**ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE**
**Fakulta elektrotechnická**
**Katedra mikroelektroniky**

# Diplomová práce

**Rozpoznávání textu a znaků s využitím vývojového prostředí ST Microelectronics „Discovery kit with STM32F746NG MCU“**

**Text and character recognition using the development environment ST Microelectronics „Discovery kit with STM32F746NG MCU“**

Studijní program: Elektronika a komunikace
Studijní obor: Elektronika
Vedoucí práce: prof. Ing. Miroslav Husák, CSc.

**Bc. Michal Lánský**

**Praha 2020**

# ZADÁNÍ DIPLOMOVÉ PRÁCE

## I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Lánský** Jméno: **Michal** Osobní číslo: **435010**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra mikroelektroniky**

Studijní program: **Elektronika a komunikace**

Studijní obor: **Elektronika**

## II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce:

**Rozpoznávání textu s využitím vývojového prostředí s STM32F746NG MCU**

Název diplomové práce anglicky:

**Recognition of the text, using the development environment with STM32F746NG MCU**

Pokyny pro vypracování:

1. Proveďte rozbor stávajícího stavu poznání o řešení rozpoznávání textu (hardware, metody a způsoby řešení, software, zpracování informace včetně ukládání, porovnávání podle vzorů uložených v databázi).
2. Navrhněte a realizujte laboratorní vzorek systému pro identifikaci textu s využitím „Discovery kit with STM32F746NG MCU" od firmy ST Microelectronics. Při návrhu software řešení se zaměřte na rozlišovací schopnosti, rychlost a chybovost vyhodnocování informace. Při návrhu využijte možnosti poskytované vývojovým kitem.
3. Vyhodnoťte dosažené parametry a shrňte základní poznatky dosažené při návrhu a realizaci systému.

Seznam doporučené literatury:

1. Neumann, P., Uhlíř, J.: Elektronické obvody a funkční bloky (I, II), ČVUT 2001
2. Chaudhuri, A., Mandaviya, K., Badelia, P., K Ghosh, S. Optical Character Recognition Systems for Different Languages with Soft Computing, Springer, 2017.
3. Yasser Alginahi (2010). Preprocessing Techniques in Character Recognition, Character Recognition, Minoru Mori (Ed.), http://www.intechopen.com/books/characterrecognition/preprocessing-techniques-in-character-recognition

Jméno a pracoviště vedoucí(ho) diplomové práce:

**prof. Ing. Miroslav Husák, CSc., katedra mikroelektroniky FEL**

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) diplomové práce:

Datum zadání diplomové práce: **15.02.2019** Termín odevzdání diplomové práce: _____

Platnost zadání diplomové práce: **20.09.2020**

| prof. Ing. Miroslav Husák, CSc.<br>podpis vedoucí(ho) práce | podpis vedoucí(ho) ústavu/kated... | prof. Ing. Pavel Ripka, CSc.<br>podpis děkana(ky) |
|---|---|---|

## III. PŘEVZETÍ ZADÁNÍ

Diplomant bere na vědomí, že je povinen vypracovat diplomovou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v diplomové práci.

Datum převzetí zadání _____ Podpis studenta _____

**Abstrakt:**
Diplomová práce se zabývá strojovým rozpoznáním textu s použitím vývojového kitu STM32F746NG. V první části je popsána teorie rozpoznávání textu a současný stav tohoto technologického odvětví. Druhá část řeší praktický návrh jednoduchého OCR implementovaného na zmíněném vývojovém kitu a popisuje realizaci použitých algoritmů. Práce je zakončena otestováním a proměřením základních parametrů, zhodnocením dosažených výsledků a uvedením možného postupu práce.

**Klíčová slova**
Rozpoznávání textu, OCR, zpracování obrazu, předzpracování, porovnávání šablon, vývojový kit, Embedded.

**Abstract:**
This master thesis deals with machine text recognition using the STM32F746NG development kit. The first part describes the theory of text recognition and the current state of this scientific branch. The second part solves practical design of simple OCR implemented on the given development kit and describes the implementation of used algorithms. The work is concluded by testing and measuring the basic parameters and evaluating the achieved results.

**Keywords**
Text recognition, OCR, image processing, pre-processing, template matching, developer board, Embedded.

# Prohlášení

Prohlašuji, že jsem diplomovou práci rozpoznávání textu vypracoval samostatně a použil k tomu pouze literaturu, kterou uvádím v seznamu přiloženém k diplomové práci.

Nemám námitky proti půjčování, zveřejnění a dalšímu využití práce, pokud s tím bude souhlasit katedra mikroelektroniky.

V Praze dne ……

……..
podpis studenta

# Poděkování

Děkuji svému vedoucímu diplomové práce prof. Ing. Miroslavu Husákovi, CSc. za čas, který mi věnoval při zpracovávání této diplomové práce. Dále bych chtěl poděkovat své rodině a kamarádům za podporu během celého studia.

# Seznam obrázků

# Seznam tabulek

# Seznam zkratek a symbolů

**Zkratky**

| | |
|---|---|
| OCR | Optical character recognization – Optické rozpoznávání symbolů. |
| RGB | Označení barevného formátu Red, Green, Blue – Červená, zelená, modrá. |
| C# | C sharp – Název programovacího jazyku |
| PDF | The Portable Document Format |
| microSD | Micro Secure Digital |

**Symboly**

| | |
|---|---|
| $R$ | Hodnota červené barvy [-] |
| $G$ | Hodnota zelené barvy [-] |
| $B$ | Hodnota modré barvy [-] |
| $m$ | Průměrná hodnota pixelů v okně [počet pixelů] |
| $s$ | Standardní odchylka okna |
| $S$ | Dynamicky rozsah variace [počet pixelů] |
| $R$ | Konstanta binarizace [-] |
| $k$ | Konstanta binarizace [-] |
| $px$ | Pixel |
| $pt$ | Obrazové body [-], 1 pt = 0,35 mm |
| $N_{Minimal}$ | Minimální hodnota šedé stupnice v obrazu [-] |
| $N_{Maximal}$ | Maximlní hodnota šedé stupnice v obrazu [-] |
| $\theta$ | Úhel natočení obrázku [°] |
| $x_{Horizontální}$ | Souřadnice pixelu v obrázku [-] |
| $y_{Vertikální}$ | Souřadnice pixelu v obrázku [-] |
| $x_{Posunutí}$ | Vzdálenost posunutí v horizontální ose [počet pixelů] |
| $y_{Posunutí}$ | Vzdálenost posunutí ve vertikální ose [počet pixelů] |
| $x_{Roztažení}$ | Faktor roztažení či zkrácení v horizontální ose [-] |
| $y_{Roztažení}$ | Faktor roztažení č zkrácení ve vertikální ose [-] |
| $Plocha$ | Velikost obrázku v pixelech |
| $Dm$ | Počet hodnot šedé stupnice |

# Obsah

# Úvod

V současnosti ve světě stále postupuje trend modernizace a automatizace. Automatizace jako proces nahrazení člověka přístrojem spočívá v převedení co nejvíce úkonů na stroj. Aby toto bylo možné, je potřeba stroji poskytnout schopnost reagovat na okolní prostředí. Z tohoto důvodu se v dnešní době velmi rozvijí zájem o algoritmy umožňující rozpoznávat objekty za pomocí kamery, či jiného senzoru. Pod skupinou těchto algoritmů jsou algoritmy zabývající se rozpoznáváním textu, ve světě známé jako OCR „Optical character recognization“. Cílem rozpoznávání textu je identifikovat text v okolí a převést ho do textové podoby v nějaké elektronickém zařízení, a umožnit tak člověku s textem dále provádět dodatečné operace. Využití takových to systémů je možné například při rozpoznávání SPZ automobilů, archivaci tištěných dokumentů, atd…

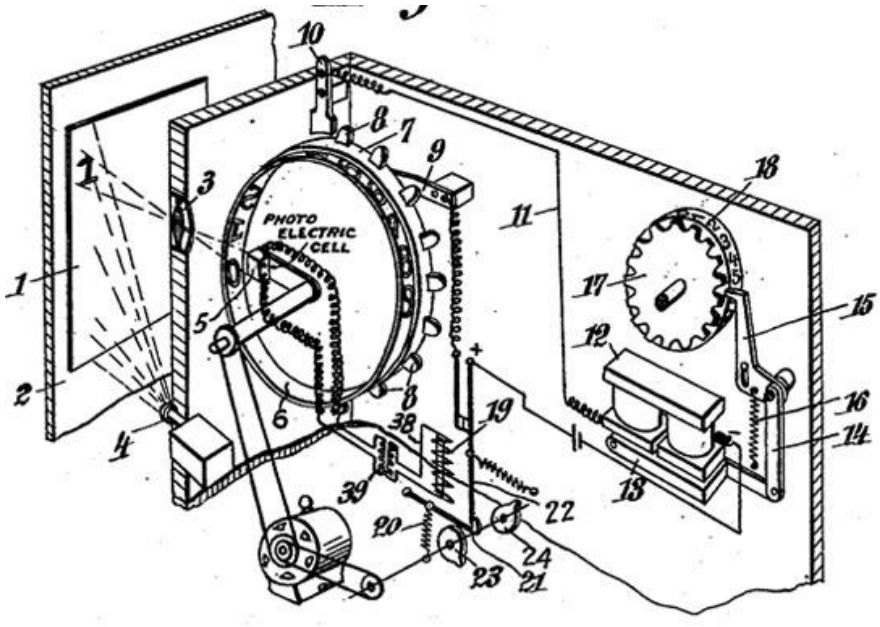
Tato práce se zabývá tématem rozpoznávání textu. V první části je uvedena sumarizace základních informací o rozpoznávání textu včetně historického vývoje, současného stavu, popisu typických metod a běžných parametrů. V druhé části je popsán postup vlastní implementace systému pro převod textu v omezených podmínkách daným vývojovým kitem „Discovery kit with STM32F746NG MCU“. Ve třetí části jsou shrnuty dosažené parametry jak jednotlivých metod, tak celého vyvinutého OCR systému. Poslední částí je návrh dalšího postupu této práce.

# 1 Teorie rozpoznání textu

## 1.1 Historický vývoj

Rozpoznávání textu je dynamicky se vyvíjející vědní obor, který má své základy již v 19 století. První pokusy o čtení textu za pomoci čtecího zařízení byly zaznamenány již v roce 1870, kdy C.R. Carey z Bostnu Massachussetts vytvořil sítnicový skener, který přenášel obrázky pomocí struktury z fotovoltaických článků. Další důležitý krok bylo objevení sekvenčního skeneru panem P. Nipkowem, jehož výtvor dal základ pro televize a čtecí zařízení [1] [2].

V roce 1929 byl v Německu podán první patent zabývající se rozpoznáváním textu. Jednalo se o prototyp čtecího zařízení, které vymyslel australský vědec Gustav Tausheck. Jednalo se o mechanický přístroj využívající principu porovnávání znaků. Zpracovávaný text byl osvětlen a umístěn před optickou čočku. Ta usměrňovala obraz písmene skrz otvory v rotujícím disku přímo do fotodetektoru. Otvory byly ve tvaru písmen a pokud nastalo zakrytí celého otvoru, systém zaznamenal dané písmeno [3]. Motivace v této době byla především snaha pomoci zrakově postiženým jedincům. Na několik let byl toto vrchol rozpoznávání textu, jelikož technologie neumožňovala výraznější vývoj. Až polovině 20. století došlo k technologické revoluci, která vedla k rozvoji elektroniky a zpracování dat. Dosavadní stroje se začaly stávat komerčně dostupné a začaly vznikat nové prototypy. První opravdové čtecí zařízení splňující dnešní definici OCR vzniklo v roce 1954. Toto zařízení bylo použito pro převod zpráv o prodeji do děrových karet, které se následně vkládaly do počítače. Následně vývoj postupoval až do současnosti ve třech až čtyřech generacích v závislosti na zvolené literatuře [1] [2].




*Obrázek 1 Taucheckovo čtecí zařízení [3].*

Každá etapa OCR systémů je charakteristická různými vlastnostmi, především účinností, dostupností a počtem podporovaných fontů [2].

### 1.1.1 První generace

Do této skupiny spadají OCR vytvořené v rozmezí od 1960 do 1965. Jedná se o komerční systémy, které využívaly velmi omezené tvary jednotlivých písmen. Ve většině případů byly symboly navrženy speciálně pro daný stroj a z počátku vykazovaly špatnou čitelnost. Mezi nejznámější systémy patřily NCR 420 a Farrington 3010. První široce rozšířený komerční OCR systém této doby byl IBM 1287, který fungoval pro převádění speciálního IBM fontu. Po čase se začaly zavádět více fontová zařízení, která byla schopná rozlišovat více fontů. Nevýhodou této etapy bylo omezení počtu znaků, která bylo zařízení schopné rozeznat. To bylo závislé především na typu použitého algoritmu. V této době byly OCR systémy založené na principu zvaném porovnávání šablon, který spočívá v porovnávání jednotlivých znaků v obrázku s databází znaků a hledá nejlepší shodu [1] [4].


*Obrázek 2 Font OCR-A nahoře a font OCR-B dole [1].*

### 1.1.2 Druhá generace

Čtecí zařízení druhé generace se objevily v polovině 60. let 20. století. Tyto zařízení byly schopny rozeznávat strojově vytištěné znaky a také již ručně psané tiskací znaky. Ručně psané znaky byly omezeny na číslice, několik písmen a pár speciálních symbolů. Na světovém veletrhu v New Yorku v roce 1965 vznikl první systém tohoto druhu. Toshiba vyvinula automatický třídící stroj pro řazení zásilek na poště podle směrovacího čísla a Hitachi vytvořilo první OCR s vyšším výkonem a menší cenou, který si mohla dovolit širší veřejnost [4].

Po roce 1965 začaly vznikat světové standardy pro zpracování textu. Jejich cílem bylo nastavit globální používání určitých fontů a stylů textu s ohledem na dobrou čitelnost pro lidi i pro stroje. První vznikl Americký standard OCR-A, který se zaměřil důrazněji na strojovou čitelnost. Později vznikl Evropský standard OCR-B, který již zaváděl fonty člověku více přirozené, než měl Americký protějšek [1]. Oba fonty jsou vidět na obrázku 2.

### 1.1.3 Třetí generace

Poslední historická generace produkovala OCR systémy v roce 1970. Jejich cílem byla snaha o zlepšení čitelnosti textu z dokumentů se špatnou kvalitou a ručně psanými znaky, při udržení nízké ceny a vysoké výkonosti. Díky pokroku v hardwaru se začaly rozvíjet mnohem sofistikovanější zařízení. Největší výhodou OCR zařízení v tomto období, bylo odstranění nutnosti používat speciální font což vedlo k možnosti převádět dokumenty, v té době běžně napsané na psacím stroji [4].

Po roce 1986 klesly ceny procesorů, čímž došlo k zvýšení prodejů OCR systémů. Tyto systémy měly již dobrou přesnost čtení, podporovaly více běžných fontů a dosahovaly nižších cen oproti předešlým systémům [1].

*Tabulka 1 Souhrn historického vývoje OCR systémů [1].*

| | |
|---|---|
| 1870 | První pokusy s rozpoznáním textu |
| 1940 | Návrhy OCR |
| 1950 | Objevují se první OCR systémy |
| 1960 - 1965 | První generace |
| 1965 - 1975 | Druhá generace |
| 1975 - 1985 | Třetí generace |
| 1986 | Rozšíření OCR mezi lidi |

## 1.2 Současnost

Po roce 1990 se začal vývoj OCR ubírat jiným směrem než doposud. Pozvolna se začalo upouštělo od speciálních hardwarových řešení a spíše se přecházelo na kompaktní softwarová řešení. V současnosti OCR systémy fungují jak na osobních počítačích, výkonných serverech tak i na mobilních zařízeních. Podle výsledné aplikace, jsou OCR vyvíjeny na různých platformách tak, aby klíčové parametry dosahovaly nejlepších hodnot.

Se stále lepším výpočetním výkonem a dokonalejším technickým vybavením, jako jsou digitální kamery nebo skenery, došlo k uzpůsobení OCR algoritmů, aby byly schopné zpracovat větší množství stupních dat. Algoritmy se stávají komplexnějšími a využívají sofistikovanějších metod přístupů jako například využití umělé neuronové sítě, Skrytého Markovova modelu nebo fuzzyho rozhodování. Pro zpracování obrazu se v současné době využívají pokročilé techniky strojového učení a dnešní OCR spadají do vědní kategorie počítačového vidění a bývá označováno jako „text recognition“ rozpoznání textu. Uplatnění OCR systémů již není cíleno pouze na rozpoznávání dokumentů, ale nachází své uplatnění při rozpoznávání libovolného textu v nehomogenním, neohraničeném prostředí jako například převedení SPZ automobilů, či popisných čísel domů atd [5].

### 1.2.1 Hardwarové řešení

V OCR technologiích lze hardware rozdělit do dvou sekcí. První sekce je zaznamenávací zařízení, které zachytí daný obraz textu ať už z okolí, nebo z dokumentu v papírové formě. V praxi se jedná o kamery, fotoaparáty nebo skenery. Cílem těchto komponent je zachytit co možná nejjasnější, čistější obraz daného textu. Pokud by dané obrazy byly rozmazané či matné, výsledné rozeznání jednotlivých písmen by nemuselo proběhnout správně. To, zda použijeme fotoaparát nebo skener záleží na dané aplikaci použití. Pro převod tištěného dokumentu se zpravidla používají skenery, protože celý obraz je rovnoměrně převeden se stejnými parametry. V případě, že chceme rozpoznávat text z okolí je nutné použít kameru nebo fotoaparát. Důležité je vybrat správnou kameru k OCR algoritmu. Čím lepší zaznamenávací zařízení, tím je sice kvalitnější vstupní obrázek pro algoritmus, ale také tím sofistikovanější musí být samotný algoritmus, aby zvládl daný převod uskutečnit. Proces převodu pro nekvalitní algoritmus a obrázek se spoustou pixelů může trvat výrazně delší dobu.

Druhá sekce hardwaru poskytuje výpočetní sílu pro daný OCR algoritmus. Jednotlivé dílčí kroky algoritmu bývají výpočetně náročné, a tudíž je důležité správně zvolit typ hardwaru. Ten jako takový má vliv především na rychlost zpracování daného algoritmu. OCR algoritmus je založen na zpracování obrazu, tudíž čím větší obraz a lepší rozlišení máme, tím více pixelů je obsaženo v obrazu, a proto jednotlivé metody algoritmu budou trvat déle. Delší dobu se tvořily

OCR systémy pouze na počítačích, ale po čase začaly vznikat i na jiných platformách. V dnešní době lze nalézt aplikace fungující na osobních počítačích, mobilních zařízeních či výkonných serverech. To, jaký hardware se použije záleží na aplikaci dané ho OCR. Reálně nelze zvolit ideální řešení, jelikož mezi parametry existuje rovnováha [6] [7].


*Obrázek 3 Princip fungování OCR na serveru [8].*

### 1.2.2 Softwarové řešení

Software dnešních moderních OCR se dá charakterizovat různě. Z hlediska použitého programovacího jazyka OCR systémy nemají žádné hranice. V dnešní době existuje několik provedení OCR systémů psaných jak v jazycích C, C++ tak i C#, Java, Javascript. Každý ze zmíněných jazyků má své uplatnění dané především platformou. Na mobilních zařízení se vyskytují OCR především psané v Javě. OCR systémy využívající webové rozhraní využívají výhod javascriptu a počítačová řešení jsou psaná v jazycích C++ nebo C#.

Z hlediska použitého algoritmu většina aktuálních OCR systémů využívá konvenční rozdělení převodu na preprocessing, klasifikaci písmen a postprocessing. Tyto tři fáze převodu jsou detailně rozebrány v kapitole 2. Použité individuální metody pro OCR se již ovšem liší systém od systému. Některé firmy se snaží docílit co nejdokonalejšího předzpracování tak, aby se vyvarovaly chybám při převodu. Některá provedení OCR naopak počítají se vznikem většího množství chyb a snaží se vzniklé chyby opravovat v koncové fázi převodu. Neexistuje žádná norma ani ideální cesta, protože OCR algoritmus pro správné fungování musí být uzpůsoben přesně na míru danému použití. V dnešní době existuje několik dostupných funkčních OCR

systémů. Některé z nich jsou dostupné pouze komerčně, ale existuje i pár nekomerčních OCR, která jsou dostupná zdarma.

***OmniPage*** **–** Jedná se o komerční OCR aplikaci zprostředkovanou vývojáři z Nuance Communication. Tento systém byl první OCR program běžící na osobních počítačích. OmniPage se specializuje na převod psaných dokumentů do elektronické podoby. Podporuje rozpoznávání více jak 100 jazyků bez nutnosti specifikace daného jazyka. Další výhodou je možnost využívání cloudového uložiště. To zajišťuje snadný přístup k systému ze vzdálených zařízení. Výstupní dokument systému lze nastavit do několika formátů např: MicrosoftWord, Excel, PowerPoint, HTM, … [5].

***Abbyy FineReader*** **–** Komerční OCR vytvořený vývojáři z ruské firmy ABBY. Program umožňuje konverzi obrazů dokumentů do elektronické podoby jako je MicrosoftWord, Excel, PowerPoint, HTML atd. Jedná se o placený nástroj, jehož licenci použila řada velkých firem jako Fujitsu, Panasonic, Xerox, Samsung [9].

***Adobe Reader*** **–** Adobe reader je aplikace pro práci s PDF dokumenty, určená pro jednoduché sledování, tištění a komentování dokumentů. Placená PRO verze obsahuje systém OCR. Díky cloudovému provedení, je možné mobilním zařízením vytvořit fotku a implementovaný OCR systém se pokusí nalézt v obrazu text a převést jej do nového PDF dokumentu.

***GOCR*** **–** OCR program který založil Jörg Schulenburg. Je použitelný k převodu obrázků s textem na textový soubor v počítači. Hlavní výhoda je univerzálnost systému, díky které je možné snadno daný systém napojit na různá uživatelská rozhraní. Program je použitelný zcela zdarma pod podmínkou nekomerčního šíření [10].

***Google*** **–** Google do svých cloudových serverů „Google Drive“ také zabudoval OCR systém. Služba je přidána do Google Docs v roce 2010. Zpočátku systém podporoval pouze hlavní světové jazyky, ale po pár letech přidal podporu dalších jazyků včetně češtiny [11].

***Tesseract*** **–** OCR systém pracující na několika operačních systémech. Jedná se o free open source projet, který lze bez limitace využívat a dále rozvíjet. Tesseract byl vyvinut v laboratořích Hewlett Packard a Greeley v Coloradu v letech 1985 až 1994. V roce 1996 se zavedla podpora pro Windows a v roce 1998 došlo k převodu z programovacího jazyka C do C++. V roce 2005 byl změněn status projektu na open source za podpory společnosti Google. V roce 2006 byl Tesseract zhodnocen jako jeden z nejrychleji rostoucích open source projektů zabývající se OCR [12].

Aplikace podporuje více než 100 jazyků, ovšem je nutné uživatelem zvolit o který jazyk se jedná. Novější verze tesseractu jsou založeny na principu neuronových sítí. To zajišťuje vyšší přesnost převodu textu na úkor výpočetní náročnosti. Projekt pro trénování využívá více jak 3 tisíc řádků textu ve více jak 4 tisíc fontech. I přes takto obrovské množství dat je uživatel nucen, speciálně při snaze přidat nový font, trénovat síť na vlastních datech [5].

### 1.2.3 Typické vlastnosti OCR

Dnešní OCR systémy se z technického hlediska dají popsat několika parametry. Z hlediska převodu daného textu do digitální podoby jsou nejdůležitějšími parametry hodnoty rozpoznání, odmítnutí a chybovost. Pro uživatele důležitý parametr je rychlost daného převodu a rozlišovací schopnost vzhledem k počtu podporovaných fontů [1]. Moderní OCR cílí na globální využití po celém světě, a proto jedním z parametrů je také množství podporovaných jazyků.

- **Hodnota rozpoznání** z anglického „Recognition rate“ je poměr úspěšně rozpoznaných písmen ku celkovému počtu.
- **Hodnota odmítnutí** z anglického „Rejection rate“ je poměr písmen, která OCR systém nedokázal rozpoznat, ku celkovému počtu písmen v daném textu.
- **Chybovost** z anglického „Error rate“ je poměr špatně převedených písmen ku celkovému počtu písmen. Tato hodnota se dá považovat za chybovost daného systému.
- **Rychlost převodu** je hodnota času udávající, jak dlouho trvá provést kompletní algoritmus rozpoznání textu. Tento parametr je závislý na využívaném hardwaru.
- **Rozlišovací schopnost** udává, jak moc je systém schopný reagovat na různá vstupní data například různé typy a velikosti fontů.
- **Podporované jazyky –** jedná se o seznam světových jazyků, které OCR dokáže převést. Moderní OCR dokáží sami rozeznat o jaký jazyk se jedná a podle toho uzpůsobit následující algoritmus rozpoznání.

### 1.2.4 Využití OCR

Rozpoznávání textu je využíváno v mnoha odvětvích a lze dělit podle typu vstupních dat. Specializované OCR systémy, které nají omezenou vstupní sadu dosahují vyšší přesnosti a rychlosti převodu. Mezi tyto aplikace lze zařadit například bankovní aplikace, kde aplikace načítají číslo účtu a následně zákazníka identifikují. Omezená množina znaků také snižuje nároky na kvalitu vstupních dat. Druhou skupinou jsou systémy využívající textový vstup. Pro převádění textu je důležitý typ písma a kvalita tištěného dokumentu. Jelikož dokumenty obsahují velké množství textu, rychlost takového převodu je zpravidla nižší. Mezi takovéto systémy lze zařadit kancelářské aplikace například použití na poštách při elektronické archivaci dokumentů. [5]

**Fakturace** je nedílnou součástí každodenního života mnoha firem i jedinců. Z hlediska zákona je nutné uchovávat své faktury několik let zpět v čase. V případě velkých firem se může jednat o obrovské množství dat. OCR umožňuje provádět jednoduší sběr dat a jejich dodatečnou analýzu [13].

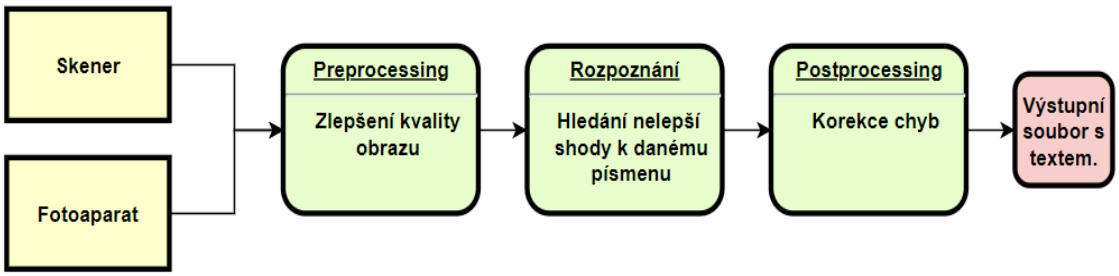
**Captcha** je typ Turingova testu, který se běžně používá k určení, zda danou aktivitu vykonává člověk nebo stroj. Jedná se o obranu proti hackerům, kteří se snaží získat citlivá data z webů pomocí opakujících se programů, jako například nalezení hesla zkoušením hesel ze slovníků. V Captcha je uživateli zobrazen obrázek obsahující několik písmen nebo čísel a uživatel je nucen přepsat dané znaky, aby došlo k potvrzení, že se jedná o člověka a ne stroj. Pokročilé OCR lze použít pro překonání této obrany. Bývá využito ve stahovacích aplikacích, kde dojde k automatickému vyplnění [13].

**Zdravotnictví** také zaznamenává zvýšení používání OCR technologií. Doktoři musí ze zákona všechny informace o pacientech udržovat a archívovat. To při počtu lidí a množství dat je velice komplikované, a proto se využívá OCR systémů pro převedení informací do elektronické databáze. Následně je možné efektivně přistupovat k datům pomocí jednoduchého rozhraní [13].

**Identifikace vozidel** je také oblast, ve které OCR systémy nacházejí své uplatnění. Kamery na dálnicích měří rychlost projíždějících automobilů a v případě překročení povolené rychlosti dojde k pořízení fotografie poznávací značky. Kamera bývá velice kvalitní, aby se zajistilo, že fotka není rozmazaná. Systém následně se rozpoznají daná čísla a písmena na značce a výsledek se odešle do databáze. Z té se následně vrátí profil daného uživatele a automaticky se mu pošle pokuta [13].

# 2 Algoritmus rozpoznání textu

V této kapitole jsou rozebrány jednotlivé části algoritmu pro rozpoznání textu. Ve světě existuje spousta OCR systémů, využívajících různé metody a postupy. Všechny tyto postupy využívají základní strukturu tvořenou třemi kroky. Prvním a zároveň velice důležitým krokem je předzpracování (preprocessing). Na vstupu tohoto kroku jsou data, typicky digitální fotografie textu, ze které chceme rozpoznat text a převést ho do textového souboru. V rámci předzpracování se snažíme docílit zlepšení kvality obrázku tak, aby následné rozpoznání bylo jednodušší. Druhým krokem algoritmu je rozpoznání jednotlivých písmen a jejich klasifikace. Klasifikace spočívá v analýze jednotlivých písmen v obrázku a následně v přiřazení odpovídajícího písmene. Opět ve světě existuje několik metod, které lze použít jako například: porovnávání šablon, statistické postup nebo neuronové sítě využívající strojové učení. Každá ze zmíněných metod má své výhody a nevýhody a postupuje k řešení klasifikace písmen různými postupy. Jakmile proběhne proces rozpoznání přes všechny písmena v obrázku, dojde k poslednímu kroku, kterému se ve světě říká „post-processing“. Cílem tohoto kroku je nalézt všechny chyby vniklé špatným rozpoznáním a pokusit se je opravit [5] [13].




*Obrázek 4 Algoritmus převodu OCR.*

## 2.1 Preprocessing

Existuje mnoho faktorů ovlivňující přesnost převodu textu OCR systémů. Patří mezi ně například kvalita skeneru či fotoaparátu, typ tištěného dokumentu, kvalita samotného povrchu, na kterém je text napsán, typ fontu a přítomnost extra objektů v okolí atd. Při špatném zachycení textu fotoaparátem může dojít k výskytu různých defektů jako rozmazaných míst, šumu, nekonzistentního jasu, atd Všechny tyto zmíněné jevy znesnadňují OCR systémům rozeznat text a znesnadňují celkový převod. Cílem předzpracování je pomocí určitých technik snížit množství šumu a ostatních defektů v obrazech a provést normalizaci a kompresi dat tak, aby z této procedury vyšel čistý obraz, který pak může být dále efektivně rozpoznáván. Dalším krokem předzpracování je rozdělit obraz na pozadí a text. V této kapitole jsou uvedeny nejběžnější metody, které upravují kvalitu obrazu pro jeho následné rozpoznání. [14]

### 2.1.1 Gray Scale

V dnešní době již snímky nebývají černobíle nýbrž využívají plnou stupnici barev. Pro účely zpracování textu ovšem není potřeba využívat barevný obraz, spíše je to na obtíž, protože to zpracování komplikuje a zpomaluje. Z tohoto důvodu se používá metoda na převedení obrázku z barevného RGB formátu na šedý obrázek přesněji obraz reprezentovaný množstvím světla. Takovému obrázku se říká Gray Scale obrázek. Algoritmus spočívá v iteraci přes jednotlivé pixely obrázku a zprůměrování daných RGB hodnot čímž se získá výsledná hodnota na stupnici šedi. Zprůměrování je možné podle rovnice (2.1). Výsledek ovšem visuálně nedosahuje zcela rovnovážných hodnot, jak je vidět na obrázku 5b). Obrázek se nachází v tmavší části šedé stupnice a kontrast je nízký. Problém je způsoben tím, že každá barva má jinou vlnovou délku [15].

$$\text{Grayscale} = (\text{R} + \text{G} + \text{B} / 3) \tag{2.1}$$

Metoda vlnové délky neboli také svítivosti, je metoda, která právě zohledňuje vlnovou délku dané barvy a podle toho upravuje jejich poměrné zastoupení v rovnici. Při upravení poměrů vzniká rovnice (2.2), která dosahuje lepšího kontrastu. Výsledek převodu pomocí metody vlnových délek je viditelným na obrázku 5c).

$$\text{Grayscale} = (0{,}3 \cdot \text{R} + 0{,}59 \cdot \text{G} + 0{,}11 \cdot \text{B}) \tag{2.2}$$

*Obrázek 5 a) Originální obrázek, b) šedá stupnice podle rovnice (2.1), c) šedá stupnice podle rovnice (2.2) [15].*

### 2.1.2 Binarizace

Poté co je obrázek převedený v šedé stupnici, je nutné rozlišit co na fotce je text a co pozadí. K tomuto účelu slouží proces binarizace, který obraz převede na bílou a černou barvu. Tento krok je v OCR převodu velice důležitý, protože velmi ovlivňuje výsledky převodu. Existuje několik různých implementací tohoto algoritmu lišících se podle komplexnosti provedení a použitelnosti. Reálně se nedá učit nejlepší algoritmus, protože daný výsledek je odlišný pro různá vstupní data. Obecně se binarizační algoritmy dělí na globální a lokální [16].

**Globální binarizace** používá jednu prahovou hodnotu pro celý obrázek. Nejjednodušší možností je algoritmus s fixní prahovou hodnotu. Fixní prahová hodnota je navolena předem uživatelem a určuje, kdy je daný pixel označen jako černý a kdy jako bílý. Trochu složitějším algoritmem je metoda binarizace Otsu. Ta na rozdíl od fixní metody prahovou hodnotu získává automaticky pomocí histogramové metody. Na počátku se provede výpočet počtu pixelů pro každou úroveň šedé stupnice. Poté se vyberou dvě úrovně s největším zastoupením a mezi nimi se nastaví výsledná prahová hodnota. Třetí používanou metodou je algoritmus zvaný „Kilter and Illingworth“. Tato metoda využívá Gausovské rozložení [16].

**Lokální binarizace,** na rozdíl od globální, počítá vlastní prahovou hodnotu pro jednotlivé oblasti v obrázku. Jednou z možných implementací tohoto druhu algoritmu je Adaptivní binarizace. Tento algoritmus rozdělí obrázek na skupiny NxN pixelů a vypočítá prahovou hodnotu pro každou oblast individuálně. Tento postup dosahuje lepších výsledků obzvlášť pokud vstupní obraz obsahuje stíny, rozostřené oblasti, nízké rozlišení a nejednotné osvětlení. Nejpoužívanější metody jsou Niblackova, Sauvolova nebo Bersenova metoda [16].

**Niblackova metoda** počítá prahovou hodnotu pro dané okno zprůměrováním a standartní odchylkou v oblasti okna jednotlivých pixelů. Postup je popsán rovnicí (2.3), kde m je průměr hodnot pixelů v lokálním okně, s je standartní odchylka okna a k je konstanta stanovena autorem algoritmu na 0,2 [16].

$$Prahová\ hodnota_{Niblack} = m + k \cdot s \tag{2.3}$$

**Sauvolavola metoda** je modifikací Niblackovy metody snažící se o zlepšení výkonu a efektivnosti. Algoritmus je definován podle rovnice (2.4), kde m je průměr, S je dynamický rozsah variance. Hodnoty R a k jsou konstanty stanovené autorem na k = 0,5 a R = 128.

$$Prahová\ hodnota_{Sauvolova} = \mathrm{m} \cdot \left(1 - k \cdot \left(1 - \frac{S}{R}\right)\right) \tag{2.4}$$

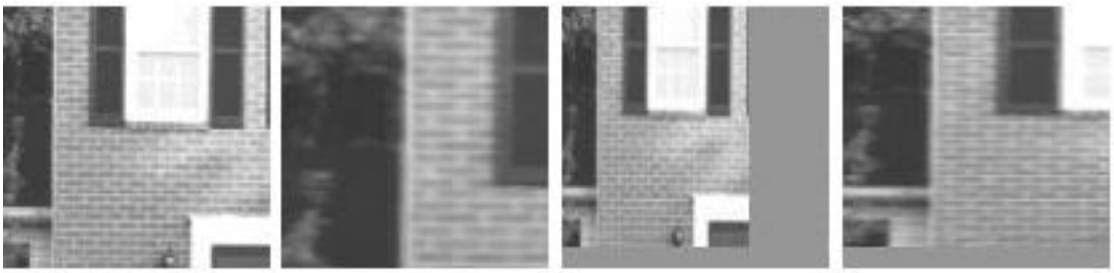
**Bersenova metoda** je lokální binarizační algoritmus který vypočítává prahovou hodnotu daného okna zprůměrováním nejmenší a největší hodnoty šedé stupnice v daném okně. Rovnice (2.5) popisuje postup získání prahové hodnoty použitím Bersenova algoritmu.

$$Prahová\ hodnota_{Bersen} = \frac{(N_{Minimal} + N_{Maximal})}{2} \tag{2.5}$$

### 2.1.3 Geometrické modifikace

Mezi základní operace s obrazy patří také rotace obrazu, roztažení či smrštění a posunutí. Tyto úkony spadají do kategorie geometrických modifikací a v rámci OCR převodu jsou běžně využívané pro upravení vstupního obrázku či nějaké části v průběhu převodu.

Posunutí funguje na principu přesunutí určitého počtu pixelů ve vertikálním či horizontálním směru. Tato metoda není v rámci převodu textu příliš potřebná. Roztažení či smrštění obrázků je činnost, při které se mění celková velikost objektů v obrázku. Jednotlivé objekty se mohou roztáhnout či smrštit v horizontální či vertikální ose. jak je znázorněno na obrázku 6. Taktéž je možné provádět v jedné ose roztažení a v druhé smrštění [17] [14].



*Obrázek 6 a) Originální obrázek, b) roztažení v horizontální a vertikální ose, c) smrštění v horizontální i vertikální ose, d) roztažení v horizontální a smrštění ve vertikální ose [17].*

Rotace je poslední geometrickou modifikací, která se využívá v rámci OCR. V případě, že byl vstupní obraz při skenování či focení natočen, může dojít k problém s rozpoznáním textu, jelikož algoritmy většinou předpokládají, že řádky textu jsou na obrázku kolmo k hraně. Z tohoto důvodu se provádí detekce úhlu náklonu vyfoceného textu a v nutnosti se provede rotace obrazu. Příklad rotace je vidět na obrázku 7 [14]. Existuje několik algoritmů pro detekování tohoto úhlu a jsou založené na různých principech. Nejpoužívanějším je Houghonova transformace [17].

*Obrázek 7 a) Originální obrázek, b) natočený obrázek pomocí rotace [17].*

Všechny zmíněné geometrické operace lze implementovat zvlášť, nebo využít výhody rovnic (2.6) a (2.7), které provádí všechny operace pro daný pixel současně.

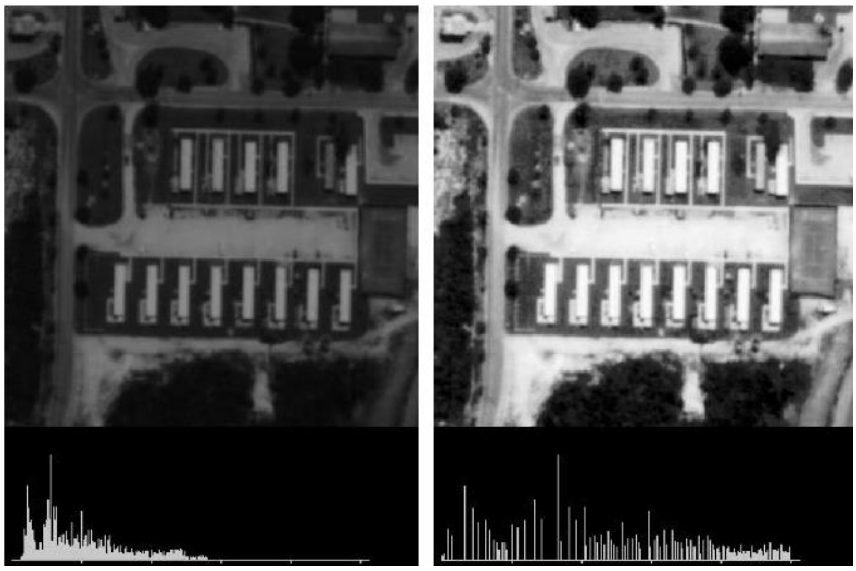
$$X = x \cdot \cos\theta + y \cdot \sin\theta + x_{posun} + x \cdot x_{roztažení} \tag{2.6}$$

$$Y = y \cdot \cos\theta + x \cdot \sin\theta + y_{posun} + y \cdot y_{roztažení} \tag{2.7}$$

Hodnoty $x_{posun}$ a $y_{posun}$ určují o jakou vzdálenost se mají pixely posunout. Hodnota $x_{roztažení}$ a $y_{roztažení}$ provádí roztažení daného pixelu a cos a sin provádí rotaci o úhel $\theta$.

### 2.1.4 Histogramové srovnání

V případě, že vstupní obraz má špatný kontrast je vhodné použít metody, které daný kontrast upraví. Mezi tyto metody patří algoritmus histogramové srovnání. Ten je založen na vytvoření histogramu z obrázku, který je ve formátu šedé stupnice. Histogram se vytvoří sečtením počtu pixelů jednotlivých úrovní šedé stupnice. Ukázka takového histogramu je na obrázku 8a). Kontrast obrázku se nachází v tmavé polovině šedé stupnice, tudíž některé objekty na obrázku splývají. Tento problém lze odstranit využitím histogramového srovnání [17], které lze popsat rovnicemi (2.8) a (2.9).



*Obrázek 8 a) Obrázek se špatným kontrastem a jeho histogram,*
*b) obrázek po aplikaci histogramového srovnání a jeho histogram [17].*

Algoritmus spočívá ve vypočtení hustoty pravděpodobnosti výskytu jednotlivých pixelů v obrazu. Z té lze odvodit počet prvků šedé stupnice ve výstupním obrázku. Výsledný počet by měl být menší, jelikož dochází k redukci některých hodnot za účelem zlepšení jasu. Následně se vypočte kumulativní distribuční funkce. Výsledek celé operace je dán rovnicí (2.9), kde $D_m$ je počet hodnot šedé stupnice výstupního obrázku, *plocha* je velikost obrázku, hodnota $a$ představuje hodnotu daného pixelu a $H_{Original}$ je histogram původního obrázku [17].

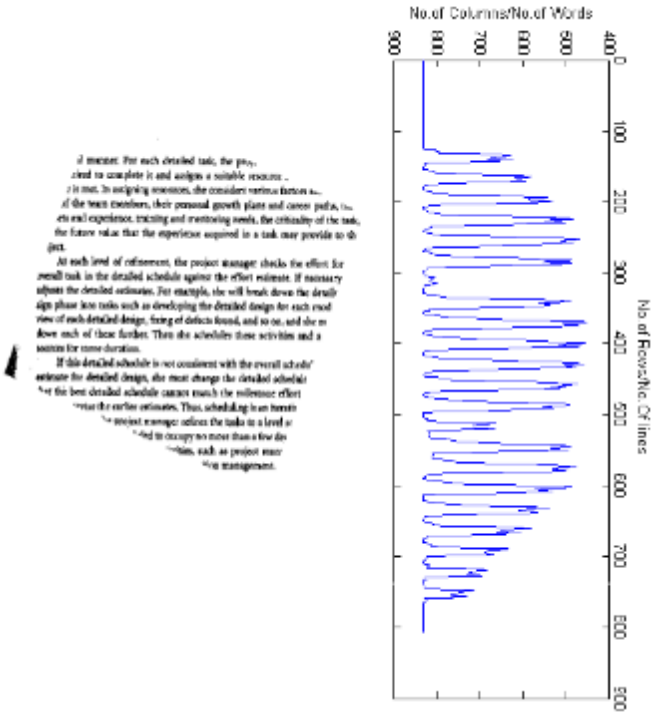
$$P_1(a) = \frac{1}{Plocha} \sum_{i=0}^{a} H_{Original}(a) \tag{2.8}$$

$$f(a) = D_m \frac{1}{Plocha} \sum_{i=0}^{a} H_{Original}(a) \tag{2.9}$$

### 2.1.5 Segmentace

Posledním krokem v rámci předzpracování je segmentace. Cílem tohoto kroku je rozdělit obraz na jednotlivé segmenty, které se budou dále zpracovávat. V rámci OCR se v prvním kroku jedná o vyhledání a rozdělení jednotlivých řádků textu v obrazu a následně rozdělení individuálních písmen v jednotlivých řádcích. Poté je na každé písmeno aplikován proces rozpoznání a klasifikace.

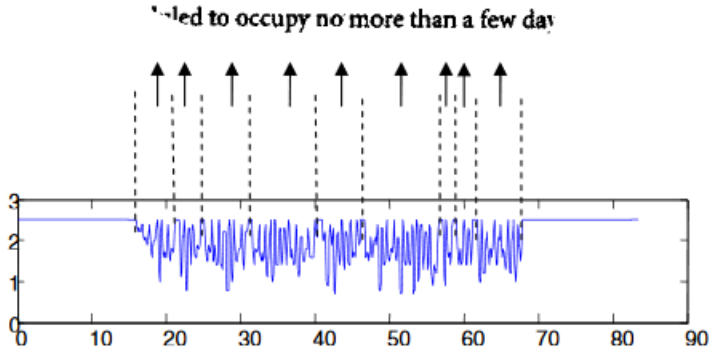
Vertikální a horizontální histogramová segmentace je jedním z používaných algoritmů pro segmentaci řádků a slov v obrázků. Segmentace řádků je založena na vertikální projekci. Na počátku se pro každý řádek na obrázku sečtou všechny černé pixely. Poté se výsledek vynese do histogramu, který je znázorněn na obrázku 9.




*Obrázek 9 Histogram vertikální segmentace řádků [5].*

Na horizontální ose je počet černých pixelů a na vertikální ose je index daného řádku pixelů v obrázku. Z výsledného histogramu vyplývá rozmístění textu. V místě, kde je hodnota černých pixelů rovna nule, či velice malé hodnotě při ne zcela kvalitním vstupním obrázku, lze prohlásit místo za prázdné bez textu. V případě míst, kde se objevují lokální maxima hodnot černých pixelů, lze daná místa prohlásit za řádky. Střední hodnota těchto špiček určuje výšku daného řádku či dokonce velikost použitého fontu. Z jednotlivých špiček lze odhadnout počet písmen obsahující háčky. Poté co se získají řádky textu, lze podobným principem aplikovat horizontální segmentaci, která řádky rozdělí na jednotlivá slova nebo dokonce i samotná písmena. Histogramové

segmentaci musí předcházet algoritmus odstraňující jiné objekty z obrázku tak, aby obraz obsahoval pouze text, jinak by došlo k problémům při detekci řádků. [5]




*Obrázek 10 Histogram horizontální segmentace písmen [5].*

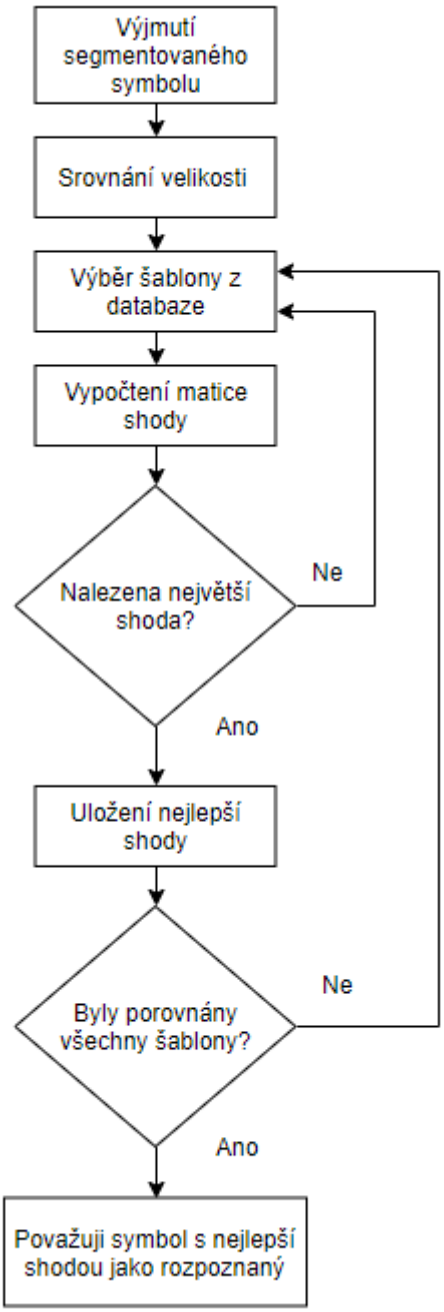
## 2.2 Rozpoznání

Velmi důležitým krokem OCR algoritmu je rozpoznání a následná klasifikace daného písmene. Častokrát se v literatuře uvádí že rozpoznání je tvořeno segmentací obrázku a následně jeho klasifikací. V této práci byla segmentace již popsána v rámci kapitoly preprocessing, a proto se tato kapitola zabývá technikami klasifikace. Hlavní smysl je získat informace o daném objektu na obrázku, v případě OCR písmeno, a na základě těchto informací ho identifikovat. Existuje spousta algoritmů, které by zde mohly být zařazeny, ale většina z nich je založena či nějak odvozena z tří základních postupů. Jedná se o porovnávání šablon, detekce rysů a metody založené na neuronových sítích.

### 2.2.1 Porovnání se šablonou

Metoda porovnání se šablonou je jedna z jednodušších metod využívaných pro klasifikaci textu. Algoritmus je popsán diagramem na obrázku 11. V prvním kroku se převezme z obrázku jedno segmentované písmeno, které se algoritmus dále snaží rozpoznat. Následně si z databáze vyčte šablonu prvního písmena a upraví jeho velikost tak, aby odpovídal segmentovanému písmenu z originálního obrázku. Poté nastane výpočet matice shody, který probíhá porovnáváním určitých sekcí v obrázku, typicky jednotlivé pixely. Algoritmus si uchovává poslední nejlepší shodu a postupuje dále. Jakmile se tento proces provede se všemi šablonami v databázi, písmeno s nejvyšší shodou je uvedeno jako klasifikované. Nevýhodou tohoto řešení je vyšší časová

náročnost a slabší schopnost rozpoznávat odlišné typy a velikosti fontů, než které používá šablona [18].




*Obrázek 11 Vývojový diagram průběhu metody porovnávání šablon.*

### 2.2.2 Detekce rysů

Dalším používaným postupem je detekce rysů. Tento mechanizmus spočívá v rozložení obrázku na menší segmenty a následné analýze jednotlivých částí odděleně. Mezi hlavní rysy patří strukturální analýza zahrnující natažení, zkroucení, rotace objektu a celkovou transformaci. Získané rysy jsou pak porovnány a program na jejich základě určuje o jaké písmeno se jedná. Detekce rysů se často používá v kombinaci s metodou porovnávání šablon nebo metodou soft computing [1].

### 2.2.3 Soft computing

Soft computing je důležitou matematickou kategorií pro rozpoznávání vzorů a umělou inteligenci. Cílem této metody je se co nejvíce přiblížit uvažováním ke člověku tím že napodobuje fungování lidského mozku. V současnosti se jedná o nejmodernější metodu využívanou v OCR systémech. Důvod proč se v nyní tato metoda používá je, protože dokáže docílit dobré kvality konverze, rychlosti a robustnosti pomocí strojového učení [4] [5].

Algoritmus se skládá z několika kroků. Prvním krokem je vytvoření modelu a sesbírání dostatečného množství dat. Následně začíná fáze trénování, kdy se systém učí korektně klasifikovat daná data. Trénování představuje důležitý krok v procesu a typicky trvá velice dlouhou dobu. Trénování se dá také rozdělit na offline learning, které probíhá před používáním daného algoritmu, nebo na online v případě, že systém provádí trénování přímo v době používání algoritmu. Další možné řazení je trénování s učitelem nebo bez učitele [5].

Aby trénování bylo úspěšné a daný model byl použitelný, je nutné správně data reprezentovat. Učení bez učitele je podmnožina algoritmů, které nepotřebují data speciálně reprezentovat a jsou schopné z neoznačených dat správně generovat výslednou množinu znaků. Takovéto algoritmy jsou použity v řadě odvětví zabývajícím se tématem soft computing, jako například počítačové vidění, zpracování dat, rozpoznávání textu, analýza trendu a clustering.

Trénování s učitelem spočívá v tom, že vstupní data mají rovnou specifikovanou danou třídu, ke které má vzor náležet. Tedy výstup trénovaného systému lze korigovat učícím algoritmem, který přizpůsobuje parametry tak, aby nejlépe odpovídaly dané třídě. Tyto úpravy jsou prováděny opakovaně a postupně zlepšují výsledek daného převodu [5].

Soft computing je velice moderní metoda, která dosahuje skvělých výsledků, ovšem nevýhodou je vysoká výpočetní náročnost. Další nutností je dlouhá doba pro natrénování daného modelu. Více informací ohledně soft computing je k nalezení v literatuře [5].

## 2.3 Postprocessing

V kapitole 2.2 jsme řešily rozpoznávání jednotlivých písmen a snažili jsme se najít nejlepší shodu jednotlivých písmen z textu a určit tak o které písmeno se jedná. Ačkoliv se metodiky rozpoznání neustále zdokonalují, v současnosti stále nedosahují 100% úspěšnosti. Předmětem této kapitoly je popsat typy jednotlivých chyb, jak tyto chyby identifikovat a jak je eliminovat. Prakticky hodnota chybovosti běžných současných OCR dosahuje hodnoty 10 – 15 %. Pokud obrázek na vstupu algoritmu má větší množství defektů jako například špatný fyzický stav, špatnou kvalitu tisku, ztrátu barvy či sešlost stářím, chybovost algoritmu roste. V situacích, kdy dojde k selhání převodu a přiřazení špatného písmene do výsledného souboru vzniká chyba. Například písmeno B může být převedeno jako číslo 8 nebo písmeno O jako 0. V OCR systémech existují dva typy chyb. Jedná se o chyby neexistujících slov a o chyby reálných slov. Oba tyto typy jsou závažným problémem a každý se řeší jiným postupem [19].

Chyba neexistujícího slova je taková, která vzniká chybou jednoho či více písmen a způsobuje, že výsledné slovo vzhledem k danému jazyku neexistuje. Například věta: „Jak se máš?“ převedena na „Jak se móš?“. Slovo móš je chyba neexistujícího slova.

Chyba reálného slova vzniká v případě, že dojde k chybnému převodu slova, ale výsledné slovo stále existuje v daném jazyce. Například věta „Podej mi maso.“ převedena na „Podej mi laso.“. Slovo laso v českém jazyce existuje, ale významově v této větě nedává smysl.

Chyby se dají dále dělit do skupin podle toho jakým způsobem chybné slovo vzniklo. V případě, že ve výsledném slově jedno či více písmen chybí, jedná se o chybu mazání. Pokud ve výsledném slově nějaké písmeno přebývá, jedná se o chybu vložení písmena. Poslední skupinou je nahrazení jednoho písmene za jiné. Kvůli často nevhodnému stavu dokumentů docházelo k častým chybám, nebo dokonce úplnému selhání převodu. Z tohoto důvodu vzniklo několik algoritmů, které se dají členit do tří hlavních skupin. Jedná se o manuální opravu chyb, opravu chyb pomocí slovníku, nebo pomocí kontextu věty [19].

### 2.3.1 Manuální korekce

Nejjednodušší způsob opravy chyb je manuální korekce. Tento způsob spočívá kontrolování výstupu OCR systému samotným člověkem. V případě nalezení chyby osoba následně provede opravu. Tento způsob byl použit kolem roku 2000 Charles Franksem. Ten vytvořil web, pomocí kterého lidi mohli procházet elektronicky uložené knihy převedené z papírové formy pomocí OCR. Dobrovolníci z celého světa mohli procházet jednotlivé knihy a hledat chyby. Po několika opakováních byl text prohlášen jako bezchybný. Manuální způsob je zbytečně pracný a nákladný a časově náročný z tohoto důvodu se příliš nepoužívá [19].

### 2.3.2 Lexikální korekce

Ve snaze najít automatický způsob detekce a opravy chyb při převodu OCR vznikla metodika založená na slovníkové korekci též nazývaná lexikální korekce. Tento postup využívá lexikon nebo slovník, pomocí kterého dochází k porovnávání jednotlivých slov a v případě nalezení chyby i k její opravě. V některých případech tato metoda vrací více možných slov, o která se může jednat. Reálně existuje mnoho komplexních lexikálních algoritmů [19].

I když se jedná o velice účinný a užitečný prvek při převodu, má své limity a nevýhody. Pro správné fungování korekce je nutné mít rozsáhlý slovník který obsahuje ideálně všechny slova daného jazyka. V případě českého jazyka existuje více jak 300000 slov s různými tvary a skloněním. To klade velké nároky na vytvoření slovníku. Druhou nevýhodou je to, že slovníky jsou cíleny na jeden daný jazyk a nepodporují více jazyčná řešení. Třetí problém je absence jmen a zkratek ve slovníkách jako například „RAM“, „CPU“. V neposlední řadě je problémem nutnost aktualizovat slovník o nová slova a případně upravovat či odstraňovat stará. Lexikální korekce se i přes nevýhody hojně používá, protože dokáže velice dobře detekovat chyby neexistujících slov [19].

### 2.3.3 Kontextová korekce

Na rozdíl od lexikální korekce se kontextová korekce zabývá sémantikou věty a její větnou skladbou. Hledá kontext jednotlivých slov, a snaží se najít chyby v gramatice a následně je opravit. Většinou není použita samostatně, ale spíše doplňuje lexikální korekci o možnost detekovat chyby reálných slov. Na základě větné skladby dokáže detekovat podstatné jméno a sloveso ve větě a snaží se hledat nekonzistentnosti. Znalost o daném jazyce je zprostředkována pomocí jazykového module. Většina algoritmů snažící se o kontextovou korekci využívají statistický jazykový model SLMs [19]. V oblasti modelování jazyka se používají dva modely, formální a stochastický [20].

**Formální jazykový model –** Tento model je odvozen z Chromského formální teorie jazyka a je tvořen gramatikou a rozdělovacém algoritmem. Gramatika určuje specifickou skladbu daného jazyka a parsovací algoritmus rozhoduje, zda daný text splňuje danou gramatiku. Model rozhoduje jednoznačně zda posloupnost slov naleží či nenáleží danému jazyku. Nevýhodou je, že pokud vstupní text není gramaticky správně napsaný, tento model nebude mít velkou účinnost [20].

**Stochastický jazykový model** – Tento model se řídí pravděpodobností spojení mezi jednotlivými slovy. Nastavuje pro posloupnosti slov pravděpodobnost od 0 až 1 podle toho, zda do daného jazyka patři. Tento model má lepší účinnost s neurčitým všeobecným textem [20].

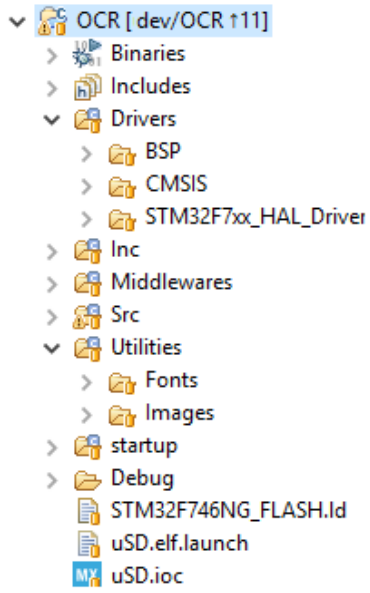
# 3 Vlastní návrh a implementace

V této kapitole popisuji můj vlastní návrh a implementaci systému pro rozpoznávání textu provozovaném na vývojovém kitu 32F746GDISCOVERY, který mi byl darován společností STMicroelektronics. Tato kapitola je rozdělena na dvě hlavní části. V kapitole 3.1 popisuji použité periferie v mé aplikaci, vysvětluji opodstatnění jejich použití a u každé individuálně popisuji postup implementace. Druhá část je popsána v kapitole 3.2, kde uvádím svoji implementaci algoritmu pro převod textu z fotografie do binárního souboru v textového formátu.

Pro vývoj této aplikace jsem zvolil vývojové prostředí Atolic True Studio především kvůli doporučení uváděné výrobcem kitu. Výhoda Atolic True Studia je především ve snadném propojení kitu a počítače pomocí ST Link debbugeru, který umožňuje krokovat kód běžící v daný moment na vývojové desce.

Programovací jazyky podporované kitem jsou C a C++. Vzhledem k tomu, že většina podpůrných firmwarů od firmy ST Microelectronics je vytvořena v C, držel jsem se tohoto jazyka i při vývoji mé aplikace. Výhodou jazyka C je možnost efektivně kontrolovat zacházení s paměťovým prostorem na úkor typicky delší doby vývoje.

Verzování celé aplikace probíhalo v systému GIT, kde jsem se snažil rozdělovat jednotlivé pod kroky implementace do samostatných commitů. Celý postup vývoje je uveden ve větvi dev/OCR. Repositář s aplikací a souborem readme.txt, popisující mimo jiné postup instalace se spuštěním aplikace, je umístěn na přiloženém CD v kořenovém adresáři. Struktura celého projektu je zobrazena na obrázku 12. Složka Drivers obsahuje 3 typy firmwarů sloužících




*Obrázek 12 Adresářová struktura projektu.*

pro nakonfigurování a snadnější používání periferií. Složka Middlewares obsahuje balíček FatFs sloužící pro manipulaci s SD kartou. Ve složce Inc se nacházejí hlavičkové soubory projektu a ve složkce Src je má vlastní implementace. Složka Utilities obsahuje fonty a obrázky, které využívá displej.

Z teorie vyplynulo, že algoritmy OCR jsou výpočetně a paměťově náročné a normálně se provozují na počítačích nebo serverech. Taktéž by měly být navrhovány cíleně k jejich budoucímu použití. Jelikož cílem práce bylo vytvořit OCR systém na vývojovém kitu s omezenými zdroji a bez specifikovaného použití, rozhodl jsem se vytvořit jednoduchý OCR pomocí nejjednodušších metod a ten otestovat na několika různých vstupních datech. V případě dostatečného času následně provést optimalizaci algoritmu. Taktéž jsem si dal za cíl vytvořit architekturu aplikace tak, aby ji bylo možné v budoucnu snadno rozšiřovat.

## 3.1 Architektura aplikace a použité periferie

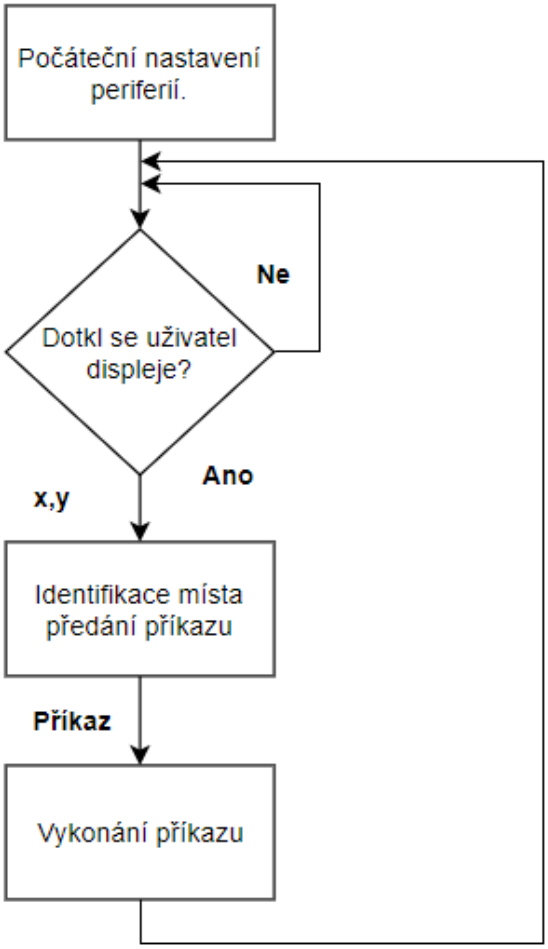
Typický OCR algoritmus již byl popsán v kapitole 1.2.1. Z teorie vyplývá, že prvním krokem OCR aplikace je zachycení obrázku s textem, který chceme rozpoznat. Jelikož moderní OCR systémy využívají kamery jako zdroj obrázků, rozhodl jsem se implementovat podporu kamerového modulu. Cílem této snahy bylo zachytit jeden snímek z kamery a poskytnout ho OCR algoritmu k dalšímu zpracování. Při vývoji nastala potřeba snímek pořízený kamerou někam uložit, především kvůli možnému využití později při testování a ladění programu. Z tohoto důvodu jsem se rozhodl implementovat také podporu microSD karty.

Při návrhu jsem použil microSD kartu s velikostí 16 GB. Na kartě jsou při běhu aplikace ukládány nejen snímky pořízené kamerou, ale také ladící data z jednotlivých dílčích metod. Jelikož jsem si v počátku vývoje nemohl být jistý funkčností a kvalitou dané kamery, rozhodl jsem se nejprve pořízený snímek z kamery ukládat a až následně aktivovat OCR algoritmus.

Důležitým krokem bylo dát uživateli možnost zvolit obrázek na SD kartě, který by se měl podrobit převodu. Z tohoto důvodu jsem se rozhodl implementovat jednoduchý průzkumník pro procházení SD karty. Ten umožňuje vyčítat složky a soubory na microSD kartě a provádět nad nimi speciální operace. Tím je uživateli umožněno za běhu programu efektivně pracovat s jednotlivými daty.

Poslední důležitou periferií, kterou jsem se rozhodl implementovat byl dotykový LCD display. Hlavním důvodem bylo dát uživateli možnost snadno a elegantně ovládat celou aplikaci. LCD display je využit jak při snímání obrazu přes kamerový modul, tak při vizualizaci průzkumníku microSD karty a prostřednictvím něho uživatel získává informace o probíhajícím programu.

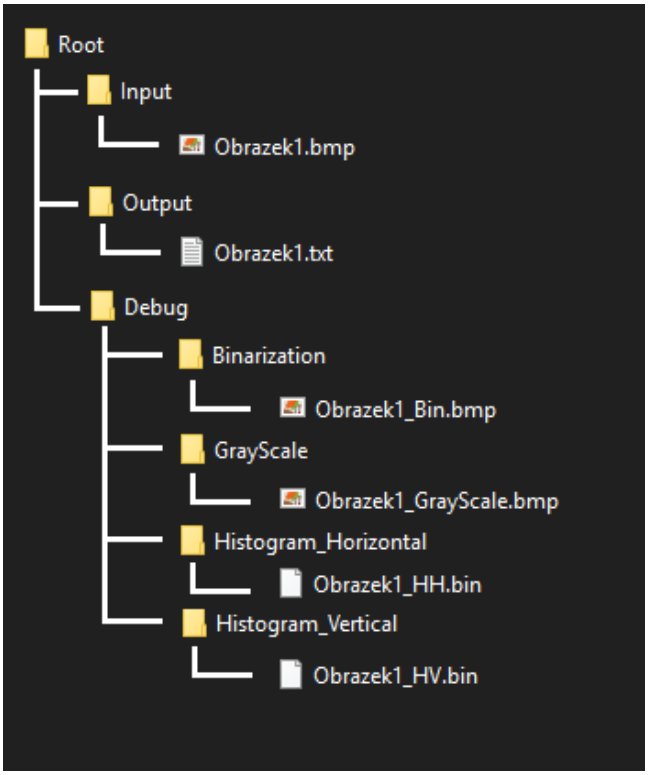
Celková architektura řešení vyvinuté aplikace je zobrazena na obrázku 13. Hlavní program běží v metodě int main(void). Na začátku metody probíhají funkce zajišťující nakonfigurování jednotlivých periferií kitu. Následně program vstupuje do nekonečného while cyklu, ve kterém setrvává po celou dobu života aplikace. V daném cyklu se nacházejí dvě sekce. První sekce kontroluje, zda došlo ke kontaktu prstu s displejem a v případě že jo, souřadnice x a y jsou předány funkci, která vrací specifický příkaz pro danou obrazovku. V případě, že uživatel klikne mimo aktivní element, funkce vrátí hodnotu nic nedělej. V druhé sekci cyklu je přepínač, který na základě obdrženého příkazu provede specifický úkon, například příkaz zapni kameru, přepni obrazovku na menu, atd… Cílem tohoto řešení byla snaha rozdělit logiku daných periferií do vlastních souborů a dané funkce používat z hlavni funkce main. Tímto řešením by se měla zvýšit znovu použitelnost daného kódu a snížit množství závislostí.




*Obrázek 13 Algoritmus hlavního programu.*

### 3.1.1 MicroSD karta

Implementace podpory microSD karty byla především motivována možností na SD kartu v průběhu aplikace ukládat výstupy z jednotlivých metod a tím mít možnost jednoduše ladit program. Za tímto účelem jsem vytvořil fixní adresářovou strukturu, která je vyobrazena na obrázku 14. V kořenové složce SD karty se nachází 3 složky. První z nich je Input složka. V ní jsou uloženy všechny obrázky, které je možné dále převádět na text. Do této složky se ukládá i obrázek získaný z kamerového modulu. Druhá složka má název Debug, v níž se nacházejí složky s daty získanými z jednotlivých metod. Výsledné soubory mají stejný název jako počáteční soubor a pro odlišení obsahují navíc koncovku podle dané metody. Třetí složka je Output, do které se uloží výstupní soubor s převedeným textem. Výhodou tohoto systému je snadné nalezení části algoritmu, která způsobuje problém, ať už se jedná o chybu programu, nebo nedostatečně kvalitní metodu v OCR převodu. Nevýhodou tohoto řešení je ale nutnost opakovaně provádět zápis dat na SD kartu, který je časově náročný. Z tohoto důvodu jsem se rozhodl vytvořit dva režimy ve kterých může program běžet. Jedná se o testovací režim také nazývaný jako debug mód, který využívá již zmíněný proces zapisování všech výstupních dat a druhý režim release mód, který přeskakuje ukládání testovacích dat a je pouze limitován na vyčtení obrazu, rozpoznání a uložení výstupu.




*Obrázek 14 Adresářová struktůra použitá v microSD kartě.*

Implementace zahrnovala použití HAL knihoven od firmy ST Microelektronics, jejichž úkolem je zprostředkování přístupu k microSD kartě. Úpravu a práci se soubory umožnují FATfs knihovny, které jsou podporovány mikrokontrolerem. V rámci diplomové práce vznikly soubory sd.c a sd.h. obsahující několik metod pro zápis obrázku, vyčtení obrázku, zápis textového souboru atd.

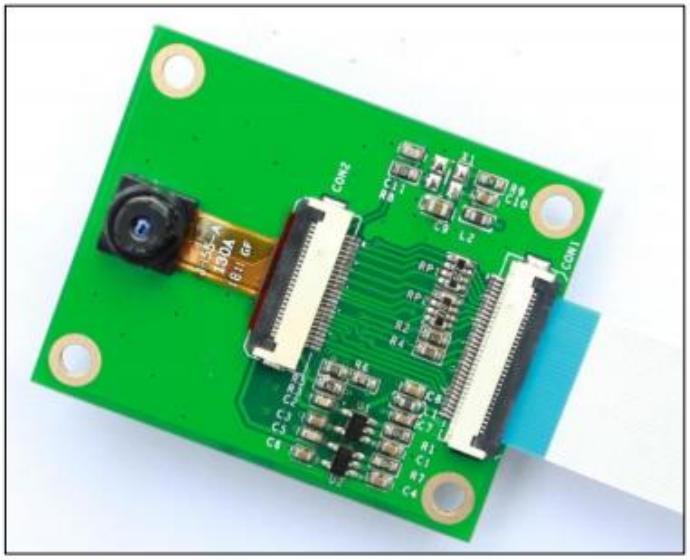
Důležitým krokem celé práce bylo zvolit formát ve kterém dané obrázky budou uloženy. Každý soubor v reálném operačním systému potřebuje mít specifikovaný formát, aby systém věděl, jak má s daným souborem zacházet. I obrázky mají několik použitelných formátů, kde každý má jinak nadefinovanou strukturu a své vlastní výhody a nevýhody. Nejznámější formáty jsou bmp, tiff, jpeg, png. V této práci jsem se rozhodl použít formát bmp, protože se jedná o jednoduší formát, který data reprezentuje v nekomprimované podobě přímo pomocí pixelů. Soubor je tvořen hlavičkou o velikosti 54 bajtů, která popisuje základní informace o obrázku viz. tabulka 2. Dále následují bajty tvořící jednotlivé pixely obrázku ve formátu RGB. Existuje několik typů bmp formátů podle toho, kolika bity je daná barva zastoupena. V rámci této práce byla naimplementována nejběžnější varianta RGB888 neboli jeden byte na barvu.

*Tabulka 2 Popis hlavičky souboru bmp.*

| Offset | Velikost | Popis |
|---|---|---|
| 0 | 2 | ID, musí být rovno 0x4D42 ("BM") |
| 2 | 4 | Velikost BMP obrázku v bytech |
| 6 | 2 | Rezervováno, musí být 0 |
| 8 | 2 | Rezervováno, musí být 0 |
| 10 | 4 | Offset do začátku pole pixelů obrázku, musí být 54 |
| 14 | 4 | Velikost hlavičky od této pozice, musí být 40 |
| 18 | 4 | Šířka obrázku v pixelech. |
| 22 | 4 | Výška obrázku v pixelech. |
| 26 | 2 | Počet vrstev obrázku, musí být 1 |
| 28 | 2 | Počet bitů na pixel. |
| 30 | 4 | Typ komprese, 0 pokud žádná není. |
| 34 | 4 | Velikost obrázku včetně hlavičky. |
| 38 | 4 | Horizontální rozlišení v pixelech na metr (Nespolehlivé) |
| 42 | 4 | Vertikální rozlišení v pixelech na metr (Nespolehlivé) |
| 46 | 4 | počet barev v obrázku, nebo 0 |
| 50 | 4 | Počet důležitých barev v obrázku. |

### 3.1.2 Kamera

Jak již bylo zmíněno v kapitolách 1.2.1 a 2, prvním krokem OCR systémů je zachycení textu jako obrázku v libovolném formátu. Po již implementované podpoře microSD karty má uživatel možnost vytvořit obrázek pomocí externího média a následně ho nahrát na kartu a zpracovat. Tento způsob ovšem není úplně pohodlný a neodpovídá klasickému použití OCR. Proto jsem se rozhodl dále implementovat kamerový modul. Vybral jsem si kamerový modul doporučovaný výrobcem desky a tj. STM32F4DIS-CAM. Jedná se o CMOS kameru. Pro připojení k vývojové desce slouží 30 pinový FPC konektor. Kamera podporuje fotografie a snímkovací frekvence je 15 fps pro SXGA a 30 fps pro VGA a CIF. Pro komunikaci s vývojovým kitem kamera používá DCMI rozhraní „Digital camera interface“ pomocí kterého je možné snadno propojit modul s mnoha různými zařízeními. Obrázky, které kamera zachytí je nutné uložit někam do datového prostoru vývojového kitu. Jednou možností je použít interní paměti FLASH 1M nebo RAM 340 kBit. Vzhledem k tomu že tyto paměti nemají dostatečnou velikost jsem se rozhodl využít externí paměti. Použil jsem SDRAM pamět, která má úložný prostor velikosti 128 Mbit, z čehož je zpřístupněno až 64 MBits.

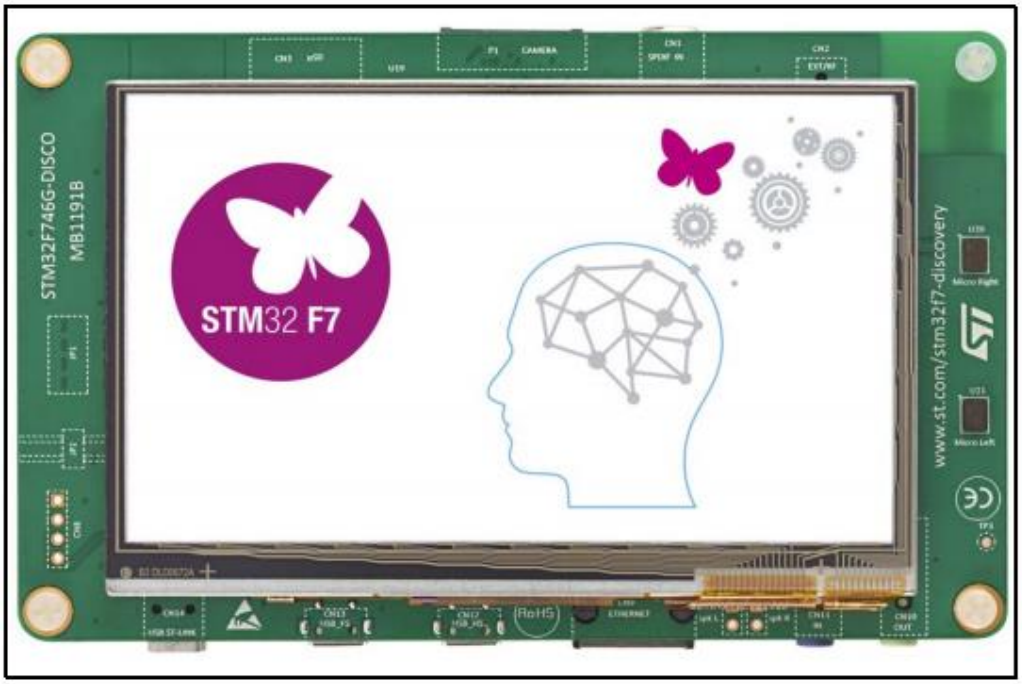


*Obrázek 15 Kamerový modul STM32F4DIS-CAM.*

Implementace ohledně kamerového modulu je umístěna v souborech camera.c a camera.h. Soubory obsahují funkce pro zahájení snímání snímků, převod snímků na displej, přerušení snímání a kompletní vypnutí kamery. Kamera je nastavena na snímání v režimu RGB565 a plynule dokáže obraz převádět obraz na ARGB888 formát, který využívá LCD displej.

### 3.1.3 LCD

Jelikož jsem chtěl vytvořit program který by se snadno ovládal, musel jsem vybrat vhodné uživatelské rozhraní. Za tímto účelem jsem se rozhodl přidat podporu LCD displeje. Vývojový kit 32F746GDISCOVERY má na své horní straně barevný LCD displej s úhlopříčkou 11 cm a velikostí 480x272 pixelů. Displej zároveň obsahuje kapacitní vrstvu, která umožnuje provádět libovolné úkony při interakci uživatele s displej. Rozhodl jsem se vytvořit jednoduché GUI pro ovládání kamery, procházení souborů na microSD kartě a možné měnění parametrů OCR algoritmu. Všechny vytvořené obrazovky byly naimplementovány za pomoci BSP driverů discovery_lcd.c Ty umožňují vykreslování jednoduchých geometrických tvarů (čár, obdélníků, kruhů). Graficky sice nepůsobí nejlépe, ale svůj účel splňují. Během vývoje vznikly čtyři hlavní obrazovky, které jsou implementovány v souborech lcd.c, lcd.h. Mezi obrazovkami lze v průběhu aplikace přepínat pomocí šipky umístěné v pravém horním rohu. Aby přepínání bylo možné, je nutné nastavit správné fungování kapacitní vrstvy. Pro tyto účely jsou vytvořeny soubory

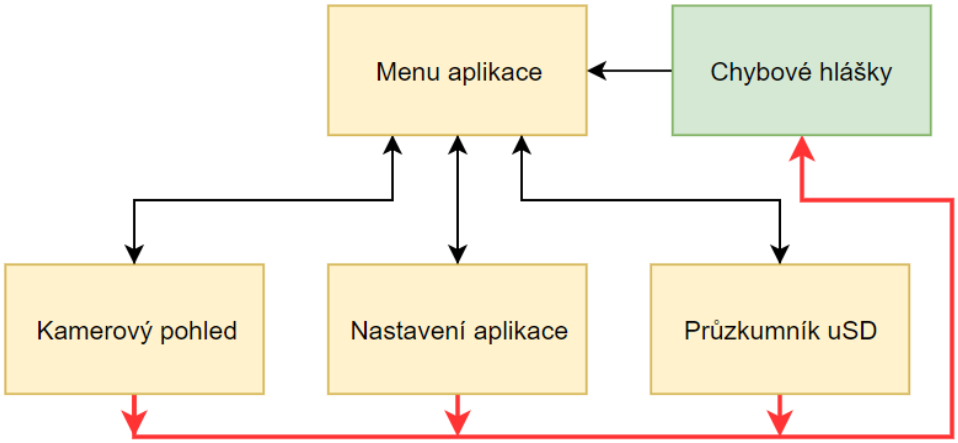



*Obrázek 16 LCD displej 480x272 s kapacitní vrstvou.*

touchscreen.c a touchscreen.h. Tyto soubory disponují funkcemi pro správu kapacitního displeje a starají se o adekvátní změnu dotykové vrstvy v případě změny obrazovky. Algoritmus pro zmíněné přepínaní je vidět na obrázku 17.

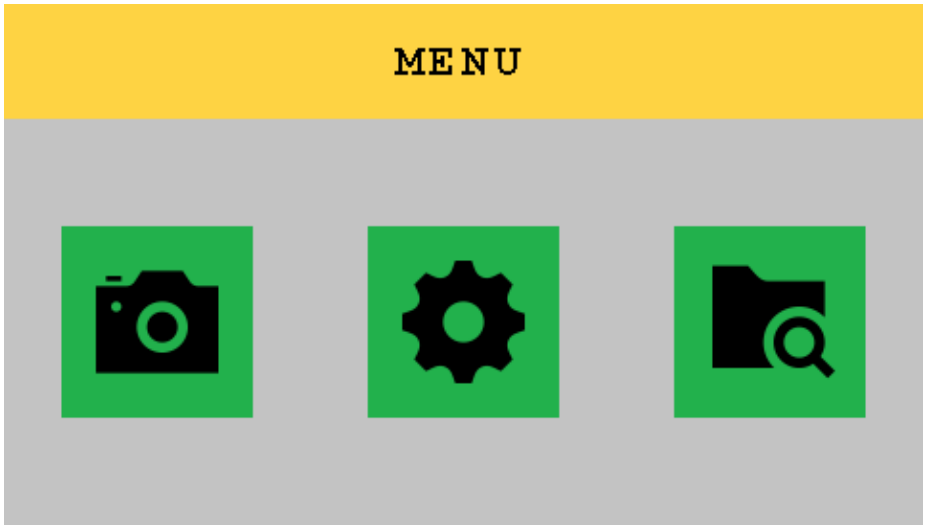
Při spuštění aplikace nejprve proběhnou inicializační funkce jednotlivých periferií a dojde k ověření přítomnosti microSD karty. V případě, že microSD karta není přítomna v socketu vývojového kitu, na displeji se zobrazí hláška missing SD card. V tomto stavu program každých

5 sekund znovu kontroluje, zda SD karta není vložena. Pokud vývojový kit detekuje SD kartu, dojde k vykreslení obrazovky menu.



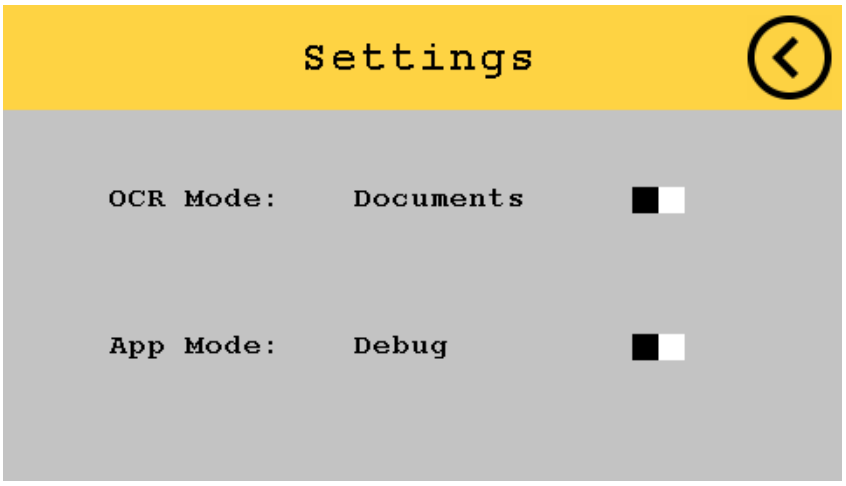

*Obrázek 17 Schéma přechodů mezi obrazovkami.*

**Menu** – První obrazovkou je hlavní menu aplikace. Ta obsahuje tři tlačítka sloužící pro přechod mezi ostatními obrazovkami. Levé tlačítko přepne uživatelovu obrazovku na kamerový pohled, pravé tlačítko přepne obrazovku na průzkumník souboru a tlačítko uprostřed přepne obrazovku do nastavení aplikace. Konečná podoba obrazovky je zobrazena na obrázku.

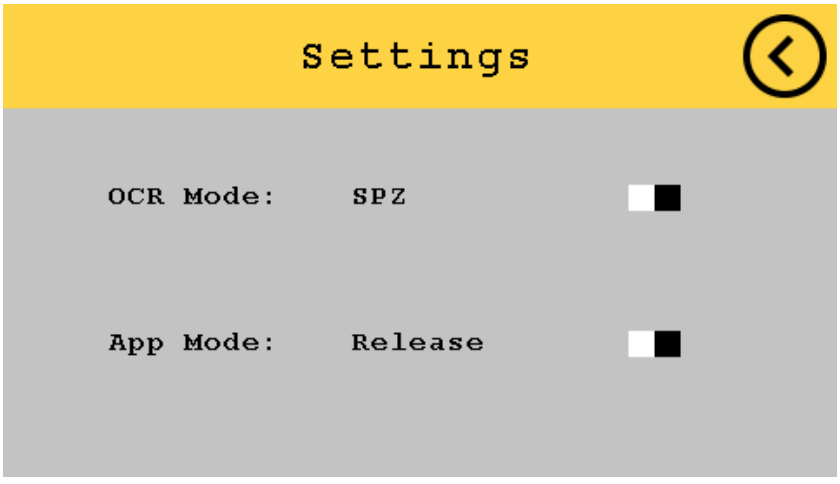



*Obrázek 18 Menu obrazovka LCD displeje.*

**Nastavení aplikace** – Počáteční záměr této obrazovky byl umožnit měnit jednotlivé parametry procesu rozpoznání textu při běhu programu. Bohužel vlivem nedostatku času byla implementace limitována pouze pro vytvoření přepínače módu aplikace. Obrazovka umožňuje přepínat aplikaci mezi debug a release modem použitím dolního přepínače. Horní přepínač umožňuje přepínat celý typ algoritmu OCR. Současná verze aplikace nemá implementováno více druhů algoritmů, ale budoucnu by bylo možné specifikovat algoritmus například pro převod dokumentů obrázek 19, nebo pro identifikaci silničních poznávacích značek obrázek 20.




*Obrázek 19 Obrazovka LCD pro nastavení módu aplikace. Zvolen debug mód s použitím pro dokumenty.*

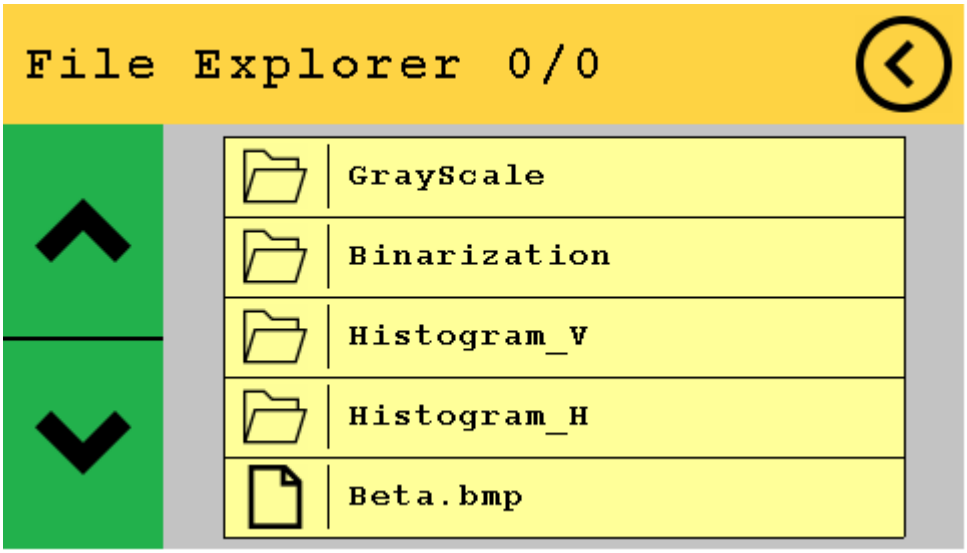



*Obrázek 20 Obrazovka LCD pro nastavení módu aplikace. Zvolen release mód s použitím pro SPZ.*

**Kamerový pohled** – Tato obrazovka přenáší aktuální data z kamery a zobrazuje je na displeji v plné velikosti 480x272 pixelů. V pravém horním rohu se nachází dotyková oblast pro návrat do hlavního menu. Pravý dolním roh slouží pro přerušení zachytávání obrázku a provedení uložení posledního snímku na microSD kartu do složky Input. Po uložení se kamera opět aktivuje. Tyto dvě dotykové oblasti nemají viditelní tlačítka, protože by zbytečně zakrývaly viditelnou oblast kamerou a tím by znesnadňovali focení.

**Průzkumník souborů** – Jelikož implementovaný průzkumník, zmíněný v kapitole 3.1.1 potřeboval pro efektivní použití vizualizaci, rozhodl jsem se pro něj vytvořit vlastní obrazovku. Jelikož souborů na SD kartě začalo značně přibývat a nebylo možné všechny vykreslit na displej současně, rozhodl jsem se omezit počet zobrazených položek v jeden moment. Z tohoto důvodu zobrazuji vždy maximálně pětice položek a pomocí šipek v levém panelu mezi danými pěticemi přepínám. V horní liště je vidět indikátor počtu pětic souborů. Složky jsou graficky odlišeny od souborů.

V případě že uživatel klikne na složku, program načte soubory v dané složce a vykreslí odpovídající pětici souborů. Následně může uživatel zmáčknout šipku zpět pro návrat do původní složky. Pokud uživatel klikne na soubor, nastane akce podle typu koncovky souboru. Pro potřeby vytvořené aplikace funguje pouze spuštění OCR převodu pro daný bmp soubor. V budoucnu by bylo možné přidat úkony pro jiné datové formáty.




*Obrázek 21 Obrazovka LCD pro průzkumník SD karty, obsahuje 4 složky a jeden soubor bmp.*

## 3.2 OCR algoritmus

Poté co jsem dokončil implementaci jednotlivých periférií a vytvořil hlavní strukturu celkového programu jsem začal pracovat na tvorbě vlastního OCR algoritmu. V rámci této kapitoly popisuji myšlenkový postup vývoje daného algoritmu. Jelikož nebylo v zadání specifikováno konečné použití systému, rozhodl jsem se vytvořit jednoduší systém, který by šlo v budoucnu snadno modifikovat a rozšiřovat jeho funkčnost pro specifickou situaci.

Vzniklé funkce spojené s OCR algoritmem jsou umístěny v souborech ocr.c a ocr.h a při jejich tvorbě nebyly použity žádné externí funkce třetích stran zaměřených na OCR. Podle typické struktury OCR algoritmů byl samotný vývoj rozdělen do tří částí. První částí byl preprocessing, druhou rozpoznání a třetí postprocessing. Taktéž bylo nutné vždy otestovat jednotlivé metody, zda fungují správně a z tohoto důvodu vzniklo několik speciálních funkcí pro ukládání dat na microSD kartu. Na základě toho došlo k rozdělení aplikace na dva módy, jak již bylo uvedeno v kapitole 3.1.1. Funkce *RunOCRInDebugMode* je volána v případě, že uživatel má navolen režim debug a funkce *RunOCRInReleaseMode* je volána v případě release modu. Obě tyto funkce využívají stejné metody OCR převodu, a liší se pouze v ukládání dodatečných dat na microSD kartu v debug modu.

Prvním krokem tvorby OCR algoritmu bylo vyčíst obrázek z microSD karty a uložit si RGB hodnoty jednotlivých pixelů do pole, aby bylo možné provádět snadnější přístup k jednotlivým hodnotám v dalším zpracování. Mimo těchto hodnot se také ukládá šířka a výška daného obrázku. Tyto dvě hodnoty jsou důležité pro celý algoritmus, jelikož se používají při procházení celého obrázku po pixelech. Po načtení obrázku do paměti, nastal vývoj předzpracovávajících metod.

### 3.2.1 Implementace preprocessing fáze

**GrayScale**

Typickým krokem předzpracování je převedení obrázku do šedé stupnice. Zatím to účelem provádím implementaci rovnice (2.2). Ve funkci ImageToGrayScaleArray procházím uložené hodnoty RGB a provádím výpočet hodnoty šedé stupnice, která může nabýt hodnoty v intervalu 0 až 255. V případě že bych nahradil hodnoty RGB výslednou hodnotou šedé stupnice, mohl bych uložit daný obrázek jako šedý. Jelikož nechci plýtvat paměťovým prostorem, vytvářím nové datové pole, do kterého ukládám výsledné hodnoty šedé stupnice. Předešlé pole vytvořené na základě bmp souboru obsahovalo tři různé hodnoty RGB daného pixelu, tedy vyžadovalo 3 bajty. Po aplikaci konverze do šedé stupnice jsou všechny tři hodnoty identické, a tudíž není nutné danou hodnotu uchovávat v paměti 3krát. Touto optimalizací se velikost potřebného místa třikrát zmenší a rychlost daných metod se třikrát zrychlí. Při vytváření bmp souboru, jsou jednotlivé

hodnoty opět použity třikrát, aby nedošlo k narušení formátu bmp. Po konverzi obrázku do šedé stupnice v debug módu proběhne uložení nově upraveného obrázku do složky GrayScale.

**Binarizace**

Druhým krokem algoritmu je převedení šedého obrázku na černobílý obraz tak, abych mohl jednoduchým způsobem rozlišit které pixely reprezentují písmena, a které pozadí. Za tímto účelem zavádím metodu globální binarizace. Počáteční implementace využívala uživatelem zvolené prahové hodnoty, podle které se rozhodlo, zda daný pixel obrazu je pozadí či text. Jelikož jsem předpokládal, že hlavním účelem aplikace by mělo být převést tištěný dokument, který má z pravidla bílé pozadí a černé písmo, bylo možné nastavit prahovou hodnotu více méně kdekoliv kolem poloviny šedé stupnice. V intervalu 0 až 255, kde 0 znamená zcela černý pixel a hodnota 255 zcela bílý, jsem zvolil hodnotu 100. Při implementaci vznikla ještě vylepšená metoda, která adaptivně určuje prahovou hodnotu binarizace na základě úrovní šedé stupnice v obrazu. Funkce sečte četnost jednotlivých úrovní a následně najde nejvyšší nejnižší úroveň, která se v obrázku vyskytuje. Předpokládá se, že nejtmavší pixely jsou právě pixely textu a nižší hodnoty šedé stupnice představují bílé pozadí. Tato metoda vyžaduje delší dobu výpočtu, ale měla by zaručovat lepší výsledky v případě rozmazání jednotlivých písmen. Pokud by jednotlivá písmena v obrázku byla rozmazaná, mohli by se rozmazané pixely kolem písmen analyzovat jako pixely textu a tím pádem spojit některá písmena do sebe. To by mohlo způsobit problém při segmentaci jednotlivých písmen. V případě debug módu opět dojde k uložení nově vytvořeného binarizovaného obrazu na SD kartu do složky Binarization.

### 3.2.2 Implementace segmentace a klasifikace

**Segementace**

Poté, co je celý obraz rozdělen na text a pozadí, je nutné provést rozložení daného obrázku na řádky, popřípadě jednotlivá písmena. Já se rozhodl použít princip histogramové segmentace popsané v kapitole 2.1.5. V prvním kroku vytvářím pole reprezentující histogram o velikosti výšky daného obrázku. Následně iteruji přes celý obraz po pixelech a sčítám počet černých pixelů v každém řádku. Výstupem dané metody je horizontální histogram určující rozložení jednotlivých řádků textu v obrazu. Pokud daný řádek neobsahuje žádné černé pixely, jedná se o prázdný místo, pokud ovšem obsahuje nějaké pixely, jedná se o řádek s textem. Výsledný histogram se ukládá do složky Histogram_H. Pro šetření paměti jsem se rozhodl vytvořit pro každý řádek dva indexy, které by značily spodek a vršek každého řádku a neinicializovat novou proměnnou, která by si uchovávala celý řádek. Tyto indexy jsou získány pomocí metody GetIndexes a ty jsou uchovány v paměti. Jelikož jsem chtěl vidět, že opravdu dochází ke správné segmentaci řádků, pro každý

řádek vytvářím nový bmp soubor, do kterého ukládám odpovídající data. Tyto obrázky jsou uloženy ve složce Histogram_H.

Dalším krokem segmentace bylo identifikovat jednotlivá písmena v daných řádkách. Tento proces je implementován stejným postupem jako v případě řádků. Nejprve se vytvoří vertikální histogram podobným způsobem jako u segmentace řádků akorát velikost daného pole je nyní rovna šířce daného obrázku. Druhou odlišnosti je to, že už se neprovádí sčítání pixelů přes celý obraz, nýbrž pouze přes úsek daného řádku. To je zajištěné pomocí indexů řádků získaných z předešlé metody. Po vytvoření histogramu opět provádím detekci na nulové hodnoty, abych identifikoval mezery mezi písmeny a opět uložím indexy indikující levý počátek písmene a pravý konec písmene.

Důležitým krokem, který je nutné zde provést, je identifikace mezer mezi slovy. Pokud by tento krok nebyl proveden, všechny slova výsledného textu by byly spojeny bez mezer. Detekci mezer jsem implementoval analýzou vzdáleností mezi jednotlivými indexy písmen obdržených z vertikálního histogramu. V případě, že vzdálenost mezi písmeny je větší než 4 pixely, mezi danými písmeny musí být umístěna mezera.

**Klasifikace**

Poté co došlo k segmentování všech písmen v obrazu, nastává fáze rozpoznání. V tomto kroku jsou postupně jednotlivé úseky obrazu, segmentované pomocí získaných indexů z předešlých metod, porovnány se šablonami písmen načtených z databáze. Šablony pro podporovaná písmena jsou uloženy v souboru font_Calibri.c a font_Calibri.h. V těchto souborech jsou vytvořeny pole bytů charakterizující pixely jednotlivých podporovaných písmen fontu Calibri velikosti 11 pt.

Jelikož jsem chtěl docílit podpory různých velikostí fontů, implementoval jsem metodu pro změnu velikosti daného obrázku. Této funkci je předán segmentovaný obraz písmene a jedna šablona písmene z databáze. Funkce se postará o srovnání výšky a šířky rozpoznávaného segmentu se šablonou. V případě shody šířky a výšky nastane výpočet matice shody. Výpočet matice shody probíhá porovnáváním jednotlivých pixelů šablony a rozpoznávaného segmentu. Nakonec se vypočítají procenta shody a uloží se do pole na index daného písmene. Tento proces se opakuje s další šablonou, dokud program neporovná všechny.

**Zápis textu do souboru**

Po projití všech šablon se vybere největší shoda, a daný index v poli je přeposlán do funkce EncodeLetterToUTF8 nacházející se v souboru encoder.c. Jelikož jsem chtěl zavést podporu českých znaků, bylo nutné výsledky textový soubor ukládat s nějakým kódování, které tyto znaky podporuje. Já jsem zvolil kódování UTF-8. Funkce EncodeLetterToUTF8 na základě indexu písmene rozpozná, o jaké písmeno se jedná a vrací odpovídající bajty. Většina písmen je definována jedním bajtem, některé dvěma bajty. Získaná hodnota je pro každé písmeno uložena do souboru. V případě nalezení mezery se uloží hodnota prázdného znaku, v případě konce řádku se do souboru uloží hodnota CF, kterou operační systém Windows chápe jako konec řádku. Toto zvolené řešení umožňuje přidávat další symboly do souboru encoder.c a přidávat tak podporu speciálních znaků z celého světa. Pokud by cílový uživatel měl zájem o výstup v jiném kódování, bylo by možné vytvořit další funkci v souboru encoder.c, která by danou funkcionalitu obsahovala. Následně by bylo možné nastavit přepínání mezi těmito funkcemi, tudíž by uživatel mohl dynamicky měnit výstupní formát.

### 3.2.3 Implementace postprocessing fáze

Posledním krokem daného převodu bývá konečné zpracování, které by mělo nalézt chyby klasifikace, pokud k některým došlo, a pokusit se je opravit. V rámci této práce nebyla tato funkce naimplementována, protože cílem bylo vytvořit jednoduché OCR a postprocessing fáze zahrnuje sofistikovanou skupinu algoritmů které by bylo nutné přidat. Výsledná aplikace spoléhá na manuální korekci v případě chyby. Teoreticky by bylo možné tento krok algoritmu přidat s použitím slovníků pro kontrolování jednotlivých textových řetězců, ovšem na úkor větší časové náročnosti.

# 4 Výsledky a diskuze

V této kapitole shrnuji výsledky své práce pří návrhu OCR systému vyvinutém na vývojovém kitu Discovery kit with STM32F746NG MCU. Kapitola je rozdělena do tří částí. V první částí popisuji výhody a nevýhody vytvořeného uživatelského rozhraní a zmiňuji nalezené nedostatky. Zbylé dvě časti tvoří testování vytvořeného OCR systému. Kapitola 4.2 postupně zmiňuje jednotlivé implementované metody a provádí jejich testovaní na sadě vstupních dat. Cílem bylo nalézt hranice použitelnosti daných metod, aby bylo možné v budoucnu provést případnou optimalizaci či dané metody zcela nahradit nebo odstranit z celkového algoritmu OCR. Po otestování jednotlivých metod jsem provedl celkový test převodu OCR a změřil jeho rychlost převodu, chybovost a rozlišovací schopnosti vzhledem k různým typům a velikostem fontů. Toto měřené je uvedeno v kapitole 4.3.

## 4.1 Uživatelské rozhraní a periferie

V této kapitole popisuji výsledky implementace uživatelského rozhraní popsané v kapitole 3.1. Postup testování spočíval v dlouhodobějším používání a hledání limitů pro jednotlivé obrazovky.

Aplikace byla podrobena 50 přechodům mezi navrženými obrazovkami a dalším dodatečným testům jednotlivých obrazovek. Přitom testování bylo nalezeno několik problémů.

Pokud je na microSD kartě přítomno větší množství souborů například 1000, průzkumník souborů se neotevře a aplikace zamrzne pravděpodobně nedostatkem paměti. Tento problém by bylo možné ošetřit nastavením omezení počtu načtených souborů.

Při testování obrazovky kamery se občas objevil problém, že se vykreslená obrazovka posunula o 20 pixelů níže. Tím pádem vrchní část snímku, která přetekla display se objevila na spodku displeje. Tato chyba se objevovala jen zřídka a vždy stačilo jen znovu načíst kamerovou obrazovku a vše již bylo v pořádku. Z tohoto důvodu nebylo blíže zkoumáno, co danou chybu způsobilo.

Testování obrazovky průzkumníku SD karty prokázalo funkční načítání pětic souborů jak při přepínání pomocí šipek, tak i při výběru složky. V rámci testování byla nalezena nahodilá chyba v čítači pětic, který je umístěn v horním středu obrazovky. V několika případech se druhé číslo nastavilo rovno nule namísto správné hodnoty. Druhým zvláštním stavem čítače byla hodnota -1 v případě načtení prázdné složky. Vzhledem k nízkému dopadu na chod aplikace nebyla tato chyba opravena. Druhou chybou této obrazovky je špatně vykreslený název složky či souboru. V programu je nastavena velikost povoleného textu na 20 znaků. Pokud soubor na SD kartě obsahuje delší název, než je tato hodnota, do proměnné se neuloží poslední znak textového řetězec, kterým je symbol oznamující ukončení textového řetězce.

Poslední testovanou obrazovkou byla obrazovka nastavení. Testování prokázalo funkčnost obou přepínačů a nenašlo žádný problém.

Při testování funkcí microSD karty se objevil problém při načítání obrázků větších než 200000 pixelů. Při hlubším zkoumání se ukázalo, že obrázky jsou ukládány automaticky do paměti RAM, která neměla dostatek místa pro uložení dat takový rozměrů. Reakce na tuto chybu byla snaha ukládat tyto data na SDRAM, kde má větší paměťovou kapacitu. Po několika dnech snažení nebylo nalezeno funkční řešení a jelikož pro testovací účely stačilo použít menší obrázek, bylo rozhodnuto toto velikostí omezení akceptovat.

## 4.2 Testy jednotlivých metod OCR

V rámci této kapitoly jsou provedena měření jednotlivých metod separátně kvůli otestování jejich funkčností a nalezení jejich limitů v navrhnutém OCR algoritmu. Pro měření bylo vytvořeno několik vstupních dat, pomocí nichž se měřily parametry jako rychlost dané funkce, chybovost a rozlišovací schopnost.

Vstupní data jsou rozřazena podle způsobu vzniku na tři skupiny. Jedna skupina zahrnuje snímky pořízené přes kamerový modul. Počáteční odhad byl, že tyto snímku budou mít nižší kvalitu tudíž bude rozpoznávání komplikovanější. Druhou skupinou jsou snímky pořízené přes skener v obyčejné laserové tiskárně. U této skupiny se odhadovala lepší kvalita než u kamery s možností občasného výskytu lokálních nečistot vlivem nekonzistentního skenování. Poslední skupinou dat byly snímky pořízené přímo v počítači přes funkci PrintScreen. Tato funkce vytvoří snímek odpovídající aktuálnímu obrazu na monitoru. Tato třetí skupina snímků by teoreticky měla být strojově nejčitelnější, protože nedochází k žádnému negativnímu vlivu okolního prostředí. Jednotlivá vstupní data jsou uložena ve složce data v kořenovém adresáři aplikace.

Měření rychlosti probíhalo za pomocí interního časovače, který postupně čítá hodnoty s přesností ms. Na začátku a konci každé metody byla získána hodnota časovače a výsledný časový interval dané metody se dopočítal jako rozdíl obou hodnot. Měření pro každá vstupní data proběhlo několikrát a za konečnou hodnotu byla zvolena průměrná hodnota ze všech provedených měření. Jelikož se předpokládalo, že módy debug a release mají velice odlišné doby trvání jednotlivých funkcí, byla všechna měření provedena separátně pro každý mód.

**Konverze obrazu do šedé stupnice**

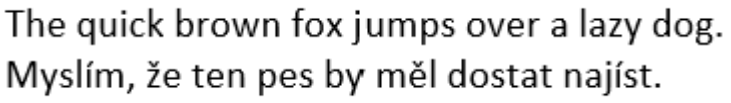
Testování této funkce spočívalo v změření její rychlosti. Měřená data jsou zobrazena v tabulce 3. Z naměřených hodnot vyplývá, že rozdělení aplikace na release a debug mód byl adekvátní krok, jelikož interval trvání funkce v debug módu je v průměru 15krát delší než v release modu. Z toho vyplívá, že zápis na SD kartu je mnohonásobně časově náročnější než

průběh dané metody. Některé výstupy z funkce jsou zobrazeny na obrázku 22 a 23. Na obrázku 22 je vidět vliv skeneru na snímaný text. Daná písmena jsou výraznější, ale jejich okolí obsahuje rozmazané pixely. Obrázek 23 obsahuje text získaný zkopírováním textu z monitoru počítače metodu PrintScreen. Při testování všech vstupních datech nenastal žádný problém.

*Tabulka 3 Měření rychlosti funkce GrayScale.*

| Šedá stupnice | | | | |
|---|---|---|---|---|
| | | | Debug | Release |
| Měření | Nazev Obrazku | Velikost [px] | t [ms] | |
| 1 | calibri_11p.bmp | 24064 | 823 | 51 |
| 2 | calibri_14p.bmp | 34944 | 2184 | 74 |
| 3 | calibri_11p_huge.bmp | 134652 | 2816 | 180 |
| 4 | calibri_11s.bmp | 65688 | 1182 | 138 |
| 5 | calibri_12s.bmp | 65688 | 2189 | 138 |
| 6 | calibri_14s.bmp | 32204 | 982 | 68 |
| 7 | calibri_11.bmp | 49920 | 1631 | 109 |


*Obrázek 22 Výstup metody ImageToGrayScaleArray, obrázek s textem fontu Calibri s velikostí 14 pořízený ze skeneru.*



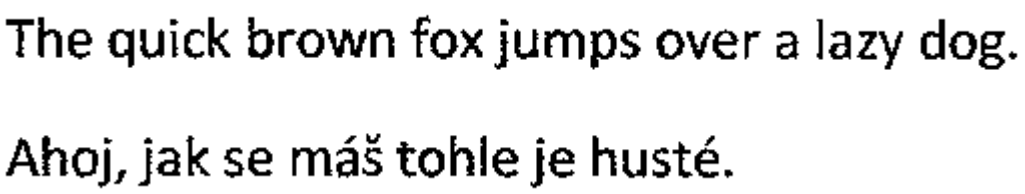

*Obrázek 23 Výstup metody ImageToGrayScaleArray, obrázek s textem fontu Calibri s velikostí 11 pořízený metodou PrintScreen.*

**Binarizace**

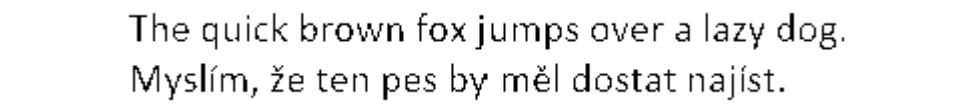
Druhou testovanou metodou spadající do fáze předzpracování je binarizace. Toto měření probíhalo podobně jako měření předešlé metody. Provedlo se především měření rychlosti a opticky se zhodnotila funkčnost dané funkce pro všechna vstupní data. Výsledky rychlostí binarizace jsou uvedené v tabulce 4. V tomto měření je stejně jako v předešlém vidět velký vliv zapisování ladících dat v módu debug. Taktéž lze pozorovat nižší doba trvání oproti předešlé metodě. Z testování vyplynulo, že metoda funguje dobře na data vytvořená skenerem. Na obrázku 24 lze pozorovat odfiltrování rozmazaných pixelů vzniklých kolem jednotlivých písmen při převodu do šedé stupnice. Naopak u obrázku 25, který vznikl metodou PrintScreen lze pozorovat odfiltrování většího počtu pixelů tvořící jednotlivá písmena a u některých písmen si lze všimnout zdvojení pixelů. Zmíněné problémy vznikají především kvůli nedostatečně homogennímu pozadí na vstupním obrazu, nebo při špatném výběru prahovací hodnoty. Při použití této funkce na font velikost 11, občas docházelo k odfiltrování spolu s okolním šumem i větší části písmene. Tento jev není žádoucí a může způsobit problém při segmentaci. Snímek pořízený kamerovým modulem je vidět na obrázku 26. Na obrázcích 27 až 29 je k vidění vliv prahovací hodnoty na binarizaci. Jelikož každý snímek pořízený kamerou měl různé množství světla, nebylo možné implementovanými metodami provést úspěšnou binarizaci pro tento typ dat.

*Tabulka 4 Měření rychlosti funkce Binarization.*

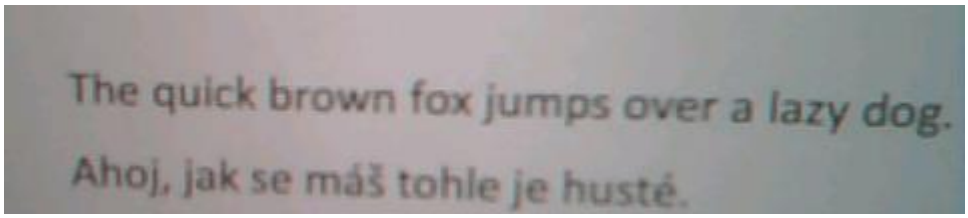
| Binarization | | | | |
|---|---|---|---|---|
| | | | Debug | Release |
| Měření | Nazev Obrazku | Velikost [px] | t [ms] | |
| 1 | calibri_11p.bmp | 24064 | 734 | 4 |
| 2 | calibri_14p.bmp | 34944 | 1097 | 6 |
| 3 | calibri_11phuge.bmp | 134652 | 2568 | 14 |
| 4 | calibri_11s.bmp | 65688 | 2078 | 10 |
| 5 | calibri_12s.bmp | 65688 | 2052 | 10 |
| 6 | calibri_14s.bmp | 32204 | 906 | 5 |
| 7 | calibri_11c.bmp | 49920 | 1552 | 7 |



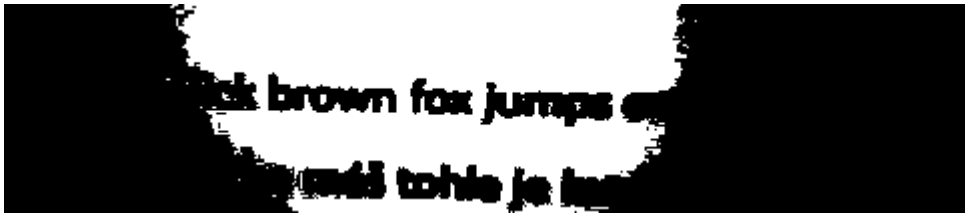

*Obrázek 24 Výstup metody Binarizaztion, obrázek s textem fontu Calibri s velikostí 14 pořízený ze skeneru.*




*Obrázek 25 Výstup metody Binarization, obrázek s textem fontu Calibri s velikostí 11 pořízený metodou print screen.*

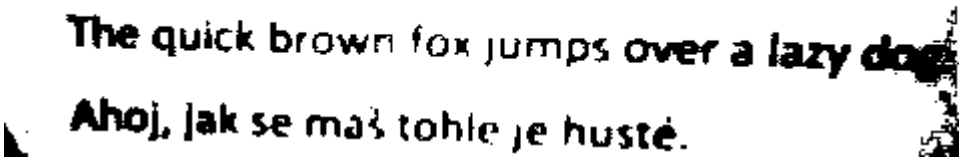



*Obrázek 26 Vstupní data, obrázek pořízený kamerovým modulem.*



*Obrázek 27 Výstup metody binarization pro obrázek 26 při prahové hodnotě 160.*

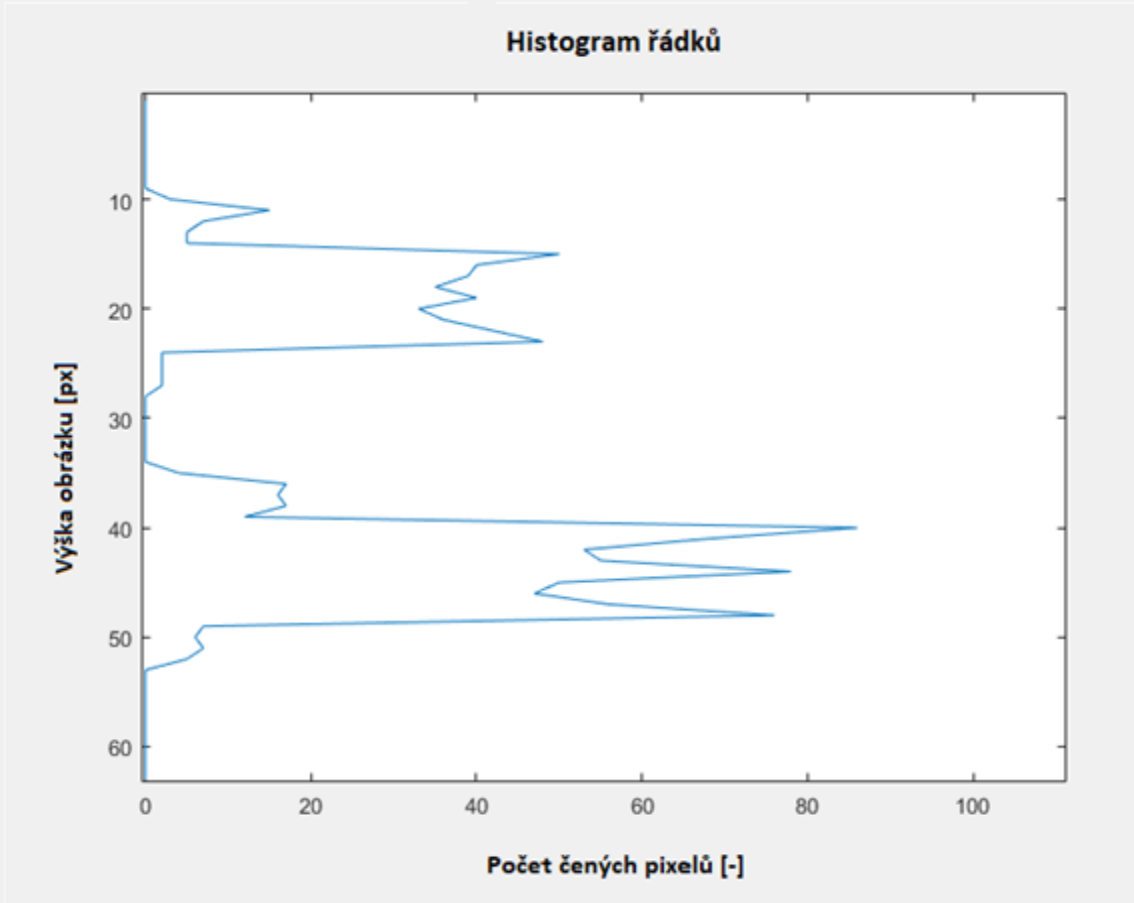
*Obrázek 28 Výstup metody binarization pro obrázek 26 při prahové hodnotě 100*



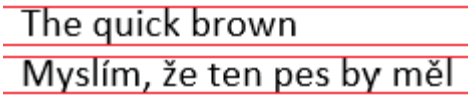

*Obrázek 29 Výstup metody binarization pro obrázek 26 při prahové hodnotě 130*

**Segmentace a histogramy**

Testování segmentace bylo provedeno pouze vizuální kontrolou obdržených ladících dat z jednotlivých metod. Důraz byl především kladen na ověření schopnosti rozdělit daný obraz na řádky a následně na písmena. Na obrázku 30 je vidět vertikální histogram řádků vytvořený v programu matlab z ladících dat obdržených z funkce *HorizontalHistogram*. Z daného histogramu lze identifikovat že se jedná o dva řádky, kde první má začátek kolem pixelu 10, končí kolem pixelu 28 a obsahuje výrazně jméně písmen než druhý řádek. Ten začíná na 33 pixelu a končí na 52 pixelu. Obdržené indexy z histogramu jsou vyznačeny na obrázku 31. Testování horizontální histogramové metody probíhalo na textech s různou velikostí mezery mezi řádky. Měření prokázalo, že metoda je funkční pro různé velikosti mezer mezi řádky. Při testování byl také nalezen jeden nedostatek. Algoritmus má problém správně segmentovat háček od velkého písmene. V těchto případech algoritmus nalezne mezeru mezi háčkem, písmenem a háček analyzuje jako další řádek.




*Obrázek 30 Histogram řádků.*




*Obrázek 31 Vyznačení indexů nalezených pomocí histogramu z obrázku 30.*

Testování vertikální segmentace probíhalo stejným stylem jako testování horizontální. Při měření bylo nalezeno několik nevyhovujících stavů. Jedním problémem bylo špatné rozdělení dvou písmen, pokud mezi nimi nebyl sloupec bílých pixelů. Algoritmus nedokáže taková písmena rozdělit a dále s nimi pracuje, jako by to byl jeden symbol. V případech, že prahová hodnota je špatně nastavena může také dojít k druhému problému, a to odmazání některých pixelů z písmen. V takovém případě algoritmus pracuje s chybně segmentovaným písmenem jako by to byly dvě písmena. Příklady zmíněných chyb jsou na obrázku 32. Vlevo a uprostřed jsou chyby spojení dvou písmen a vpravo je chyba chybného rozdělení písmene.

*Obrázek 32 Chyby vertikální segmentace.*

## 4.3 Globální test celého OCR

Po ověření funkčnosti jednotlivých metod navrženého OCR algoritmu proběhlo také testování celého algoritmu. V rámci testování se změřila doba trvání daného algoritmu, chybovost a rozlišovací schopnost daného algoritmu. Měření bylo rozděleno na tři části.

První část měření se soustřeďuje na typické texty psané ve fontu Calibri s velikostí 11. Výsledky tohoto měření jsou popsané v tabulce 5. Názvy jednotlivých souborů jsou tvořeny pomocí podtržítka a písmene určující původ daného obrázku. Písmeno „c“ reprezentuje kameru, písmeno „s“ skener a písmeno „p“ zkopírovaný obrázek z počítače. Z měření vyplývá, že implementovaný OCR algoritmus nedokáže převádět text z obrázků pořízených kamerou, kde hodnota chybovosti dosahuje 100 %. Hlavním důvodem je špatné rozlišení daného obrázku, a především nevhodně nastavená prahová hodnota binarizace. Nejnižší chybovosti dosahují obrázky vytvořené zkopírováním z počítače v průměru 21,3 %. To je dáno absencí šumu v pozadí vstupních obrázků. Průměrná hodnota chybovosti pro data vytvořená pomocí skeneru je 30.3 %. Důvodem této chybovosti je nejspíše především nedostatečně kvalitně naimplementovaná funkce měnící velikost testovací šablony. Jelikož výstupní snímek ze skeneru má tučnější jednotlivá písmena, jak již bylo zjištěno z předešlého měření v kapitole 4.2, je nutné danou šablonu zvětšit. Zvětšená šablona pravděpodobně neodpovídá danému segmentu.

*Tabulka 5 Výsledky chybovosti OCR algoritmu pro různá vstupní data.*

| Calibri font 11 | |
|---|---|
| Název obrázku | Chybovost [%] |
| a_p.bmp | 20 |
| b_p.bmp | 18 |
| c_p.bmp | 26 |
| d_s.bmp | 35 |
| f_s.bmp | 32 |
| g_s.bmp | 34 |
| h_c.bmp | 100 |
| i_c.bmp | 100 |

Ve druhé části měření se měřil vliv velikosti daného textu na chybovost. Výsledky tohoto měření jsou popsané v tabulce 6 a podporují tezi o nevyhovující funkci pro měnění velikosti šablony. Font s velikostí 11 v průměru dosahuje standardní chybovosti implementovaného algoritmu 23 %. Tato hodnota se shoduje s výsledky předešlého měření. Font s velikostí 12 dosahuje mírného zhoršení, ale stále je jeho chybovost pod 30 %. Tyto dva fonty mají velice podobnou velikost blížící se velikosti jednotlivých šablon. Z tohoto důvodu funkce pro změnu

velikosti šablony nemá takový význam. Naopak při použití fontů s velikostí větší než 12 funkce měnící velikost již nedokáže správně provést roztažení a chybovost roste ke 100 %. Zlepšení výsledků by šlo dosáhnout přidáním šablon s různými velikostmi písmen.

*Tabulka 6 Výsledky chybovosti pro různé velikosti fontů.*

| Calibri různé velikosti fontů | |
|---|---|
| Typ fontu | Chybovost [%] |
| Font11 | 23 |
| Font12 | 32 |
| Font14 | 82 |
| Font18 | 100 |

Třetí měření spočívalo v testování různých fontů. Vytvořily se 4 obrázky s identickými texty v odlišných fontech. Při měření se vyzkoušely 4 různé fonty a výsledky tohoto měření jsou uvedené v tabulce 7. Z výsledků vyplývá, že i při použití jiného fontu něž který používají šablony je možné docílit částečného převodu. Font Times New Roman dosahuje vysoké hodnoty chybovosti 76 %. Hlavní důvod takto vysoké chybovosti jsou problémy při segmentaci. Jelikož tento font obsahuje patky u jednotlivých písmen, velice často dochází k spojení několika písmen ve fázi binarizace. Jelikož hodnoty chybovosti jsou pro jiné fonty relativně vysoké, není vhodné navržený algoritmus takto používat. Ideálnější možnost by byla přidat novou databázi písmen pro daný font a nechat uživatele přepínat mezi fonty k rozpoznání, nebo vytvořit novou funkci, která určí, o jaký font se jedná.

*Tabulka 7 Výsledky chybovosti pro různé fonty.*

| Různé typy fontů velikost 11 | |
|---|---|
| Název fontu | Chybovost [%] |
| Calibri | 23 |
| Times New Roman | 76 |
| Arial | 35 |
| Consolas | 60 |

Posledním měřením bylo zjištění doby trvání celého algoritmu OCR. Výsledky měření jsou zobrazeny v tabulce 8. Měření proběhlo na pěti vstupních obrázcích, kde každý měl různý počet pixelů a různý pročet textových znaků. Měření rychlosti proběhlo pro oba módy aplikace. Výsledky ukazují obrovský rozdíl rychlosti programu mezi jednotlivými módy. Největší vliv na to má počet znaků v textu, protože v debug modu dochází k ukládání každého segmentovaného písmene na microSD kartu jako nový bmp soubor. Jelikož každý znak má různou velikost, nebylo možné stanovit rychlost rozpoznání pro jeden symbol.

*Tabulka 8 Výsledky měření doby trvání OCR algoritmu.*

| Měření rychlosti algoritmu. | | | | |
|---|---|---|---|---|
| | | | Debug | Release |
| Název obrázku | Velikost [px] | Počet znaků [N] | t [ms] | |
| a.bmp | 15000 | 30 | 3504 | 435 |
| b.bmp | 21000 | 60 | 5508 | 785 |
| c.bmp | 50000 | 90 | 7653 | 1137 |
| d.bmp | 60000 | 180 | 23076 | 2146 |
| e.bmp | 70000 | 270 | 41622 | 3185 |

# 5 Návrh dalšího postupu

Při vývoji a testování bylo nalezeno několik problémů či nedokonalostí. V této kapitole je zmíněno pár úkonů, které by bylo možné provést pro zlepšení výsledného produktu.

Z hlediska uživatelského rozhraní by bylo vhodné rozšířit obrazovku nastavení o možnosti modifikovat OCR algoritmus za běhu programu. Uživatel by tak mohl provádět přepínání mezi různými funkcemi a mohl tak přizpůsobovat algoritmus přesně danému typu použití. Druhou obrazovkou, kterou by bylo možné rozšířit, je průzkumník microSD karty. Tato obrazovka má velký potenciál pro rozšíření, jako například interakci se soubory. Bylo by možné přidat funkce pro vykreslení obsahu textového souboru na displej, vykreslení bmp souboru na display. Poslední možností rozšíření uživatelského rozhraní je přidání zcela nové obrazovky pro prezentování ladících dat získaných z OCR algoritmu.

Z hlediska OCR algoritmu existuje mnoho možných optimalizací. Důležitým krokem by bylo zvolit konečné použití a začít podle toho OCR algoritmus specializovat. Vhodným krokem by jistě bylo přidat více metod zabývajících se předzpracováním a zároveň se snažit zlepšit kvalitu vstupních obrazů použitím lepšího zaznamenávacího zařízení. V případě požadavku využívat kamerový modul jako zdroj vstupních dat by bylo vhodné zavést funkce pro modifikaci jasu a kontrastu. Největším problémem implementovaného algoritmu je nevyhovující funkce, která mění rozměry šablon jednotlivých písmen. Vhodnou modifikací by bylo se na tuto funkci zaměřit, popřípadě na samotné porovnávání a provést vylepšení. Jednou možností by bylo například naimplementovat funkci dvoudimenzionální korelace, která by porovnávala šablonu se segmentem obrazu a hledala by ideální shodu.

Dalším zajímavým krokem by bylo vytvořit postprocessing, který by dokázal opravovat chyby vzniklé při rozpoznání v relativně krátkém čase.

Poslední možností dalšího postupu by bylo vyzkoušení možnosti implementace strojového učení na daném vývojovém kitu.

# Závěr

Diplomová práce se zabývá studiem metodik rozpoznávání textu. V první části je uveden rozsáhlý teoretický rozbor o problematice rozpoznání textu včetně historického rozboru, současného stavu a celkového algoritmu rozpoznání textu. Druhá část popisuje návrh a implementaci vlastního systému pro rozpoznání textu na vývojovém kitu STM32F746NG. V třetí části je změřen výsledný algoritmus a otestován celý program. V poslední části je uveden další možný postup práce.

Z hlediska praktické části vznikl program s možností převádět text z obrázku do textového formátu *txt. Stěžejní práce byla implementace na vývojovém kitu, jelikož většina současných OCR funguje na výkonných zařízeních s operačním systémem. Kvůli obavám z omezené výpočetní síly a omezené datové paměti vývojového kitu, byl naimplementován jednodušší algoritmus založený na porovnávání šablon. Aby bylo možné na vývojovém kitu řádně vyvíjet, proběhla také implementace podpory několika periferií například: microSD karty, LCD, kamerového modulu. Vzniklo několik LCD obrazovek, mezi kterými si uživatel může pomocí kapacitní vrstvy displeje přepínat a ovládat tak celý program.

Testování OCR algoritmu prokázalo, že navržený algoritmus dokáže převádět text z obrázku na text, ale nedosahuje příliš dobrých rozlišovacích schopností ve srovnání s moderními OCR systémy založenými na strojovém učení. Hodnoty chybovosti pro podporovaný font Calibri s velikostí 11, dosahuje algoritmus 23% chybovost. Při použití nepodporovaného fontu dochází k zvýšení chybovosti. Pokud je daný font podobný fontu Calibri, může dojít k převodu s relativně slušnou chybovostí jako v případě fontu Arial s chybovostí 35 %. Algoritmus dokáže rozpoznávat pouze velikosti fontů 11 a 12. Pro vyšší velikosti již dochází k problému se správným roztažením šablon jednotlivých písmen a při nižší velikosti již nezvládají preprocessing funkce rozlišit jednotlivá písmena. Kamera jako zdroj dat neposkytuje obraz v dostatečné kvalitě pro implementované metody předzpracování. Algoritmus lze použít pro převod obrázků vložených na SD kartu získaných ze skeneru, nebo z počítače překopírováním.

Vzhledem k omezeným zdrojům vývojového kitu, problematickému počátečnímu vývoji, lze výsledný algoritmus považovat za úspěch. Výsledek se ovšem nemůže porovnávat s komerčními systémy vyvíjenými několik let. Na druhou stranu, pokud by došlo ke specifikaci využití daného OCR, a přizpůsobení jednotlivých metod tomuto účelu, bylo by možné tento výsledný algoritmus využít i v praxi.

# Literatura


[1] L. Eikvil, OCR Optical Charakter Recognition, 1993.

[2] H. F. Schantz, The history of OCR, optical character recognition, Recognition Technologies Users Association, 1982.

[3] „History-Computers.com," [Online]. Available: https://history-computer.com/ModernComputer/Basis/OCR.html. [Přístup získán 1 10 2019].

[4] A. Chaudhuri, K. Mandaviya, P. Badelia a S. K Ghosh, Optical Character Recognition Systems for Different Languages with Soft Computing, Springer, 2017.

[5] M. Čermák, *Metody využívané pro OCR,* Brno, 2017.

[6] SimpleSoftware, „https://www.simpleocr.com/ocr-servers/," [Online]. Available: https://www.simpleocr.com/ocr-servers/. [Přístup získán 3 10 2019].

[7] J. Duffy, „Zapier," 3 9 2018. [Online]. Available: https://zapier.com/blog/best-mobile-scanning-ocr-apps/. [Přístup získán 8 10 2019].

[8] „Oki Europe," [Online]. Available: https://www.oki.com/cz/printing/services-and-solutions/smart-solutions/ocr-with-abbyy/index.html. [Přístup získán 6 10 2019].

[9] „ABBYY," [Online]. Available: https://www.abbyy.com/en-ee/finereader/?gclid=CjwKCAiA3abwBRBqEiwAKwICA6WtZO7qG9o0wsWV1LUvbSp9l7a_vuXbn1kwbrMYYURCticEsnOD1hoCT0AQAvD_BwE. [Přístup získán 5 10 2019].

[10] „GORC," [Online]. Available: http://jocr.sourceforge.net/. [Přístup získán 30 10 2019].

[11] „Cloud Google," 30 10 2019. [Online]. Available: https://cloud.google.com/vision/docs/ocr.

[12] L. Vincent, „The official Google Code blog," [Online]. Available: http://googlecode.blogspot.com/2006/08/announcing-tesseract-ocr.html. [Přístup získán 2019 10 30].

[13] H. Modi a M. C. PhD Parikh, „A Review on Optical Character Recognition Techniques," v *International Journal of Computer Applications (0975 – 8887),* 2017.

[14] Y. Alginahi, Preprocessing Techniques.

[15] „TutorialPoint," [Online]. Available: https://www.tutorialspoint.com/dip/grayscale_to_rgb_conversion.htm#:~:targetText=So%20the%20new%20equation%20that,and%20Blue%20has%20contributed%2011%25.. [Přístup získán 22 11 2019].

[16] P. Puneet a K. G. Naresh, „Binarization Techniques used for Grey Scale Images," v *International Journal of Computer Applications*, 2013.

[17] D. Phillips, Image Processing in C Second Edition Dwayne PhillipsThis first edition of "Image Processing in C".

[18] Hossain, M. A. a. Afrin a Sadia, „Optical Character Recognition By Using Template Matching," *Global Journal of Computer Science and Technology,* pp. 31-35, 5 2019.

[19] Y. Bassil a M. Alwani, „OCR Post-Processing Error Correction Algorithm,“ v *Journal of Emerging Trends in Computing and Information Sciences*, 2012.

[20] P. Veselý, „Diplomová práce Masarykova univerzita,Fakulta informatiky, Brno 2006,“ [Online]. Available: https://is.muni.cz/th/51661/fi_m/.

[21] „Wikipedie,“ [Online]. Available: https://cs.wikipedia.org/wiki/%C4%8Ce%C5%A1tina#Po%C4%8Det_slov. [Přístup získán 22 11 2019].

[22] „Processing,“ [Online]. Available: https://processing.org/tutorials/color/. [Přístup získán 22 11 2019].