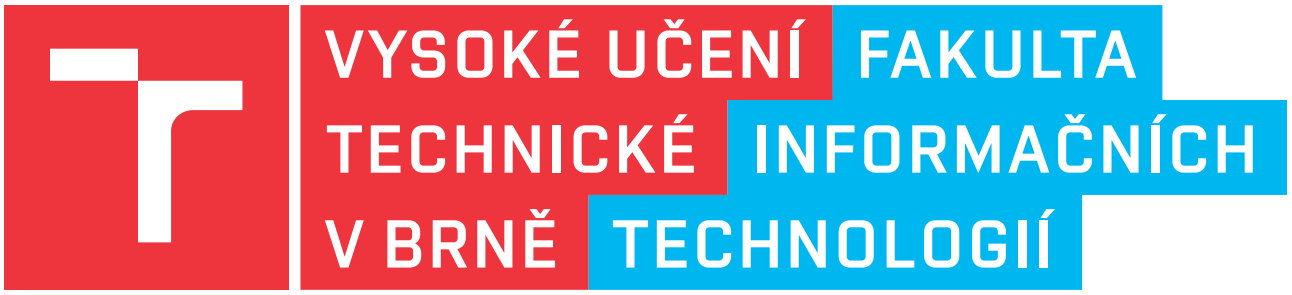


Klasifikátor třídy otisků prstů
Biometrické systémy

Nikola Valešová (xvales02)
Ondřej Valeš (xvales03)

19. listopadu 2017

1 Úvod

Technologie rozpoznávání osob na základě jejich otisků prstů se začala používat již na počátku 20. století [1], až během posledních let se však dočkala významného rozšíření i do běžného života. Díky tomu, že na snímání této biometrické charakteristiky lze použít relativně malé a levné senzory, můžeme dnes tento způsob zabezpečení vidět například i na mobilních telefonech, noteboocích, počítačových myších a přístupových systémech.

Cílem tohoto projektu je vytvořit aplikaci, která provede předzpracování požadovaného otisku prstu, poté v něm nalezne význačné atributy a na jejich základě daný otisk klasifikuje do jedné ze známých tříd.

2 Třídy otisků prstů

V otiscích prstů obecně můžeme pozorovat dva typy míst význačných pro klasifikaci, a to delty (místa, kterými probíhají papilární linie do tři různých směrů) a jádra (nacházející se ve středu otisku, jedná se o nejvyšší, či nejnižší vyklenutí papilární linie). Na základě počtu a vzájemné polohy těchto rysů můžeme poté rozdělit otisky prstů do několika tříd:

- oblouk neobsahuje žádnou deltu ani jádro, všechny papilární linie probíhají vodorovně, případně s mírným vyklenutím,
- klenutý oblouk obsahuje jedno jádro a jednu deltu, které se nachází pod sebou přibližně uprostřed prstu,
- spirála se vyznačuje dvěma deltami a dvěma jádry, přičemž jádra se nachází výše než delty a zároveň jsou blíže u sebe,
- levá smyčka obsahuje jedno jádro a jednu deltu, kde delta je napravo od jádra,
- pravá smyčka je podobná levé smyčce, delta je však nalevo od jádra,
- dvojitá smyčka sestává ze dvou delt a dvou jader, kde se každá singularita nachází v jiné výšce.

Klasifikace otisků prstů má velký význam především při identifikaci osob, jelikož místo prohledávání celé databáze můžeme výběr zúžit na danou třídu a snížit tak dobu potřebnou k získání výsledku.

3 Implementace

Projekt se skládá ze dvou částí. První je nedokončený klasifikátor v jazyce C/C++, který neměl být založen na žádném frameworku. V průběhu práce na projektu však vyšlo najevo, že tuto verzi nebudeme schopni v rámci projektu dovést do úspěšného konce. Druhou částí je proto klasifikátor v jazyce Python založený na existujícím frameworku, který je již plně funkční, ale využívá některé elementy frameworku.

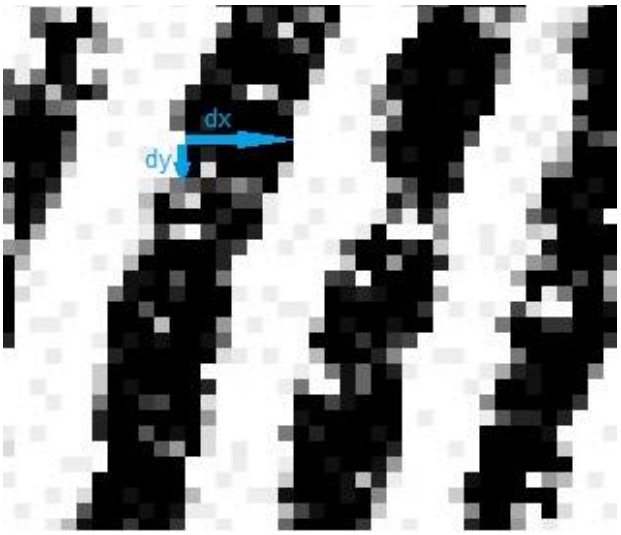
3.1 Projekt v C/C++

První část projektu měla za cíl vytvořit klasifikátor otisků prstů v jazyce C/C++, který není vybudovaný nad žádným existujícím frameworkem. Jeho součástí mělo být předzpracování sejmutého otisku, určení pole orientací, skeletonizace a následné určení počtu a poloh jader i delt, podle kterých měla být provedena výsledná klasifikace.

Předzpracování bylo provedeno prahováním. V rámci projektu jsme vyzkoušeli prahování konstantní hodnotou, adaptivní prahování průměrem, adaptivní prahování pomocí mediánu a na závěr i výpočet pole orientací z nepředzpracovaného otisku, která se paradoxně ukázala jako nejlépe fungující.

3.1.1 Orientace papilárních linií

Výpočet pole orientací je nastíněn na obrázku 1. Nejprve byly vypočteny gradienty dx a dy obrázku otisku, k tomu jsme použili nejprve Sobelův operátor a poté i přímý výpočet gradientu ze sousedních hodnot. Z nich jsme potom vypočítali normály a orientace papilárních linií v jednotlivých bodech (podle vzorců uvedených na obrázku 1). Celý otisk byl poté rozdělen na čtvercové oblasti o šířce W, pro které byla určena průměrná hodnota orientací papilárních linií.



Obrázek 1: Ilustrace výpočtu pole orientací

3.1.2 Výpočet průměrné orientace

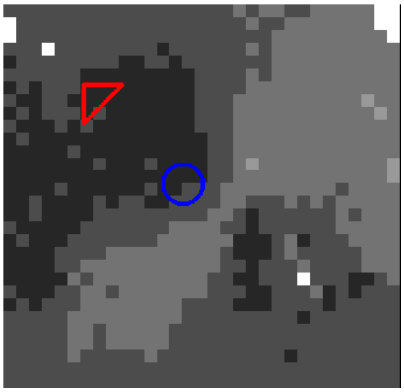
Experimentovali jsme s několika způsoby výpočtu průměrných orientací. Nejprve jsme použili výpočet jednoduchého průměru, tento přístup však vedl na pole orientací, ve kterém vůbec nebyl zřetelný původní otisk a veškeré hodnoty orientací se blížily nule. Došli jsme k závěru, že je to způsobeno průměrováním s hodnotami, kde je okolí přibližně stejné intenzity jako daný bod (například veprostřed šířky papilární linie nebo v mezerách mezi liniemi). Kvalita se nijak nelepšila ani s použitím výše uvedených způsobů předzpracování.

Dalším vyzkoušeným přístupem bylo ořezání extrémních hodnot. Z vypočtených orientací v jednotlivých bodech bylo tedy pro každou oblast zanedbáno $10\,\%$ nejmenších hodnot. Tento přístup

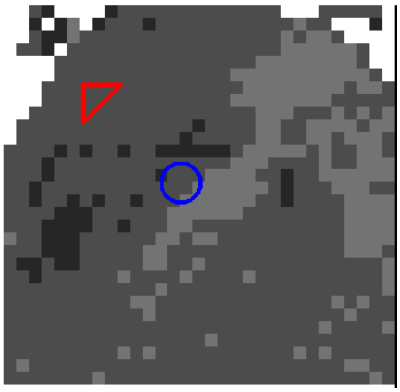
přinesl problémy v oblastech, kde skutečná hodnota orientací byla blízká nule. V těchto oblastech vznikaly nepřesnosti způsobené ořezáním hodnot, ze kterých měl být průměr ve skutečnosti počítán, a naopak byly započteny velké hodnoty vzniklé náhodným šumem. Náhodný šum se nám sice podařilo do jisté míry odstranit s využitím předzpracování, ale výsledky zůstaly silně závislé na vstupním otisku. Zvláště otisky, které obsahovaly příliš silné papilární linie nebo mezery mezi nimi, produkovaly nepoměrně více bodů s malým gradientem v okolí a ořezání extrémů se s tímto nedokázalo vyrovnat.

Posledním a nejúspěšnějším přístupem byl výpočet orientací pouze z bodů, jejichž gradient byl větší než pevně daný práh. To odstranilo problémy se silnými papilárními liniemi a mezerami. Dále také umožnilo výpočet pole orientací i pro vodorovné a svislé papilární linie (protože na jejich krajích měly body gradient dx nebo dy vždy větší než je hodnota prahu). Ve výsledném poli orientací byl poprvé zřetelný původní otisk a metoda fungovala pro otisky s různě silnými papilárními liniemi. Tento přístup je v kombinaci s výpočtem gradientu přímo ze sousedních hodnot a bez metod předzpracování obrazu v odevzdaném archivu.

(a) Původní snímek otisku



(b) Pole orientací vypočtených s ořezáním extrémních hodnot



(c) Pole orientací vypočtené se zanedbáním pozic s malým gradientem

Obrázek 2: Výsledky výpočtu pole orientací pro různé postupy

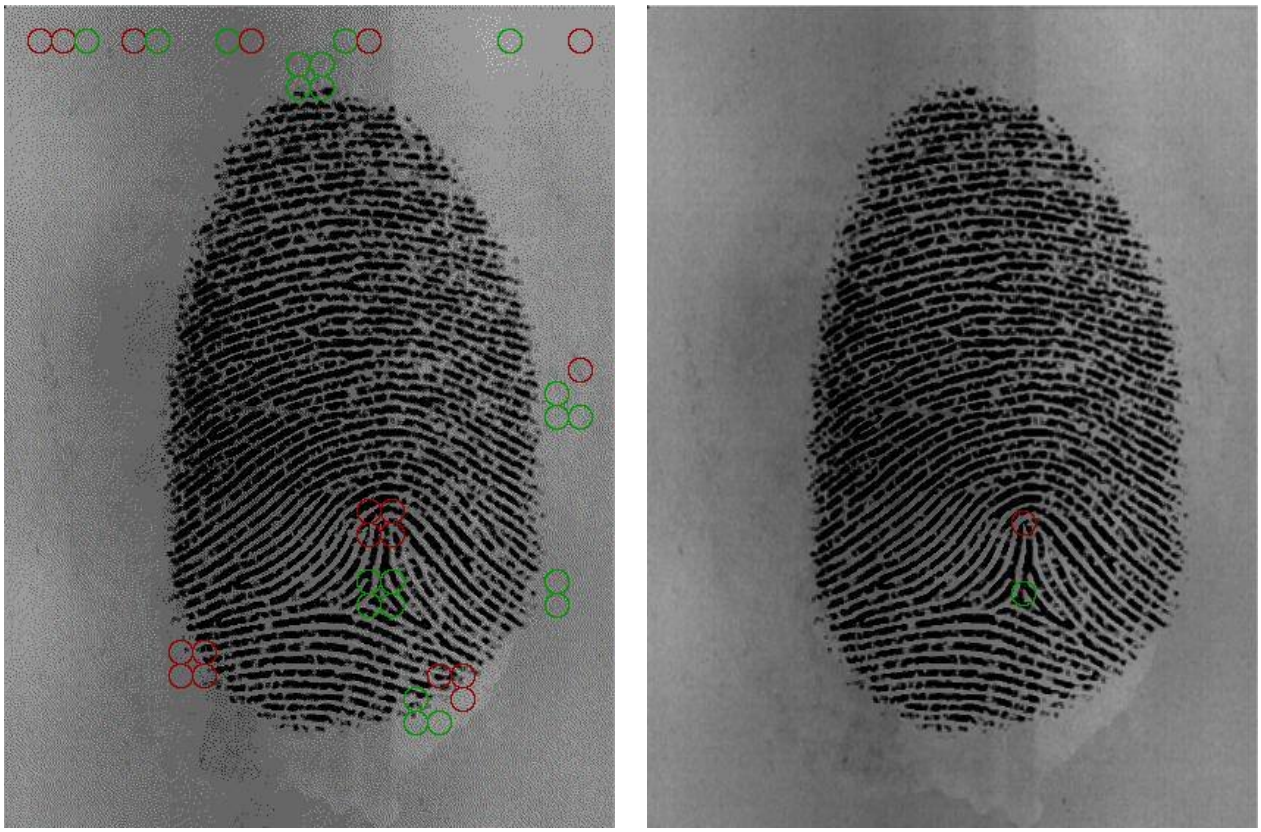
Naprogramování výpočtu pole orientací nám zabralo nepoměrně mnoho času. Z důvodu nejistoty ohledně vhodnosti našeho pole orientací pro hledání delt a jader jsme byli nuceni uznat, že naprogramování celého procesu klasifikace by bylo v rámci tohoto projektu nad naše možnosti. Proto jsme se rozhodli pro změnu – využít existující framework, který bude vyhledávat delty a jádra, a vybudovat náš projekt nad ním.

3.2 Projekt v jazyce Python

Použili jsme framework rtshadow [3] (přiložen v archivu) napsaný v jazyce Python, z tohoto důvodu je druhá část našeho projektu v témže jazyce. Tento framework jsme vybrali, protože implementuje pouze část námi řešeného problému, což nám dalo prostor daný framework rozšířit a ne pouze replikovat jeho existující funkcionalitu.

3.2.1 Algoritmus hledání singularit v rtshadow

V původní implementaci funguje algoritmus hledání delt a jader nad nezpracovaným otiskem a produkuje značné množství nesprávně určených singularit. Jedná se o výpočet Poincareho indexu [2], který určuje typ singularit na základě změn orientací v osmi-okolí. V průměru v každém otisku identifikuje 10 až 20 singularit (viz obrázek 3 (a)). Mezi hlavní nedostatky patří duplikování singularit, nalezení neexistujících singularit na okraji otisku a nalezení neexistující singularity v šumu a na okraji snímku.



(a) Delty a jádra nalezené algoritmem v rtshadow [3] (b) Výsledek po odstranění duplicit a falešných nálezů

Obrázek 3: Porovnání původního algoritmu na vyhledávání singularit (a) a námi upravené verze (b)

V našem projektu jsme provedli vylepšení algoritmu pro hledání singularit a následně klasifikaci otisku na základě počtu nebo poloh nalezených delt a jader, přičemž z původního frameworku jsme nevyužili žádnou další funkcionalitu kromě zmíněného algoritmu pro hledání singularit.

Odstranění duplicit
V prvním kroku jsme se snažili odstranit duplicitně nalezené singularity. Zjistili jsme, že množství duplicitních nálezů závisí hlavně na velikosti oblastí, pro které jsou počítány průměrné orientace. Experimentálně jsme se pro snímky otisků o velikosti 512 px $\times$ 512 px dostali na šířku oblasti 12 px až 20 px (tedy přibližně 1/32 šířky snímku). Při využití této šířky oblasti algoritmus nalezne duplicitně singularity pouze v osmi-okolí skutečné pozice, dalším vylepšením je tedy kontrola duplicit v osmi-okolí a odstranění duplicitních nálezů.

Odstranění falešně detekovaných singularit
Odstranění neexistujících singularit na okraji otisku bylo provedeno výpočtem průměrné intenzity

v oblastech sousedících se singularitou (pokud se oblast nachází na okraji otisku, tak alespoň v jedné ze sousedních oblastí se nenachází papilární linie, a tato oblast je v průměru světlejší). Tento přístup dokázal odstranit i chybnou detekci singularit způsobenou šumem. Dále byl řešen problém s detekcí neexistujících singularit na okraji snímku, řešením je jednoduché odstranění singularit nacházejících se v konstantní vzdálenosti od okraje.

3.2.2 Námi upravený algoritmus detekce singularit

Po provedení těchto úprav byl počet detekovaných singularit v otisku snížen na $0-4$ (viz obrázek 3 (b)), to je množství vhodné pro provedení klasifikace. Klasifikátor funguje na jednoduchém principu porovnání počtu rozpoznaných jader a delt. V případě levé smyčky, pravé smyčky a klenutého oblouku dochází k porovnání vzájemné pozice singularit.

Výsledný program v Pythonu spolehlivě klasifikuje do čtyř kategorií, kterými jsou:

- levá smyčka,
- pravá smyčka,
- dvojitá smyčka, nebo vír,
- oblouk, nebo klenutý oblouk.

Problémy v rozlišení dvojité smyčky a víru plynou ze způsobu detekce, který využívá framework rtshadow. Ten sice umožňuje detekovat i víry, ale v praxi je to možné pouze v případě, že vír má téměř dokonale kulatý tvar a stejnou šířku, jako je šířka použitých oblastí. Rozlišení oblouku a klenutého oblouku je závislé na výšce vyklenutí. Pokud se jedná o dokonalý klenutý oblouk, je naším programem klasifikován správně, ale pokud není v otisku jasně zřetelná delta nebo jádro, nebo pokud částečně splývají, otisk je vždy klasifikován jako oblouk.

4 Instalace a spuštění

Aplikace využívá prostředky knihovny *Python Imaging Library*, kterou je možno doinstalovat s využitím příkazu `pip install pil`. Poté lze aplikaci spustit pomocí příkazu:

```
python fingerprint_classifier.py [image_path] [width] [tolerance],
```

kde prvním parametrem je cesta k souboru s otiskem prstu určenému ke klasifikaci. Druhý parametr udává šířku bloku, přičemž zde se nejvíce osvědčila hodnota 16. Posledním parametrem je tolerance, jejíž ideální hodnota je rovna 1.

5 Závěr

V rámci tohoto projektu vznikly dvě aplikace, první v jazyce C/C++, která neprovádí klasifikaci velmi spolehlivě, avšak je kompletně naším autorským dílem. Druhá aplikace je napsána v Pythonu a využívá framework rtshadow pro detekci singularit v klasifikovaných otiscích, díky čemuž se nám podařilo vytvořit spolehlivější klasifikátor, jehož výsledky jsme se co nejvíce snažili zpřesnit předzpracováním vstupního obrazu a filtrováním nalezených singularit.

Reference

[1] Prof. Ing., Dipl.Ing. Martin Drahanský, Ph.D. *Biometrické systémy, Studijní opora* [online], [cit. 2017-11-15]. Dostupné z: https://www.fit.vutbr.cz/study/courses/BIO/private/BIO_Studijni_opora.pdf.

[2] Maltoni, Davide a spol. Handbook of fingerprint recognition. 2. London: Springer, 2009. ISBN 1848822537.

[3] Przemysław Pastuszka. *rtshadow/biometrics: fingerprint recognition, etc.* [online], [cit. 2017-11-18]. Dostupné z: https://github.com/rtshadow/biometrics.