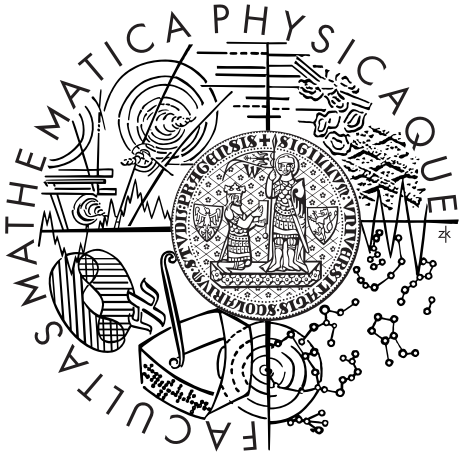
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE



Petr Vyhlas

Porovnávání podpisů pomocí metrik pro oční pohyby

Kabinet software a výuky informatiky

Vedoucí bakalářské práce: Mgr. Filip Děchtěrenko
Studijní program: Informatika
Studijní obor: Programování

Praha 2016

Chtěl bych poděkovat hlavně svému vedoucímu Mgr. Filipu Děchtěrenkovi za odborné vedení, trpělivost, rady, ochotu a pomoc kdykoliv jsem potřeboval. Také bych chtěl poděkovat své rodině za podporu, bez které bych práci nedokončil. Jsem vděčný všem, kteří se účastnili experimentu. Nakonec bych chtěl poděkovat RNDr. Haně Vyhlasové za pomoc s experimentem a korekturu této práce.

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Univerzita Karlova v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle §60 odst. 1 autorského zákona.

V ..... dne ..... Podpis autora

Název práce: Porovnávání podpisů pomocí metrik pro oční pohyby

Autor: Petr Vyhlas

Katedra: Kabinet software a výuky informatiky

Vedoucí bakalářské práce: Mgr. Filip Děchtěrenko, Kabinet software a výuky informatiky

Abstrakt: Porovnávání podpisových vzorů je součástí procesu ověření identity. Práce se zabývá porovnáváním podpisových vzorů pomocí metrik pro porovnávání očních pohybů. První část práce tvoří shrnutí algoritmů pro ověřování podpisových vzorů a představení metrik pro porovnávání očních pohybů. Využili jsme Levenshteinovu vzdálenost, Fréchetovu vzdálenost a vzdálenosti pomocí korelačního koeficientu k porovnávání podpisových vzorů. Zjistili jsme chování těchto metrik a zvolili arbitrární vzorec k výpočtu procentuální podobnosti dvou podpisů. Navrhli a naimplementovali jsme univerzální Windows aplikaci, která digitalizuje podpis, porovnává dva podpisy a určí míru jejich podobnosti. Provedli jsme experiment, ve kterém účastníci opakovaně vytvářeli vlastní podpis nebo věrohodně napodobovali cizí podpisový vzor. Z výsledků nevyplynulo, že se zlepšovali.

Klíčová slova: Porovnávání podpisových vzorů, Levenshteinova vzdálenost, Fréchetova vzdálenost, Korelační koeficient, UWP aplikace, PRISM

Title: Comparison of signatures using metrics for the eye movements

Author: Petr Vyhlas

Department: Department of Software and Computer Science Education

Supervisor: Mgr. Filip Děchtěrenko, Department of Software and Computer Science Education

Abstract: Comparison of signatures is part of the identity verification. The study focuses on comparison of signatures using metrics for the eye movements. In first part of study we review algorithms for the signature verification and we introduce metrics for comparing eye movements. We used Levenshtein distance, Fréchet distance and correlation coefficient for the comparison of signatures. We discovered behavior of these metrics and choose their combination for the computation of percentage similarity between two signatures. We designed and implemented universal Windows application which digitizes signatures, compares two signatures and determines their similarity. We conducted an experiment in which participants tried to sign themselves or tried to sign someone else. We did not find improvement during the signing.

Keywords: Comparison of signatures, Levenshtein distance, Fréchet distance, Correlation coefficient, UWP application, PRISM

Obsah

Úvod

Podpisový vzor je ručně psané a často stylizované vypsání jména nebo jiného identifikačního znaku osoby, která podpis napsala (Burton, 2007). Podpis je velmi významný při ověřování identity díky jeho obtížnému padělání a také jeho snadnému vytvoření. Vzhledem k velké popularitě chytrých mobilních telefonů, tabletů a malých počítačů s kapacitním displejem má ověřování podpisového vzoru stále větší potenciál při ověření identity uživatele. V poslední době se pomalu objevují speciální pera, která v kombinaci s dotykovým dispelejem dávají podobné informace jako speciální hardware pro záznam podpisů (Faundez-Zanuy, 2007). Takový systém ověřování má potenciál nahradit PIN nebo podobné přístupy pro odemykání operačního systému. Využití tohoto systému má stejné výhody jako u všech biometrických metod (De Luis-García et al., 2003), že si identifikovaný subjekt nemusí pamatovat nic navíc a tato metoda jednoznačně identifikuje danou osobu. Hlavní nevýhodou je velká variabilita při podepisování. Cílem této práce je vytvořit aplikaci, která pomáhá lidem stabilizovat jejich podpisový vzor a získat větší jistotu při jeho tvorbě. V této aplikaci využijeme metriky pro porovnávání očních pohybů k porovnávání podpisových vzorů.

Nejprve se seznámíme s konceptem ověřování podpisového vzoru (Faundez-Zanuy, 2007; Fierrez et al., 2007). Poté ukážeme podobnost mezi digitalizovaným podpisovým vzorem a očními pohyby, které vznikají při úloze plynulého sledování objektu (Smooth pursuit). V této práci jsme navrhli a naimplementovali univerzální windows aplikaci(Microsoft, 2016f), která digitalizuje podpisový vzor a umožňuje procvičování podpisu proti zvolenému vzoru. K porovnávání podpisů jsme zvolili tři metriky pro porovnávání očních pohybů. Levenshteinovu vzdálenost (Levenshtein, 1966b), která reprezentuje vzdálenost mezi dvěma textovými řetězci. Fréchetovu vzdálenost (Eiter and Mannila, 1994), která reprezentuje délku nejkratšího vodítka potřebného pro psa a pána k překonání cesty po dvou křivkách a vzdálenost pomocí korelačního koeficientu dvou trojrozměrných charakteristických map (Meur and Baccino, 2012). Vyřešili jsme problém normalizace zaznamenaného podpisu pomocí projekce do středu pevně definovaného prostoru 80x80 pixelů při zachování poměru stran. Dále jsme prozkoumali chování jednotlivých metrik v závislosti na různých transformacích aplikovaných na podpisový vzor a porovnali podpisový vzor s různými primitivními podpisy (například vodorovná čára, svislá čára, ...). Zvolili jsme arbitrární vzorec, který kombinuje výsledky těchto tří metrik a převede je na procentuální podobnost dvou podpisových vzorů srozumitelnou pro koncové uživatele. Experimentálně jsme zjišťovali, jestli se účastníci experimentu zlepšují při tvorbě vlastního nebo cizího podpisového vzoru.

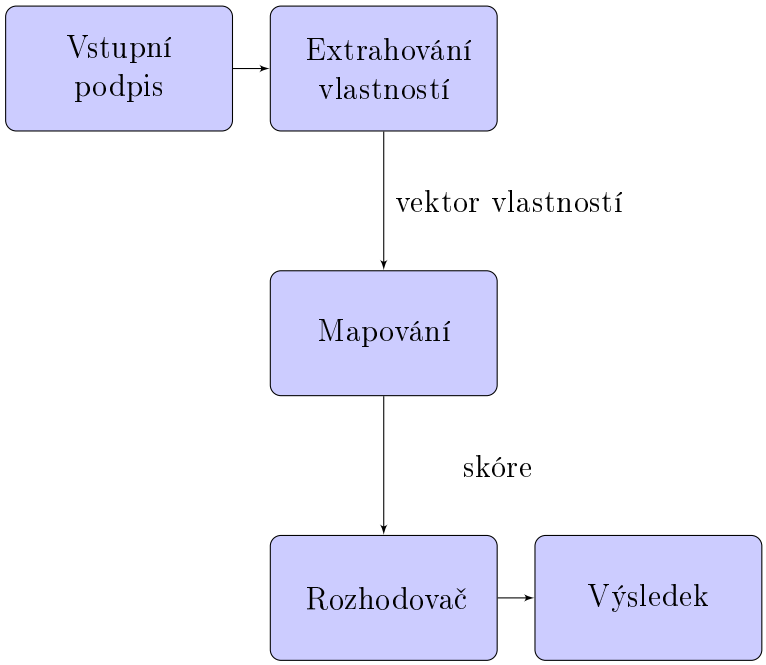
Na závěr představíme problémy, které jsme vyřešili a nevyřešili a nabídneme témata pro rozšíření této práce a přidružené projekty.

1. Ověření podpisu

V současné době existuje mnoho biometrických metod k ověření identity (De Luis-García et al., 2003). Mezi ně patří například rozpoznávání otisků prstů, oční duhovky, sítnice, podpisového vzoru a dalších (De Luis-García et al., 2003).Tyto metody většinou vyžadují speciální hardware, který je drahý. Vytvoření podpisového vzoru na papír je levné a snadné, proto se podpisové vzory používají na šecích a také v právním systému jako identifikátor podepisovatele (Burton, 2007).V této kapitole vysvětlíme, jak obvykle funguje ověřování podpisových vzorů (De Luis-García et al., 2003; Faundez-Zanuy, 2007; Fierrez et al., 2007; Piyush Shanker and Rajagopalan, 2007). Poté zmíněné algoritmy rozdělíme podle vstupních dat na statické (Piyush Shanker and Rajagopalan, 2007) a dynamické (Faundez-Zanuy, 2007; Fierrez et al., 2007). Následně tyto skupiny popíšeme.

1.1 Jak funguje ověřování podpisových vzorů

Existuje velké množství různých algoritmů, které řeší tento problém (De Luis-García et al., 2003; Faundez-Zanuy, 2007; Fierrez et al., 2007; Piyush Shanker and Rajagopalan, 2007). Ukážeme obecný postup, jak tyto algoritmy většinou fungují. Algoritmus nejprve dostane na vstupu ověřovaný podpis. Poté si z něj extrahuje důležité vlastnosti. Podpis reprezentovaný vlastnostmi zkusí namapovat na již známý podpis. Podle úspěšnosti mapování buď podpis úspěšně ověří, nebo zamítne. Celé schéma je dobře vidět v následujícím obrázku.



Obrázek 1.1: Schématické zobrazení obecného postupu pro ověřování podpisových vzorů

Tyto algoritmy se obvykle liší v extrakci vlastností, mapování a přijímání výsledků. Obvykle se extrahují tři typy vlastností. Vlastnosti závislé na celém podpisu, například vyplnění plochy podpisem (Qi and Hunt, 1994). Dále závislé

na jednotlivých čarách (Qi and Hunt, 1994) a nakonec na jednotlivých bodech (dynamicky zaznamenané body). Způsobů mapování na již známý podpis je velké množství a je to hlavní část, kde se jednotlivé algoritmy liší. Výsledky se dále přijmou nebo zamítnou podle stanovené limitní hodnoty pro dané mapování.

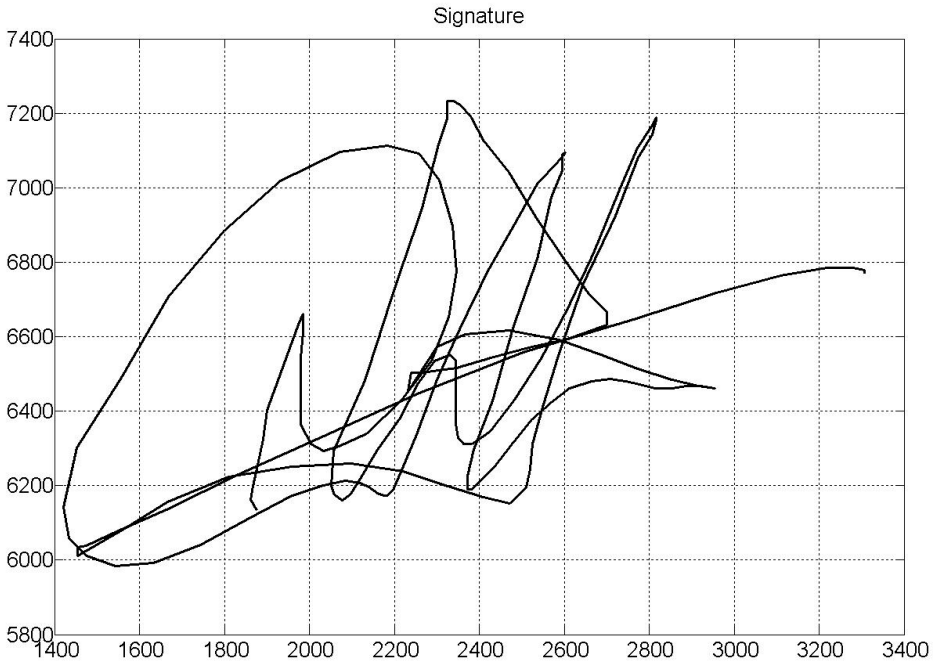
1.2 Statické algoritmy (offline)

Statické neboli offline algoritmy (Qi and Hunt, 1994; Piyush Shanker and Rajagopalan, 2007) se vyznačují tím, že porovnávají naskenované podpisy. Jejich vstupem je obrázek, který převedou na podpis. Z toho vyplývá, že ze vstupních dat není známa časová posloupnost podpisu. To je hlavní rozdíl oproti dynamickým algoritmům. Při použití naskenovaného podpisu hrozí mnohem větší nebezpečí padělání podpisu, protože celá informace o podpisovém vzoru bývá často veřejně dostupná buď na šecích, smlouvách nebo jiných dokumentech. Padělatel může takový podpis nastudovat a pro úspěšné padělání stačí vytvořit stejně vypadající podpis. Toto je nevýhoda uvedeného přístupu. Existuje mnoho offline algoritmů, mezi ně patří skryté Markovovy modely (Justino et al., 2001), neuronové klasifikátory (Bajaj and Chaudhury, 1997), dynamic time warping (Piyush Shanker and Rajagopalan, 2007) a mnoho dalších.

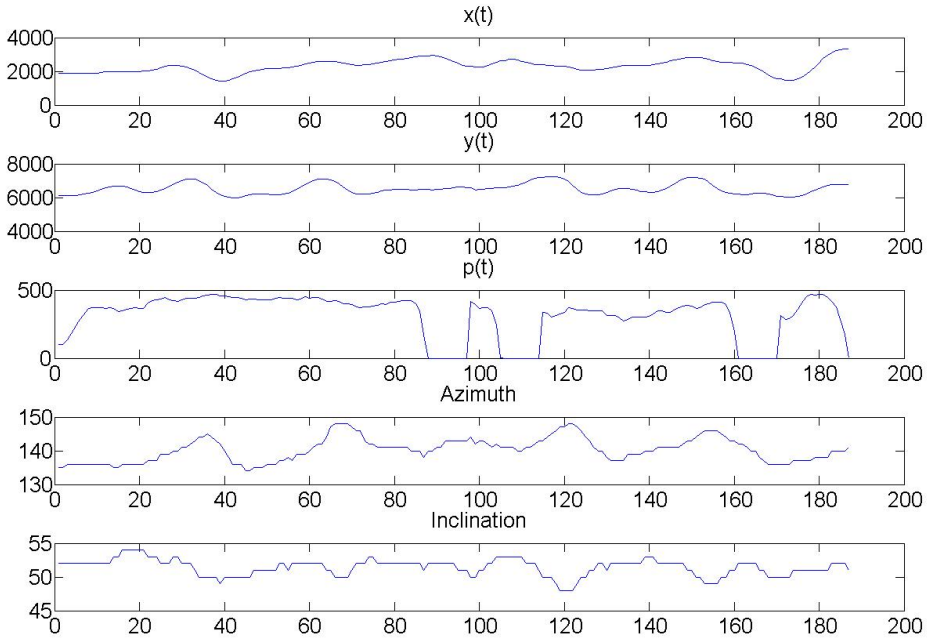
Obrázek 1.2: Příklad podpisového vzoru pro statické algoritmy.

1.3 Dynamické algoritmy (online)

Dynamické neboli online algoritmy (Faundez-Zanuy, 2007; Fierrez et al., 2007) přijímají vstupní podpis, který obsahuje časovou značku. Obvykle se pro záznam používá speciální hardware, který zaznamenává koordináty x a y, přítlak pera, azimut a sklon v čase (Faundez-Zanuy, 2007). Pro záznam se také mohou použít zařízení s kapacitním displejem a stylusem. Stylus je pero, které má na konci speciální hrot potažený gumou. Slouží pro psaní na kapacitní displeje. Tato zařízení zaznamenávají pouze koordináty x a y v čase. Padělání takto zaznamenaných podpisů je velice obtížné (Faundez-Zanuy, 2007; Fierrez et al., 2007). Při znalosti podpisu pouze z obrázku nemá padělatel znalosti o tempu podpisu, přítlaku v jednotlivých částech, náklonu ani azimutu pera. Stejně vypadající podpis jako v případě statických algoritmů se velikou jistotou zamítne. I při znalosti kompletně zaznamenaného podpisu je padělání obtížné, protože padělatel musí věrohodně



Obrázek 1.3: Příklad dynamického podpisového vzoru. (Dynamický podpis)



Obrázek 1.4: Dynamické parametry podpisu x, y, přítlak, azimut a sklon v čase. (Dynamický podpis parametry)

napodobit mnohem více parametrů než u statických podpisů. Existuje celá řada dynamických algoritmů, mezi ně patří například vector quantization, nejbližší soused, dynamic time warping a další (Faundez-Zanuy, 2007).

1.4 Shrnutí

V této kapitole jsme vysvětlili, jak obvykle funguje ověřování podpisových vzorů. Algoritmus nejprve dostane vstupní podpis. Poté si extrahuje vlastnosti. Dále namapuje vstupní podpis na předem známé podpisy. Nakonec přijme nebo zamítne výsledek mapování. Rozdělili jsme tyto algoritmy na statické a dynamické podle typu vstupu. Statické s naskenovaným obrázkem jako vstup a dynamické na vstup závislý na čase.

2. Metriky pro oční pohyby

Pro porovnávání očních pohybů existuje celá řada metod (Meur and Baccino, 2012). V této kapitole se nejprve seznámíme s očními pohyby. Ukážeme, jak znormalizovat podpis. Dále představíme jednotlivé metriky využité pro porovnávání podpisových vzorů. Představíme Levenshteinovu vzdálenost (Levenshtein, 1966b), Fréchetovu vzdálenost (Eiter and Mannila, 1994) a vzdálenost pomocí korelačního koeficientu (Meur and Baccino, 2012). Pro každou metriku ukážeme algoritmus pro její výpočet a určíme časové a prostorové složitosti. Následně porovnáme použité metriky s algoritmy pro ověřováním podpisů.

2.1 Úvod do měření očních pohybů

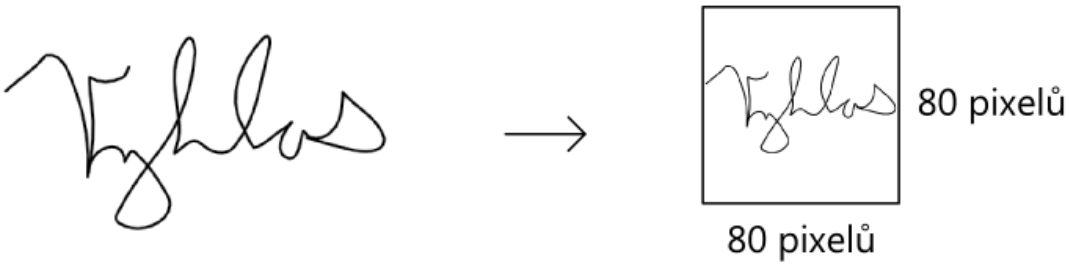
Měření očních pohybů u lidí se primárně provádí pomocí bezkontaktního kamerového zařízení eyetracker (například EyeLink II). Hlavním výstupem tohoto zařízení jsou čas, x, y a velikost zorničky. Začátky sakád a fixací. Sakády jsou krátké rychlé pohyby pro skenování vizuální scény (Holmqvist et al., 2011). Při čtení se naše oči nepohybují plynule po řádcích, ale provádí sérii krátkých rychlých pohybů neboli sakád a zastaveních (Rayner, 1998). Hlavním použitím sakadických pohybů je schopnost rychle přeskakovat a tím naskenovat větší plochu pomocí žluté skvrny v oku. Mezi jednotlivými sakádami jsou zastavení neboli fixace. V tuto dobu je oko relativně v klidu. Fixace většinou trvá mezi 225 a 330 milisekundami (Holmqvist et al., 2011). Dalším typem očního pohybu je plynulé sledování (Holmqvist et al., 2011), je to mnohem pomalejší pohyb než sakáda, při kterém se oko plynule pohybuje a soustředí se na nějaký pohybující se objekt (Cavanagh and Alvarez, 2005). Pohyby oka při plynulém sledování mají určitou podobnost s podpisovým vzorem. Jedná se o velmi spojité pohyby, proto je možné využít algoritmy pro porovnávání očních pohybů k porovnávání podpisových vzorů.

2.2 Normalizace podpisu

Záznam podpisového vzoru bude probíhat na různých zařízeních s různým rozlišením a dpi. Navíc aplikace pro záznam podpisu může mít pokaždé jinou velikost okna. Musíme tedy zajistit, aby se vzor i nově zaznamenaný podpis porovnávaly ve stejném prostoru. Podpisy budeme normovat. Zavedeme pevný rozměr 80x80 pixelů a do tohoto prostoru vždy převedeme zaznamenaný podpis. Převod provedeme následujícím způsobem. Určíme, jestli je podpis širší nebo vyšší. Poté naškálujeme větší stranu podpisu tak, aby odpovídala 80 pixelům při zachování poměru stran. Následně podpis posuneme tak, aby střed našeho podpisu odpovídal středu podpisu. Příklad normalizace podpisu je na obrázku 2.1.

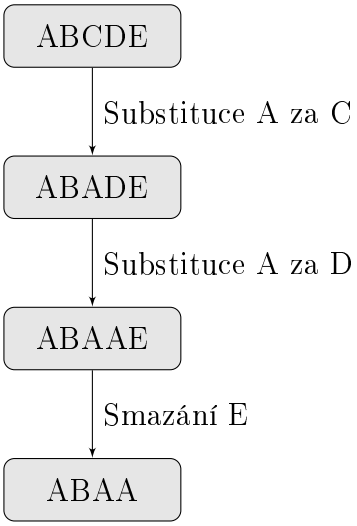
2.3 Levenshteinova vzdálenost

Levenshteinova vzdálenost (Levenshtein, 1966a) byla původně vytvořena k určení vzdálenosti mezi dvěma řetězci pomocí počtu vložení, mazání a substitucí, které jsou potřebné k převedení jednoho řetězce na druhý. Vzdálenost mezi řetězci



Obrázek 2.1: Příklad normalizace podpisového vzoru do prostoru 80x80 pixelů. Střed podpisu je zarovnaný se středem prostoru a šířka podpisu je zarovnaná na 80 pixelů.

je definovaná jako minimální počet takovýchto vložení, mazání a substitucí potřebných k převedení. V naivní implementaci bychom použili rekurzivní volání a dostali bychom exponenciální složitost. V našem řešení jsme použili Wagner-Fischerův algoritmus (Wagner and Fischer, 1974), který využívá dynamického programování (Eddy, 2004) a výrazně urychlí celkový výpočet. Nejprve ukážeme, jak tato metrika funguje při převodu řetězce ABCDE na ABAA na obrázku 2.2.



Obrázek 2.2: Příklad výpočtu vzdálenosti mezi ABCDE a ABAA. Vzdálenost je v tomto případě 3 a získá se například substitucí A za C, poté substitucí A za D a nakonec smazáním E.

Nyní bychom měli mít představu o funkci této metriky. Dále ukážeme, jak pomocí dynamického programování (DP) nalezneme LV pro libovolné řetězce. DP obvykle funguje tak, že řeší trochu jiný problém než je původní úloha a postupně si napočítává optimální řešení do matice, až se na konci algoritmu z této nové úlohy nechá vyčíst řešení původní úlohy. Budeme řešit úlohu LV všech prefixů daných řetězců a výsledky budeme ukládat do matice. Každé políčko v matici reprezentuje vzdálenost mezi prefixem tvořeným od začátku až do indexu sloupečku a prefixem od začátku až do indexu řádku. Algoritmus postupně vypočítává jednotlivá políčka matice po řádcích. Každé políčko je závislé na levém, horním a levém horním sousedovi. Pokud je poslední písmenko v prefixech stejné, převez-

me se hodnota z levého horního souseda. Jinak se vezme minimální hodnota ze zmíněných sousedů a přičte se k ní jednička. Postup tohoto řešení je dobře vidět v následující tabulce 2.1.

		k	**i**	**t**	**t**	**e**	**n**
	0	1	2	3	4	5	6
s	1	1	2	3	4	5	6
i	2	2	1	2	3	4	5
t	3	3	2	1	2	3	4
t	4	4	3	2	1	2	3
i	5	5	4	3	2	2	3
n	6	6	5	4	3	3	2
g	7	7	6	5	4	4	3

Tabulka 2.1: Názorná ukázka výpočtu LV mezi řetězci Sitting a Kitten pomocí dynamického programování. Výslednou vzdálenost 3 nalezneme v pravém dolním rohu.

Můžeme si všimnout, že není nutné mít v paměti celou matici, ale stačí nám vždy jen 2 řádky. Výsledek nalezneme na konci posledního řádku. Nyní ukážeme celý algoritmus pro výpočet LV 1. Nejprve zkontrolujeme triviální případy. Poté inicializujeme minulý řádek, ve kterém uchováváme napočítané hodnoty. Poté spustíme výpočet vzdáleností mezi prefixy prvního a druhého řetězce. Wagner-Fisherův algoritmus (WF) používá standardně celou matici pro uchovávání výsledků. Toto řešení se dá zlepšit na dva řádky matice. Algoritmus spočítá celou matici, z toho plyne časová složitost $O(l \cdot k)$, kde l a k jsou délky vstupních řetězců. Pří výpočtu využíváme pouze dva řádky matice, díky tomu je prostorová složitost $O(2 \cdot Min(l, k))$.

Pomocí tohoto algoritmu umíme porovnat dva řetězce mezi sebou, nyní zbývá převod podpisového vzoru na řetězec. Podpis reprezentujeme jako sérii koordinátů x a y v čase. Nejprve nalezneme ohraničující obdélník pro sjednocení obou podpisů. Do tohoto obdélníku promítneme tabulku s osmi sloupci a osmi řádky rovnoměrně rozdělenými. Každému políčku přidělíme zvláštní znak. Každému koordinátu x, y přiřadíme znak podle toho, do které buňky v tabulce patří. Tímto algoritmem převedeme podpis na řetězec.

LV může nabývat hodnot od 0 až do délky delšího řetězce. Tuto vzdálenost znormujeme pomocí následujícího vzorce $1 - \frac{distance}{Max(s_1.Length, s_2.Length)}$, kde $distance$ je vzdálenost spočítaná pomocí LV a s_1 a s_2 jsou řetězce (Meur and Baccino, 2012).

2.4 Fréchetova vzdálenost

Fréchetova vzdálenost (Eiter and Mannila, 1994) je vzdálenost mezi dvěma křivkami, určená minimální vzdáleností potřebnou k spojení psa a majitele vodítkem jdoucích každý po své křivce. Majitel ani pes se nesmí vracet zpátky. Oba začínají v prvním bodě a končí v posledním. Můžou se lišit v rychlosti chůze, můžou se i zastavit. Fréchetova vzdálenost (FD) je délka nejkratšího vodítka, pomocí kterého

Algorithm 1: Levenshteinova vzdálenost pomocí dynamického programování.

```
   Input: s1, s2:string
   Result: Vzdálenost mezi vstupními řetězci
 1 begin
 2     if s1 is empty then return s2.Length
 3     if s2 is empty then return s1.Length
 4     if s1 == s2 then return 0
 5     pLine ← ushort[s1.Length + 1]                    // předchozí řádek
 6     aLine ← ushort[s1.Length + 1]                     // aktuální řádek
                                              // Initializace nultého řádku
 7     for i = 0 to s1.Length - 1 do
 8         pLine[i] ← i
 9     end
                                                  // Výpočet řádků matice
10     for j = 0 to s1.Length - 1 do
11         aLine[0] ← j + 1
12         for i = 0 to s2.Length - 1 do
13             if s1[i] == s2[j] then
14                 aLine[i + 1] ← pLine[i]
15             else
16                 aLine[i+1] ← Min(pLine[i+1]+1, pLine[i]+1, aLine[i]+1)
17             end
18         end
19         tmp ← pLine                                  // prohození řádků
20         pLine ← aLine
21         aLine ← tmp
22     end
23 end
24 return pLine[s1.Length]
```

můžou oba přejít celou cestu. V našem případě máme na vstupu diskrétně reprezentovaný podpis. Můžeme proto využít diskrétní variantu FD, které se také říká dynamic time warping. Tato varianta využívá dynamického programování stejně jako u WF algoritmu. Pomocí DP nejprve spočítáme vzdálenosti mezi prefixy těchto křivek, výsledky ukládáme do matice. Výsledek opět nalezneme v pravém dolním rohu napočítané matice 2.1. Výpočet probíhá podobným způsobem, liší se ve způsobu výpočtu nové vzdálenosti. Pro výpočet vzdálenosti používáme euklidovskou vzdálenost mezi dvěma body.

Algorithm 2: Fréchetova vzdálenost pomocí dynamického programování

```
   Input: k1, k2:List<SignaturePoint>
   Result: Vzdálenost mezi vstupními podpisy
 1 begin
 2     pLine ← double[k1.Length + 1]                    // předchozí řádek
 3     aLine ← double[k1.Length + 1]                     // aktuální řádek
                                          // initializace prvního řádku
 4     for i = 1 to k1.Length - 1 do
 5         pLine[i] ←
             Max(SignaturePoint.Distance(k1[i − 1], k2[0]), pLine[i − 1])
 6     end
                                                  // výpočet řádků matice
 7     for j = 0 to s1.Length - 1 do
 8         aLine[0] ← Max(SignaturePoint.Distance(k1[0], k2[j]), pLine[0])
           for i = 0 to s2.Length - 1 do
 9             aLine[i + 1] ← Max(
                 Min(pLine[i + 1], pLine[i], aLine[i]),
                 SignaturePoint.Distance(k1[i], k2[j])
                 )
10         end
11         tmp ← pLine                                   // prohození řádků
12         pLine ← aLine
13         aLine ← tmp
14     end
15 end
16 return pLine[s1.Length]
```

Časová složitost i prostorová složitost je stejná, jako v případě LV, pouze nahradíme délku řetězce za počet bodů, kterými je definovaný podpis. Tedy časová složitost $O(l \cdot k)$, kde l a k jsou počty bodů v jednotlivých podpisech. Prostorová složitost je $O(2 \cdot Min(l, k))$, protože ukládáme pouze dva řádky matice.

FD nabývá vzdálenosti od 0 do úhlopříčky ohraničujícího obdélníku obou podpisů. Tuto vzdálenost znormalizujeme pomocí následujícího vzorce $1 - \frac{d}{max}$, kde max je úhlopříčka ohraničujícího obdélníku a d je vzdálenost spočítaná pomocí FD.

2.5 Vzdálenost pomocí korelačního koeficientu

Tento algoritmus spočítá korelaci dvou charakteristických map (Meur and Baccino, 2012). Nejprve vstupní podpisy převedeme na trojrozměrné charakteristické mapy 3. Charakteristická mapa má tři osy, X, Y a čas. Každý bod z podpisového vzoru převedeme podle jeho souřadnic do této mapy. Následně konvolujeme mapu s Gaussovým rozostřením (Mathematics and Mathematics, 2008). Dále spočítáme korelační koeficient takto získaných charakteristických map 4.

Algorithm 3: ComputeSaliencyMap - výpočet charakteristické mapy z podpisu.

Input: s_1:Signature, k,l,m,t,gl:int
Result: double[„] saliencyMap

begin
  $sm \leftarrow newdouble[k, l, m]$
  for $i = 0$ **to** $s_1.Points.Count - 1$ **do**
    $p \leftarrow s_1.Points[i]$ // uložení aktuálního bodu do p
    $xres \leftarrow p.X - Round(p.X, 0)$ // X za des. čárkou
    $yres \leftarrow p.Y - Round(p.Y, 0)$ // Y za des. čárkou
    $lx \leftarrow (int)p.X$ // menší celé X
    $ly \leftarrow (int)p.Y$ // menší celé Y
    $lt \leftarrow p.Timestamp/t$ // přihrádkování podle parametru
    $hx \leftarrow xres \geq 0 \ ? \ (lx + 1) : (lx - 1)$ // větší celé X
    $hy \leftarrow yres \geq 0 \ ? \ (ly + 1) : (ly - 1)$ // větší celé Y
    $d \leftarrow (x, y, x_2, y_2) \rightarrow \sqrt{(x - x_2)^2 + (y - y_2)^2}$
    $d_1 \leftarrow d(lx, ly, p.X, p.Y)$ // vzdálenosti od celých bodů
    $d_2 \leftarrow d(lx, hy, p.X, p.Y)$
    $d_3 \leftarrow d(hx, ly, p.X, p.Y)$
    $d_4 \leftarrow d(hx, hy, p.X, p.Y)$
    $dsum \leftarrow d_1 + d_2 + d_3 + d_4$ $w_1 \leftarrow dsum/d_1$ // kolik bod dostane
    $w_2 \leftarrow dsum/d_2$
    $w_3 \leftarrow dsum/d_3$
    $w_4 \leftarrow dsum/d_4$
    $wsum \leftarrow w_1 + w_2 + w_3 + w_4$
      // rozdělení mezi celé body, podle jejich vzdálenosti
    $sm[lx, ly, lt] \leftarrow sm[lx, ly, lt] + w_1/wsum$
    $sm[lx, ly, lt] \leftarrow sm[lx, ly, lt] + w_2/wsum$
    $sm[lx, ly, lt] \leftarrow sm[lx, ly, lt] + w_3/wsum$
    $sm[lx, ly, lt] \leftarrow sm[lx, ly, lt] + w_4/wsum$
  end
  // konvoluce s Gaussovým rozostřením
  $ker \leftarrow Gaussian.ThreeDimKernel(length = gl, deviation = 1)$
  $sm \leftarrow Convolution.Apply(sm, GaussianBlur, ker)$
end
return sm

Při převodu podpisu na charakteristickou mapu 3 postupně procházíme všechny body podpisu a přenášíme je do charakteristické mapy. U každého bodu spočí-

táme, mezi které souřadnice patří a rozdělíme pro tyto 4 body, kolik má každý dostat. Následně celou charakteristickou mapu konvolujeme s Gaussovým rozostřením (Mathematics and Mathematics, 2008).

Algorithm 4: Výpočet korelačního koeficientu.

Input: s1, s2:Signature
Result: Korelace mezi podpisy

begin
  $sm_1 \leftarrow ComputeSaliencyMap(s1, dimensions)$;
  $sm_2 \leftarrow ComputeSaliencyMap(s2, dimensions)$;
  $mean_1 \leftarrow Mean(sm_1)$;
  $mean_2 \leftarrow Mean(sm_2)$;
  $sumSq \leftarrow Sum(sm_1, 2)$;
  $sumSq_2 \leftarrow Sum(sm_2, 2)$;
  $cov \leftarrow 0$;
  for *i = 0* **to** *sm.Length(1) - 1* **do**
    for *j = 0* **to** *sm.Length(2) - 1* **do**
      for *k = 0* **to** *sm.Length(3) - 1* **do**
        $a \leftarrow sm_1[i, j, k] - mean_1$;
        $b \leftarrow sm_2[i, j, k] - mean_2$;
        $cov \leftarrow cov + (a * b)$;
      end
    end
  end
  $\sigma \leftarrow \sqrt{\frac{1}{|sm_1|} * sumSq - mean_1^2}$;
  $\sigma_2 \leftarrow \sqrt{\frac{1}{|sm_1|} * sumSq_2 - mean_2^2}$;
end
return $(Max(0, \frac{cov/n}{(\sigma * \sigma 2)}))^2$;

Spočítáme korelační koeficient dvou charakteristických map a ten omezíme zdola nulou, protože nás zajímají pouze kladně korelující hodnoty. K výpočtu použijeme vzorec $cor(sm_1, sm_2) = \frac{cov(sm_1, sm_2)}{\sigma_1 \sigma_2}$, kde $cov(sm_1, sm_2)$ je kovariance mezi s_1 a s_2 a σ_1, σ_2 jsou směrodatné odchylky sm_1 a sm_2. Výsledek dále umocníme na druhou.

Nyní určíme časovou a prostorovou složitost. Buď l,k,m velikosti charakteristické mapy v jednotlivých dimenzích. Algoritmus nejprve vytvoří charakteristickou mapu, poté ji konvoluje s Gaussovým rozmazáním. Toto má složitost $O(l \cdot k \cdot m)$. Algoritmus dále vypočítá sumy jednotlivých map a průměry. To má také $O(l \cdot k \cdot m)$. Následně se spočítá kovariance se složitostí $O(l \cdot k \cdot m)$. Poté se podle vzorce spočítá výsledný korelační koeficient se složitostí $O(1)$. Celkově tedy dostaneme $O(l \cdot k \cdot m)$. Časová složitost je tedy lineární s celkovou velikostí charakteristické mapy. Prostorová složitost je $O(2 \cdot l \cdot k \cdot m)$, tedy $O(l \cdot k \cdot m)$.

2.6 Porovnání se systémy pro ověření podpisu

V našem řešení nejprve načteme podpis pomocí tabletu se stylusem. Tento podpis reprezentujeme jako sérii vektorů s koordinátem X, Y a časovou značkou. Jedná se tedy o dynamické řešení 1.3. Dále porovnáváme podpis proti zvolenému vzoru pomocí Levenshteinovy vzdálenosti, Frechetovy vzdálenosti a vzdálenosti pomocí korelačního koeficientu. Z těchto výsledků vytvoříme procentuální podobnost podpisových vzorů. Toto číslo je konečným výsledkem. Hlavní rozdíly proti obvyklému ověřování podpisů jsou v rozhodovací části. Naše řešení slouží primárně pro trénování podpisů a nerozhodujeme, jestli je podpis pravý nebo ne. Pokud bychom chtěli vytvořit rozhodovací část, tak využijeme pevně zvolenou hranici a všechny podpisy s vyšší podobností budou přijaty a ostatní budou zamítnuty (například 85%).

Diskrétní varianta Fréchetovy vzdálenosti se také používá při porovnávání a ověřování podpisových vzorů (Piyush Shanker and Rajagopalan, 2007; Faundez-Zanuy, 2007). Přesto jsme se jí rozhodli použít, protože je využitá také pro porovnávání očních pohybů (Le Meur and Liu, 2015). Levenshteinova vzdálenost se obvykle používá, pokud je ve vstupní prostoru předem známá oblast zájmů (Meur and Baccino, 2012). V našem řešení nemáme oblast zájmů, proto při převodu na řetězec rovnoměrně rozdělíme prostor a každé části přiřadíme znak.

2.7 Shrnutí

V této kapitole jsme se seznámili se záznamem očních pohybů a úlohou plynulého sledování objektů. Ukázali jsme, jak normalizujeme podpisový vzor. Představili jsme tři metriky a vysvětlili, jak fungují. Levenshteinovu vzdálenost, která porovnává vzdálenost textových řetězců. Fréchetovu vzdálenost, která určí jaké nejkratší vodítko je potřebné pro psa a majitele, aby spolu přešli od začátku do konce po křivkách. Vzdálenost pomocí korelačního koeficientu dvou charakteristických map. Ukázali jsme, jak se tyto problémy řeší pomocí dynamického programování a určili časové a prostorové složitosti. Poté jsme porovnali naše řešení se systémy pro ověřování podpisových vzorů.

3. Implementace

Navrhli a naimplementovali jsme aplikaci pro záznam a vyhodnocení podobnosti podpisových vzorů (Příloha 1 a 5). K implementaci jsme využili jazyk `C#` (Microsoft, 2016a) a XAML (Microsoft, 2010). Zvolili jsme univerzální windows aplikaci (Microsoft, 2016f) využívající .NET pro univerzální aplikace. Použili jsme návrhové vzory model-view-viewmodel (MVVM) (Smith, 2009), vzor pro příkazy (Microsoft, 2009) a úložiště (Microsoft, 2009). Zvolili jsme PRISM (Prism library) pro univerzální aplikace k realizaci MVVM. K záznamu podpisů jsme využili rychle se vyvíjející platformu Windows inking (Karl-Bridge-Microsoft, 2016), která poskytuje výkonnostně dobré řešení. V této kapitole vysvětlíme, proč jsme použili zmíněné technologie, návrhové vzory a frameworky. Dále uvedeme pojem lokalizace a na závěr se zmíníme o testování a dokumentaci aplikace.

3.1 Univerzální windows aplikace

S příchodem Windows 8 Microsoft vytvořil nový typ aplikací. Po update na Windows 8.1 a Windows Mobile 8.1 přišla snaha o tvorbu univerzálních aplikací pro tablety, mobily a stolní počítače. S příchodem Windows 10 tato platforma konverguje k univerzálním aplikacím (Microsoft, 2016f), které jsou vyvíjené v jednom prostředí a jednom frameworku. Tyto aplikace běží na mobilech, stolních počítačích, Xboxu a dalších zařízeních (Microsoft, 2016f). Je jasné, že různé typy zařízení mají různou funkcionalitu a z tohoto důvodu byl zaveden přístup api kontraktů. Aplikace se zeptá, jestli je daná funkcionalita k dispozici a pokud ano, může ji použít. Nyní porovnáme základní vlastnosti univerzálních aplikací s klasickými Win32 aplikacemi (Microsoft, 2004) v následující tabulce 3.1.

	Klasické Win32	**Univerzální aplikace**
.NET Framework	používá se nainstalovaný ve windows	používá .NET pro univerzální aplikace (podmnožina .NET frameworku)
Přístupy	WinRT, COM, Win32 APIs	WinRT, COM, podmnožina Win32 APIs
Sandbox	ne	ano
Nasazení	uživatel si musí stáhnout instalační balíček	distribuce pomocí .appx, primárně se aplikace stahují přes obchod
Odinstalace	aplikace většinou po sobě něco v systému zanechají	odinstalace je úplná a plně ji zajišťuje systém

Tabulka 3.1: Ukázka rozdílů mezi klasickými Win32 aplikacemi a univerzálními windows aplikacemi. Jedná se o orientační výčet, který není kompletní.

V našem řešení jsme zvolili pro univerzální aplikaci(Microsoft, 2016f), která se zaměřuje na tablety a stolní počítače. Pro rozšíření na další zařízení je nutné pouze optimalizovat uživatelské rozhraní. Hlavním důvodem pro univerzální aplikaci je snadná distribuce pomocí oficiálního obchodu. Díky tomu můžeme aplikaci nabídnout po celém světě.

3.2 Architektura

V našem řešení jsme využili návrhový vzor Model-View-Viewmodel (MVVM) (Smith, 2009). Jedná se o návrhový vzor vhodný pro aplikace s datovou vrstvou a netriviálním uživatelským rozhraním. Na následujícím obrázku je názorně vidět schéma 3.1.

Obrázek 3.1: Vrstevnatý model architektury MVVM.

Modely jsou datové objekty, které většinou reprezentují řešený problém. V našem případě je hlavním modelem podpisový vzor. Viewmodel je vrstva mezi pohledy a modely. Většinou načítá a ukládá data do úložiště, transformuje je podle potřeby a zveřejňuje důležité vlastnosti a příkazy pro pohledy. V prostřední vrstvě bývá logika aplikace. Poslední vrstva pohledů je uživatelské rozhraní. Pomocí binding se připojuje na vlastnosti a příkazy z viewmodel vrstvy.

Existuje mnoho frameworků pro usnadnění implementace MVVM. Například MVVMLight (MvvmLight), MvvmCross (MvvmCross), Prism (Prism library) a další. Mají mezi sebou mnoho drobných rozdílů, ale všechny implementují tento návrhový vzor. V našem řešení jsme využili PRISM pro univerzální windows aplikace (Prism library). Použili jsme Unity (Unity Container) jako kontejner pro závislosti a implementovali jsme vzor úložiště pro datovou vrstvu. Kontejner pro závislosti funguje jako černá skříňka, do které jednotlivé části aplikace zaregistrují zveřejněné rozhraní a implementace. Poté každá část aplikace používá uvedená rozhraní a při tvorbě objektů kontejner dosadí správné implementace podle požadovaných rozhraní. Výhoda spočívá v tom, že jednotlivé komponenty nejsou přímo pevně provázané, ale vždy požadují určitou funkcionalitu. Výměna různých komponent je díky tomuto vzoru jednoduchá. Datová vrstva využívá lokální úložiště, které má své definované rozhraní a je také registrované v Unity kontejneru (Unity Container). Díky tomu je možné snadno vyměnit naimplementovanou lokální variantu za jiné úložiště.

Hlavním důvodem, proč jsme zvolili MVVM, je separace logiky aplikace, datové vrstvy a uživatelského rozhraní. Dále tento vzor funguje velmi dobře s jazykem xaml a jeho funkcionalitou. Nevýhody, které tento přístup přináší v našem řešení, nevadí. Mezi ně patří větší spotřeba procesorového času a nutnost správně strukturovat aplikaci. I pro jednoduché aplikace je potřebné napsat značné množství kódu. Rozsáhlé aplikace mohou pocítit pomalejší běh z důvodu velkého množství bindingů (Binding).

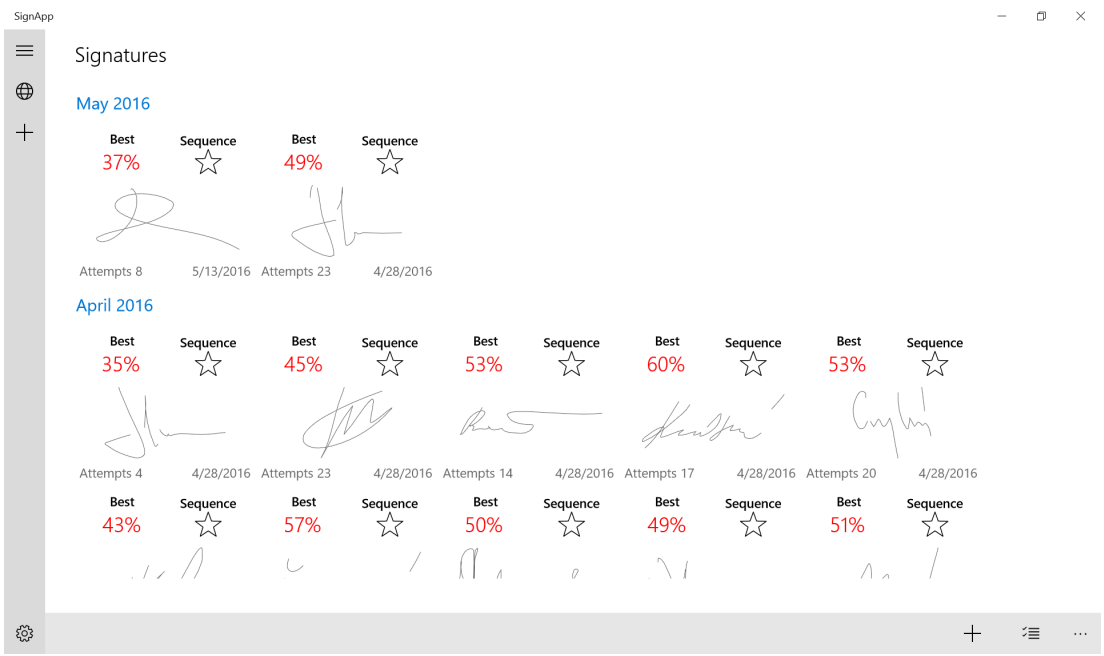
3.3 Implementační detaily jednotlivých algoritmů

Levenshteinovu vzdálenost 1 a Fréchetovu vzdálenost 2 jsme naimplementovali přímočaře podle uvedených algoritmů. Využili jsme dynamického programování s dvěma řádky matice v paměti. U vzdálenosti pomocí korelačního koeficientu jsme také postupovali podle zmíněného algoritmu 3. Naimplementovali jsme dvě

varianty. Dvourozměrnou, která všechny body promítá do dvourozměrné charakteristické mapy. Trojrozměrnou, která používá časovou osu jako třetí rozměr. U vzdálenosti pomocí korelačního koeficientu používáme Gaussovo rozostření. Protože Gaussovo rozostření je cyklické (Waltz and Miller, 1998), tak ho lze provádět po jednotlivých rozměrech, nejprve řádky, poté sloupce a nakonec podle časové osy. Namísto použití vícerozměrných polí používáme pole polí, protože vícerozměrné pole není správně optimalizované pro přístup pomocí indexů (Microsoft, 2009).

3.4 Uživatelské rozhraní

Návrh kvalitního uživatelského rozhraní je velmi složitá úloha. Většinou vyžaduje tým odborníků a mnoho iterací pokusů a omylů. V našem řešení jsme použili hamburgerové menu (Hamburger) pro navigaci. Tento navigační systém pravděpodobně není nejlepší pro naše řešení, ale uživatelé cílové platformy jsou na něj zvyklí a umí s ním pracovat. Na následujícím obrázku je vidět hlavní obrazovka. 3.2



Obrázek 3.2: Úvodní obrazovka aplikace, po levé straně je vidět menu, vlevo nahoře je tlačítko hamburgeru. Dole je vidět příkazový panel. Zbytek prostoru je vyplněn kolekcí již nasbíraných podpisů a informací o nich.

V levé části je vidět menu. Zde je možné přidat nový podpis, jít do nastavení nebo na hlavní obrazovku. Hamburger vlevo nahoře potom toto menu otevře a zobrazí popisky jednotlivým symbolům. Ve spodní části je vidět příkazový panel. Pomocí tohoto panelu lze vykonávat různé funkce v závislosti na daném kontextu. Uživatelé univerzální platformy windows jsou na tento design zvyklí.

Uživatel provádí s aplikací primárně dvě činnosti. Za prvé vytváří podpisový vzor 3.3. Za druhé se snaží tento podpisový vzor napodobit. Pro vytvoření podpisového vzoru je na hlavní stránce a v menu tlačítko +. Pro zkoušení podpisového

vzoru je potřeba vybrat vzor na hlavní stránce a klepnout na něj. Poté se otevře strana pro testování.

Obrázek 3.3: Stránka pro tvorbu nového podpisového vzoru. Uživatel se podepíše do šedého rámečku a pokud se mu podpis líbí, zmáčkne na disketu. V opačném případě může celý proces restartovat pomocí tlačítka znovu.

3.5 Windows inking

Windows Ink platforma (Karl-Bridge-Microsoft, 2016) spolu s perem poskytuje přirozenou cestu k vytváření ručně psaných poznámek a kreslení. Tato platforma podporuje záznam, generování, manipulaci, vykreslování a rozpoznávání psaného textu. Při použití speciálního pera (Surface Pen) určeného pro tuto platformu je možné zaznamenávat také přítlak. Tento parametr v našem řešení nepoužíváme, protože při použití stylusu nebo prstu není k dispozici. Hlavním cílem této platformy je poskytnout vývojářům snadně integrovatelné řešení pro podporu pera. Stačí přidat InkCanvas a kreslení funguje.

Při záznamu dat z pera obvykle vzniká prodleva mezi fyzickým kreslením a vykreslením na obrazovce Ted Miller (2016). Tuto prodlevu můžeme rozdělit na hardwarovou a softwarovou. Hardwarová prodleva vzniká při záznamu dat z pera a přenosu do počítače, s touto prodlevou nemůžeme nic dělat. Softwarová prodleva vzniká po přenosu dat z hardwaru a před vykreslením kreslené čáry na obrazovku. Windows ink velmi dobře minimalizuje softwarovou prodlevu Ted Miller (2016). Proces záznamu běží v samostatném vlákně a je nezávislý na zbytku aplikace.

Tato platforma je značně otevřená a některé základní části se dají nahradit. V našem řešení používáme windows inking pro záznam, vykreslování, ukládání a načítání podpisů. Používáme InkCanvas, InkPresenter a InkStrokeContainer. InkCanvas slouží jako zastřešovací třída, pomocí které lze vykonávat všechny podporované scénáře. InkPresenter slouží pro vykreslování uložených čar. InkStroke-

Container je kolekce nakreslených čar. Pomocí této třídy načítáme a ukládáme data do obrázku s koncovkou .gif.

Při implementaci jsme narazili na trochu neobvyklý problém. Potřebovali jsme odchytávat události o pohybu pera, abychom mohli využít naše metriky. Nicméně klasické přihlášení k odběru údálostí o PointerMoved u InkCanvas se nikdy nevykonalo. Pro navazující vlastní zpracování těchto dat je nutné vytvořit nezávislý zdroj a přes ten se poté přihlásit k PointerMoved. U této události se nedoporučuje vykonávat žádné déle trvající operace, protože to negativně ovlivňuje softwarovou odezvu vykreslování.

3.6 Lokalizace

Každá vícejazyčná aplikace musí řešit lokalizaci. My jsme lokalizovali aplikaci do anglického a českého jazyka. Lokalizaci jsme udělali pomocí zdrojových souborů .resw (Microsoft, 2016e). Každý lokalizovaný prvek v aplikaci má vlastní identifikátor, pomocí kterého dokážeme dohledat správný zdroj pro daný jazyk. Prvkům v jazyce xaml jsme přiřadili unikátní identifikátor pomocí x:Uid. Ostatním prvkům jsme přiřadili unikátní řetězec. Při spuštění aplikace se vytvoří instance načítače zdrojů, který následně využíváme k načítání lokalizovaných prvků. Zdroje aplikace jsme nejprve vytvořili v anglickém jazyce a následně pomocí Multilingual application toolkit (Microsoft, 2016c) lokalizovali do českého jazyka.

3.7 Testování

Testování aplikace patří k dobrým praktikám. V našem řešení jsme k testování využili testovací projekt a PEX framework (Microsoft, 2016b), který je součástí visual studia enterprise. Pomocí tohoto frameworku se snadněji generují generické testy pokrývající veliký rozsah kódu a vstupních parametrů. Takto jsme otestovali všechny součásti hlavního projektu pro výpočty jednotlivých metrik. Protože spouštění s dlouhou dobou běhu se neprovádí dobře ve vývojovém prostředí, pro testování výsledků metrik jsme využili benchmark.

3.8 Dokumentace

Součáští aplikace je dokumentace (Příloha 2). Ta se skládá z uživatelské dokumentace, programátorské dokumentace a technické dokumentace. Uživatelská dokumentace obsahuje návod pro uživatele a popisuje jednotlivé funkce aplikace. Programátorská dokumentace obsahuje popis organizace řešení, architektury a dalších důležitých částí. Technickou dokumentaci generujeme z kódu a dokumentačních komentářů (SandCastle).

3.9 Shrnutí

Navrhli a naimplementovali jsme univerzální windows aplikaci pro záznam a vyhodnocení podobnosti podpisových vzorů. Aplikaci jsme vytvořili pomocí jazyka

C#. Využili jsme návrhového vzoru MVVM, z důvodu dobré separace uživatelského rozhraní, logické a datové vrstvy. K implementaci tohoto vzoru jsme využili PRISM pro univerzální aplikace. V aplikaci používáme Unity kontejner k řešení závislostí mezi komponentami. Dále využíváme vzor úložiště a také vzor příkazů. Pro záznam podpisů jsme zvolili Windows inking. Aplikaci jsme lokalizovali do českého a anglického jazyka. Knihovnu pro výpočet jednotlivých metrik jsme otestovali pomocí Intellitest ve Visual Studiu. Poté jsme vytvořili uživatelskou a programátorskou dokumentaci spolu s vygenerovanou technickou dokumentací.

4. Benchmark

Vytvořili jsme aplikaci pro záznam podpisových vzorů a jejich porovnávání. Naimplementovali jsme tři zmíněné metriky (Kap. 2) a pokusili se zjistit, jak jsou tyto metriky citlivé a jakým způsobem dokážou rozeznávat různé podpisy. Pomocí této znalosti jsme zkombinovali výsledky v jedno číslo reprezentující procentuální podobnost dvou podpisů. Použitím vzorce získáme procentuální vyjádření vzdálenosti. Takto reprezentovaný výsledek je dobře srozumitelný koncovým uživatelům.

4.1 Postup

Zvolili jsme transformace posunutí x, posunutí y, rotaci a zkosení, abychom zjistili, jak se metriky chovají. Při testování jednotlivých transformací jsme testovanou transformaci vždy aplikovali pro každý bod transformovaného podpisu. Posunutí x posune body o t pixelů po ose X. Posunutí y je stejné jako posunutí po x, jen po ose Y. Rotace rotuje každý bod kolem středu podpisu o t stupňů. Transformace zkosení zkosí osy X a Y každou o t stupňů a aplikuje toto zkosení pro každý bod.

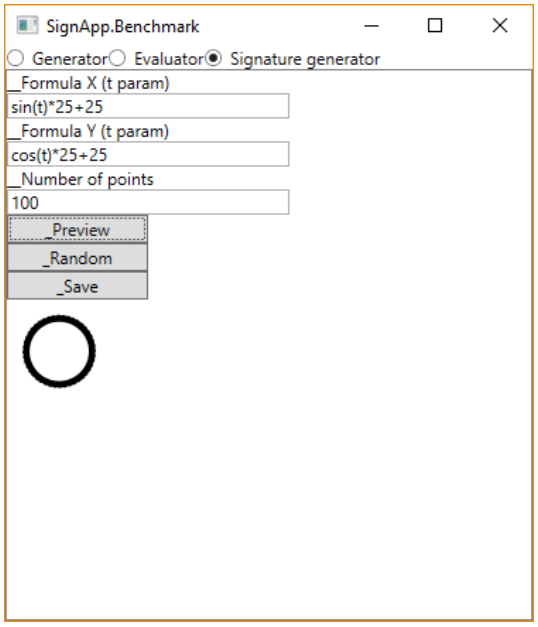
Vygenerovali jsme padesát náhodných podpisů. Každý náhodně vygenerovaný podpis obsahuje 100 bodů zaznamenaných po sedmi milisekundách. Pevně jsme zvolili první bod se souřadnicemi $[100, 100]$. Dále jsme náhodně vygenerovali vektor směru, podle kterého bude podpis pokračovat. Poté jsme v cyklu vygenerovali všech 100 bodů a při každém průchodu jsme změnili vektor směru nejvýše o 10% v ose X a o 10% v ose Y.

Vytvořili jsme testy pro jednotlivé transformace. Posunutí x od 0 do 40 pixelů s 20 kroky. Posunutí y stejně jako posunutí x. Rotaci kolem středu podpisu od 0 do 40 stupňů s 20 kroky. Zkosení od 0 do 45 stupňů s 20 kroky. Při každém kroku se parametr t lineárně posouvá tak, aby při prvním kroku byl na začátku rozmezí a při posledním na konci rozmezí. Tedy pro každou transformaci jsme definovali 20×50 testů. Pro každý náhodně vygenerovaný podpis jsme generovali 20 testů s parametrem t určující míru transformace. Dále jsme testy spustili pomocí benchmark aplikace. Každý test nejprve načte vzor, ten znormalizuje podle 2.2, aplikuje definovanou transformaci a spustí vyhodnocení metrik na vzoru a jeho transformovaném obraze. Takto získané výsledky jsme zobrazili do grafů 4.2 a pomocí těchto informací jsme zvolili vzorec pro výpočet procentuální podobnosti.

4.2 Aplikace Benchmark

Potřebovali jsme využít naimplementované metriky, které jsou v přenosné knihovně tříd (Microsoft, 2016d). Proto jsme vytvořili novou aplikaci (Příloha 1). Jedná se o klasickou Win32 (Microsoft, 2004) aplikaci napsanou v jazyce `C#`, která využívá knihovnu s metrikami. Klíčové funkce této aplikace jsou generování a ukládání definovaných podpisů podle vzorce nebo náhodných podpisů, tvorbu testovacích souborů a jejich vyhodnocení. Pro vygenerování grafů z vyhodnocených dat využíváme skript v `F#` (Delimarsky, 2016) a grafy generujeme pomocí R (R Core Team, 2016) a balíčku ggplot2. Cílem této aplikace je usnadnit zkoumání chování

metrik a vytvořit procentuální podobnost z těchto metrik. Součástí je uživatelská i programátorská dokumentace (Příloha 2).



Obrázek 4.1: Ukázka benchmark aplikace. V horní části je vidět překlikávání mezi generováním testovacích souborů a vyhodnocovací částí a na obrázku je vidět část pro generování podpisů s vygenerovaným kruhem.

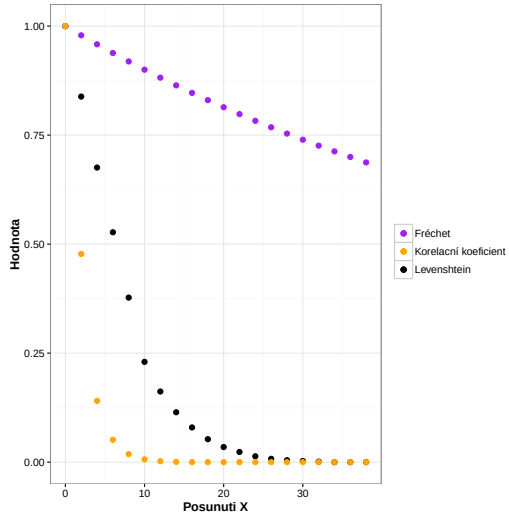
4.3 Vyhodnocení dat

Nejprve ukážeme výsledky 4.2 získané pro jednotlivé transformace (data v Příloze 3).

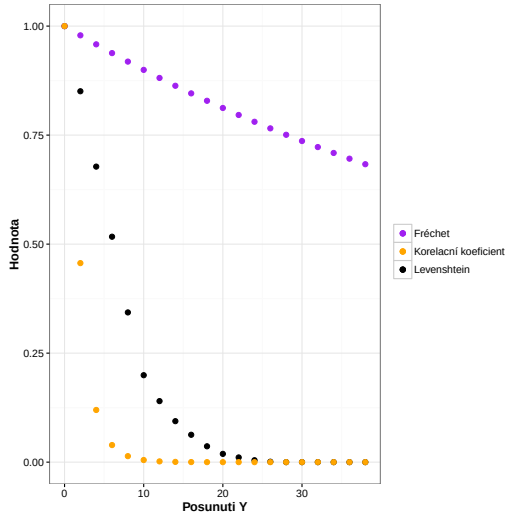
Z grafů je vidět, jak se různé metriky chovají v závislosti na různých transformacích. Fréchetova vzdálenost je velmi odolná na všechny zvolené operace. Ukazuje se, že FD dává příliš dobré výsledky i pro značně transformované podpisy. Levenshteinova vzdálenost dává dobré výsledky pro stejné podpisy a již při menších transformacích nepovažuje podpisy za podobné. Vzdálenost pomocí korelačního koefiecientu také pozná stejné podpisy, ale i málo transformované podpisy považuje za zcela nepodobné. Pomocí tohoto měření jsme získali základní informace o chování metrik pro zvolené transformace.

4.4 Porovnání reálného podpisu s primitivními tvary

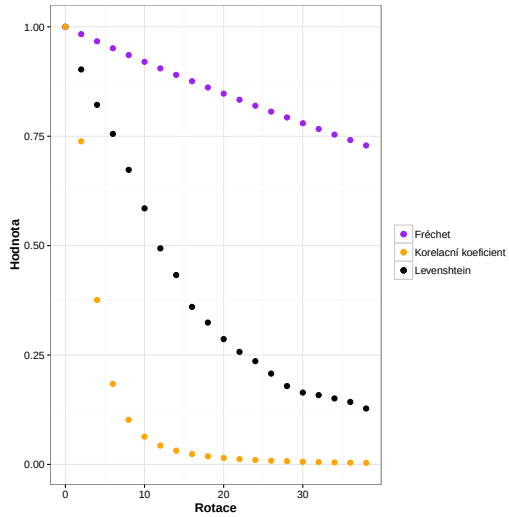
Předchozí měření nám neposkytlo dostatečné informace o chování metrik při různých podpisech. Transformovali jsme podpis jako celek a nevyzkoušeli jsme, jak metriky reagují na jiné operace nebo jiné podpisy. Nyní potřebujeme zjistit, jestli metriky dokáží rozeznat podpisy na první pohled rozdílné.



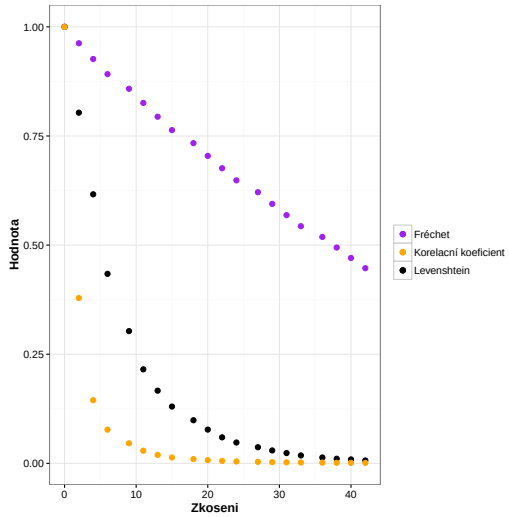
(a) Transformace podpisu podle X v pixelech.



(b) Transformace podpisu podle Y v pixelech.



(c) Transformace podpisu pomocí rotace ve stupních.

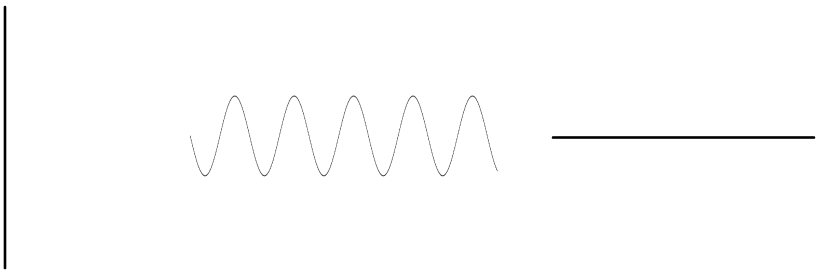


(d) Transformace podpisu pomocí zkosení ve stupních.

Obrázek 4.2: Grafy zobrazující zprůměrované výsledky pro 50 náhodně vygenerovaných podpisů.

4.4.1 Postup

Zvolili jsme pět zaznamenaných podpisových vzorů (Příloha 3). Tyto podpisy porovnáme se zvolenými primitivními podpisy 4.3 (Příloha 3). S vodorovnou čárou zaznamenanou po směru 4.3c a také proti směru 4.3c. Dále se svislou čárou 4.3a a nakonec s vlnovkou 4.3b napodobující zvolený podpis. Vygenerujeme testovací soubor pro aplikaci benchmark, ve kterém bude test pro všechny kombinace zvolených podpisů a primitivních podpisů. Celkem tedy 5×4 testů. Při každém testu se nejprve oba podpisy znormují, a poté se provede vyhodnocení metrik. Takto získané výsledky 4.1 vyhodnotíme.



(a) Svislá čára. (b) Vlnovka. (c) Vodorovná čára.

Obrázek 4.3: Obrázky zobrazující zvolené primitivní podpisy.

4.4.2 Vyhodnocení

V tabulce 4.1 zobrazujeme zprůměrované hodnoty pro všechny zvolené primitivní podpisy a jednotlivé metriky.

	Levenshtein	**Fréchet**	**Korelační koeficient**
Vod. čára po směru	0,19	0,75	0,01
Vod. čára proti směru	0,05	0,17	0,00
Svislá čára	0,04	0,51	0,00
Vlnovka po směru	0,14	0,78	0,01

Tabulka 4.1: Porovnání klasických podpisů s primitivními tvary (zprůměrované hodnoty).

Vzdálenost pomocí korelačního koeficientu správně rozeznává nepodobné podpisy, průměrná podobnost všech testů je kolem 0. Levenshteinova vzdálenost nabývá maximálních hodnot okolo 0,19 při vodorovné čáře po směru. Tyto hodnoty jsou velmi malé pro nepodobné podpisy, LV tedy vrací nízké % podobnosti pro nepodobné podpisy. Fréchetova vzdálenost při porovnání podpisu s horizontální čárou nabývá hodnoty okolo 0.75 a při porovnání s vlnovkou po směru podpisu dokonce okolo 0,78. Tato metrika nedokáže správně rozpoznávat nepodobné podpisy. Nyní máme lepší představu o funkci těchto metrik a můžeme lépe stanovit vzorec pro celkovou procentuální podobnost.

4.4.3 Vzorec pro výpočet celkové podobnosti

Pokud Fréchetova vzdálenost nabývá hodnot menších než 0,5, podpisy si nejsou podobné, při větší podobnosti to znamená, že mají stejný směr a pokud tato metrika nabývá hodnot větších než 0.85, jsou si podpisy opravdu podobné. Levenshteinova vzdálenost nabývá nízkých hodnot při nepodobných podpisech a je velmi citlivá na různé transformace. Vzdálenost pomocí korelačního koeficientu je nejcitlivější metrika, a pokud nabývá vyšších hodnot, podpisy jsou si velmi podobné. Pro celkovou procentuální podobnost jsme zvolili arbitrární vzorec

$$k(y) = -\frac{5}{3}x^2 + 1.5x + \frac{2}{3} \quad \text{kde } x = Min(y, 0.4)$$

$$l(x) = Min(x, 0.6) + 0.4$$

$$d = \begin{cases} \frac{fd+k(kk)l(ld)}{3} & \text{kde } fd \in [0.85, 1], ld \in [0.3, 1], kk \in [0.1, 1] \\ \frac{fd+k(kk)}{2} - 0.1 & \text{kde } fd \in [0.85, 1], kk \in [0.1, 1] \\ \frac{fd+l(ld)}{2} - 0.1 & \text{kde } fd \in [0.85, 1], ld \in [0.3, 1] \\ \frac{k(kk)+l(ld)}{2} - 0.1 & \text{kde } ld \in [0.3, 1], kk \in [0.1, 1] \\ \frac{2*fd+3*ld}{5} & \text{jinak} \end{cases}$$

fd, ld, kk jsou výsledky jednotlivých metrik

. Pokud všechny metriky překročí stanovenou hranici, FD 0.85, LV 0.3 a vzdálenost pomocí korelačního koeficientu 0.1, vzorec zkombinuje všechny metriky a vrátí vysokou podobnost. Pokud stanovenou hranici překročí pouze dvě ze tří, aplikuje se na jejich kombinaci penalizace 10%. V ostatních případech vzorec kombinuje pouze FD a LV v poměru 2 : 3. Určitě existuje optimální vzorec pro kombinaci zmíněných metrik, který dává lepší výsledky. Naším cílem bylo tuto podobnost nějakým způsobem kvantifikovat.

4.5 Shrnutí

Vytvořili jsme benchmark aplikaci pro vyzkoušení chování metrik při jednotlivých transformacích. Otestovali jsme metriky proti posunutí po ose x a y, rotaci a zkosení. Dále jsme vyzkoušeli chování metrik při porovnání podpisů s primitivními tvary. Zjistili jsme, že Fréchetova vzdálenost dává velmi vysoké hodnoty i pro nepodobné podpisy. Levenshteinova vzdálenost dává vysoké hodnoty pro podobné a nízké pro nepodobné a rychle klesá při použití různých transformací. Vzdálenost pomocí korelačního koeficientu je nejcitlivější metrika a dává vysoká čísla pouze pro téměř stejné podpisy a pro ostatní kolem nuly. Pomocí těchto zjištění jsme zvolili arbitrární vzorec pro výpočet celkové procentuální podobnosti.

5. Experiment

V předchozí kapitole jsme zvolili arbitrární vzorec pro výpočet podobnosti dvou podpisových vzorů. Máme připravenou aplikaci pro reálné použití. Experimentálně zjistíme, jestli se účastníci dokáží zlepšit při napodobování vlastního podpisu nebo při napodobení cizího podpisového vzoru (data v příloze 4).

5.1 Metoda

5.1.1 Účastníci

Třicet účastníků experimentu. Deset účastníků při trénování vlastního podpisového vzoru. Dvacet účastníků při pokusech o napodobení cizího podpisového vzoru. Účastníci jsou z našeho okolí a nezaznamenávali jsme jejich pohlaví, věk ani jiné informace. Všichni se účastnili experimentu dobrovolně a žádný z nich se předtím nezúčastnil podobného experimentu.

5.1.2 Zařízení

Pro záznam a cvičení podpisových vzorů jsme použili tablet Lenovo Miix 2 10 a stylus CONNECT IT CI-183. Pro odstínění ruky od dotykového displeje jsme využili čistící hadřík na brýle.

5.1.3 Procedura

Experiment jsme rozdělili na dvě části. První část trénování vlastního podpisu a druhou část pokus o napodobení cizího podpisového vzoru.

Při trénování vlastního vzoru každý účastník nejprve zaznamenal svůj podpisový vzor. Při zaznamenání podpisového vzoru otevřel příslušnou část aplikace a podepsal se. Pokud s ním byl spokojený, tento vzor uložil. Pokud ne, záznam opakoval. Poté přešel k trénování vytvořeného vzoru. Desetkrát za sebou se podepsal.

Pro druhou část jsme vybrali 4 podpisové vzory (Příloha 4) a nechali účastníky tyto vzory napodobovat. Každý vzor jsme nechali napodobit pětkrát. Každému účastníkovi jsme vybrali podpisový vzor a ten ho nechali zkoušet. Účastník nejprve přešel ke zkoušení zvoleného podpisového vzoru v aplikaci. Tento vzor zobrazil a přehrál animaci ukazující tempo podpisu. Poté skryl podpisový vzor a desetkrát se ho pokusil napodobit.

5.2 Výsledky

5.2.1 Zpracování dat

V zaznamenaných datech byly chyby způsobené ztrátou kontaktu mezi stylusem a dotykovým displejem. Občas se zaznamenala ruka místo stylusu. Chybné záznamy jsme vyloučili. Poté jsme pro každou část vytvořili testovací soubory z nasbíraných

dat a spustili vyhodnocení celkové podobnosti podle zvoleného vzorce pomocí benchmark aplikace. Vyhodnocená data jsme zobrazili v grafech 5.1.

5.2.2 Napodobení vlastního vzoru

Vyhodnocená data pro první část jsme zobrazili v grafu 5.1a. Při napodobení vlastního podpisového vzoru jsme dostali hodnoty převážně ve dvou skupinách. Mezi 30% a 60% a druhou skupinu mezi 75% a 90%. Regresní křivka ukazuje na negativní korelaci mezi hodnotou a číslem pokusu.

5.2.3 Napodobení cizího vzoru

Vyhodnocená data pro druhou část jsme zobrazili v grafu 5.1b. Při napodobení cizího podpisového vzoru nabývá hodnota vypočtená pomocí zvoleného vzorce okolo 40%.

5.3 Diskuze

Vytvořili jsme experiment, který zjišťoval, jestli se účastníci dokáží zlepšit při napodobování vlastního nebo cizího podpisu s využitím zvoleného vzorce pro výpočet podobnosti dvou podpisů. Ze získaných dat nevyplývá, že se účastníci zlepšili při tvorbe vlastního nebo napodobení cizího podpisu. Při napodobování vlastního vzoru dosahovali účastníci mnohem vyšší průměrné úspěšnosti než u napodobování cizích vzorů. Regresní křivka u vlastních podpisů ukazuje na negativní korelaci mezi hodnotou a číslem pokusu. Tento jev může být způsobený nevhodně zvoleným vzorcem pro výpočet podobností, rostoucí únavou účastníků s přibývajícím číslem pokusu nebo jiným neznámým jevem. Nesetkali jsme se s jinými pracemi zkoumající schopnost tvorby podpisového vzoru.

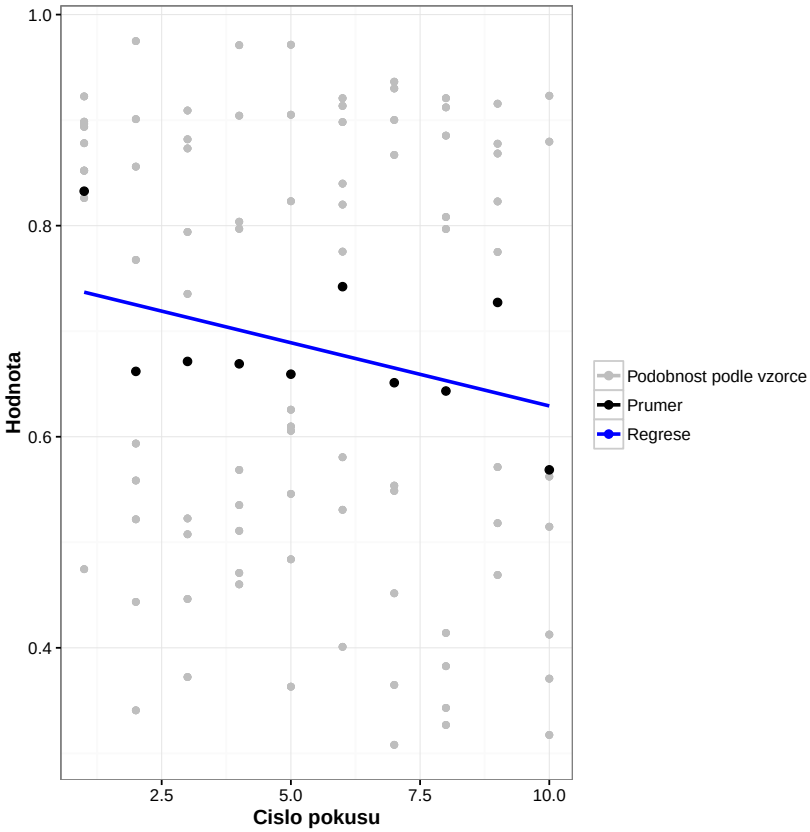
5.3.1 Aplikace

Při experimentu jsme využili naimplementovanou aplikaci, která porovnává podpisové vzory pomocí zvoleného arbitrárního vzorce. Ukázalo se, že tato aplikace je velmi dobře použitelná pro trénování podpisových vzorů. Při nahrazení využitých metod pro porovnávání podobnosti za nejmodernější metody v této oblasti, poskytne aplikace velmi dobré řešení pro tréning podpisových vzorů.

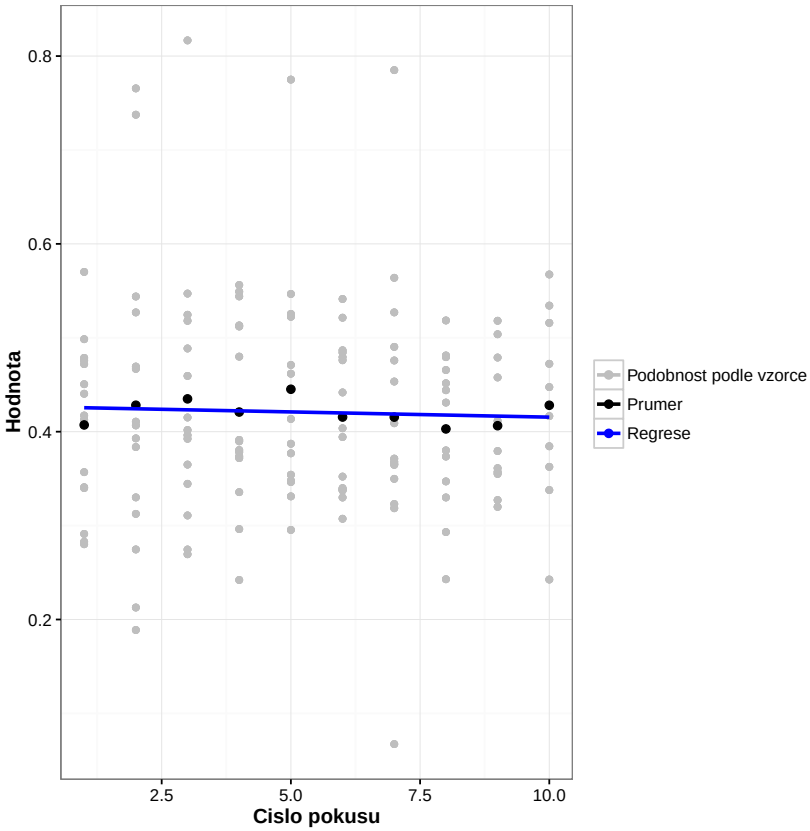
5.4 Shrnutí

Provedli jsme experiment s cílem zjistit, jestli se účastníci dokáží zlepšit při napodobení vlastního vzoru nebo při napodobení cizího vzoru. Celkem se experimentu účastnilo 30 lidí, 10 lidí napodobovalo vlastní podpisový vzor a 20 lidí napodobovalo cizí podpisový vzor. Podpisy jsme zaznamenávali pomocí tabletu se stylusem. Odstranili jsme špatně zaznamenaná data a vyhodnotili jsme podobnosti pomocí benchmark aplikace. Poté jsme zobrazili výsledky do grafů a diskutovali výsledky. Z naměřených dat jsme nezjistili, že by se účastníci zlepšovali. Při napodobování

vlastního podpisu účastníci dosáhli mnohem lepších výsledků než při napodobování cizích podpisů. Naměřená data u napodobování vlastního podpisu ukázují na negativní korelaci mezi podobností a číslem pokusu. Tento jev je pravděpodobně způsobený nevhodnou volbou vzorce pro výpočet podobnosti. Využitá aplikace poskytuje velmi dobré řešení pro tréning podpisových vzorů.



(a) Podpisy podle vlastního podpisového vzoru.



(b) Pokus o netrénovaný cizí podpisový vzor.

Obrázek 5.1: Grafy zobrazují výsledky vyhodnocených dat.

6. Možná rozšíření

Při tvorbě této práce jsme narazili na řadu zajímavých možností rozšíření, které jsou nad rámec jejího obsahu. Nejprve uvedeme možnosti přímého rozšíření této práce.

1. Zvolený vzorec pro výpočet celkové podobnosti není optimální. Bylo by dobré provést mnohem robustnější benchmarking a vyhledat optimální kombinaci zvolených metrik pro výpočet celkové procentuální podobnosti.

2. Nahradit použité řešení pro výpočet podobnosti za nejmodernější systém v oblasti ověřování podpisových vzorů za účelem maximálního využití potenciálů naimplementované aplikace.

3. Rozšíření o backend aplikace. Zavedení přihlašování uživatelů. Aplikace je připravená pro využití Azure Mobile App service (Microsoft, 2015). Ukládání, stahování a sdílení podpisů do cloudu. Přidání sdílení úspěchů na sociální sítě. Zavedení odměn za dobře natrénované podpisové vzory.

4. Inteligentní filtrování špatně zaznamenaných částí při záznamu podpisu pomocí stylusu. Vyloučení občasné zaznamenání položené ruky.

5. Lokalizace aplikace do dalších jazyků. Globalizování aplikace a testování pomocí speciálních řetězců, které obsahují maximální délky pro daná pole.

6. Rozšíření cílové platformy z tabletů a stolních počítačů s perem na další zařízení (Mobily, možná Xbox a Hololense).

7. Experimentálně ověřit na větším vzorku lidí a po delší dobu, jestli aplikace pomáhá stabilizovat podpisový vzor.

Dále uvedeme zajímavé projekty, které souvisí s touto prací.

1. Lockscreen aplikace pro mobily, kde si uživatel nejprve definuje vzor pomocí stylusu nebo prstu. Poté si zvolí úroveň bezpečnosti, neboli jak bude nastavená míra pro přijetí podpisu. Poté bude místo zadávání PINu, vykonání gesta nebo jiné metody využívat biometrické ověření podpisového vzoru k přístupu do zařízení.

2. Využití použitých metrik pro porovnávání dat získaných z akcelerometru, gyroskopu nebo jiných senzorů při sportovních činnostech. Při sportech se většinou vykonávají opakované pohyby. Nejprve tedy vytvoříme databáze vzorů (například různé údery při tenisové hře). Poté uživatelé provádí záznam a jejich sportovní aktivita se následně vyhodnotí pomocí metrik a databáze. Výsledkem bude co, kdy a jak dobře daný uživatel dělal.

Závěr

Představili jsme problematiku ověřování podpisových vzorů. Zařadili jsme naše řešení mezi dynamické algoritmy a neřešili jsme poslední krok verifikace, rozhodnutí, jestli je podpis pravý nebo ne. Ukázali jsme, že problematika porovnávání podpisových vzorů je podobná porovnávání očních pohybů a vybrali jsme tři metriky, které jsme následně naimplementovali v našem řešení. Levenshteinovu vzdálenost, která slouží k porovnávání dvou textových řetězců. Fréchetovu vzdálenost, která reprezentuje délku nejkratšího vodítka mezi pánem a psem potřebnou k překonání cesty po dvou křivkách a vzdálenost pomocí korelačního koeficientu, který počítá korelaci mezi dvěma charakteristickými mapami.

Navrhli a naimplementovali jsme univerzální windows aplikaci pro procvičování podpisových vzorů pomocí zmíněných metrik. Použili jsme návrhový vzor MVVM a PRISM pro univerzální aplikace. Využili jsme Windows inking. Aplikaci jsme lokalizovali do českého a anglického jazyka. Připravili jsme aplikaci pro publikování do obchodu jako .appx balíček. Provedli jsme benchmark zvolených metrik na náhodných podpisech pomocí vybraných operací a vytvořili procentuální podobnost pro dva podpisy.

Provedli jsme experiment s cílem zjistit, jestli se účastníci dokáží zlepšit při tvorbě vlastního nebo cizího podpisu při porovnávání podobnosti s využitím zvoleného vzorce. Z naměřených dat se nám toto nepodařilo prokázat. Vyzkoušeli jsme, že se naimplementovaná aplikace dá využít pro procvičování podpisového vzoru.

Doufáme, že tato práce mnohým pomůže při nácviku podpisu a inspiruje k využití metrik pro porovnávání očních pohybů při tvorbě systému pro ověřování podpisových vzorů.

Seznam použité literatury

Reena Bajaj and Santanu Chaudhury. Signature verification using multiple neural classifiers. *Pattern Recognition*, 30(1):1–7, 1997. ISSN 00313203. doi: 10.1016/S0031-3203(96)00059-3. URL http://www.sciencedirect.com/science/article/pii/S0031320396000593.

Binding. Wpf binding, May 2016. URL https://msdn.microsoft.com/en-us/library/aa480224.aspx.

William C. Burton. *Burton's Legal Thesaurus*. 2007. ISBN 0071472622.

Patrick Cavanagh and George A. Alvarez. Tracking multiple targets with multifocal attention, jul 2005. ISSN 13646613. URL http://www.ncbi.nlm.nih.gov/pubmed/15953754http://www.sciencedirect.com/science/article/pii/S136466130500149X.

Rodrigo De Luis-García, Carlos Alberola-López, Otman Aghzout, and Juan Ruiz-Alzola. Biometric identification systems, 2003. ISSN 01651684.

Den Delimarsky. Fsharp language reference, May 2016. URL https://msdn.microsoft.com/visualfsharpdocs/conceptual/fsharp-language-reference.

Dynamický podpis. Dynamický podpis, 2001-. URL https://upload.wikimedia.org/wikipedia/commons/4/44/Online_signture.jpg.

Dynamický podpis parametry. Dynamický podpis parametry, 2001-. URL https://upload.wikimedia.org/wikipedia/commons/3/35/Dynamic_information_of_a_signature.jpg.

S.R. Eddy. What is dynamic programming? *Nature biotechnology*, 22(7): 909–910, 2004. ISSN 1087-0156. doi: 10.1038/nbt0704-909. URL http://scholar.google.com/scholar?hl=en{&}btnG=Search{&}q=intitle:What+is+dynamic+programming?{#}0.

Thomas Eiter and Heikki Mannila. Computing Discrete Frechet Distance. *See Also*, 1994. URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.90.937{&}rep=rep1{&}type=pdf.

Marcos Faundez-Zanuy. On-line signature recognition based on VQ-DTW. *Pattern Recognition*, 40(3):981–992, 2007. ISSN 00313203. doi: 10.1016/j.patcog.2006.06.007.

Julian Fierrez, Javier Ortega-Garcia, Daniel Ramos, and Joaquin Gonzalez-Rodriguez. HMM-based on-line signature verification: Feature extraction and signature modeling. *Pattern Recognition Letters*, 28(16):2325–2334, 2007. ISSN 01678655. doi: 10.1016/j.patrec.2007.07.012.

Hamburger. Hamburger menu, 2001-. URL https://en.wikipedia.org/wiki/Hamburger_button.

Kenneth Holmqvist, Marcus Nyström, Richard Andersson, Richard Dewhurst, Halszka Jarodzka, and Joost Van de Weijer. *Eye tracking: A comprehensive guide to methods and measures*. OUP Oxford, 2011.

Edson J R Justino, Flávio Bortolozzi, and Robert Sabourin. Off-line signature verification using HMM for Random, simple and skilled forgeries. In *Proceedings of the International Conference on Document Analysis and Recognition, ICDAR*, volume 2001-Janua, pages 1031–1034, 2001. ISBN 0769512631. doi: 10.1109/ICDAR.2001.953942.

Karl-Bridge-Microsoft. `https://msdn.microsoft.com/en-us/windows/uwp/input-and-devices/pen-and-stylus-interactions`, 05 2016.

Olivier Le Meur and Zhi Liu. Saccadic model of eye movements for free-viewing condition. *Vision Research*, 116:152–164, 2015. ISSN 18785646. doi: 10.1016/j.visres.2014.12.026.

V I Levenshtein. Binary Codes Capable of Correcting Deletions, Insertions, and Reversals. *Soviet Physics Doklady*, 10:707–710, 1966a. ISSN 00385689. URL `http://adsabs.harvard.edu/abs/1966SPhD...10..707L`.

V I Levenshtein. Levenshtein. *Soviet Physics Doklady*, 10(8):707–710, 1966b. ISSN 00385689.

Applied Mathematics and Recreational Mathematics. Gaussian Function. *History*, (6):8–10, 2008.

Olivier Meur and Thierry Baccino. Methods for comparing scanpaths and saliency maps: strengths and weaknesses, 2012. ISSN 1554-3528.

Microsoft. General overview of win32s, 04 2004. URL `http://support.microsoft.com/kb/83520`.

Microsoft. MSDN Library, 2009. URL `https://msdn.microsoft.com/en-us/library/ee658117.aspx`.

Microsoft. Extensible application markup language. `http://www.microsoft.com/downloads/details.aspx?FamilyID=52a193d1-d14f-4335-aa86-c53193e1885d&displayLang=en`, 06 2010.

Microsoft. Microsoft Azure. *The Cloud for Modern Business*, page 1, 2015. doi: 10.1007/978-1-4842-1043-7. URL `http://www.windowsazure.com/en-us/`.

Microsoft. `https://msdn.microsoft.com/en-us/library/618ayhy6.aspx`, May 2016a.

Microsoft. Intellitest, May 2016b. URL `https://msdn.microsoft.com/en-us/library/dn823749.aspx`.

Microsoft. Multilingual app toolkit, May 2016c. URL `https://developer.microsoft.com/en-us/windows/develop/multilingual-app-toolkit`.

Microsoft. Cross-platform development with the portable class library, May 2016d. URL https://msdn.microsoft.com/en-us/library/gg597391(v=vs.110).aspx.

Microsoft. Creating and retrieving resources in windows runtime apps, May 2016e. URL https://msdn.microsoft.com/en-us/library/hh694557.aspx.

Microsoft. Uwp app, 05 2016f. URL https://msdn.microsoft.com/en-us/windows/uwp/get-started/whats-a-uwp.

MvvmCross. Mvvmcross, May 2016. URL https://github.com/MvvmCross/MvvmCross.

MvvmLight. Mvvm light toolkit, May 2016. URL http://www.mvvmlight.net/.

A. Piyush Shanker and A. N. Rajagopalan. Off-line signature verification using DTW. *Pattern Recognition Letters*, 28(12):1407–1414, 2007. ISSN 01678655. doi: 10.1016/j.patrec.2007.02.016.

Prism library. Prism library. https://github.com/PrismLibrary/Prism, 05 2016.

Yingyong Qi and Bobby R. Hunt. Signature verification using global and grid features. *Pattern Recognition*, 27(12):1621–1629, 1994. ISSN 00313203. doi: 10.1016/0031-3203(94)90081-7.

R Core Team. R: A language and environment for statistical computing. *R Foundation for Statistical Computing, Vienna, Austria.*, pages –, 2016. ISSN 16000706. doi: ISBN3-900051-07-0,. URL http://www.r-project.org/.

K Rayner. Eye movements in reading and information processing: 20 years of research. *Psychological Bulletin*, 124(3):372–422, 1998. ISSN 0033-2909. doi: 10.1037/0033-2909.124.3.372.

SandCastle. Sandcastle, May 2016. URL https://github.com/EWSoftware/SHFB.

Josh Smith. Patterns - wpf apps with the model-view-viewmodel design pattern. *MSDN Magazine*, 02 2009.

Smooth pursuit. Smooth pursuit. https://en.wikipedia.org/wiki/Smooth_pursuit, May 2016.

Surface Pen. Surface pen, May 2016. URL http://www.microsoftstore.com/store/msusa/en_US/pdp/Surface-Pen/productID.325724000.

Xiao Tu Ted Miller, Scott Stacey. Pen and ink: Inking at the speed of thought, 03 2016. URL https://channel9.msdn.com/Events/Build/2016/B865.

Unity Container. Unity container, May 2016. URL https://msdn.microsoft.com/en-us/library/ff647202.aspx.

Robert A. Wagner and Michael J. Fischer. The String-to-String Correction Problem, 1974. ISSN 00045411.

FM Waltz and JWV Miller. An efficient algorithm for Gaussian blur using finite-state machines. *SPIE Conf. on Machine Vision ...*, (July):1–8, 1998. ISSN 0277786X. doi: 10.1117/12.326976. URL `http://mii-prakse.googlecode.com/svn/prakse/AtteluApstrade/Subprojects/PixelMaster/trunk/doc/materi{%}C4{%}81li/Gaussianblurusingfinite-statemachines.pdf`.

Seznam tabulek

Seznam použitých zkratek

- LV - Levenshteinova vzdálenost
- DP - Dynamické programování
- FD - Fréchetova vzdálenost
- WF - Wagner-Fisherů algoritmus
- MVVM - Model-View-Viewmodel

Přílohy

Pokud struktura nebo formát příloh vyžaduje vysvětlení, přiložili jsme také soubor readme.txt s vysvětlením.

1. Zdrojové kódy k celému řešení. Zde jsou zdrojové kódy k aplikaci Podpis a také k aplikaci Benchmark. Organizace projektu je popsaná v programátorské dokumentaci. Tato příloha je ve složce /Zdrojové kódy/

2. Programátorská, uživatelská a technická dokumentace k aplikacím. Tato příloha je ve složce /Dokumentace/

3. Náhodně vygenerované podpisy a testovací soubory pro Benchmark aplikaci. Tato příloha je ve složce /Benchmark - data/

4. Podpisy nasbírané v experimentu při záznamu vlastního podpisu a při pokusech o napodobení cizího podpisu a testovací soubory použité pro benchmark aplikaci. Tato příloha je ve složce /Experiment/

5. Instalační balíček .appx pro aplikaci Podpis a spustitelný binární soubor a jeho reference pro benchmark aplikaci