VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

ROZPOZNÁNÍ RUČNĚ PSANÝCH ČÍSLIC

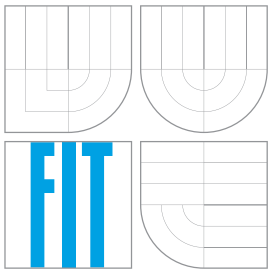
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

MICHAL HEKRDLA

BRNO 2009

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

ROZPOZNÁNÍ RUČNĚ PSANÝCH ČÍSLIC

RECOGNITION OF HANDWRITTEN DIGITS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — MICHAL HEKRDLA
AUTHOR

VEDOUCÍ PRÁCE — Ing. ADAM HEROUT, Ph.D.
SUPERVISOR

BRNO 2009

Abstrakt

Tato bakalářská práce se zabývá problematikou rozpoznání ručně psaných číslic metodou rozhodovacích stromů. Popisuje princip metody, využití databáze NIST (Národní institut standardů a technologie) pro účely učení algoritmu, konstrukci stromu tagů a rozhodovacího stromu. Popisuje také Implementaci této metody na demonstračním programu, který je její programovou částí. Na závěr se zabývá testováním rozpoznávání programu a jeho zhodnocením.

Abstract

This bachelors thesis inspects an issue of recognition of handwritten digits with decision trees method. It describes principle method, usage database NITS (National Institute of Standards and Technology) for purposes teaching algorithm, construction tags tree and decision tree. It describes too implementation those method on demonstrational program, which is its programme part. Finally it deal with testing recognition program and its estimation.

Klíčová slova

Rozpoznání ručně psaných číslic, tag, strom tagů, rozhodovací strom, NIST, Národní institut standardů a technologie

Keywords

Recognition of Handwritten Digits, tag, tags tree, decision tree, NIST, National Institute of Standards and Technology

Citace

Michal Hekrdla: Rozpoznání ručně psaných číslic, bakalářská práce, Brno, FIT VUT v Brně, 2009

Rozpoznání ručně psaných číslic

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Ing. Adama Herouta.

.....
Michal Hekrdla
19. května 2009

Poděkování

Chtěl bych zde poděkovat vedoucímu mé bakalářské práce Ing. Adamu Heroutovy za poskytnuté rady a hlavně za trpělivost.

Obsah

Kapitola 1

Úvod

Problematika rozpoznávání ručně psaných číslic je v dnešní době jistě významné téma. Setkáváme se s ním v běžném životě, stačí vyplnit složenku na zaplacení účtů a na poště ji jen naskenují a o ostatní se již postará příslušný program.

Způsobů pro rozpoznávání jakéhokoli psaného textu je několik. Nejznámější je asi využití neuronových sítí. Cílem této práce je však implementace rozpoznávání ručně psaných číslic metodou rozhodovacích stromů. Program je psán v programovacím jazyce `C/C++` a je funkční jak pod systémem `Windows` tak `Linux`.

V této práci je vysvětlen princip této metody. Jednotlivě se věnuje databázi číslic pro učení programu, předzpracováním obrazů číslic, stromu tagů, rozhodovacímu stromu a využití více stromů pro účely testování. Poté popisuje jak byly jednotlivé části implementovány, možnostmi nastavení a definici kontrolních výpisů. Na závěr řeší problematiku testování na testovací databázi a vyhodnocením testů.

Kapitola 2

Rozpoznání ručně psaných číslic

Tato kapitola se zabývá principem metody rozpoznávání číslic pomocí rozhodovacího stromu. Popisuje použitou databázi číslic, konstrukci stromu tagů, konstrukci rozhodovacího stromu a princip testování. Tato metoda je popsána v článku [1], uvedeném v literatuře u zadání této práce. Popis principu této metody v této kapitole vychází z tohoto článku.

2.1 Databáze číslic

Protože zvolená metoda rozhodovacích stromů potřebuje pro učení algoritmu velké množství obrázků číslic, není reálně možné tuto databázi vytvořit ručně. Pro tyto účely je vhodné využít již hotovou databázi z Národního institutu standardů a technologie (NIST) v USA [7], zabývajícím se touto problematikou. Databáze z tohoto institutu obsahuje přibližně po 6 tisících obrázků jedné číslice pro učení algoritmu a po 1 tisíci pro následné testování. Ukázku pár vzorků můžete vidět na obrázku 2.1.

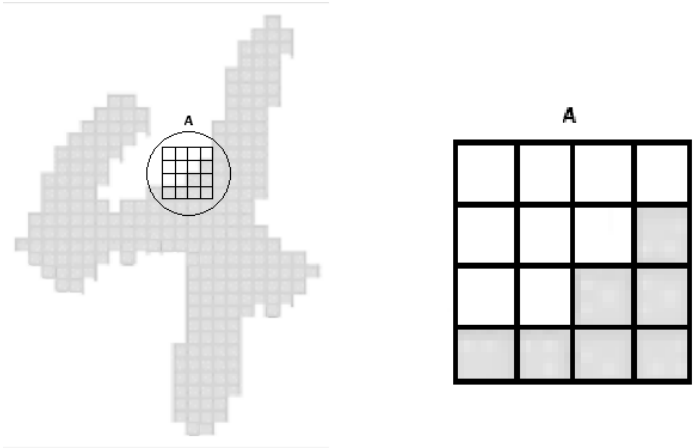
Obrázek 2.1: Příklad číslic z databáze. Převzato z [3]

2.2 Výběr významných bodů

Jako první zpracování obrázků číslic je výběr významných bodů. Významnými body se myslí specifické části v číslic, jako je koncový bod, roh, případně vysoké zakřivení.

2.3 Tagy

Pojmem tag se zde rozumí matice 4x4 pixelů z obrázku číslice. Tato matice tvoří okolí bodu (pixelu) a to tak, že tento bod je v jejím středu. Proto tuto matici, kromě pojmu tag, někdy též označujeme jako čtyřokolí. Toto okolí charakterizuje umístění každého bodu. Pro snažší práci nebereme pixely v odstínech šedi, ale podle zvoleného prahu je převádíme na černo-bílé. Takto přiřadíme čtyřokolí ke každému významnému bodu ze všech číslic. Tím nám vznikne množina tagů pro učení algoritmu. Příklad jednoho tagu je na obrázku 2.2, obrázek číslice převzat z [1].



Obrázek 2.2: Ukázka jednoho tagu z číslice 4.

2.4 Strom tagů

Strom tagů slouží k rozdělení tagů na typy. Počet typů je dán úrovní stromu, jelikož jeden typ odpovídá jednomu koncovému uzlu. Počet typů tagů tedy lze spočítat vzorcem: $pocet - typu = 2^{uroven-stromu}$. Tagy zde jsou na typy rozděleny na základě barvy některých pixelů. V každé úrovni stromu se porovná určený pixel a podle jeho barvy se pokračuje stromem hlouběji. Ukázka jak vypadají tři úrovně stromu tagů je na obrázku 2.3. Na obrázku je také ke každému typu tagu (koncovému uzlu) přiřazený pro ukázku jeden tag. Z tohoto rozdělení je patrné že jednotlivé typy tagů definují zda je tag v číslici umístěn na vodorovné, svislé, šikmé lince číslice. Čím více typů máme tím specifičtější část číslice definuje.

2.4.1 Struktura stromu

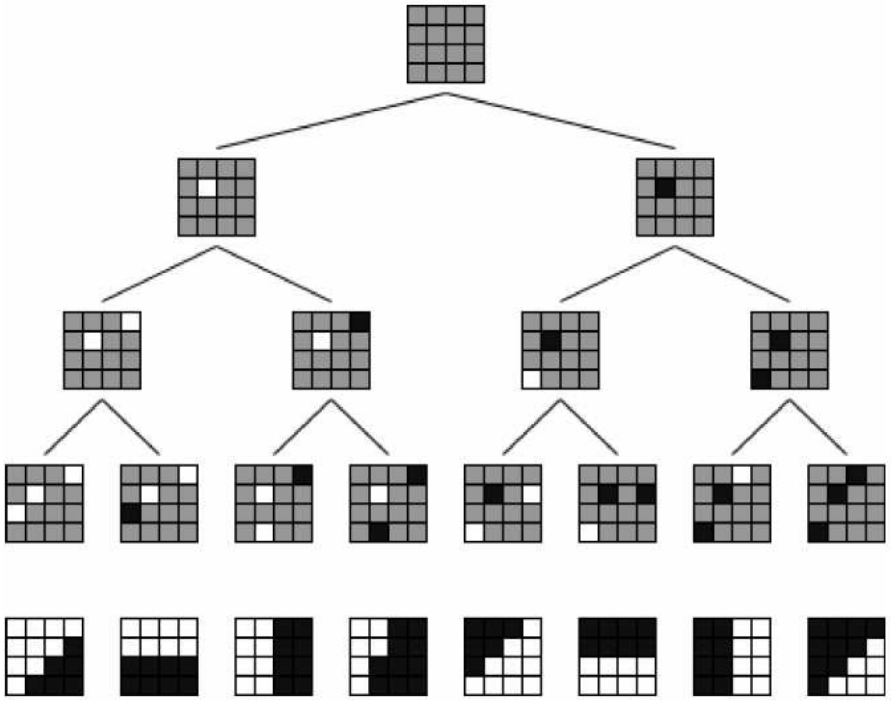
Každý uzel obsahuje číslo jednoho pixelu z tagu. Tento pixel větví strom vždy na dva podstromy. Levý, který má tento pixel bílý a praví, který má tento pixel černý.

2.4.2 Princip větvení stromu

V kořenovém uzlu vezmeme celou množinu tagů. Podle hodnoty pixelu určeným kořenovým uzlem rozdělíme množinu na dvě tak, že k levému podstromu přiřadíme tagy, jež mají tento pixel bílý, k pravému tagy, jež mají tento pixel černý. Poté vezmeme množinu tagů v levém podstromu a znovu ji rozdělíme podle hodnoty pixelu určeném tímto uzlem. Obdobně rozdělíme množinu tagů v pravém podstromu. Tímto způsobem pokračujeme až do zvolené úrovně stromu. V každém koncovém uzlu tedy vznikne množina tagů s několika stejnými pixely. Tyto pixely charakterizují tuto množinu a určují typ tagů v této množině. Číslo pixelu podle kterého se má strom v daném uzlu větvit se vybírá tak, aby se množina tagů, která odpovídá tomuto uzlu, rozdělila co možná nejrovnoměrněji na poloviny.

2.4.3 Práce se zkonstruovaným stromem

Pokud máme strom tagů již zkonstruovaný a chceme určit jakého je zvolený tag typu, Projdeme stromem od kořene a v každém uzlu testujeme příslušný pixel a na jeho základě pokračujeme levým či pravým podstromem. Až takto skončíme v koncovém uzlu, tento uzel nám udává typ tagu.



Obrázek 2.3: První tři úrovně stromu tagů s příkladem tagů. Převzato z [1]

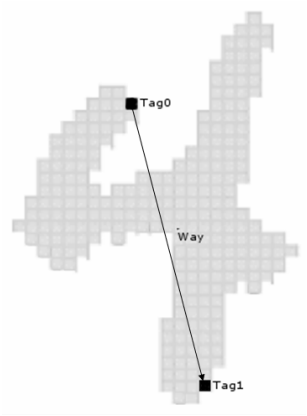
2.5 Rozhodovací strom

Rozhodovací strom již slouží k rozpoznávání číslic. Tento strom pracuje s „trojkombinacemi“ popsanými níže. Jeho konstrukcí získáme seznamy těchto kombinací, které se v číslici mají, nebo naopak nemají nacházet. Tento seznam nám jasně definuje druh číslice s určitou

pravděpodobností. Jak tyto seznamy definují typ číslice je znázorněno na obrázku 2.5. Na obrázku je vidět jak mají číslice 5, 6 a 8 společné tři „trojkombinace“, poté jsou rozděleny čtvrtou, která se u číslice 8 nachází, ale u číslic 5 a 6 nikoliv. Takto pracuje celý strom. Hledá „trojkombinaci“, jež by rozlišila dvě číslice. Takto pokračuje až oddělí všechny typy.

2.5.1 trojkombinace

Pojmem „trojkombinace“ se při konstrukci rozhodovacího stromu myslí kombinace dvou tagů a jejich vzájemná poloha. V obraze číslice vezmeme jeden tag (`Tag0`) a druhý tag (`Tag1`) a zjistíme jejich vzájemnou polohu (`Way`). Vzájemnou polohu určujeme tak, že určíme směr jakým se nachází druhý tag od prvního. Tento směr udáváme pomocí světových stran. Konkrétně tedy že druhý tag je na sever, severovýchod, atd. od prvního. Toto je ilustrováno na obrázku 2.4, obrázek číslice je převzat z [1].



Obrázek 2.4: Příklad „trojkombinace“ v obrázku.

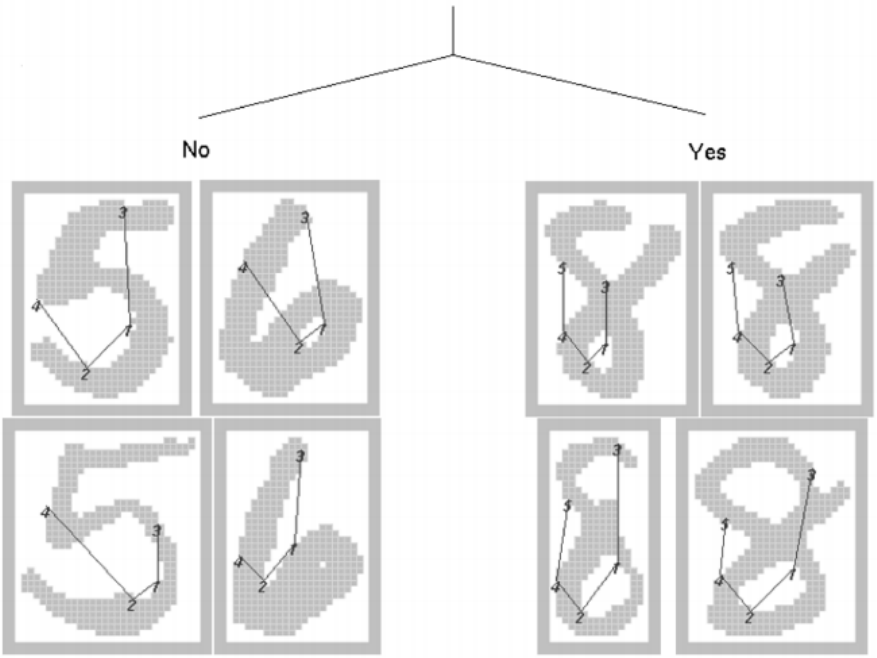
2.5.2 Struktura stromu

V každém uzlu stromu je uložena „trojkombinace“ podle níž se strom větví. Tato je vyhledána v obrázku číslice, pokud je v něm nalezena pokračuje se jedním podstromem, jinak druhým. Druhá položka uložená v uzlu je tabulka pravděpodobností. V této tabulce je pro každý typ číslice uložen procentuální odhad, že se jedná právě o ni. Pro číslici jež se nachází v tomto uzlu máme tedy informaci na kolik procent se jedná o číslici 0, 1, … až 9.

2.5.3 Princip větvení stromu

Jak je již zmíněno výše, strom se větví na základě, zda se v obrázku číslice nahází, nebo nenachází daná „trojkombinace“. Tímto způsobem při konstrukci stromu rozdělujeme množinu číslic. Vždy se snažíme u konkrétního uzlu vybrat „trojkombinaci“ z množiny obrázků tak, aby jsme rozvětvením tohoto uzlu co nejvíce od sebe oddělily dva druhy číslic. Takto postupně získáme uzly, kde bude převažovat jeden druh.

Postup výběru nejlepší „trojkombinace“ je následující. Náhodně si vygenerujeme tabulku s různými kombinacemi. Do této poté jednotlivě spočítáme pro všechny druhy číslic z množiny u daného uzlu, v kolika se kombinace nachází. Z této tabulky poté vybereme tu nejvhodnější „trojkombinaci“.



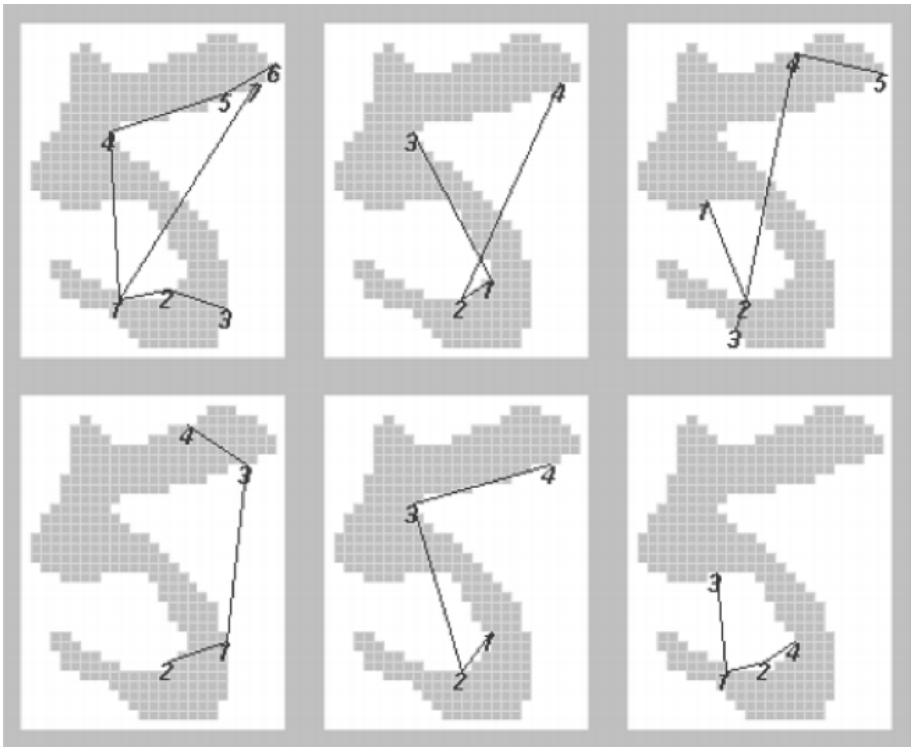
Obrázek 2.5: Ukázka větvení rozhodovacího stromu. Převzato z [1]

2.5.4 Práce se zkonstruovaným stromem

Když máme strom vytvořený, můžeme s ním již rozpoznávat číslice. V číslici nalezneme tagy pro všechny významné body a pomocí stromu tagů u nich určíme typ. Nyní procházíme stromem od kořene a pomocí získaných tagů testujeme zda se v tomto obrázku nachází „trojkombinace“ uložená v daném uzlu. Takto projdeme stromem až ke koncovému uzlu. V tomto uzlu se nachází tabulka pravděpodobností pro tuto číslici. Vybereme položku s nejvyšší pravděpodobností a můžeme říci, že s touto pravděpodobností jde o tuto číslici.

2.6 Využití více klasifikačních stromů

Protože konstrukce rozhodovacího stromu je ovlivněna náhodným generováním tabulky „trojkombinací“, je vždy vygenerován trochu jiný strom. Ukázka jak může být charakterizována číslice 5 šesti různými stromy je na obrázku 2.6. Celková úspěšnost těchto stromu se sice příliš razantně nemění, ale tyto stromy jsou schopné některé číslice rozpoznávat různě. Aby se potlačila tato náhodnost je pro rozpoznávání číslic použito více stromů a jejich výsledky jsou zprůměrovány. Díky tomu se zlepší i celková úspěšnost rozpoznávání, jelikož jestliže jeden strom číslici rozpozná chybně a ostatní správně je jeho špatná funkce potlačena. U jiné číslice může zas chybně rozpoznávat jiný strom a tak dále.



Obrázek 2.6: Charakteristika číslice 5 nalezená šesti různými stromy. Převzato z [1]

2.7 Testování

Pro testování je určena speciální databáze číslic [7]. Tato databáze obsahuje 10000 číslic přibližně po 1000 od každé číslice. Při testování je porovnána skutečný druh číslice s rozpoznaným. Takto je zjištěna celková úspěšnost rozpoznávání i úspěšnost jednotlivých číslic a za jaké číslice jsou nejvíce zaměňovány.

Kapitola 3

Návrh a Implementace

V této kapitole je popsána vlastní realizace programu. Je zde vysvětlena implementace jednotlivých částí programu zmíněných výše v teorii metody rozpoznávání číslic.

První verzi programu tvořil jeden program. Protože jsou však jednotlivé části metody časově náročné a při jejich vývoji a testování zdržovaly již funkční části, byl program rozdělen na tři. První na konstrukci stromu tagů, druhý na konstrukci rozhodovacího stromu a třetí na testování. Program na testování je vytvořen ve dvou verzích. Jedné s využitím jednoho rozhodovacího stromu a druhé s využitím pěti stromů. Jednotlivé části programu jsou od sebe odděleny za pomoci ukládání a zpětnému načítání zkonstruovaných stromů do xml souboru.

Při testování se ukázalo spouštění na mém počítači pod systémem `windows` jako příliš zdlouhavé, proto jsem v této fázi program upravil pod systém `Linux` a testoval na školním serveru `merlin`.

3.1 Vývojové prostředí

Protože pracuji převážně s operačním systémem `windows` projekt jsem vyvýjel pod tímto systémem ve vývojovém prostředí `Microsoft Visual Studio 2005` jež mají studenti `FIT` pro studijní účely licencované školou. Toto prostředí poskytuje i ladící nástroje a tak se ukázalo jako vhodné. Zvoleným programovacím jazykem je `C/C++`, protože s ním mám největší zkušenosti. Pro tento jazyk existuje navíc volně dostupná knihovna `OpenCV` [5] pro práci s obrazem a videem. Ta se ukázala jako velice vhodná pro práci s databází číslic. Protože jak jazyk `C/C++` tak i knihovna `OpenCV` je plně přenositelná je program přeložitelný a spustitelný pod operačním systémem `Linux`, což se v konečné fázi ukázalo jako přínosné. Díky vyššímu výkonu školního servru je tedy finální verze upravena pro tento systém.

3.2 OpenCV

`OpenCV` (Open Computer Vision library) [5] je volně dostupná grafická knihovna od společnosti `intel`. Poskytuje mnoho funkcí pro práci s obrazem a videem.

V programu je knihovna využita pro své možnosti zpracování obrazu. Slouží pro práci s databází číslic. Načítaní z databáze a hledání významných bodů v obraze.

3.3 Databáze číslic

Jak již bylo zmíněno výše, pro účely učení a testování je vhodné využít databázi z Národního institutu standardů a technologie (NIST) v USA [7]. V této databázi jsou bílé číslice na černém podkladu velikosti 20x20 px. V mém programu jsem využil databáze MNIST z webové stránky [4]. Jedná se o upravenou NIST. V této jsou obrázky také bílé na černém poli, ale velikosti 28x28 px. Databázi MNIST netvoří sada obrázků, ale pouze 2 soubory pro trénovací a 2 pro testovací sadu. V jednom ze souborů jsou data obrázků číslic a v druhém jakou číslici zobrazuje. Struktura souboru s popisem je popsána v tabulce 3.1, struktura souboru s obrázky je popsána v tabulce 3.2. Toto přináší jisté výhody i nevýhody. Výhodou je jednodušší přenositelnost databáze, paměťová velikost a rychlejší zpracování. Nevýhodou je nutnost přídavného algoritmu pro práci s touto databází. Pro tyto účely jsem vytvořil třídu `class_mnist`, která umožňuje načítaní jednotlivých obrázků i s informacemi o nich.

[offset]	[typ]	[hodnota]	[popis]
0000	32 bit integer	0x00000801(2049)	speciální číslo (MSB)
0004	32 bit integer	60000	počet položek
0008	unsigned byte	??	popis číslice
0009	unsigned byte	??	popis číslice
.....			
xxxx	unsigned byte	??	popis číslice

Tabulka 3.1: Soubor s popisem číslic. Převzato z [4]

[offset]	[typ]	[hodnota]	[popis]
0000	32 bit integer	0x00000803(2051)	speciální číslo (MSB)
0004	32 bit integer	60000	počet obrázků
0008	32 bit integer	28	výška obrázků
0012	32 bit integer	28	šířka obrázků
0016	unsigned byte	??	pixel
0017	unsigned byte	??	pixel
.....			
xxxx	unsigned byte	??	pixel

Tabulka 3.2: Soubor s daty obrázků číslic. Převzato z [4]

3.4 Výběr významných bodů

Pro výběr významných bodů jsou implementovány v programu 2 funkce, `getImg2Points()` a `getImg2Points2()`. Obě tyto funkce požadují jako vstup obrázek číslice a výstupem je pole nalezených bodů.

getImg2Points() V této funkci je hledání významných bodů implementováno pomocí funkce `cvGoodFeaturesToTrack()` z knihovny `OpenCV` [5]. Ukázala se však jako nevhodná, protože i při nastavení nízké kvality byl nalezený počet bodů pro program nedostačující. Jednalo se v průměru o přibližně 7 bodů na číslici.

getImg2Points2() Tato funkce využívá pro hledání významných bodů pouze Cannyho hranový detektor [2]. Konkrétně funkci `cvCanny()` z knihovny `OpenCV` [5]. Prahy Cannyho detektoru jsou nastaveny na hodnoty 120 a 230, jejichž hodnota byla zjištěna experimentálně. Poté jsou vráceny všechny body z takto nalezených hran, průměrně 85 bodů na číslici. Bodů je sice poněkud více, ale pro funkčnost je lepší bodů více než méně.

3.5 Tagy

Pro přiřazení tagu některému bodu je implementována funkce `xy2tag()`. Funkci se zadá obrázek číslice a souřadnice bodu, tag vrátí v poli 16 pixelů. Tag je uložen jako vektor, pro práci jako s maticí je nutné souřadnice v programu ručně přepočítat. Problémem je umístění bodu do středu matice. Jelikož se jedná o matici 4x4 pixelů, nelze nalézt přesný střed. Ten je ve funkci umístěn na souřadnice $[1, 1]$, počítáme-li souřadnice od nuly. Druhá věc, kterou bylo nutné ošetřit, je situace, kdy je okolí bodu již mimo obrázek. Tehdy jsou pixely mimo obrázek doplněny do tagu jako černé.

3.6 Strom tagů

Pro strom tagů je v implementaci programu vytvořena třída `class_tagTree`. Tato třída je uložena v hlavičkovém souboru `class_tagTree.h`. V tomto souboru jsou také definice datových typů se kterými tato třída pracuje. Třída obsahuje strom tagů a metody pro jeho práci. V konstruktoru je připravena inicializace kořenového uzlu a v destruktoru algoritmus pro jeho zrušení. Třída umožňuje konstrukci tohoto stromu, jeho uložení do xml formátu a jeho opětovné načtení. Na závěr metodu pro jeho využití, jež vrací typ daného tagu. Ještě zde jsou metody pro kontrolní výpisy. Obsahuje také různé metody, jež tvoří součást jijž zmíněných metod. Vstupem pro konstrukci tohoto stromu je vektor všech tagů získaných z databáze číslic.

3.6.1 Struktura stromu

Jedná se o binární strom. V každém uzlu je informace o úrovní uzlu a číslu uzlu v dané úrovní. Takto lze identifikovat každý uzel. Toho je využito při kontrolních výpisech a ukládání stromu do xml souboru. Pak obsahuje číslo pixelu z tagu, podle něhož se strom větví. V poslední úrovni je uložena hodnota -1. Poslední položkou je maska uzlu. Ta je tvořena maticí 4x4 pixelů, u každého je informace zda má být bílý, černý, nebo jeho barva nerozhoduje. Pomocí masky lze identifikovat tagy náležící danému uzlu.

3.6.2 Konstrukce stromu

Při konstrukci stromu jím rekurzivně procházíme od kořene až ke koncovým uzlům. Takto se dostaneme ke všem koncovým uzlům jež chceme rozvětvit, čímž vznikne nová úroveň stromu. Při větvení se postupuje tak, že se vyberou všechny tagy vyhovující masce daného uzlu. U těchto tagů spočítáme počet bílých a černých pixelů u všech 16 pixelů v tagu jednotlivě. Poté vybereme pixel, u něhož je co nejrovnoměrněji zastoupen bílý i černý. Na základě takto vybraného pixelu se uzel rozvětví. Tato celé opakujeme pro všechny úrovně stromu, jež chceme vytvořit.

3.6.3 Práce se zkonstruovaným stromem

Využití tohoto stromu je jednoduché. Chceme-li určit typ některého tagu, projdeme cyklem strom od kořene a testujeme příslušný pixel až dojdeme ke koncovému uzlu. Číslo koncového uzlu je zároveň typ tohoto tagu.

3.6.4 xml formát pro uložení stromu

Konstrukce tohoto stromu je poměrně náročná, jak časově tak převážně paměťově. Proto se strom po zkonstruování uloží do xml souboru a při jeho potřebě v jiném programu ho stačí načíst. Načtený strom je shodný se zkonstruovaným. Pro potřebu xml formátu je využita knihovna `TinyXml` [6].

Struktura xml souboru je následující.

```
<Tree LEVELTREE=,,4‘‘>        - kořenový element, obsahuje počet úrovní
  <level_0>                   - element pro uložení jedné úrovně stromu
      <node_0 ifTag=,,14‘‘ /> - element pro uložení uzlu z dané úrovně,
  </level_0>                    obsahuje pixel, podle nějž se uzel větví
  .........
  <level_4>
      <node_0 ifTag=,,-1‘‘ />
      .......
      <node_15 ifTag=,,-1‘‘ />
  </level_4>
</Tree>
```

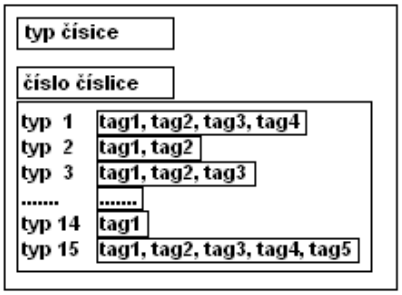
3.7 Rozhodovací strom

Rozhodovací strom je tvořen třídou `class_wayTree` v souboru `class_wayTree.h`. K této třídě jsou ve stejném souboru také přidány definice datových typů s nimiž pracuje. Třída obsahuje rozhodovací strom a tabulku „trojkombinací“ (`wayTable`), která je popsána níže. Disponuje metodou pro konstrukci stromu, jeho uložení do xml souboru, jeho načtení z xml souboru a metodu pro vyhodnocení obrázku číslice pomocí tohoto stromu, jež tvoří vlastní rozpoznávání programu. Metoda pro vyhodnocení vrací rozpoznaný typ číslice a tabulku pravděpodobností k ní přiřazenou. Poslední funkční metody slouží pro kontrolní výpisy. Ostatní metody jsou pouze součásti těchto již zmíněných. Vstupem této třídy jsou opět tagy, jsou již ale uspořádané. Jejich struktura je popsána v následující podkapitole.

3.7.1 Tagy

Tento strom pro svou konstrukci i rozpoznávání využívá tagy, již ale nestačí jejich množina jak tomu bylo u stromu tagů. Tyto tagy jsou uskupeny pro každý obrázek číslice zvlášť a navíc utříděny. Schéma jak vypadá struktura tagů pro jeden obrázek je na obrázku 3.1. Tato struktura obsahuje typ číslice z níž jsou tagy, pořadové číslo číslice z databáze a roztříděné tagy podle jejich typu. Utřídění tagů podle typu se ukázalo jako nutné pro zkrácení časové náročnosti konstrukce stromu. Při konstrukci dochází k neustálému vyhledávání těchto tagů a při jejich nesetřídění byla doba běhu programu neúnosná. Tyto tagy netvoří již matice

4x4 pixelů, ale pouze souřadnice bodu k němuž byl tag přidělen. Matice již není potřeba, pro funkci algoritmu stačí pouze jeho typ získaný stromem tagů. Tímto způsobem jsou zpracovány všechny obrázky.



Obrázek 3.1: Strukturované uložení tagů ke konstrukci rozhodovacího stromu.

3.7.2 Tabulka trojkombinací

Jak je již zmíněno v teorii konstrukce stromu výše, při konstrukci je potřeba tabulka „trojkombinací“ pro výběr kombinace podle níž se strom větví. Tato tabulka je obsažena uvnitř třídy. Její velikost není v teorii jasně určena a je do jisté míry ovlivněna počtem typů tagu, jelikož čím více typů, tím více kombinací existuje. Netvoříme však celou tabulku pro všechny kombinace, ale pouze tak velkou jak je určená v nastavení. Spoléháme se na náhodné generování kombinací. Tato tabulka je naznačena tabulkou 3.3 . První tři položky tvoří zmiňovanou „trojkombinaci“, zbytek je počet číslic pro jednotlivé druhy v níž se kombinace vyskytuje. Samozřejmě pouze z množiny číslic příslušící konkrétnímu uzlu.

`Tag0`	`Tag1`	`Way`	0	1	2	3	4	5	6	7	8	9
17	28	4	24	0	69	78	58	233	30	117	39	164
22	21	2	64	7	110	49	724	42	23	469	33	217
31	6	2	2	1	41	21	78	3	2	159	5	12

Tabulka 3.3: Ukázka struktury wayTable.

3.7.3 Hledání trojkombinace v číslici

Nejvíce využívaný algoritmus této konstrukce je právě test zda se v číslici nachází daná „trojkombinace“. Z tohoto důvodu jsou tagy setříděny, aby algoritmus pracoval co nejefektivněji. Vstupem je typ tagů `Tag0` a `Tag1` a jejich vzájemná poloha (`Way`). Postupně se otestují všechny kombinace těchto tagů a zjistí se jejich vzájemná poloha. Ta se zjišťuje nejdříve tak, že se umístí první tag na nulové souřadnice a určí se polorovina kde leží druhý tag ve směru osy x a y. Poté je spočtena tangenta úhlu mezi tagy pro upřesnění polohy. Pokud poloha souhlasí s hledanou, vrací `true`. Pokud projde všechny kombinace beze shody vrací `false`.

3.7.4 Struktura stromu

Jedná se opět o binární strom. Uzel obsahuje jeho identifikaci pomocí úrovně a čísla uzlu. Další položkou je vektor obsahující čísla řádků z `wayTable` podle nichž se k uzlu dostalo od kořenového uzlu. „Trojombinace“ jež nemá číslice obsahovat jsou značeny zápornou hodnotou. Tohoto se využívalo v první fázi vývoje, v konečné fázi zůstal tento seznam využit pouze pro kladnou a zápornou hodnotou a pro kontrolní výstup. Další vektor obsahuje podobné informace, ale „trojkombinace“ jsou v něm přímo uloženy. S nimi koresponduje první pro určení zda má číslice kombinaci obsahovat. Poslední „trojkombinace“ v tomto seznamu současně udává „trojkombinaci“ podle níž se strom větví. Předposlední položka je již tabulka pravděpodobností o jakou číslici se jedná. Na závěr je pro zrychlení algoritmu přidán vektor ukazatelů na číslice, které k uzlu patří, aby se nemusely vždy procházet všechny a testovat je.

3.7.5 Princip větvení stromu

Strom se větví podle určité „trojkombinace“. K vybrání té nejlepší se jich náhodně nageneruje několik do `wayTable` a spočítá jejich výskyt v jednotlivých číslicích. Teď následuje nejdůležitější rozhodnutí algoritmu a sice vybrání té nejvhodnější pro roztřídění číslic. V průběhu vývoje jsem postupně vyzkoušel tři varianty výběru. Postupně se vyskytovali nové parametry, jež bylo nutné ošetřit. Všechny tři jsou popsány níže.

1. **funkce** pracovala s celkovým počtem jednotlivých číslic v daném uzlu. V příslušné tabulce `wayTable` porovnávala dvě nejpočetněji zastoupené číslice. Hledala největší rozdíl u nich tak, aby byla „trojkombinace“ u jedné číslice zastoupena nejpočetněji a u druhé co nejméně. Tímto došlo k rozdělení těchto číslic podle dané „trojkombinace“. Problémem bylo že nezohledňovala ostatní typy číslic, jež mohli být zastoupeny jen o něco méně a bylo by vhodnější se zaměřit na ně.

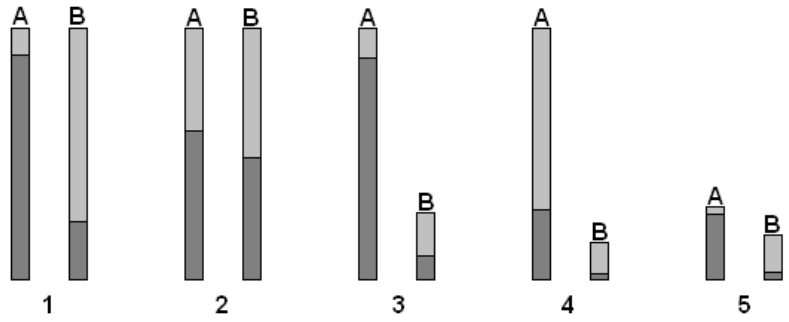
2. **funkce** pracovala na stejném principu jako předchozí, ale porovnávala všechny typy číslic navzájem. V prvních úrovních rozhodovacího stromu pracovala celkem úspěšně. Problém nastal v nižších úrovních kde již nejsou typy číslic zastoupeny rovnoměrně. Při porovnání dvou číslic, kde jedné byl velký počet a druhé malý, došlo k výběru pravidla, jelikož byl jejich rozdíl největší. Takto však nedošlo k žádnému rozlišení, protože se „trojkombinace“ vyskytovala u obou.

3. **funkce** je jakousi kombinací předchozích dvou. Zohledňuje celkový počet číslic a porovnává všechny typy navzájem. Navíc nepočítá pouhý rozdíl číslic v nichž se daná „trojkombinace“ vyskytuje. K tomuto rozdílu přičítá i rozdíl doplňků k celkovému počtu.

   Ilustrace této metody v různých situacích je zobrazena na obrázku 3.2.

   1. **varianta** zobrazuje ideální případ kde jsou číslice `A` a `B` zastoupeny rovnoměrně a v jedné se kombinace vyskytuje hojně a ve druhé ne. Rozdíl výskytů i doplňků je velký.
   2. **varianta** zobrazuje případ kde jsou číslice `A` a `B` zastoupeny rovnoměrně, ale v obou se kombinace vyskytuje přibližně stejně. Rozdíl výskytů i doplňků je malí. Výběrem této možnosti bychom jen obě množiny rozdělily napůl.

3. **varianta** zobrazuje případ kde je číslice `A` zastoupena podstatně více než `B`. V `A` se kombinace vyskytuje hojně a v `B` ne. Rozdíl výskytů je velký, ale rozdíl doplňků malý. Zde je vidět, že velký rozdíl výskytů nestačí pro vhodnou možnost.
4. **varianta** zobrazuje případ kde je číslice `A` zastoupena podstatně více než `B`. V obou se kombinace vyskytuje poměrně málo k celkovému počtu. Rozdíl výskytů i doplňků není výrazný.
5. **varianta** zobrazuje také ideální případ, je však dána přednost variantě 1 s vyšším počtem, protože součet rozdílů je větší.



Obrázek 3.2: Ilustrace 3.funkce pro výběr „trojkonbinace“.

3.7.6 Konstrukce stromu

Při konstrukci rozhodovacího stromu rozšiřujeme strom po úrovních. Rekurzivně procházíme stromem až se dostaneme ke kořenovým uzlům a jejich rozvětvením získáme další úroveň. U každého uzlu se nachází vektor ukazatelů na struktury s tagy pro číslice jež náleží tomuto uzlu. Při větvení uzlu z těchto číslic získáme příslušnou tabulku `wayTable`. Pro každý řádek tabulky otestujeme číslice u tohoto uzlu zda se v nich nachází „trojkombinace“ z tohoto řádku. Řádek poté obsahuje v kolika v nich se vyskytuje. Poté vybereme „trojkombinaci“ podle níž se bude strom dále větvit. Způsob jakým se vybírá je popsán výše. Po rozvětvení uzlu se k němu ještě vypočte tabulka pravděpodobností. Počty jednotlivých číslic u tohoto uzlu se přepočítají na procentuální hodnoty zastoupení u tohoto uzlu. Takto jsme tedy získali informace s jakou pravděpodobností se jedná o každou číslici. Na závěr ještě rozdělíme vektor ukazatelů do poduzlů podle toho jak se uzel větví pro další zpracování těchto uzlů. Takto přidáváme úrovně až získáme rozhodovací strom.

3.7.7 Práce se zkonstruovaným stromem

Pro rozpoznání číslice potřebujeme tagy z této číslice ve struktuře 3.1 v jaké byly při jeho učení. Nyní projdeme stromem od kořenového uzlu a postupně testujeme zda se v obrázku této číslice nachází „trojkombinace“ daná tímto uzlem. Podle této informace pokračujeme levým či pravým podstromem. Takto projdeme stromem až ke koncovému uzlu. V něm se nachází tabulka pravděpodobností příslušící této číslici. Ta je vrácena jako parametr. V této tabulce je nalezena položka s nejvyšší pravděpodobností a podle ní je vráceno o jakou číslici se jedná.

3.7.8 xml formát pro uložení stromu

Aby nebylo nutné při každém spuštění strom znovu konstruovat je po zkonstruování uložen do xml souboru. Konstrukce tohoto stromu je velmi časově náročná a tak by jeho neustálá konstrukce byla značně neefektivní. Při načtení stromu již nejsou dostupné některé hodnoty, které byli využity pro konstrukci stromu. Není tedy možné načtený strom dále rozšiřovat. Pro testování a kontrolní výpisy je však shodný. Pro uložení do formátu xml je využita knihovna `TinyXml` [6].

Struktura xml souboru je následující.

```
<Tree LEVELTREE2=,,8‘‘>   - kořenový element, obsahuje počet úrovní
  <level_0>               - element pro uložení jedné úrovně stromu
      <node_0 .... />     - element pro uložení uzlu z dané úrovně
  </level_0>
  .........
  <level_8>
      <node_0 .... />
      .........
      <node_255 maskWayRow=,,-984‘‘ Tag0=,,6‘‘ Tag1=,,29‘‘ Way=,,7‘‘
                p0=,,0.262850‘‘ ................ p9=,,0.554907‘‘ />
  </level_8>
</Tree>

element pro uložení uzlu obsahuje:
-  číslo řádku s trojkombinací z wayTabel (maskWayRow)
-  trojkombinace podle níž se uzel větví (Tag0, Tag1, Way)
-  jednotlivé položky z tabulky pravděpodobností (p0 až p9)
```

3.8 Možnosti nastavení programu

V programu je několik hodnot, které nejsou přesně určené. Tyto hodnoty ovlivňují funkčnost programu. Proto jsou nastavené pomocí definic a tyto pak shromážděny v souboru `config.h`. Pokud tedy chceme vyzkoušet program s různým nastavením, stačí změnit tyto hodnoty a znovu přeložit program. Tyto definice jsou:

LEVELTREE určuje maximální úroveň stromu tagů. Lze jí ovlivnit počet druhů tagů.

CNTLEVELTREE je pouze pomocná hodnota. Udává počet druhů tagů. Musí tedy odpovídat hodnotě $2^{LEVELTREE}$.

LEVELTREE2 určuje maximální úroveň rozhodovacího stromu. S každou novou úrovní se zlepšuje schopnost rozhodovaní, ale konstrukce jedné úrovně je poměrně časově náročná. Je tedy nutno volit nastavení úrovně s rozvahou. Navíc při vyšších úrovních není podstatné zlepšení zárukou.

CNTTABLEWAY udává velikost tabulky `wayTable` do které se generují „trojkombinace“ a poté se hledá ta jež dokáže čísla nejlépe rozlišit. Čím větší tedy bude, tím je větší pravděpodobnost že se do ní nageneruje ta nejvhodnější. S její velikostí je ovšem opět spjata časová náročnost.

REINITWT zapíná či vypíná inicializaci `wayTable` v každém uzlu. Nadefinováním teto položky na hodnotu 1 se budou „trojkombinace“ v tabulce `wayTable` přegenerovávat pro každý uzel rozhodovacího stromu znova. Pokud se tedy v daném uzlu nenageneruje vhodná „trojkombinace“, je šance že v další úrovni bude úspěšnější. Při jiné hodnotě `REINITWT` se tabulka vygeneruje na začátku a již se nemění. Takto je velká část algoritmu ovlivněna štěstím, jak se tabulka vygeneruje na začátku. Toto nastavení mělo smysl spíše při vývoji programu.

GOODWAYFUNCTION určuje funkci pro výběr nejvhodnější „trojkombinace“ z tabulky `wayTable`. Při vývoji programu jsem zkoušel více funkcí, které jsem v něm ponechal implementované a takto je lze přepínat. Funkce jsou 3, lze tedy nadefinovat hodnotu 1, 2, nebo 3. Nejnovější a nejlepší je číslo 3.

3.9 Kontrolní výpisy

Při vývoji programu se ukázala nutnost nahlédnou, jak pracuje s konkrétními hodnotami, pro kontrolu jeho správné funkce. Kontrola proměnných při ladění však nebyla dostačující. Proto je implementováno několik kontrolních výpisů, které mají již jistou formu, ze které lze něco vyčíst. Struktura a význam těchto výpisů je popsána níže. Výpisy se také ukázaly jako vhodná pomůcka k upřesnění jak program skutečně funguje. Ve finální verzi aplikace jsou však již nepotřebné a proto je jejich výpis podmíněn nadefinováním příslušného názvu. Tyto definice jsou pro přehlednost umístěny v souboru `config.h`.

Seznam definic a jím odpovídajícím souborům:

- PRINTTAG - Tag.txt
- CNTTAG - cnttag.txt
- PRINTTREE - Tree.txt
- PRINTWAYTABLE - (a,b)_wayTable.txt
- PRINTCNTNUM - Tree2cntNum.txt
- PRINTTREE2 - Tree2.txt

Aby bylo možné zpětně identifikovat při jakém nastavení programu byl kontrolní výpis pořízen, osahuje každý hlavičku s těmito informacemi.

Příklad hlavičky:
```
LevelTree 4
LevelTree2 6
CntWayTable 500
REINITWT 1
GOODWAYFUNCTION 3
```

3.9.1 Soubor Tag.txt

Soubor `Tag.txt` obsahuje informace o databázi číslic a o počtech načtených tagů. V první časti je celkový počet číslic v databázi a jednotlivě pro typy číslic. V další je získaný počet

tagů celkem i jednotlivě pro druh číslice. Z tohoto lze vyčíst průměrný počet tagů na jednu číslici celkem nebo průměr pro konkrétní druh. Poslední část obsahuje počty tagů pro jednotlivé druhy tagů získané pomocí stromu tagů, rozdělené i podle jednotlivých číslic.

```
pocet cisel:60000
     0      1      2      3      4      5      6      7      8      9
[ 5923   6742   5958   6131   5842   5421   5918   6265   5851   5949 ]
pocet tagu:5100179
       0      1      2      3      4      5      6      7      8      9
[ 629226 330100 554184 574551 471004 488409 519600 471662 568546 492897 ]
-----------------------------------------------------------------------
tag:celkem        0     1     2     3     4     5     6     7     8     9
  0:330434    36276 25628 32755 28639 33016 21535 42315 36278 40337 33655
  1:385902    57501 23526 40068 35377 41099 23990 49084 37127 40877 37253
                          ......................
 14:276167    29089 14673 40084 36541 23496 39517 24884 20359 26738 20786
 15:263878    32272  8726 31656 33388 18938 32185 30971 22539 32988 20215
```

3.9.2 Soubor cnttag.txt

Soubor `cnttag.txt` ukazuje princip výběru pixelů, podle něhož se větví strom tagů. Znázornění výběru je vždy uvozeno identifikací uzlu, kterému náleží. Obsahuje v kulatých závorkách číslo úrovně, v níž je uzel a číslo uzlu v dané úrovni. Výběr pro jednotlivé uzly je navíc oddělen pár pomlčkami.

Výpis pro jeden uzel obsahuje 16 řádků. Každý řádek odpovídá určitému pixelu z tagu. První je zde uvedeno číslo pixelu. Pak následuje počet napočítaných bílých a černých pixelů z množiny tagů daného uzlu, oddělené znakem `':'`. Nakonec je zde ještě uvedeno, jaký pixel byl vybrán pro větvení uzlu.

Příklad pro jeden uzel:

```
( 2,  0)
 0) 1106278:  257683
 1)  615725:  748236
       ........
12)  647625:  716336
13)  239700: 1124261
14)       0: 1363961
15)  299509: 1064452
vybrano:12
```

3.9.3 Soubor Tree.txt

Soubor `Tree.txt` obsahuje zkonstruovaný strom tagů. Jeden řádek výpisu náleží jednomu uzlu stromu. V kulatých závorkách je identifikace uzlu. První číslo je úroveň stromu v níž se uzel nachází. Druhé je číslo uzlu v dané úrovni. Poté následuje v hranatých závorkách maska uzlu. Tečka vyjadřuje nevýznamný pixel, číslo 1 bílý a číslo 0 černý pixel. Poslední je číslo pixelu, podle nějž se daný uzel dále větví.

```
Příklad čtyř uzlů:
( 3, 0)[  .  .  1  .  .  .  .  .  .  .  .  .  1  .  1  .]  1
( 3, 1)[  .  .  1  .  .  .  .  .  .  .  .  .  0  .  1  .] 13
( 3, 2)[  .  .  0  .  .  .  .  1  .  .  .  .  .  .  1  .]  4
( 3, 3)[  .  .  0  .  .  .  .  0  .  .  .  .  .  .  1  .]  0
```

3.9.4 Soubory (a,b)_wayTable.txt

Soubor `(a,b)_wayTable.txt` obsahuje tabulku `wayTable` pro daný uzel. Pro každou tabulku je vytvořen jeden soubor. Uzel jemuž tabulka náleží je identifikován v názvu souboru v kulatých závorkách. První číslo je úroveň stromu v níž se uzel nachází. Druhé je číslo uzlu v dané úrovni. Jeden řádek výpisu obsahuje jeden řádek tabulky. První číslo je číslo řádku tabulky. V kulatých závorkách následuje „trojkombinace“ (`Tag0, Tag1, Way`). Jednotlivé položky „trojkombinace“ jsou odděleny znakem `'|'`. Nakonec jsou v hranatých závorkách, oddělené čárkou, hodnoty v kolika číslicích byla „trojkombinace“ nalezena. Jednotlivě od číslice 0 po číslici 9.

```
Příklad několika řádků:
106( 23| 15| 5)[ 1603,  404, 1827, 2618, 2635, 2573, 2990, 1174, 2713, 1383]
107(  4|  0| 1)[ 1173,  102, 1354, 2189, 1038, 1034, 2679,  988, 2552, 1621]
108( 30| 13| 0)[ 1255,    8, 1030, 1398,   93, 2527, 1016,   75, 2017,  814]
109( 23| 23| 3)[ 4028, 2483, 2483, 2710, 2923, 2093, 4141, 4121, 3564, 2858]
110( 24| 19| 7)[ 1523,    1,  470,  753,   54,  562,  339,  372,  423,  412]
111(  1| 29| 6)[ 1163,   54, 1374,  873,  417,  388,  207, 1555,  455,  517]
112( 12| 17| 3)[ 4868,   45, 1872, 2022, 4129, 3599, 4223,  449, 4768, 3481]
113( 21| 20| 7)[ 2475, 2404, 1598,  771, 2718,  420, 1829, 2824, 2331, 2314]
114( 17| 28| 0)[ 4164, 1627, 4733, 5375, 3890, 4375, 5201, 2333, 4590, 3658]
```

3.9.5 Soubor Tree2cntNum.txt

Soubor `Tree2cntNum.txt` znázorňuje postup rozdělování číslic při konstrukci rozhodovacího stromu. Obsahuje množiny jednotlivých číslic náležící konkrétnímu uzlu stromu. Pokud si v souboru nalezneme uzel i s jeho potomky, lze vyčíst jak se množiny rozdělily. Podle tohoto můžeme posoudit zda se strom větví správně. Při správné konstrukci je v nižších uzlem patrné, jak množina jedné číslice převahuje. Pokud by se podařilo osamostatnit množinu jedné číslice úplně, bylo by její rozpoznání stoprocentní. V reálu se však jen snažíme tomuto stavu co nejvíce přiblížit.

Struktura souboru je následující. V kulatých závorkách je identifikován konkrétní uzel pomocí čísla úrovně uzlu (první číslo) a čísla uzlu v dané úrovni (druhé číslo). Pak následují v hranatých závorkách jednotlivé počty číslic (množiny) pro tento uzel. Postupně od číslice 0 po číslici 9. Nakonec je přidána suma těchto číslic.

```
Příklad jedné úrovně:
( 3, 0)[  899,    1,  981, 1946,  122,  985,  332,   31, 3822,  806,] 9925
( 3, 1)[ 4652,    0,  260,  521,  171,  919, 2402,   24,  897, 1271,]11117
( 3, 2)[   54,    2,   74,  142,    6,  760,  163,   11,  141,  145,] 1498
( 3, 3)[  137,   37,  931,   38, 2846,   91, 1527,  282,  357,  391,] 6637
```

```
( 3, 4)[    9,   17,  774, 1822,  372, 1989,   94,  110,   58,  338,] 5583
( 3, 5)[   87,  210, 2716, 1484, 1866,  290,  364, 5020,  218, 2368,]14623
( 3, 6)[   75,   61,  143,  116,  239,  226,  961,   33,  312,  471,] 2637
( 3, 7)[   10, 6414,   79,   62,  220,  161,   75,  754,   46,  159,] 7980
```

3.9.6 Soubor Tree2.txt

Soubor `Tree2.txt` obsahuje zkonstruovaný rozhodovací strom. Jeden řádek výpisu obsahuje informace o jednom uzlu stromu. O jaký uzel se jedná je zapsáno v kulatých závorkách, pomocí čísla úrovně v níž je uzel (první číslo) a čísla uzlu v této úrovni (druhé číslo). Poté následuje v hranatých závorkách tabulka pravděpodobností pro jednotlivé číslice. Pokud máme obrázek číslice náležící k tomuto uzlu, tak tato tabulka udává, s jakou pravděpodobností se jedná o konkrétní číslici. Tabulka je zapsána od číslice 0 po 9.

Za tabulkou následuje seznam „trojkombinací“, jež definují tento uzel. Tento seznam tvoří jakousi masku číslice. „Trojkombinace“ zde nejsou uvedeny přímo, ale jsou zde čísla řádků do příslušné tabulky `wayTable`. Záporná hodnota některého z těchto čísel značí že daná „trojkombinace“ v číslici být nemá. Poslední číslice v tomto seznamu je číslo řádku z `wayTable` s „trojkombinací“ podle níž se uzel dále větví (Samozřejmě kromě poslední úrovně).

```
Příklad prvních dvou úrovní:
( 0, 0)[  9 11  9 10  9  9  9 10  9  9]  243
( 1, 0)[ 14  0 13 15  1 11 10  7 13 12]  243  297
( 1, 1)[  3 26  5  2 21  5  9 15  5  6] -243  322
( 2, 0)[ 34  0  5  2  0  4 17  1 20 13]  243  297  178
( 2, 1)[  4  0 17 23  1 15  6  9  9 12]  243 -297  829
( 2, 2)[  1  1  5  3 39  3  5 26  2  9] -243  322  885
( 2, 3)[  4 43  5  1  7  7 11  6  7  3] -243 -322  396
```

Kapitola 4

Testování

Pro testování programu slouží speciální sada obrázků číslic. Tato sada obsahuje 10000 vzorků, přibližně po 1000 ke každému typu číslice. Testovací databáze obsahuje oproti trénovací obrázky v horší kvalitě.

Při testování v číslici nalezneme významné body, přiřadíme k nim tagy a ty stromem tagů otypujeme. Tyto tagy setřídíme a pomocí rozhodovacího stromu určíme typ číslice na obrázku. Tuto hodnotu porovnáme s informací o skutečném typu číslice. Takto otestujeme celou databázi.

Pro zjištění úspěšnosti programu a jiné informace jsou výsledky jednotlivých testů shromažďovány v tzv. confusion matrix. Jedná se o matici kde řádku odpovídá testovaný typ číslice a sloupci pak typ číslice, jaký rozpoznal rozhodovací strom. Na hlavní diagonále této matice tedy nalezneme počet úspěšně rozpoznaných číslic. Procentuálním přepočtem k celkovému počtu číslic daného typu, zjistíme úspěšnost rozpoznávání tohoto typu číslice. Zprůměrováním pak získáme celkovou úspěšnost programu. Vezmeme-li celý řádek tabulky, máme přehled za jaké číslice je daný typ nejčastěji zaměňován.

4.1 Testovaní jedním rozhodovacím stromem

Při testování programu pomocí jednoho rozhodovacího stromu vyplynulo, že náhodnost konstrukce rozhodovacího stromu ovlivňuje jeho rozpoznávací schopnost. Celková úspěšnost rozpoznávání se pohybovala od 60% do 67% a při porovnání rozpoznání dvou odlišných stromů se ukázalo že oba stromy jsou schopné rozpoznat jiné číslice. Proto je výhodné využití více rozhodovacích stromů popsané v další kapitole.

4.2 Testování pěti rozhodovacími stromy

Jak bylo zmíněno dříve, rozpoznávání číslic jedním rozhodovacím stromem je značně ovlivněno náhodným generováním při jeho konstrukci. Pokaždé vznikne jiný strom s rozdílnou schopností rozpoznávat číslice. Proto je pro rozpoznávání využito více stromů najednou. Tím se potlačí chyba způsobená jedním stromem, neboť převáží výsledek ostatních. V programu je využito rozpoznávání pěti stromy. Od všech stromů získáme pro konkrétní číslici pět tabulek pravděpodobností. Jednotlivé hodnoty sečteme a vydělíme pěti. Takto získáme novou tabulku pravděpodobností, vzniklou zprůměrovaním všech pěti. V této pak najdeme číslici s nejvyšší pravděpodobností a tím získáme rozpoznaný typ číslice. Při sumarizaci

testů již postupujeme běžným způsobem. Výsledky testů pěti stromy jsou více jak o 10% úspěšnější než jednotlivé stromy. Přínos využití více stromů je tedy zřejmý.

4.3 Formát výstupu testu

Výstup testů se ukládá do souboru `Test_out.txt`. Stejně jako kontrolní výpisy obsahuje hlavičku s nastavením při jakém byl vytvořen. Poté již následují testy jednotlivých číslic. Testu jedné číslice odpovídá jeden řádek. První číslo je pořadové číslo číslice z testovací databáze. V kulatých závorkách je skutečný druh číslice na obrázku. V hranatých následuje pravděpodobnostní tabulka vrácevá rozhodovacím stromem a za ní o jakou číslici se podle něj jedná. Pokud je rozpoznání chybné, je na konec přidáno slovo `false`.

Příklad testů:

```
17:(7)[  0  2 11 11  2  5  1 55  1  8] 7
18:(3)[ 22  0 13 14  0  4  7  0 19 17] 0 false
19:(4)[  1 18  0  0 59  0  6  4  2  6] 4
20:(9)[  1  0  6 10  4  3  1 55  2 13] 7 false
21:(6)[ 15  0  3  0  1  2 63  0  7  3] 6
22:(6)[ 27  0  3  1  1  2 41  0  7 13] 6
23:(5)[  2  1  6 11  1 53 16  2  3  1] 5
```

Na konci souboru je zobrazena úspěšnost testů. Příklad těchto informací je níže. Pro každý typ číslice obsahuje počet správně rozpoznaných číslic (`ok`), celkový počet číslic (`sum`) a procentuální úspěšnost tohoto typu číslice. Na závěr jsou informace o celkové úspěšnosti. U každého typu číslice je navíc tabulka, za jaké číslice byla rozpoznávána.

Příklad úspěšnosti testů uložené ve výstupním souboru:

```
0) ok:  707 sum:  980  72\%   707    1    6    6    6    6   97    0  137   14
1) ok: 1093 sum: 1135  96\%     0 1093   10    1    3    0    0   20    4    4
2) ok:  630 sum: 1032  61\%    28    2  630  160   21   12   24   37  102   16
3) ok:  606 sum: 1010  60\%    17    0  250  606    3   37   11   20   55   11
4) ok:  697 sum:  982  70\%    15    7   65    7  697    6   90   29   28   38
5) ok:  393 sum:  892  44\%    41   10   15  248    5  393   58    9   90   23
6) ok:  664 sum:  958  69\%    89    9   52   18   32   17  664    1   53   23
7) ok:  877 sum: 1028  85\%     2    7   65   22   36    7    6  877    1    5
8) ok:  626 sum:  974  64\%    93    6   63   48   16   31   56    1  626   34
9) ok:  373 sum: 1009  36\%   154    7  123   18  133   28   25   46  102  373
celkem) ok: 6666 sum: 10000  66\%
```

4.4 Testy

Při testování bylo experimentálně zjištěno nejvhodnější nastavení programu na hodnoty:

```
LevelTree 5
LevelTree2 8
CntWayTable 1000
REINITWT 1
GOODWAYFUNCTION 3
```

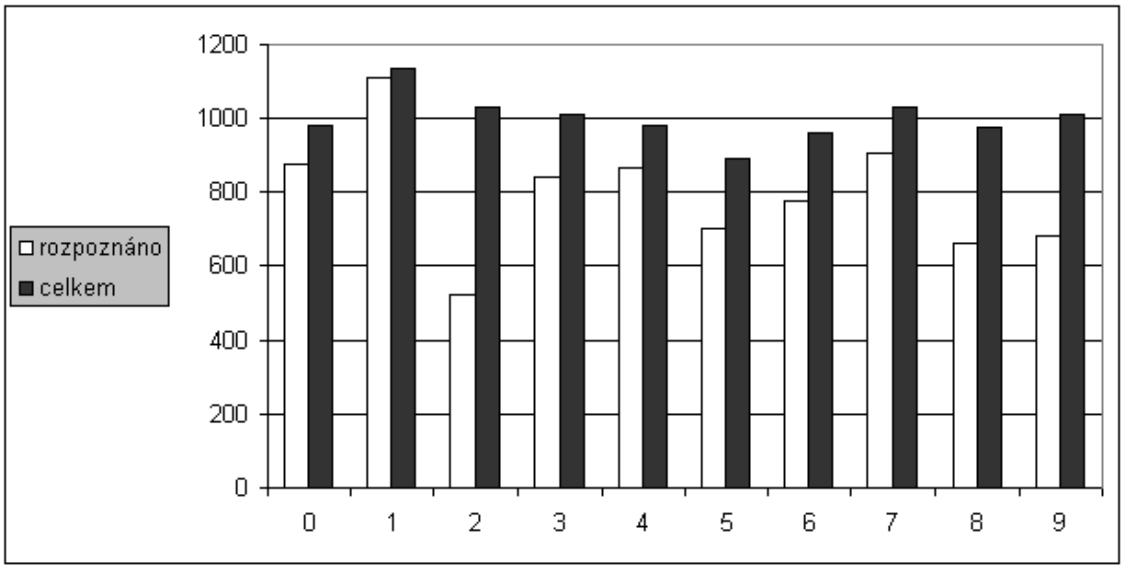
Výsledek testů s tímto nastavením a využitím pěti rozhodovacích stromů je zobrazen tabulkou 4.1, obsahující confusion matrix tohoto testu. V tabulce 4.2 je pak z této matice vypočtena procentuální úspěšnost. Graf 4.1 znázorňuje rozpoznaný počet jednotlivých číslic ku celkovému počtu. Na závěr je zde grafické znázornění confusion matrix, na němž je vidět, jak jsou jednotlivé číslice rozpoznány. Každý graf zobrazuje rozpoznání jednoho druhu číslice a v grafu pak jednotlivé počty, jak byla číslice rozpoznána.

	0	1	2	3	4	5	6	7	8	9
0	876	0	8	8	10	7	24	3	24	20
1	0	1111	4	6	1	0	5	8	0	0
2	35	4	523	287	12	17	23	70	34	27
3	10	2	35	840	2	42	5	37	26	11
4	3	11	2	4	863	6	22	22	4	45
5	19	6	9	109	5	700	20	6	14	4
6	89	10	9	4	30	20	774	1	15	6
7	3	13	29	18	30	1	1	906	5	22
8	89	6	23	35	34	43	30	7	660	47
9	41	11	12	26	123	27	10	59	26	674

Tabulka 4.1: Confusion matrix.

číslice	0	1	2	3	4	5	6	7	8	9	celkem
úspěšnost [%]	89	97	50	83	87	78	80	88	67	66	79

Tabulka 4.2: Tabulka procentuální úspěšnosti.



Obrázek 4.1: Úzpěšnost rozpoznání jednotlivých číslic.

4.5 Zhodnocení testů

Z výsledku testů je patrné, že úspěšnost mnou implementovaného programu nedosahuje dostačujících výsledků. Není to až tak špatné, ale pro reálné využití je chybovost nezanedbatelná. Návrhy jak by bylo možné program vylepšit jsou popsané v následující kapitole. Z testů lze vyčíst, že největší problém má program s rozpoznáním číslic 2, 8 a 9. K nim pak můžeme dohledat za jaké číslice jsou nejvíce zaměňovány.

4.6 Možnosti zlepšení programu

Zde bych tedy zmínil v čem vidím jisté rezervy pro zlepšení, nebo jiné postupy, které by mohli zlepšit metodu. Z časových důvodů jsem jejich efekt neprověřoval a slouží spíš jako inspirace pro pokus o zlepšení.

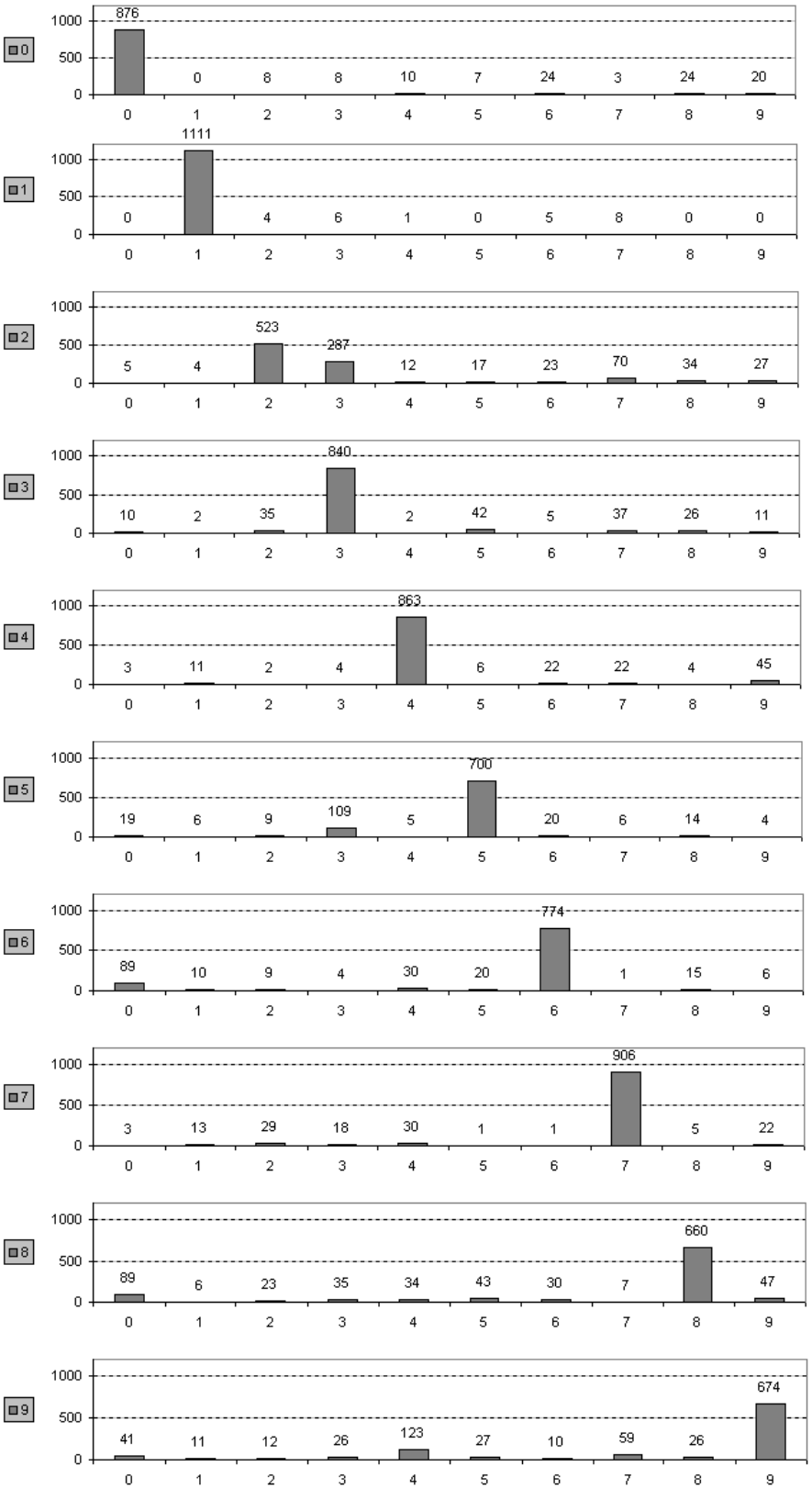
Výběr významných bodů - První zlepšení by mohlo být možné výběrem významných bodů. Vyzkoušel jsem hledat opravdu významné body, těch však bylo málo a tak jsem zvolil hranový detektor a vzal všechny body z této hrany. Dobré by bylo vyzkoušet zkombinovat oba postupy. Najít tedy významné body, ale ve větším počtu.

Výběr trojkombinace - Jistou možnost poskytuje výběr „trojkombinace“ při konstrukci rozhodovacího stromu. Použitá funkce kombinuje efektivnost s jednoduchostí. Funkci považuji za postačující, je však možné že nějakým složitějším algoritmem lze dosáhnout o něco lepších výsledků.

Trojkombinace - Při zvažování jak zlepšit úspěšnost mě napadlo, že jistou možnost poskytuje základní položka metody a sice „trojkombinace“. V obraze by se místo dvou tagů a jejich polohy hledaly tři tagy jejich vzájemná poloha. Jde však o zásadní zásah do principu metody, který by se jistě promítl na časová náročnosti a jeho přínos je pouze teoretický.

Dodatečné rozlišení - Z testů lze odhalit za jaké číslice jsou problematické číslice zaměňovány. Pokud tedy rozhodovací strom vyhodnotí číslici jako jednu z problémových typů, můžeme ještě nějakým způsobem zkontrolovat zda se nejedná o typ za který bývá zaměňována.

Příklad jak od sebe jednoduše rozlišit číslici 0 a 8 je následující. Číslicí vedeme několik vertikálních řezů a zjistíme kolikrát jednotlivé řezy protnuly číslici. Pokud některý řez protne číslici více jak dvakrát, nejedná se velmi pravděpodobně o číslici 0. Takto, nebo jiným jednoduchým způsobem můžeme tedy dodatečně pomoci metodě v problémových případech.



Obrázek 4.2: Grafy rozpoznání jednotlivých číslic.

Kapitola 5

Závěr

Běhe vývoje demonstrační aplikace pro rozpoznávání ručně psaných číslic jsem se potýkal především s otimalizací programu. V první fázi vývoje byl program neůnosně náročný, jak časově tak paměťově. To prakticky znemožňovalo průběžné testování funkčnosti. Oba problémy byli naštěstí optimalizovány. Přesto je čas běhu programu poměrně velký, to je však dáno jeho náročností. Pro lepší testování je tedy rozdělen na tři části. Konstrukci stomu tagů, rozhodovacího stromu a testování.

V článku popisujícím metodu rozpoznávání pomocí rozhodovacích sromů [1] se zmiňují, že touto metodou dosáhli 93% úzpěšnosti. Mě se bohužel podařilo dosáhnout pouze 79%. I tak je vidět, že zmíněná metoda přináší možné řešení této problematiky. Možnosti jak se pokusit ještě zlepšit mou implementaci jsem popsal v kapitole s testováním. Zmíněné návrhy jsem již nestihl realizovat a tak je jejich přínos pouze teoretický.

Možnosti využití práce k dalšímu rozšíření jsou jednak zlepšení úzěšnosti rozpoznávání a dále pak zakomponování pro reálné využití. Tím by mohli být programy na zpracování různých formulářů jež jsou vyplňovány ručně, nebo třeba i rozpoznávání číslic ve videu.

Na návrhu a realizaci bakalářské práce vidím jako největší přínos práce na komplexnějším projektu, jež má i reálné využití. Seznámení s touto metodou rozpoznávání bylo rozhodně zajímavé. Přínosem byla i práce s obrazem pomocí knihonvy `OpenCV` [5], která poskytuje opravdu velké možnosti pro zpracování obrazu a videa. Tyto zkušenosti se jistě do budoucna budou hodit.

Literatura

[1] AMIT; GEMAN; WILDER: *Joint Induction of Shape Features and Tree Classifiers.* IEEE Trans. on Pattern Analysis and Machine Intelligence, 1997.

[2] GREEN, B.: Canny Edge Detection Tutorial [online]. `http://www.pages.drexel.edu/~weg22/can_tut.html`, 2002 [cit. 2009-05-07].

[3] LECUN, Y.; BOTTOU, L.; BENGIO, Y.; aj.: Gradient-based learning applied to document recognition. [online]. `http://yann.lecun.com/exdb/publis/index.html#lecun-98`, Nov-1998, 86(11):2278-2324.

[4] LECUN, Y.; CORTES., C.: THE MNIST DATABASE of handwritten digits. [online]. `http://yann.lecun.com/exdb/mnist/index.html`, [cit. 2009-05-07].

[5] OpenCV community: OpenCV Wiki [online]. `http://opencv.willowgarage.com/wiki/`, 2009-03-18 [cit. 2009-05-07].

[6] THOMASON, L.; BERQUIN, Y.; ELLERTON, A.; aj.: TinyXml Documentation 2.5.1 [online]. `http://www.grinninglizard.com/tinyxmldocs/index.html`, 2006-04-18 [cit. 2009-05-07].

[7] WILKINSON, R.; GEIST, J.; JANET, S.; aj.: *The First Census Optical Character Recognition System Conference.* Technical Report, National Institute of Standards and Technology, 1992, nIS-TIR 4912.

Dodatek A

Obsah CD

- zdrojové texty programů
- programy ve spustitelné formě
- písemná zpráva ve formátu PDF
- zdrojový tvar písemné zprávy
- programová dokumentace
- manuál
- plakát pro reprezentaci práce
- příklad kontrolních výpisů a testů
- zkonstruované stromy v xml souborech