Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky
Katedra informatiky

Optické rozpoznávanie testov

DIPLOMOVÁ PRÁCA

Bc. Juraj Prievalský

Študijný odbor: 9.2.1 Informatika
Vedúci práce: RNDr. Richard Ostertág

BRATISLAVA 2009

Prehlásenie:

Týmto prehlasujem, že som diplomovú prácu vypracoval samostatne a všetku použitú literatúru uvádzam.

.....

Touto cestou si dovoľujem poďakovať p. RNDr. Richardovi Ostertágovi za odborné vedenie a rady počas spracovávania diplomovej práce.

Abstrakt

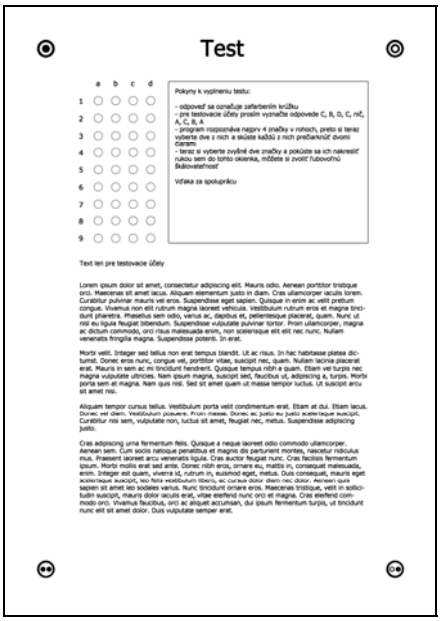
Cieľom diplomovej práce bolo navrhnúť spôsob rýchleho a spoľahlivého rozpoznávania značiek počítačom a ich využitie pri vyhodnocovaní naskenovaných testov. Test štandardného formátu papiera A4 bude pozostávať z odpoveďových políčok a špeciálne navrhnutých značiek umiestnených do rohov papiera. Program dostane na vstupe naskenovaný test a automaticky ho vyhodnotí. Systém bude robustný na afinné transformácie a šumy, ktoré môžu vzniknúť pri naskenovaní testu. K dosiahnutiu cieľa boli použité technológie .NET a jazyk C#. Aplikácia beží na platforme Windows a súčasťou je aj ukážková sada testov.

Kľúčové slová: optické rozpoznávanie testov, topologické značky, geometrické značky

Obsah

1 Úvod

V dnešnom modernom svete sa čoraz viac preferuje ukladanie a spracovávanie dát v elektronickej forme. Je bežné naskenovať si textový dokument a potom ho dať automaticky rozpoznať OCR softvéru. Ale čo keď potrebujeme vyhodnotiť zadané informácie od používateľa? Nemusíme mať pritom žiadny špeciálny skener, špeciálnu tlačiareň ani špeciálny papier. Program dostane na vstupe naskenovaný test, ktorý rozpozná a automaticky vyhodnotí.



Obr. 1.1: Ukážka testu

1.1 Cieľ práce

Hlavným cieľom práce je navrhnúť spôsob rýchleho a spoľahlivého rozpoznávania značiek počítačom a ich využitie pri vyhodnocovaní naskenovaných testov.
Ďalšími cieľmi sú:

- prehľad existujúcich riešení
- analýza značiek a algoritmov na ich rozpoznávanie
- návrh vlastných značiek
- implementácia viacerých algoritmov rozpoznávania značiek
- implementácia rozpoznávania testov

1.2 Motivácia

Hlavnou motiváciou práce je reálne použitie v praxi, napríklad na akademickej pôde pre automatické opravovanie testov. V súčasnosti ani naša fakulta nedisponuje takýmto softvérom. Existuje viacero riešení založených na neurónových sieťach alebo strojovom učení. Tieto metódy sú však častokrát založené na pravdepodobnostných algoritmoch, čo neposkytuje dostatočnú spoľahlivosť. Budeme sa teda zaoberať exaktnými deterministickými riešeniami. Navrhnuté značky môžeme ďalej použiť pri ľubovoľných formulároch, nielen testoch. Identifikujú nám plochu, ktorú chceme analyzovať a zistia, či bola celá plocha korektne naskenovaná. Rovnako nám zabezpečia zistenie orientácie papiera, keďže sa môže stať, že papier naskenujeme hore nohami. Neposledným faktorom pri výbere takéhoto softvéru je cena. Špecializovaný softvér na optické rozpoznávanie značiek stojí 995$.

1.3 Členenie práce

Práca je rozdelená do deviatich kapitol a ako príloha je CD so softvérom a ukážkovou sadou testov. Prvá kapitola je úvodná, stanovili sme si v nej ciele práce a motiváciu. V druhej kapitole ukážeme dva typy značiek podľa štruktúry, existujúce systémy na ich rozpoznávanie a algoritmy, ktoré sa pritom využívajú. Ďalšia kapitola obsahuje prehľad značiek kódujúcich nejakú informáciu, existujúce štandardy a projekty. Štvrtá kapitola prezentuje optické rozpoznávanie značiek (OMR), jeho súčasný stav a výhody i nevýhody, ktoré poskytuje. V piatej kapitole spravíme analýzu modelovej situácie a samotný návrh celého riešenia. Vyberieme najvhodnejšie značky podľa presne stanovených kritérií. Realizáciu problému tvorí šiesta kapitola. Vyberieme si v nej vhodné technológie a prezentujeme postup tvorby aplikácie a jej následné otestovanie na reálnych dátach. Posledné časti práce tvoria zovšeobecnenie počtu značiek, záver a zoznam použitej literatúry.

2 Značky podľa štruktúry

Podľa štruktúry rozlišujeme vo všeobecnosti dva typy značiek:

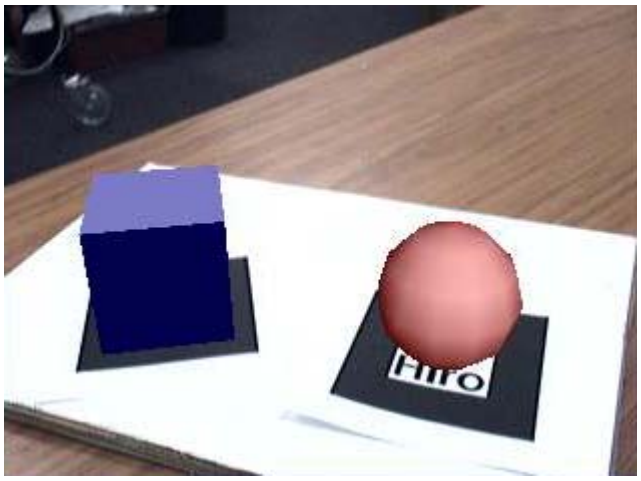
1. **geometrické značky** - dodržujú presný tvar, prípadne farbu

Obr. 2.1: Geometrická značka

2. **topologické značky** - dodržujú topológiu vrstiev, ktoré v sebe obsahujú. Tá istá topologická značka môže mať viac geometrických prevedení.

Obr. 2.2: Dve geometrické prevedenia topologickej značky

Pozrime sa na existujúce systémy rozpoznávania týchto značiek, ktoré sú využívané hlavne pri modelovaní rozšírenej reality (augmented reality). Rozšírená realita je prekrytie reálneho sveta virtuálnom počítačovou grafikou a má značné využitie v odvetviach a vedeckom výskume. Kamera sníma značky a každá snímka sa následne analyzuje. Rozpoznajú sa značky a v závislosti od konkrétneho systému sa vymodeluje napríklad 3D objekt na danej značke alebo začne hrať nejaký zvuk.



Obr. 2.3: Augmented reality

Väčšina systémov pre real-time analýzu obrázkov je založená na geometrických vlastnostiach alebo rozpoznávaní farby. Aké sú teda výhody a nevýhody topologických značiek? Značky sú reprezentované ako čiernobiele vzorky a ich rozpoznávanie je založené na prislúchajúcom grafe susedných regiónov. Boli navrhnuté predovšetkým pre dávkové (off-line) aplikácie, keďže pri výpočtoch vo všeobecnosti narážajú na problém podgrafového izomorfizmu. Ten pozostáva v nájdení a mapovaní medzi všetkými uzlami cieľového grafu reprezentujúceho hľadaný objekt a podmnožiny veľkého grafu reprezentujúceho scénu. Podgrafový izomorfizmus je NP-úplný problém a preto je jeho zložitosť exponenciálna vzhľadom k veľkosti dvoch grafov. Vhodnými obmedzeniami štruktúry grafu však možno dosiahnuť oveľa lepšiu časovú zložitosť.

Ďalší problém pre topologický prístup vzniká, keď sú značky prekrížené. Vtedy sa regióny spájajú a značne menia topologickú štruktúru. Na druhej strane, topologický prístup smeruje k slobode pri návrhu a toleruje deformáciu tak dlho, ako je dodržaná topológia. Táto trieda deformácií pokrýva širší záber než tie, ktoré sú dovolené pri algoritmoch založených na geometrických vlastnostiach, ako napríklad ohýbanie. Tým pádom môžu byť značky kreslené aj rukou alebo pripevnené na šatách. Navyše, ak je geometria značiek transparentná, môže byť použitá na zakódovanie extra informácie.

2.1 Algoritmy využívané pri rozpoznávaní značiek

2.1.1 Thresholding

Prahovanie (thresholding) je najjednoduchšia metóda segmentácie obrázku. Zo šedého (grayscale) obrázku môže byť prahovanie použité na vytvorenie binárneho obrázku Tento proces nazývame aj binarizácia. Prahovacie metódy predpokladajú, že objekty v obraze sa dajú odlíšiť od pozadia na základe jasovej hodnoty jednotlivých obrazových bodov.

Obr. 2.4: Binarizácia

Jednoduchý prah

Najprv sa zvolí prah T a podľa tejto prahovej hodnoty sa rozdelia obrazové body na body objektu a body pozadia.

Prahovanie môže byť viacerých typov:

1. globálne prahovanie $$g(x,y) = \begin{cases} f(x,y) & \text{ak } f(x,y) > T \\ 0 & \text{inak} \end{cases}$$
2. poloprahovanie $$g(x,y) = \begin{cases} 255 & \text{ak } f(x,y) > T \\ 0 & \text{ak } f(x,y) \leq T \end{cases}$$
3. spektrálne prahovanie $$g(x,y) = \begin{cases} f(x,y) & \text{ak } f(x,y) \in I \\ 0 & \text{inak} \end{cases}$$
4. multispektrálne prahovanie $$g(x,y) = \begin{cases} a_1 & \text{ak } f(x,y) \in I_1 \\ .. & \dots \\ a_n & \text{ak } f(x,y) \in I_n \\ 0 & \text{inak} \end{cases}$$

pričom I_1, ..., I_n sú disjunktné intervaly jasových hodnôt a a_1, ..., a_n sú rôzne jasové úrovne.

Adaptívny prah

V praxi sa môže stať, že obraz ktorý dostaneme na vstupe obsahuje objekty, ktoré sú zle osvetlené alebo vrhajú tieň. Tieto dôvody môžu byť príčinou, kedy jednoduché prahovacie metódy zlyhajú. Riešením situácie je adaptívne prahovanie. Pri adaptívnom prahovaní sa prahová hodnota vypočítava zvlášť pre každý bod obrazu.

Existujú dve základné techniky na výpočet prahovej hodnoty:

1. **Chow a Kanenho metóda** - Chow a Kanen boli medzi prvými výskumníkmi, ktorý navrhli adaptívny prah. Obraz rozdelíme na množinu menších častí. Skúmaním histogramu pre každú obrazovú časť zistíme príslušný optimálny prah. Prahovú hodnotu pre konkrétny bod vypočítame interpoláciou prahových

2. **Lokálne prahovanie** - pre každý bod si zvolíme malé okolie, pomocou ktorého vypočítame prahovú hodnotu. Na výpočet prahovej hodnoty máme viac možností:
    - priemer jasových hodnôt v okolí
    - medián jasových hodnôt v okolí
    - stredná hodnota minimálnej a maximálnej jasovej hodnoty v okolí

Metódy založené na lokálnom prahovaní, ktoré poskytujú najlepší výkon sú [1]:

Niblackova metóda - myšlienka je založená na lokálnom priemere a lokálnej štandardnej deviácii. Prah pixlu (x, y) je vypočítaný ako:

$$T(x, y) = m(x, y) + k * s(x, y)$$

kde m(x, y) a s(x, y) sú priemer jasových hodnôt a štandardné hodnoty deviácie porade v lokálnom okne centrovanom v (x, y). Veľkosť okna by mala byť dosť malá na to, aby odrazila lokálny stupeň osvetlenia a dosť veľká na to, aby zahrnula aj objekt aj pozadie.
Doporučené hodnoty: lokálne okno veľkosti 15 x 15 a koeficient k = -0,2.

Eikvil–Taxt–Moenova metóda - pixle vnútri malého okna S sú prahované na základe zoskupenia pixlov vo väčšom okne L. Lokálne okná L a S sú posúvané cez obrázok v krokoch veľkosti okna S. Všetky pixle v okne L sú rozdelené pomocou Otsuovho prahu T do dvoch tried. Ak platí:

$$\left|\hat{s}_1 - \hat{s}_2\right| \geq l$$

kde $\hat{s}_1$ a $\hat{s}_2$ sú priemery jasových hodnôt jednotlivých tried a l je konštanta, potom pixle vnútri okna S sú binarizované použitím prahu T. Inak sú všetky pixle vnútri okna S prepísané do triedy s najbližšou priemernou jasovou hodnotou.
Doporučené hodnoty: S = 3 x 3, L = 15 x 15 a l = 15.

Bernsenova metóda - prah pixlu (x, y) je vypočítaný ako:

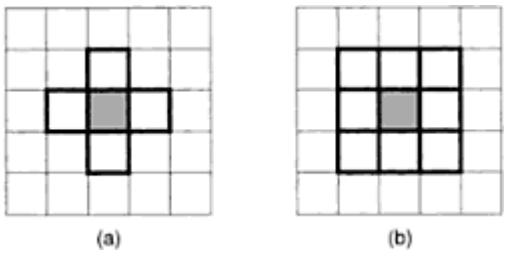
$$T(x, y) = (Z_{low} + Z_{high}) / 2$$

kde Z_{low} a Z_{high} sú najnižšia a najvyššia jasová hodnota pixlu v lokálnom okne centrovanom v (x, y). Ak je miera kontrastu $C(x,y) = Z_{low} - Z_{high} < l$, potom je celé lokálne okno považované za pozadie.

Doporučené hodnoty: lokálne okno veľkosti 15 x 15 a koeficient l = 15.

2.1.2 Connected component labeling

Označovanie súvislých komponentov (connected component labeling) [2] je segmentácia binárneho obrázku na korešpondujúce súvislé komponenty. Algoritmus pracuje s množinou pixlov, ktoré sú biele. Táto množina je rozdelená do disjunktných podmnožín. Rozdelenie je reprezentované výsledným obrázkom, v ktorom všetky biele pixle, ktoré ležia v rovnakom súvislom komponente, majú rovnakú hodnotu pixlu. Rôzne hodnoty pixlov sú priradené rôznym súvislým komponentom. Pre označovanie súvislých komponentov môže byť použitá 4-spojitosť alebo 8-spojitosť. Pri 4-spojitosti sú susedné pixle len vo vodorovnom a zvislom smere, pri 8-spojitosti aj v diagonálnom smere.



Obr. 2.5: a) 4-spojitosť, b) 8-spojitosť

Na rozlíšenie súvislých komponentov sú väčšinou použité rôzne farby alebo odtiene sivej. Preto sa označovanie komponentov uvádza aj ako farbenie kúskov.

Obr. 2.6: Farbenie súvislých komponentov

Označovanie súvislých komponentov sa dá implementovať dvoma spôsobmi. Budeme pritom uvažovať 8-spojitosť.

1. postupný algoritmus (row-by-row)

Jednoduchá implementácia sa dá spraviť dvomi prechodmi s využitím union a find algoritmov, ktoré umožňujú dynamickú konštrukciu a manipuláciu tried ekvivalencií reprezentujúcich stromové štruktúry.

Pseudokód:

```
pre každý riadok obrázka
   pre každý stĺpec obrázka
      ak je aktuálny pixel biely
         pokiaľ sa dá, skontroluj postupne ľavého, ľavého horného,
         horného a pravého horného suseda
         ak nie sú žiadny susedia biely
            označ aktuálny pixel novým číslom
         inak
            nájdi suseda s najmenším číslom a priraď jeho hodnotu
            aktuálnemu pixlu
            uchovaj si ekvivalenciu medzi susedmi (union)
pre každý riadok obrázka
   pre každý stĺpec obrázka
      ak je aktuálny pixel biely
         preznač tento pixel najmenším číslom ekvivalentného suseda
         (find)
```

2. rekurzívny algoritmus

Pseudokód:

```
pre každý riadok obrázka
   pre každý stĺpec obrázka
      ak je aktuálny pixel biely a neoznačený
         označ aktuálny pixel novým číslom
         rekurzívne skontroluj všetkých jeho susedov a priraď im
         rovnakú hodnotu, ak sú neoznačené a biele
```

2.1.3 Template matching

Template matching je technika digitálneho spracovania obrázku pre nájdenie malých častí obrázka, ktoré sa zhodujú so vzorkovým obrázkom (template). Existuje viacero

prístupov pre nájdenie vzorky. Základná metóda používa ako mieru zhody sumu absolútnych diferencií (SAD) intenzít pixlov. Výstup bude najmenší na miestach, kde sa štruktúra obrázka zhoduje najviac so štruktúrou vzorky.

Definícia: Suma absolútnych diferencií prehľadávaného obrázku a vzorky priloženej tak, že súradnice jej ľavého horného rohu sú zhodné so súradnicami x, y v prehľadávanom obrázku je:

$$SAD(x,y) = \sum_{i=0}^{T\_rows} \sum_{j=0}^{T\_cols} \left| S\_value(x+i, y+j) - T\_value(i,j) \right|$$

kde T_rows, T_cols sú riadky a stĺpce vzorkového obrázku, S_value (x, y) je intenzita pixla so súradnicami (x, y) v prehľadávanom obrázku a T_value (x, y) je intenzita pixla so súradnicami (x, y) vo vzorkovom obrázku.

Algoritmus (pre obrázky v šedej farebnej škále): [19]

```
// inicializácia prahu
int minSAD = VALUE_MAX;
Position position = new Position();

// cyklus cez prehľadávaný obrázok
for ( int x = 0; x <= S_rows - T_rows; x++ ) {
   for ( int y = 0; y <= S_cols - T_cols; y++ ) {
      int SAD = 0;

      // cyklus cez vzorkový obrázok
      for ( int i = 0; i < T_rows; i++ )
         for ( int j = 0; j < T_cols; j++ ) {
            pixel p_SearchIMG = S[x+i][y+j];
            pixel p_TemplateIMG = T[i][j];

            // výpočet sumy absolútnych diferencií
            SAD += abs( p_SearchIMG.Grey - p_TemplateIMG.Grey );
         }

      if ( minSAD > SAD ) {
         minSAD = SAD;
         // uloženie súradníc a SAD najlepšej nájdenej pozície
         position.bestRow = x;
         position.bestCol = y;
         position.bestSAD = SAD;
      }
   }
}
```

Template matching pre farebné obrázky je založený na rozklade pixlov do ich farebných zložiek a takisto meraní sumy absolútnych diferencií. Zlepšenie výkonu

template matchingu môže byť dosiahnuté použitím viac ako jednej vzorky. Tieto ďalšie vzorky môžu mať rôznu škálovateľnosť a rotáciu.

Časová zložitosť template matchingu je veľká. Ak je vzorka štvorec veľkosti m x m a prehľadávaný obrázok veľkosti N x N, potom je časová zložitosť $O(N^2m^2)$. Toto je cena za invariant pozície. Každé ďalšie parametre záujmu zväčšujú časovú zložitosť priamo úmerne k počtu hodnôt extra parametrov [3].

Pokročilejšou metódou je Scale-invariant feature transform (SIFT). SIFT [4] je algoritmus počítačového videnia na detekciu a opis lokálnych čŕt obrázku. Tento prístup transformuje obrázok na veľkú množinu lokálnych črtových vektorov, pričom každý z nich je invariant na transláciu, škálovateľnosť, rotáciu obrázku a čiastočný invariant na zmenu v osvetlení a afinné alebo 3D projekcie. Predošle prístupy postrádali predovšetkým invariant škálovateľnosti a boli viac citlivé na projektívne deformácie a zmenu osvetlenia. Črty vzorky sú efektívne detekované pomocou fázového filtrovania, ktoré identifikuje stabilné body v škálovateľnom priestore . Kľúčové lokality sú použité ako vstup pre metódu indexovania najbližších susedov, ktorá identifikuje kandidátov na zhodu v prehľadávanom obrázku. Finálna verifikácia je vykonaná hľadaním najmenej odlišných a najmenších štvorcov ako neznámych parametrov modelu.

Obr. 2.7: Vzorky

(a) (b)

Obr. 2.8: a) Prehľadávaný obrázok b) Nájdené obrysy vzoriek a kľúčové lokality

2.1.4 Erózia a dilatácia

Erózia (erosion) a dilatácia (dilation) sú dve základné morfologické operácie pri spracovaní obrázkov. Všetky ostatné operácie sú vyjadrené pomocou ich kombinácie [5].

Definícia: Erózia množiny X podľa štruktúrovacieho elementu B je označená $\varepsilon_B(X)$ a definovaná ako množina bodov x takých, že B je podmnožina X, pričom stred B je umiestnený v bode x:

$$\varepsilon_B(X) = \{\boldsymbol{x} \mid B_{\boldsymbol{x}} \subseteq X\}$$

Rovnica môže byť prepísaná podľa pravidiel prieniku translácií množín. Translácie budú určené štrukturálnym elementom:

$$\varepsilon_B(X) = \bigcap_{\boldsymbol{b} \in B} X_{-\boldsymbol{b}}$$

Táto nová formulácia môže byť priamo rozšírená na binárne a šedé (grayscale) obrázky.

Definícia: Erózia obrázku f podľa štruktúrovacieho elementu B je označená $\varepsilon_B(X)$ a definovaná ako minimum translácií f podľa vektorov -b z B:

$$\varepsilon_B(f) = \bigwedge_{\boldsymbol{b} \in B} f_{-\boldsymbol{b}}$$

A preto, erodovaná hodnota daného pixlu x je minimálna hodnota obrázku v okne definovanom štrukturálnym elementom, pričom jeho stred je v x:

$$[\varepsilon_B(f)](\boldsymbol{x}) = \min_{\boldsymbol{b} \in B} f(\boldsymbol{x} + \boldsymbol{b})$$

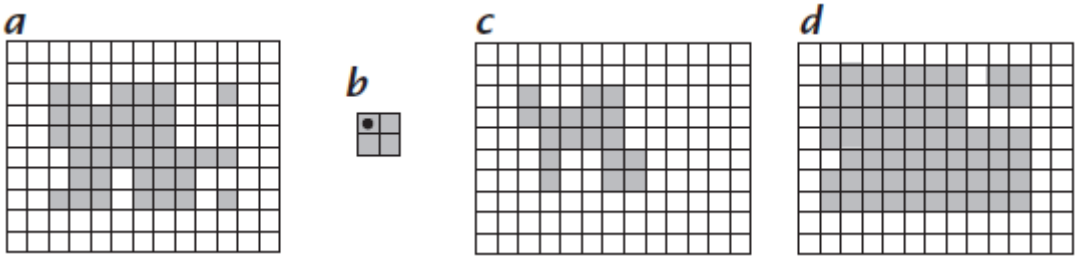
Obdobne môžeme zadefinovať dilatáciu, keďže je to opačná operácia k erózii.

Definícia: Dilatácia obrázku f podľa štruktúrovacieho elementu B je označená $\delta_B(X)$ a definovaná ako maximum translácií f podľa vektorov -b z B.

$$\delta_B(f) = \bigvee_{\boldsymbol{b} \in B} f_{-\boldsymbol{b}}$$

Inými slovami, dilatovaná hodnota daného pixlu x je maximálna hodnota obrázku v okne definovanom štrukturálnym elementom, pričom jeho stred je v x:

$$[\delta_B(f)](\boldsymbol{x}) = \max_{\boldsymbol{b} \in B} f(\boldsymbol{x} + \boldsymbol{b})$$



Obr. 2.9: Príklad erózie a dilatácie

a) binárny obrázok X

b) 2 x 2 štrukturálny element B so stredom v ľavom hornom rohu

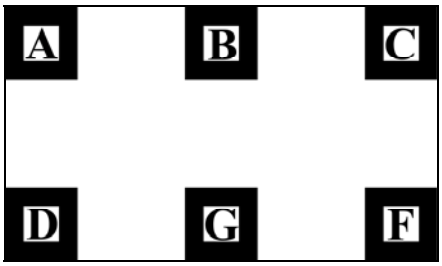
c) erózia X podľa B

d) dilatácia X podľa B

2.2 Geometrické značky

2.2.1 ARToolkit

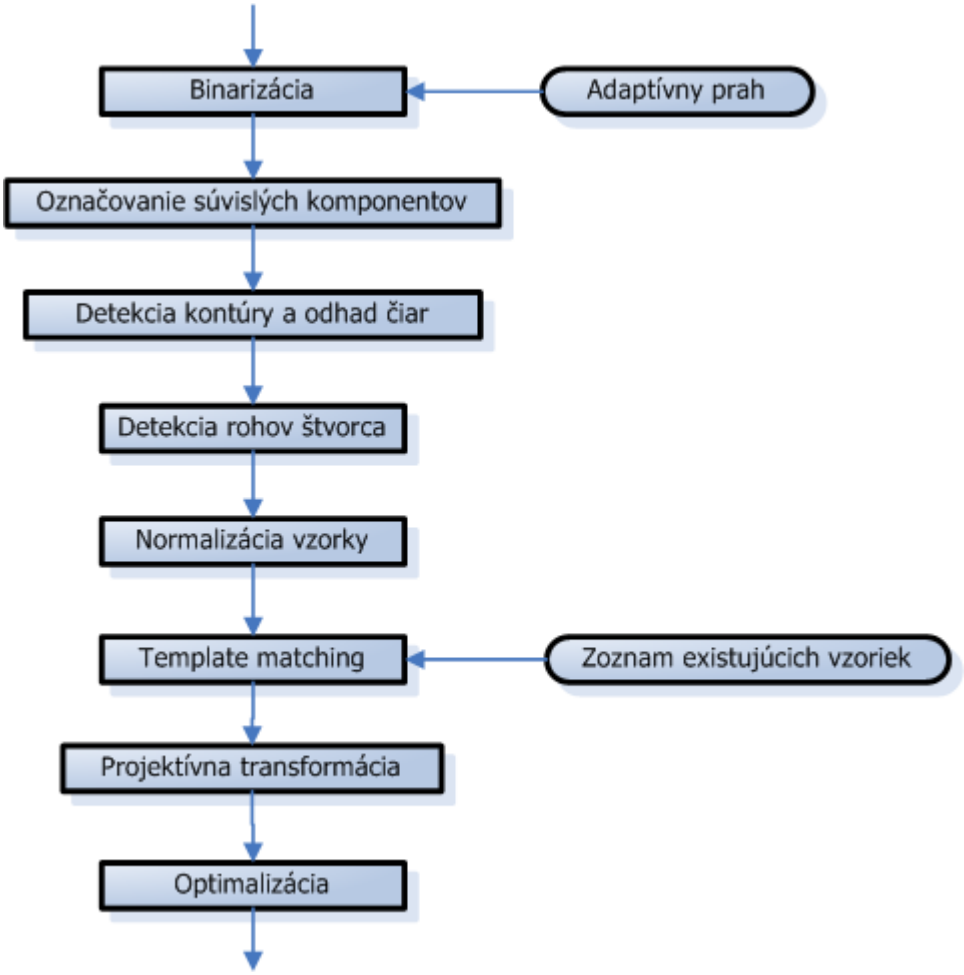
ARToolkit [6, 7, 20] je C a C++ softvérová knižnica, ktorá pomáha vývojárom ľahko vytvárať aplikácie s rozšírenou realitou (augmented reality). ARToolkit je založený na geometrických značkách v čiernom štvorci.



Obr. 2.10: ARToolkit Shared White Board

Princíp ARToolkitu:

1. kamera sníma video reálneho sveta a posiela ho počítaču
2. každý rámec sa následne analyzuje a hľadajú sa čierne štvorce
3. ak je štvorec nájdený, vypočíta sa pozícia a orientácia štvorca vzhľadom ku kamere
4. identifikuje sa značka vo vnútri štvorca
5. na pozícii štvorca je následne nakreslený počítačový model
6. finálny výstup s 3D objektom je zobrazený naspäť používateľovi



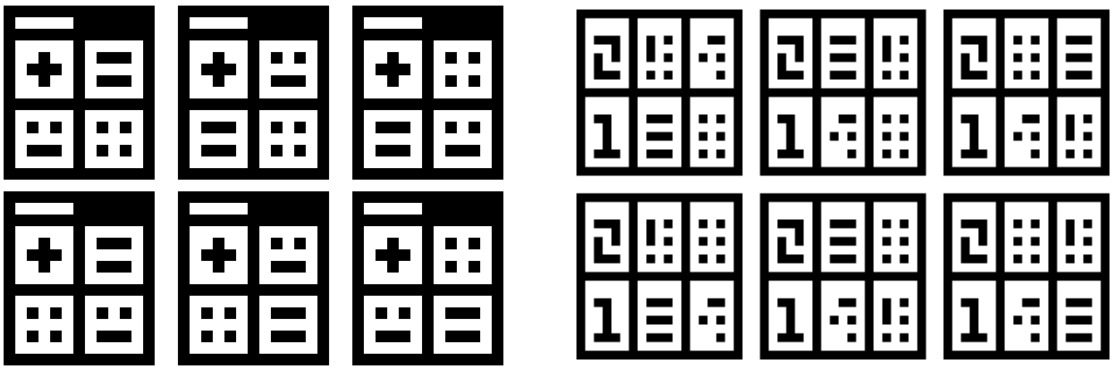
Obr. 2.11: ARToolkit algoritmus

2.3 Topologické značky

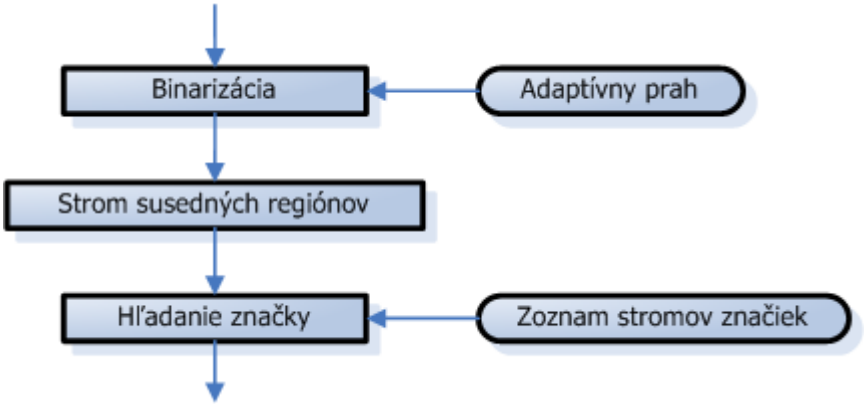
2.3.1 D-touch

D-touch [8, 9, 10] je softvérový framework pre budovanie nenákladných dotykových používateľských rozhraní a rozšírených systémov reality. Fyzické objekty sú označené topologickými značkami, ktoré ich robia rozpoznateľnými napr. počítačom vybaveným web kamerou. Rozpoznávanie funguje v reálnom čase a pri štandardnom osvetlení.

Abeceda značiek môže byť definovaná tak, že všetky symboly majú rovnakú topologickú štruktúru, ale rôznu geometriu. Týmto spôsobom môžu byť všetky symboly abecedy rozpoznané a lokalizované jedným prechodom algoritmu. Následne môžu byť lokálne použité jednoduchšie spracovávacie metódy na rozlíšenie medzi rôznymi symbolmi.



Obr. 2.12: Ukážky prvých 6 značiek abecied z 24 a 120 značiek



Obr. 2.13: D-Touch algoritmus

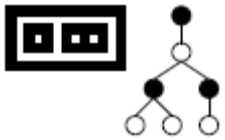
Popis algoritmu:

1. Binarizácia pomocou adaptívneho prahu

Na vstupnom obrázku budeme uvažovať lokálne okno a na ňom vypočítame prah. Vhodná je napríklad Bernsenova alebo Niblackova metóda (viď. kapitola 2.1.1).

2. Strom susedných regiónov

Jadro algoritmu je založené na štruktúre susedností súvislých regiónov obrázku. Pomocou rekurzívnej procedúry extrahujeme z obrázka súvislé komponenty a zároveň si vytvárame susednosť regiónov vo forme neorientovaného grafu G(V, E). Každý vrchol grafu $v \in V$ reprezentuje súvislý komponent obrázka. Dva vrcholy sú spojené hranou $e \in E$ práve vtedy, keď sú susedné. Počet hrán začínajúcich alebo končiacich vo vrchole nazývame stupňom vrchola. Bolo dokázané, že pre binárne obrázky je graf susedných regiónov bipartitný strom.



Obr. 2.14: Graf susedných regiónov (region adjacency graph)

3. Problém podstromového izomorfizmu

Po vytvorení grafu susedných regiónov nastáva hľadanie zhody medzi stromom topologickej značky a jednej alebo viacerých podmnožín stromu scény dodržujúc susednosť. Toto je klasický problém, ktorý môže byť vyriešený algoritmom prehľadávania grafu. Existuje algoritmus, ktorého časová zložitosť je $O\left(n * m^{1.5} / \log m\right)$, pričom n je počet vrcholov stromu scény a m je počet vrcholov stromu topologickej značky [11].

Pre zjednodušenie podstromového izomorfizmu bolo navrhnuté obmedzenie štruktúry značky tak, aby maximálnej hĺbka stromu susedných regiónov bola 3. Jednotlivé úrovne pomenujeme ako koreň, vetvy a listy. Jedna značka s n_b vrcholmi vo vetvovej úrovni môže byť reprezentovaná postupnosťou $\{a_k\}$ zloženej z (n_b+1) čísel. Jedno číslo bude určovať počet vetiev a ostatné čísla počet listov v každej vetve - označme si ich ako množinu $\{l_k\}$. S touto reprezentáciou stačí vykonať traverzovanie stromu scény iba raz. Každý vrchol n^* stromu scény so stupňom (n_b+1) je považovaný za kandidáta na koreň. Množina $\{l'_k\}$ zložená z (n_b+1) čísel bude skonštruovaná so stupňom každého vrchola spojeného s n^*. Ak je mohutnosť prieniku množín $\{l_k\}$ a $\{l'_k\}$ rovná n_b, potom je vrchol n^* akceptovaný ako koreň.

2.3.2 ReacTIVision

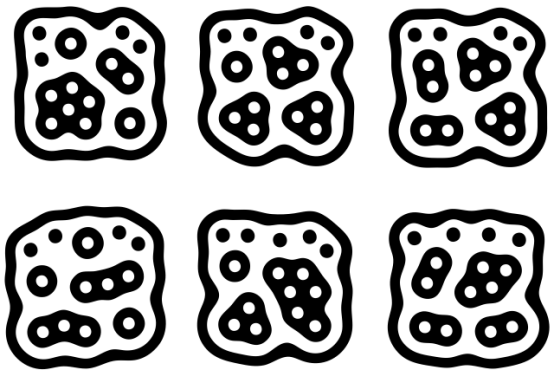
ReacTIVision [12, 13] je open source systém pre monitorovanie pozície a orientácie značiek v real-time videu. Systém bol vyvinutý pre interaktívny stôl s dotykovým používateľským rozhraním a je vylepšením D-touch systému. Rozhodnutie vyvinúť nový monitorovací systém vychádzalo z relatívne nízkej rámcovej frekvencie dosiahnutej pri D-Touchi. Cieľom bolo podporiť snímkové frekvencie, ktoré prekračovali kapacity 640x480 60Hz obrázkového snímača, zatiaľ čo by bola zachovaná robustnosť a presnosť D-touchu. Neskôr sa pridali ďalšie požiadavky ako redukcia veľkosti značiek a zvýšenie počtu jedinečne identifikovateľných značiek.

Obr. 2.15: Interaktívny stôl ReacTable

Bolo identifikovaných viacero aspektov D-touch prístupu, ktoré ponúkajú možnosti pre zlepšenie:

1. D-touch spája geometrickú extrakciu permutačného kódu špecifickej abecedy značiek.
2. D-touch neponúka výpočet polohy a orientácie, odhliadnuc od tradičných metód počítačového videnia.
3. Jednoduchá geometria D-touch značiek nebola navrhnutá na minimalizovanie veľkosti značiek.

Z týchto požiadaviek vznikol návrh nových topologických značiek, ktoré sú zároveň menšie a ponúkajú škálovacie techniky. Umožňujú meniť veľkosť značiek v závislosti od celkového počtu potrebných unikátnych značiek.

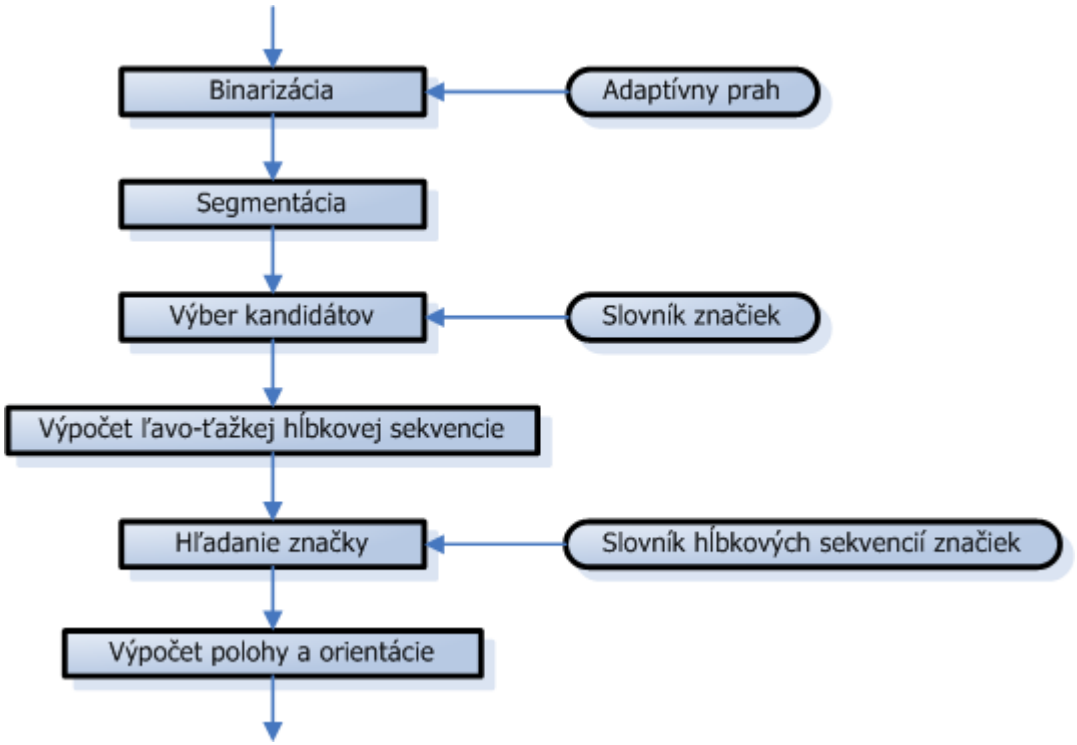


Obr. 2.16: ReacTIVision značky Amoeba (celkovo 90 unikátnych značiek) [12]

ReacTIVision značky sú teda identifikovateľné jedine podľa ich topologickej štruktúry. Každá značka v množine má jedinečnú topológiu. Otázka nastáva ako generovať jednotlivé značky. Pred generovaním samotných značiek si vygenerujeme množinu unikátnych stromov. Ak si zvolíme veľkosť množiny, potom je možné vypočítať počet vrcholov potrebných k naplneniu množiny. Avšak aj iné podmienky sú dôležité, ako napríklad zaistenie určitého počtu čiernych a bielych listov.

Počet vrcholov v strome a jeho hĺbka určuje minimálnu veľkosť geometrie, ktorá môže byť generovaná pre strom. Menšie stromy sú menej unikátne a preto majú väčší potenciál pre výskyt falošných detekcií v scéne. A preto, radšej ako by sme mali vymenovať všetky možné stromy, vygenerujeme náhodne stromy s požadovaným počtom vrcholov a vyberieme tie, ktoré spĺňajú kritéria maximálnej hĺbky a počtu čiernych a bielych listov.

Presná geometria značiek Amoeba bola dosiahnutá genetickým algoritmom, ktorý optimalizoval parametre ako tvar, veľkosť pôdorysu, ťažisko a rotačný uhol. Súčasná sada distribuovaná s ReacTIVision systémom obsahuje 90 unikátnych značiek, pričom každá z nich obsahuje 19 listov a maximálnu hĺbku stromu 2.



Obr. 2.17: ReacTIVision algoritmus

Popis algoritmu:

1. Binarizácia pomocou adaptívneho prahu

Vhodná je napríklad Bernsenova metóda. Plochy s nízkym dynamickým rozsahom sú vyhodnotené na čierno.

2. Segmentácia

Segmentácia vytvorí nekompletný graf susedných regiónov. Hlavná dátová štruktúra Region reprezentuje jeden vrchol v grafe.

```
struct Region{
      int flags;
      short left, top, right, bottom;
      ...
      int adjacent_regions_count;
      Region *adjacent_regions[ MAX_ADJACENT ];
};
```

Štruktúra obsahuje príznaky regiónu, súradnice rohov ohraničujúceho obdĺžnika alebo štvorca, počet susedných regiónov a pole s referenciami na tieto regióny. Algoritmus ide riadok po riadku a rozširuje existujúce regióny z predošlého riadku alebo vytvára nové regióny. Susedné regióny sú pridávané do spájaného zoznamu. Niektoré regióny potrebujú byť zjednotené. Vtedy sa upraví aj referencia zjednoteného regiónu v spájanom zozname.

Obr. 2.18: Príklad zjednotenia regiónov

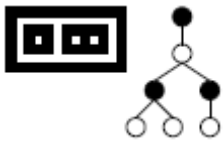
Regióny sú označené ako vyplnené, keď je ich pole susedností plné, inak sú fragmentované. Segmentáciou teda dostaneme nekompletný graf susedných regiónov, pričom podgrafy neobsahujúce vyplnené alebo fragmentované uzly sú kompletné.

3. Výber kandidátov na značky

Výber kandidátov prebieha podľa vlastností vypočítaných na slovníku existujúcich značiek. Za kandidáta je považovaný každý graf susedných regiónov, ktorý má celkový počet vrcholov a maximálnu hĺbku v intervale príslušných hodnôt existujúcich značiek.

4. Výpočet ľavo-ťažkej hĺbkovej sekvencie

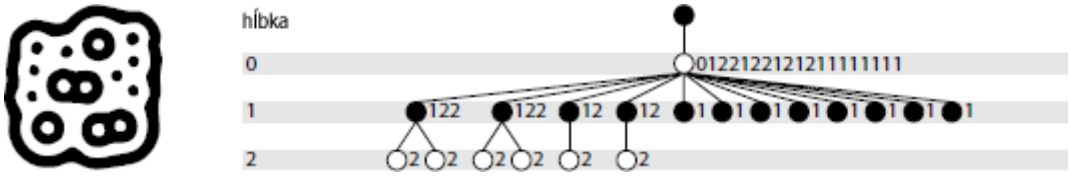
Hĺbková sekvencia grafu je definovaná ako sekvencia hĺbok vrcholov pomocou preorder prechodu (spracujeme vrchol, ľavého syna, pravého syna). Koreň má hĺbku 0 a hĺbka každého ďalšieho vrchola je počet hrán medzi ním a koreňom.



Prípustné hĺbkové sekvencie sú:

- 0, 1, 2, 3, 3, 2, 3
- 0, 1, 2, 3, 2, 3, 3

Na odstránenie nejednoznačnosti môžeme vykonať traverzovanie podľa váhy hĺbkovej sekvencie každého syna, kde ťažšie sekvencie sú tie, ktoré sú lexikálne väčšie. Ľavo-ťažká hĺbková sekvencia môže byť teda definovaná rekurzívne ako zreťazenie ľavo-ťažkých hĺbkových sekvencií detí vrcholov v lexikálne klesajúcom poradí, kde hĺbková sekvencia listu je jednoducho jeho hĺbka.



Obr. 2.19: Ľavo-ťažká hĺbková sekvencia

Najvrchnejší čierny región (najvrchnejší vrchol grafu) slúži len na poskytnutie obalu pre biely región, ktorý obsahuje.

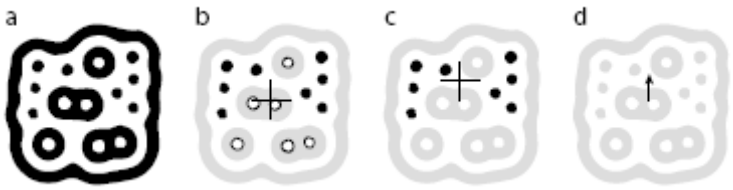
5. Hľadanie sekvencie v slovníku hĺbkových sekvencií existujúcich značiek

Vyhľadanie ľavo-ťažkej hĺbkovej sekvencie kandidáta v slovníku existujúcich značiek spočíva len vo vyhľadaní číselnej sekvencie kandidáta v poli čísel existujúcich značiek. Ak bola sekvencia nájdená pokračujeme výpočtom polohy a orientácie.

6. Výpočet polohy a orientácie

Metóda pre počítanie polohy a orientácie bola ovplyvnená návrhom segmentovacieho algoritmu, ktorý vracia pre každý región len jeho ohraničujúci obdĺžnik. Stred ohraničujúceho obdĺžnika poskytuje dobrú aproximáciu pre stred regiónu len v prípade, že región je štvorec, obdĺžnik alebo relatívne malý. Je teda vhodné, ak budú listové regióny vždy tie najmenšie regióny. Preto boli pri výpočte polohy a orientácie použité práve stredy ohraničujúcich obdĺžnikov listov.

Najprv vypočítame ťažisko značky ako vážený priemer stredov čiernych a bielych listov. Vektor z ťažiska značky do bodu váženého priemeru stredov čiernych listov (rovnako sme mohli použiť aj biele listy) nám určí orientáciu značky. Každý list je ohodnotený váhou pomocou funkcie jeho hĺbky v strome na vyjadrenie celkovej plochy, ktorú zaberá. V praxi sa ukázalo, že je nutné minimum 4 čiernych a 4 bielych listov na dosiahnutie stability, aká bola dosiahnutá s predošlým D-touch systémom.



Obr. 2.20: Výpočet polohy a orientácie značky [12]

Časová zložitosť:

Nekompletný graf susedných regiónov vytvorený pomocou segmentácie znemožňuje traverzovanie do hĺbky. Keďže graf je neorientovaný, rodič každého vrchola musí byť odvodený počas traverzovania. Použijeme teda postupné traverzovanie zdola, v ktorom prechádzame cez každý list a skúšame traverzovať nahor. Počítadlo v každom rodičovi je použité na zaistenie, že postupujeme nahor iba raz, keď už boli všetky deti rodiča navštívené. Traverzovanie nahor je zastavené, ak je vrchol fragmentovaný alebo vyplnený, alebo ak je prekročený počet potomkov.

Všimnime si, že v najhoršom prípade je každý vrchol navštívený trikrát - raz počas iterácie na hľadanie listov, druhýkrát počas traverzovania nahor na identifikovanie kandidátov podstromov a nakoniec počas rekurzívneho výpočtu ľavo-ťažkej hĺbkovej sekvencie. Časová zložitosť pre rozpoznávanie všetkých značiek je lineárna vzhľadom na počet vrcholov v grafe susedných regiónov.

3 Značky pre kódovanie informácií

Značky pre kódovanie informácií sú štandardne dvojrozmerné (2D) kódy, ktoré spadajú do dvoch kategórií:

1. **kumulované čiarové kódy** - niekoľko tradičných jednorozmerných (1D) čiarových kódov kumulovaných na seba.
2. **maticové kódy** - používajú dvojrozmernú maticu na zakódovanie informácie v horizontálnom aj vertikálnom smere pre zvýšenie kapacity úložiska. Sú to predovšetkým geometrické značky, ktoré používajú presný tvar, prípadne farbu na zakódovanie informácie.

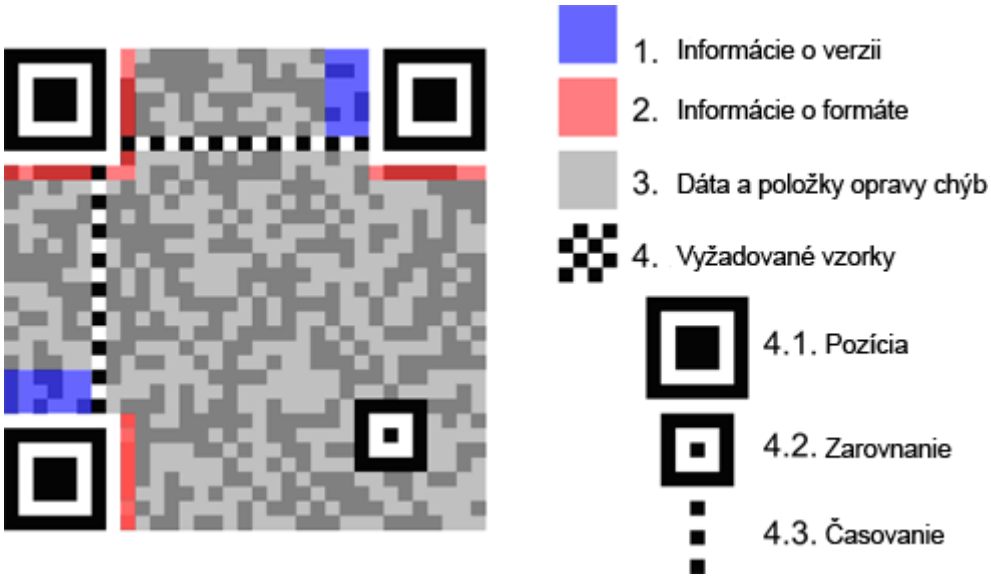
3.1 Štandardy

3.1.1 QR code

QR kód (QR code) [22] je maticový kód vytvorený japonskou korporáciou Denso-Wave v roku 1994. Skratka QR je odvodená od "Quick Response", čo naznačuje, že kód bol navrhnutý tak, aby bol jeho obsah rýchlo dekódovateľný. Je to otvorený formát a špecifikácia je dostupná priamo od jeho autora. QR kódy sú bežne používané v Japonsku, kde sú momentálne najpopulárnejším typom dvojrozmerných kódov. Rozmery kódov sú v rozmedzí od 21 x 21 do 177 x 177.

Vlastnosti:

- vysoká kapacita uloženia dát
    - 7089 numerických znakov
    - 4296 alfanumerických znakov
    - 2953 bajtov
    - 1817 znakov Kanji / Kana (čínske znaky / japonské slabičné skripty)
- Reed-Solomon korekcia chýb, maximálne 30% kódových slov môže byť obnovených
- malá tlačená veľkosť
- vysoká rýchlosť čítania
- možnosť rozdelenia kódu na viacero menších



Obr. 3.1: Špecifikácia QR kódu

3.1.2 Dátova matica

Dátová matica (data matrix) je maticový kód pozostávajúci z bielych a čiernych buniek usporiadaných v štvorcovej alebo obdĺžnikovej vzorke. Bola vytvorená spoločnosťou RVSI/Acuity CiMatrix, ktorú odkúpil Siemens AG a neskôr Microscan (2008). Dnes je pokrytá ISO štandardom a hoci je to bezplatný štandard, nie sú žiadne bezplatné dokumenty, ktoré vysvetľujú kódovací proces. Väčšinu matíc tvorí štvorec rozmerov od 8 x 8 do 144 x 144. Dáta sú kódované čiernymi bunkami.

Obr. 3.2: "Wikipedia, the free encyclopedia"

Vlastnosti:

- kapacita uloženia dát
  - 3116 numerických znakov
  - 2335 alfanumerických znakov
  - 1556 bajtov
  - 778 znakov Kanji / Kana
- Reed-Solomon korekcia chýb
- malá tlačená veľkosť

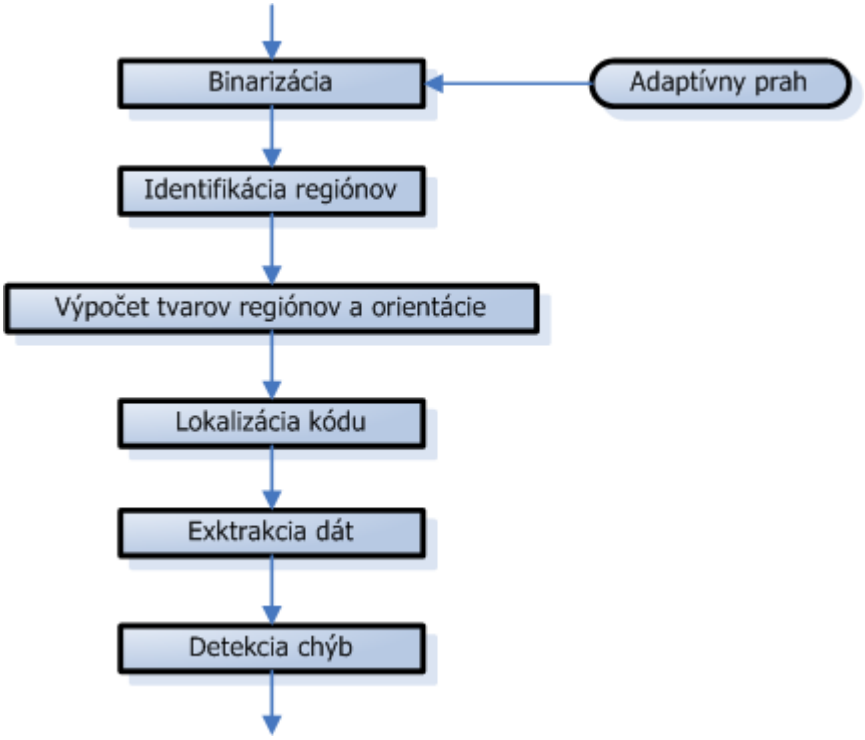
3.2 Projekty

Teraz, keď sme pokryli dva najčastejšie vyskytujúce sa maticové kódové štandardy, pozrime sa na konkrétne projekty. Niektoré používajú tieto dva štandardy, iné majú svoje vlastné. Niektoré sú komerčné, iné zase výskumné alebo oboje.

3.2.1 Visual codes

Vizuálne kódy (visual codes) [14] je výskumný projekt, ktorý používa vlastný maticový kódový formát navrhnutý a implementovaný Beatom Gfellerom a Michaelom Rohsom. Formát kódu zakóduje 83 bitov alebo 76 bitov s použitím Hammingovho kódu pre korekciu chýb.

Obr. 3.3: Špecifikácia vizuálneho kódu



Obr. 3.4: Visual codes algoritmus

Popis algoritmu:

1. Binarizácia pomocou adaptívneho prahu

2. Identifikácia regiónov

Môžeme použiť algoritmus označovania súvislých komponentov.

3. Výpočet tvarov regiónov a orientácie

Pre identifikovanie kandidátov na orientačné pásy použijeme výpočet druhých centrálnych momentov [17] pre každý región. Najjednoduchšia črta regiónu je jeho plocha. Označme si plochu regiónu R ako a, potom platí:

$$a = \|R\| = \sum_{(r,c)\in R} 1$$

kde r je riadok (row) a c je stĺpec (column).

Plocha je však len špeciálny prípad oveľa všeobecnejšej triedy čŕt, nazývaných momenty regiónu. Moment postupnosti $(p,q), p \geq 0, q \geq 0$, je definovaný ako:

$$m_{p,q} = \sum_{(r,c)\in R} r^p c^q$$

Treba poznamenať, že $m_{0,0}$ je plocha regiónu. Momenty záležia na veľkosti regiónov. Častokrát je nutné mať črty, ktoré sú invarianty k veľkostiam objektov. Na dosiahnutie takýchto čŕt môžeme jednoducho rozdeliť momenty podľa plochy regiónu ak $p+q \geq 1$ a získať normalizované momenty:

$$n_{p,q} = \frac{1}{a} \sum_{(r,c)\in R} r^p c^q$$

Najzaujímavejšia črta, ktorá môže byť odvodená z normalizovaných momentov, je ťažisko regiónu ($n_{1,0}$, $n_{0,1}$). Normalizované momenty závisia na pozícii v obrázku. Častokrát je užitočné robiť črty ako invarianty k pozícii regiónu v obrázku. Toto možno vykonať vypočítaním momentov relatívnym k ťažisku regiónu. Tieto centrálne momenty, pričom $p+q \geq 2$, sú dané:

$$\mu_{p,q} = \frac{1}{a} \sum_{(r,c)\in R} \left(r - n_{1,0}\right)^p \left(c - n_{0,1}\right)^q$$

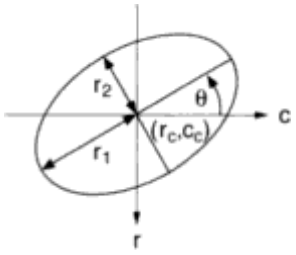
Zaujímavé sú najmä druhé centrálne momenty, keď $p+q=2$. Definujú nám orientáciu a rozlohu regiónu na základe predpokladu, že momenty postupnosti 1 a 2 regiónu boli získané z elipsy. Z týchto momentov následne vypočítame hlavnú a vedľajšiu os a uhol natočenia.

$$r_1 = \sqrt{2\left(\mu_{2,0} + \mu_{0,2} + \sqrt{\left(\mu_{2,0} - \mu_{0,2}\right)^2 + 4\mu_{1,1}^2}\right)}$$

$$r_2 = \sqrt{2\left(\mu_{2,0} + \mu_{0,2} - \sqrt{\left(\mu_{2,0} - \mu_{0,2}\right)^2 + 4\mu_{1,1}^2}\right)}$$

$$\theta = -\frac{1}{2}\arctan\frac{2\mu_{1,1}}{\mu_{0,2} - \mu_{2,0}}$$

Pomer dĺžok r_1 / r_2 je dobrou mierou pre excentricitu regiónu.



Obr. 3.5: Geometrické parametre elipsy

Parametre elipsy sú prahované na identifikáciu správnych regiónov korešpondujúcich orientačných pásov a kandidátov na rohy.

4. Lokalizácia kódu

Po identifikácii orientačných pásov pokračujeme hľadaním prislúchajúcich troch rohov. Pre každý z nich sa vypočítajú očakávané pozície a následne sa skontroluje, či sú regióny na týchto pozíciách.

5. Extrakcia dát

Na mapovanie medzi kódom a súradnicami obrázku použijeme projektívnu transformačnú maticu. Získame súradnicový systém so stredom v ľavom hornom rohu kódu a s rozlíšením jedného bitového elementu.

6. Detekcia chýb

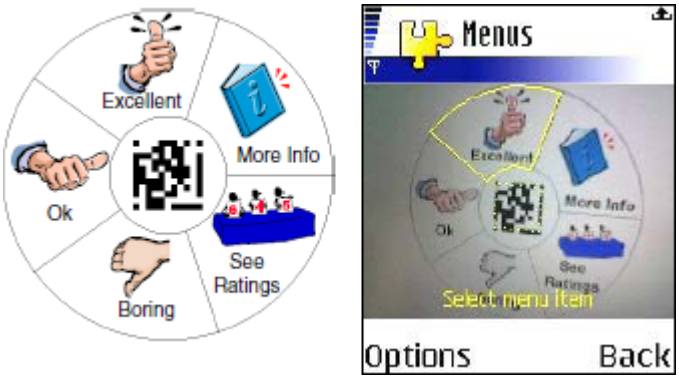
Jednotlivé bity sú chránené pomocou (83, 76, 3) lineárneho kódu s Hammingovou vzdialenosťou 3, ktorá zakóduje 76 dátových bitov na 83-bitové kódové slovo.

Vizuálne kódy môžu byť rozpoznané zariadeniami, ktoré sú vybavené fotoaparátom, ako PDA alebo mobilný telefón. Rozpoznávanie zahŕňa orientáciu zariadenia vzhľadom ku kódu. Vzhľadom k tomuto faktu je možné definovať ovládacie prvky vybavené vizuálnymi kódmi (visual code widgets) [15]. Pre hladký priebeh

interakcie sú visual code widgets rozpoznané v hľadáčikovom móde a aktualizované v reálnom čase podľa toho, ako sa hýbe zariadenie. Kvôli urýchleniu interakcie sú typ a rozvrhnutie widgetu uložené priamo v kóde.

Existuje viacero typov widgetov:

- menu - vertikálne, koláčové kruhové, koláčové štvorcové



Obr. 3.6: Visual codes menu v mobile

- zaškrtávacie políčka (checkboxes) a políčka s jednou voľbou (radio buttons)
- ohraničené vstupné polia (free-form input)
- posuvníky (sliders)

Kódy môžu byť takisto vytlačené na papierové dokumenty, zobrazované na elektronických obrazovkách alebo pripojené k fyzickým objektom s pridaním dodatočnej fukcionality.

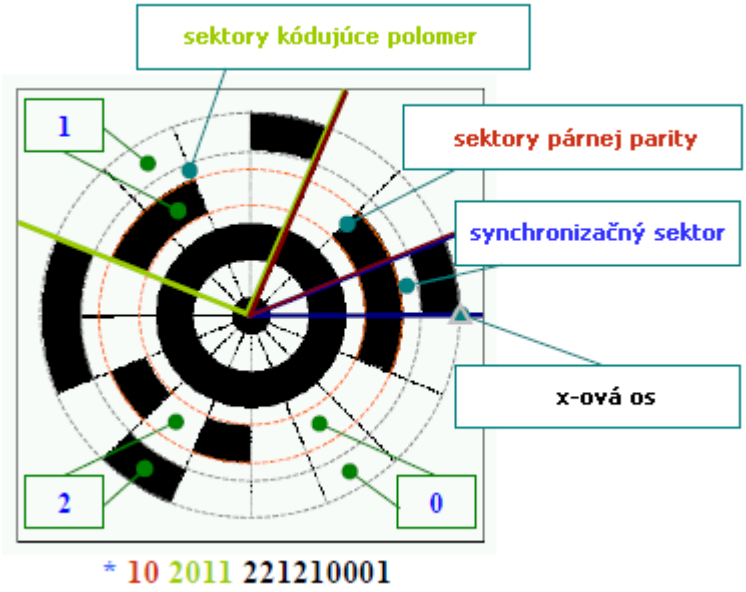
Obr. 3.7: Použitie vizuálnych kódov ako telefónnych čísel

3.2.2 Shotcodes

Shotcode [23] je kruhový kód, ktorý bol vytvorený v roku 1999 na Cambridgskej Univerzite. Používa terčový kruh s býčím okom v strede a dátovými kruhmi, ktoré ho obklopujú. Technológia číta dátové bity z dátových kruhov vypočítaním uhla a vzdialenosti od býčieho oka. Shotcode bol navrhnutý pre zariadenia s fotoaparátom ako mobilný telefón alebo webkamera bez použitia špecializovaného hardvéru. Shotcodu predchádzali názvy TRIPcode a Spotcode.

Obr. 3.8: "http://en.wikipedia.org/wiki/ShotCode"

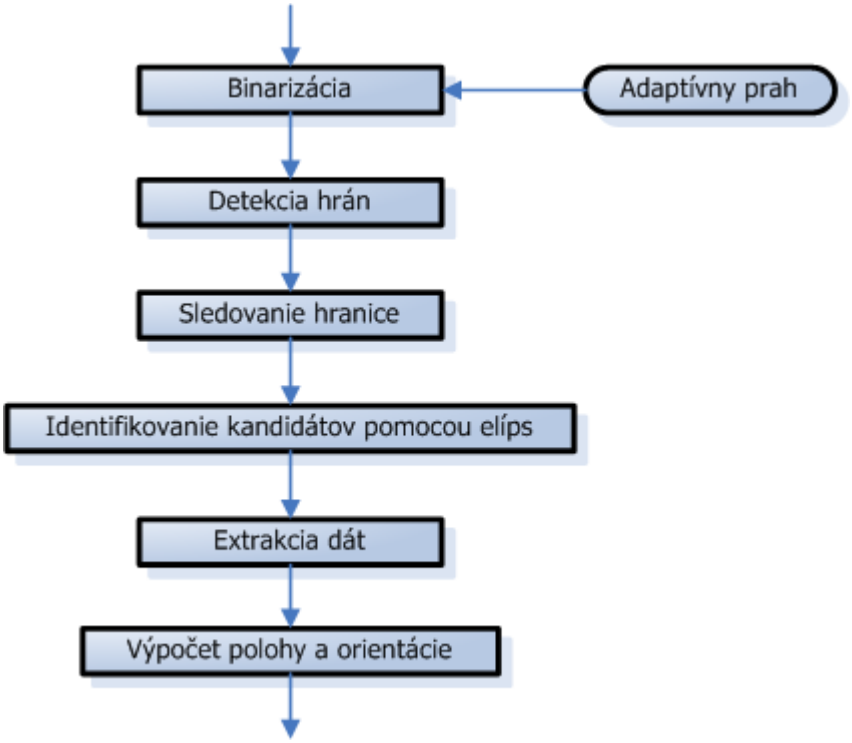
TRIP (Target Recognition using Image Processing) [16] je nenákladný monitorovací systém, ktorý používa kombináciu kruhového kódu (TRIPcode) a kamery s nízkym rozlíšením.



Obr. 3.9: Špecifikácia TRIP kódu

TRIP kód pozostáva z dvoch sústredných kruhov obklopujúcich býčie oko. Tieto dva kruhy sú rozdelené na 16 sektorov. Prvý sektor je synchronizačný a určuje, kde

TRIP kód začína. Je tvorený dvomi čiernymi bitmi, pričom nikde inde sa táto kombinácia už nemôže vyskytnúť. Kód je čítaný od tohto sektora proti smeru hodinových ručičiek. Ďalšie 2 sektory sú vyhradené na kontrolu párnej parity. Nasledujúce 4 sektory kódujú polomer centrálneho býčieho oka v milimetroch. Zvyšných 9 sektorov kóduje trojkový identifikátor. TRIP kód môže teda zakódovať číslo v rozmedzí 1 - 19683 (3^9-1).



Obr. 3.10: TRIPcode algoritmus

Sledovanie hranice - predstavuje získanie vnútornej hranice komponentu. Komponent je reprezentovaný bielymi pixlami a pozadie čiernymi. Budeme uvažovať 8-spojitosť.

Pseudokód:

```
pre každý riadok obrázka
   pre každý stĺpec obrázka
      ak je aktuálny pixel biely
         označ tento pixel ako prvý bod hranice
         preskúmaj susedov v poradí proti smeru hodinových ručičiek,
         začni ľavým dolným susedom a prvý nájdený biely pixel označ
         ako ďalší bod hranice
         pokiaľ prvý bod hranice je rôzny od posledného
```

```
preskúmaj susedov v poradí proti smeru hodinových ručičiek,
začni ľavým dolným susedom a prvý nájdený biely pixel označ
ako ďalší bod hranice
```

Poradie skúmania susedov:

7	6	5
8	*	4
1	2	3

Tento algoritmus nájde hranice všetkých komponentov v obraze. Podľa nich vyfiltrujeme len tie komponenty, ktoré dodržujú eliptický tvar.

Identifikovanie kandidátov pomocou elíps - spočíva v metóde napasovania elíps na dané hranice (least-square ellipse fitting) [18].

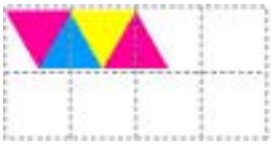
3.2.3 Microsoft Tag

Microsoft Tag [24] je sofistikovaná technológia vysokokapacitných farebných 2D kódov (High Capacity Color Barcode - HCCB) vynájdená výskumom Microsoftu. Je navrhnutá pre maximálny výkon aj na limitovaných fotoaparátoch na mobilných telefónoch. HCCB kód pozostáva z farebných trojuholníkov v geometrickej mriežke a tým zabezpečuje zakódovanie väčšieho množstva informácii na menšej ploche. Microsoft Tag používa 4 rôzne farby a mriežku rozmerov 5 x 10. Technológia rozpoznávania nie je bližšie špecifikovaná, keďže sa jedná o komerčný formát. Na rozdiel od ostatných technológií, Microsoft Tag neuchováva žiadne informácie. Jedine, čo uchováva je unikátne ID, ktoré následne posiela servrom Microsoftu.

Obr. 3.11: Porovnanie veľkosti kódov s existujúcimi štandardami

Porovnanie veľkosti 2D kódu kódujúceho 1B (8 bitov):

- použitie 8 čiernobielych symbolov



- použitie 4 farebných symbolov

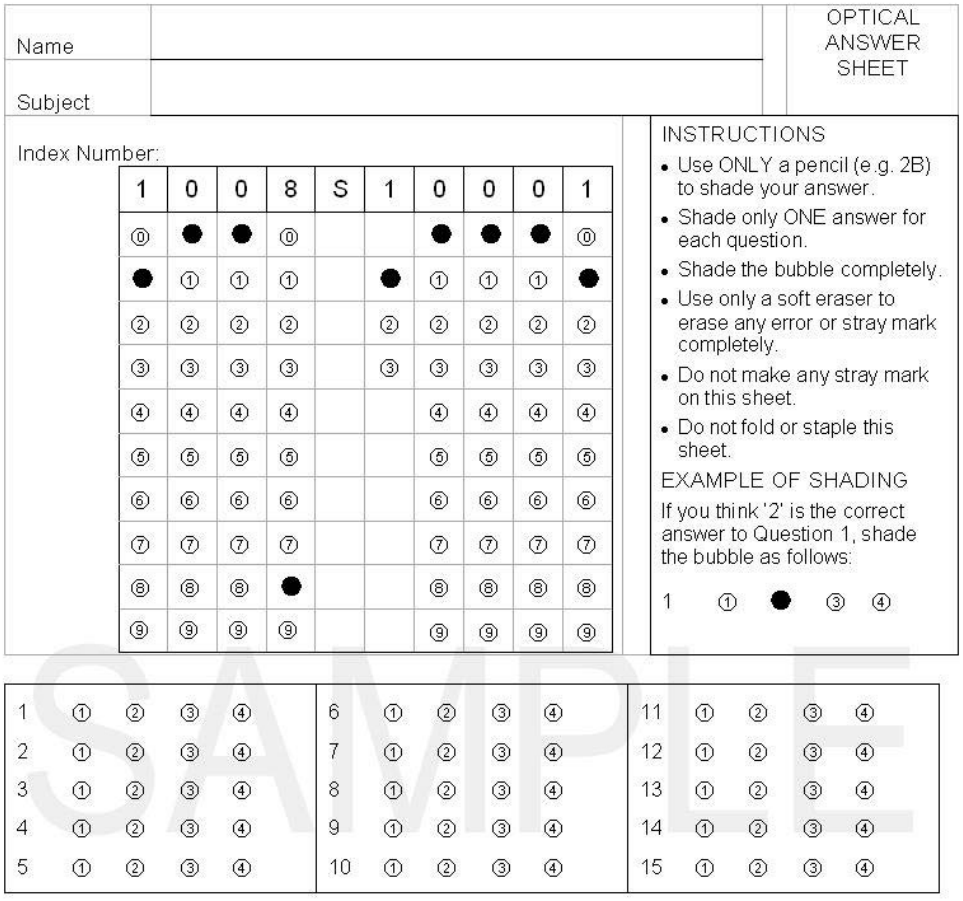
Vlastnosti:

- designovaný pre limitované kapacity fotoaparátov v mobiloch - dekódovanie aj nezaostreného obrazu
- kapacita uloženia dát 13 bajtov
- rozšírená Reed-Solomon korekcia chýb
- menší ako ostatné formáty

4 Optické rozpoznávanie značiek (OMR)

Optické rozpoznávanie značiek (optical mark recognition) je technológia elektronického extrahovania dát z predurčených oblastí, ako sú zaškrtávacie políčka a vypĺňacie oblasti na predtlačených formulároch. Optické rozpoznávanie značiek je vo všeobecnosti oveľa jednoduchšie ako optické rozpoznávanie znakov (optical character recognition). OMR technológia skenuje formulár, číta preddefinované miesta a zaznamenáva, kde sú urobené značky. Značky sú skonštruované tak, že je len minimálna šanca na zlé rozpoznanie a nevyžadujú ani špeciálny kontrast obrázku.

Jednou z najznámejších aplikácií optického rozpoznávania značiek je použitie optického odpoveďového hárku (optical answer sheet) s niekoľkými možnosťami odpovede.



Obr. 4.1: Optický odpoveďový hárok

Treba zdôrazniť, že pri takomto rozpoznávaní značiek sa nepoužívajú žiadne špeciálne značky v rohoch papiera. Tým nastáva veľká bariéra použiteľnosti. V minulosti aj súčasnosti vyžadujú niektoré OMR systémy špeciálny papier, špeciálny atrament a špeciálne čítacie zariadenie. Množstvo špecializovaných OMR skenerov používaných dnes vyrába Scantron Corporation. Obyčajné papierové OMR formuláre sú zase spracovateľné napr. špeciálnym softvérom Remark Office OMR [21]. Cena najnovšej verzie Remark Office OMR 7 je 995$.

Funkcie Remark Office OMR:

- rozpoznávanie optických značiek - krúžky a zaškrtávacie polia, počítačom generovaných značiek a čiarových kódov
- navrhovanie vlastných formulárov
- trénovanie softvéru na čítanie vlastných formulárov
- skenovanie formulárov
- export dát do viac ako 30 formátov
- analýza dát a vyhodnotenie

Obr. 4.2: Softvér Remark Office OMR

4.1 Výhody a nevýhody OMR

Pokrok v OMR umožňuje používateľom vytvárať a tlačiť svoje vlastné formuláre. Používateľ je schopný naštýlovať otázky vo formáte, ktorý vyhovuje jeho potrebám. OMR systémy dosahujú 100%-nú presnosť a rozpoznávanie značiek zaberie v priemere len 0,05 sekundy. Používatelia môžu použiť ako značky štvorce, kruhy, elipsy a hexagóny. Softvér je potom možné nastaviť na rozpoznávanie vyplnených bublín, krížikov alebo kontrolovať značky.

Existujú aj nevýhody a obmedzenia OMR. Ak chce používateľ zhromaždiť veľké množstvo textu, potom OMR komplikuje zber dát. Taktiež sa môže stať, že budú chýbať údaje do testovacieho procesu. Nesprávne alebo neočíslované stránky môžu viesť k ich skenovaniu v nesprávnom poradí. Stránky môžu byť naskenované duplicitne a pootočene.

5 Analýza a návrh

5.1 Analýza modelovej situácie

Naša modelová situácia pozostáva zo 4 krokov:

1. návrh formulára
2. vyplnenie testu
3. naskenovanie
4. vyhodnotenie počítačom

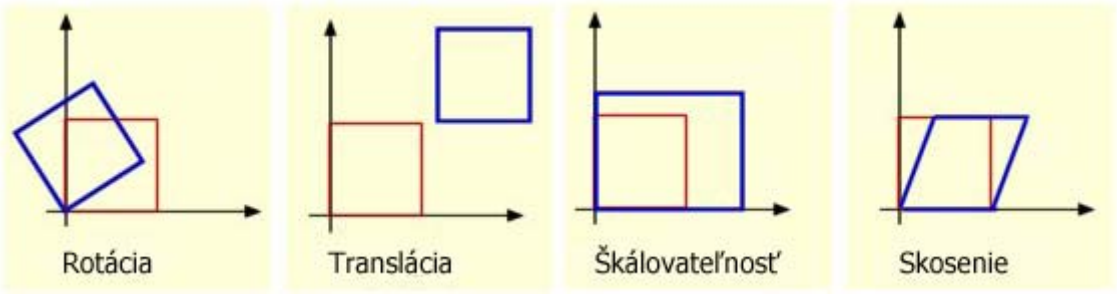
Obr. 5.1: Modelová situácia

Ako vidíme samotný proces, tak môžeme ovplyvniť jedine prvý a štvrtý krok. Môžeme teda navrhnúť formulár ako má vyzerať a potom implementovať samotný algoritmus na jeho rozpoznávanie počítačom. To už ako človek test vyplní a opravovateľ naskenuje, neovplyvníme. Ak sa však zamyslíme, náš návrh formulára ovplyvní aj samotné rozpoznávanie počítačom.

5.2 Návrh formulára

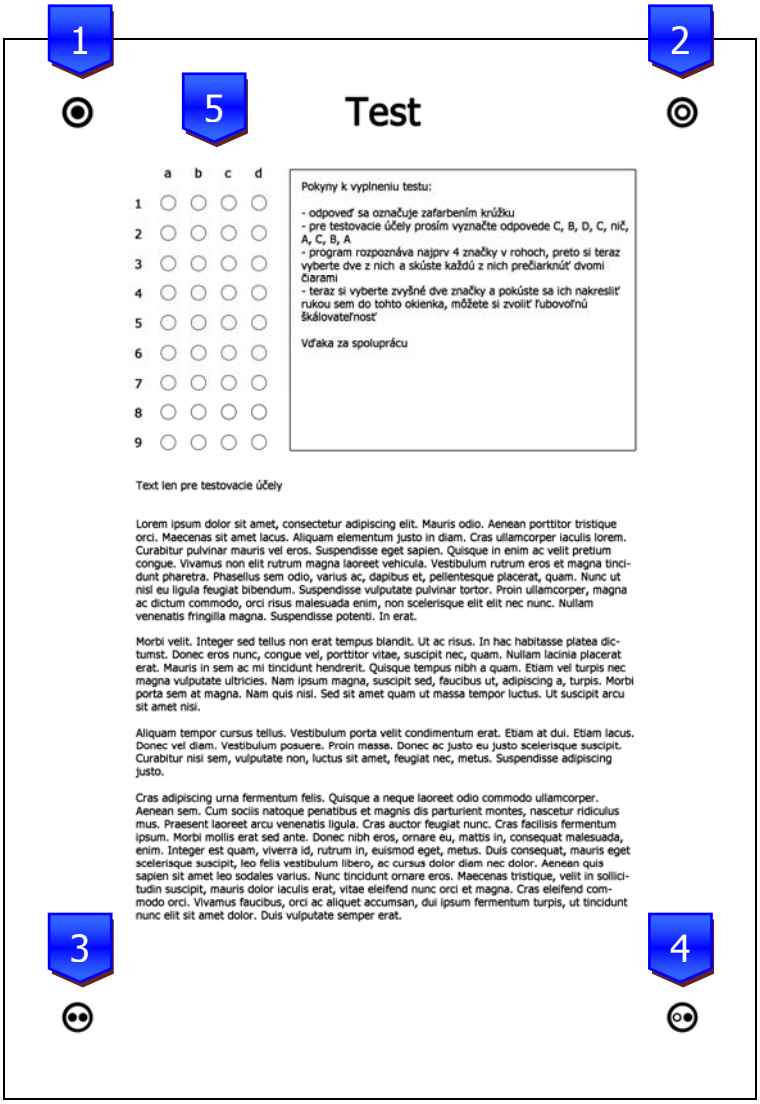
Pri návrhu formulára musíme zohľadniť:

1. možnosť zistenia skutočnosti, či bol test kompletne naskenovaný
2. robustnosť na afinné transformácie
    a. rotácia - test môže byť naskenovaný pootočene
    b. translácia - test môže byť posunutý pri skenovaní
    c. škálovateľnosť - test môže byť naskenovaný s rôznym dpi
    d. skosenie - nebudeme uvažovať



Obr. 5.2: Afinné transformácie

Všetky tieto požiadavky zabezpečíme návrhom vhodných značiek, ktoré umiestnime do 4 rohov papiera. Stredy značiek budú vymedzovať obdĺžnik samotného testu s odpoveďovými políčkami. Ak budú rozpoznané všetky 4 značky, vieme povedať, že test bol naskenovaný kompletne.



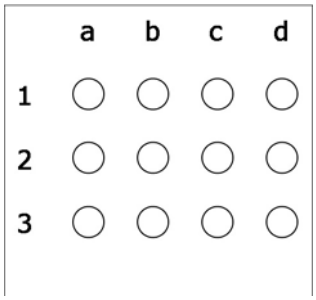
Obr. 5.3: Návrh formulára

- jednotlivé značky

- testová časť

Testová časť formulára bude pozostávať z matice otázok a odpovedí. Pre demonštračné účely budeme uvažovať pre každú otázku 4 možné odpovede A, B, C, D, pričom len jedna bude správna. Označovanie odpovedí bude realizované zafarbovaním krúžkov. Máme na výber ešte zaškrtávanie krížikov do štvorčekov, avšak rozdiel medzi zaškrtnutým a nezaškrtnutým políčkom bude predstavovať omnoho menej zafarbenej plochy ako pri zafarbovaní krúžkov a tým aj vyššiu pravdepodobnosť chyby vyhodnotenia odpoveďového políčka.

Testová časť formulára bude umiestnená do plochy, ktorú vymedzujú 4 navrhnuté značky. Takisto musíme poznať relatívne umiestnenie testu vzhľadom k prvej značke, aby sme pri vyhodnocovaní testu vedeli presné umiestnenie krúžkov. Odpoveďové políčka budú umiestnené v štvorcovej mriežke s rovnakým rozostupom. Ak bude test naskenovaný pootočene, vypočítame uhol sklonu a z neho odvodíme presné umiestnenie krúžkov.



Obr. 5.4: Návrh odpoveďových políčok

Ešte treba zvážiť možnosť, že test bude mať viac strán. Vtedy môžeme čísla strán testu zakódovať pomocou dátovej matice. Identifikácia mena je ďalší problém. Jedna možnosť je rozdať na začiatku roka nálepky s dátovými maticami každému človeku a pri vyplňovaní testu si nalepí každý tú svoju. Druhá možnosť je pomocou čísla ISIC alebo nejakého interného číslovania študentov, pričom číslo označí priamo pomocou odpoveďových políčok (viď. optický odpoveďový hárok).

5.3 Analýza značiek

Prvá otázka je koľko značiek potrebujeme a či musia byť rôzne. Keďže potrebujeme ohraničiť plochu, ktorú chceme následne analyzovať, budeme potrebovať práve 4 značky. Menej značiek by spôsobilo, že nevieme jednoznačne určiť, či bola naskenovaná celá plocha alebo nie. Viac značiek by zase predstavovalo redundanciu. Ak by boli všetky štyri značky rovnaké, nastáva problém určiť orientáciu papiera.

1	1	1	2	1	2	1	2
1	1	2	2	3	3	3	4
a) 1 typ		b) 2 typy		c) 3 typy		d) 4 typy	

Postupným vylučovaním sa dostávame k záveru, že všetky 4 značky musia byť rôzne. Po analýze existujúcich geometrických a topologických značiek sa môžeme pustiť do navrhovania vlastných značiek.

5.3.1 Kritéria pre dobré značky

Ako kritéria pre dobré značky si zvolíme:

1. rýchlosť rozpoznávania
2. robustnosť na afinné transformácie
3. odolnosť voči poškodeniu
4. nízka pravdepodobnosť výskytu duplicity

Rýchlosť rozpoznávania bude závisieť od samotného typu značky. Po analýze algoritmov pre geometrické a topologické značky jednoznačne vyhrávajú topologické značky.

Čo sa týka robustnosti na afinné transformácie, nastáva problém len s geometrickými značkami, kde bude zlyhávať škálovateľnosť a rotácia. Transláciu spĺňajú obidva typy značiek.

Odolnosť voči poškodeniu čiastočne spĺňajú obidva typy značiek. Pri geometrických značkách môžeme znížiť 100%-nú zhodu so vzorkou na nižšiu. Pri topologických značkách, ak neporušíme topológiu, tak značku nájdeme. Ak uvažujeme drobné počarbanie značky, nie úplne zaškrtanie, môžeme použiť eróziu, aby sme dostali pôvodnú topológiu.

Pri geometrických značkách je nízka pravdepodobnosť výskytu duplicity značky zabezpečená samotnou vzorkou, ktorá musí presne zapasovať na nejaké miesto v hľadanom obrázku. Pri topologických značkách sa len overí topológia, čiže sa môže stať, že nejaký náhodný objekt bude tiež spĺňať túto topológiu. Môžeme vymyslieť zložitejšiu topológiu a tým znížime pravdepodobnosť výskytu rovnakej topológie. Keď budeme mať papier a na ňom nakreslené značky, môže sa stať, že niekto sa bude chcieť pokúsiť dokresliť tú istú značku voľnou rukou. V tomto prípade opäť vyhrávajú geometrické značky.

Zhrnutie kritérií (● - spĺňa, ○ - nespĺňa):

	topologické	geometrické
rýchlosť rozpoznávania	●	○
robustnosť na afinné transformácie	●	○
odolnosť voči poškodeniu	●	●
nízka pravdepodobnosť výskytu duplicity	○	●

Vidíme, že v celkovom hodnotení je to 3:2 v prospech topologických značiek, avšak dva nedostatky topologických značiek môžeme ošetriť vhodnou topológiou. Rozhodneme sa preto pre použitie topologických značiek.

5.3.2 Návrh značiek

Teraz potrebujeme navrhnúť geometrický tvar značiek. Medzi najjednoduchšie obrazce patria kružnice a štvorce. Vzhľadom na posledné kritérium si vyberieme pre kružnice, pretože tie sa predsa len ťažšie kreslia voľnou rukou ako štvorce.

Potrebujeme vymyslieť predovšetkým topológiu, ktorá bude spoľahlivá. A preto naša topológia bude pozostávať zo sústredných kružníc. Spoľahlivosť zabezpečíme overovaním, či všetky vrstvy majú spoločný stred (sústrednosť kružníc). Ukazuje sa, že dve vrstvy je málo, ale tri by už mohli byť zaujímavé. Zoberme si teda ako značku číslo 1 kruh s tromi vrstvami, dvomi čiernymi a jednou bielou medzi nimi. Od tejto značky môžeme odvodiť ďalšie pomocou pridávania ďalších vrstiev vo forme zmenšujúcich sa sústredných kružníc.

Dostávame nasledujúci návrh značiek (fiducial):

značka č.1 (3 vrstvy)

značka č.2 (4 vrstvy)

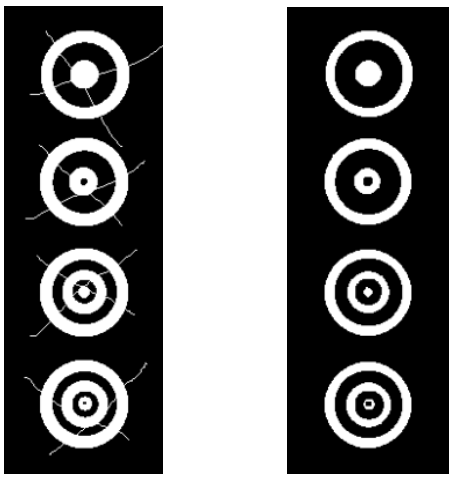
značka č.3 (5 vrstiev)

značka č.4 (6 vrstiev)

Pozrime sa teraz na našu tabuľku kritérií. Vyriešili sme spoľahlivosť topológie, čím sa nám znížila pravdepodobnosť výskytu duplicitnej značky. Takisto aj voľnou rukou sa veľmi ťažko trafíme, aby boli všetky kružnice sústredné. Jediným slabým miestom je teraz odolnosť voči počarbaniu značiek. Implicitne môžeme predpokladať, že značka bude počarbaná len čiernou farbou, keďže nikto štandardne nepíše bielym perom. Z toho vyplýva, že sa nám môže narušiť vnútorná topológia značky, ale v žiadnom prípade nám značku nerozpolí na dva samostatné komponenty. Pôvodnú topológiu sa pokúsime obnoviť pomocou erózie.

5.3.3 Vylepšenie odolnosti značiek

Pri počarbaní značiek nám erodovanie zabezpečí stenšenie všetkých čiar a teda úplné vypadnutie tenkých čiar, ktoré preškrtali značku.



Obr. 5.5: Počarbaný a erodovaný obrázok

Nastáva problém pri poslednej značke. Čo keď sa človek s perom trafí presne do bieleho kruhu v poslednej vrstve? Ten je natoľko malý, že sa nám zo značky stane identická značka ako predposledná a to aj po vykonaní erózie. Zohľadnením tohto faktu navrhneme vylepšené značky. Prvé dve značky ostanú rovnaké a ďalšie dve kritickejšie navrhneme inak. Použijeme dvojicu sústredných kružníc.

Vylepšený návrh značiek (fiducial+):

 značka č.1

 značka č.2

 značka č.3

 značka č.4

5.4 Návrh aplikácie

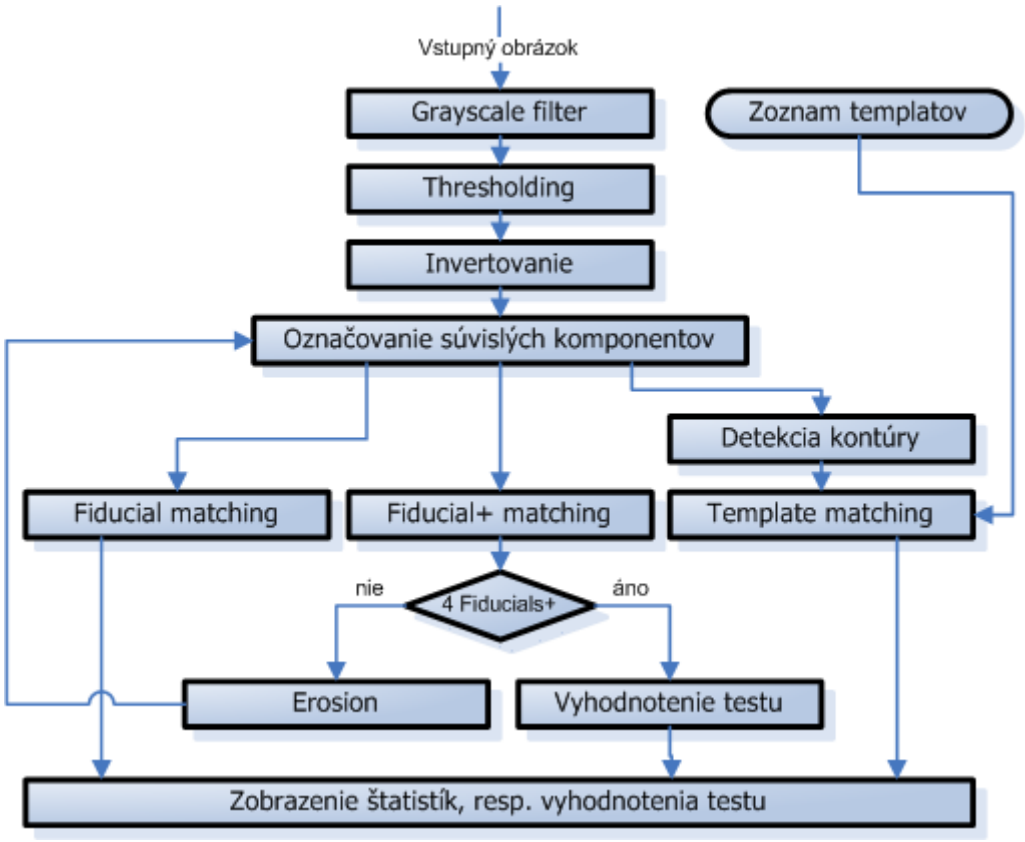
5.4.1 Návrh architektúry

Budeme implementovať algoritmy rozpoznávania pre 3 typy značiek:

1. geometrické značky (template)
2. topologické značky (fiducial)
3. vylepšené topologické značky (fiducial+)

Ak boli nájdené všetky vylepšené topologické značky prevedieme aj samotné vyhodnotenie testu. Implementácia bude teda zahŕňať aj rozpoznávanie odpoveďových políčok. Geometrické značky, ktoré budeme implementovať, budú písmená A, B, D a G v čiernom štvorci.

Obr. 5.6: Implementované topologické značky



Obr. 5.7: Návrh hlavného programu

5.4.2 Návrh používateľského rozhrania

Menu

- File / Open (CTRL+O) - načítanie a zobrazenie obrázku
- Exit - ukončenie aplikácie

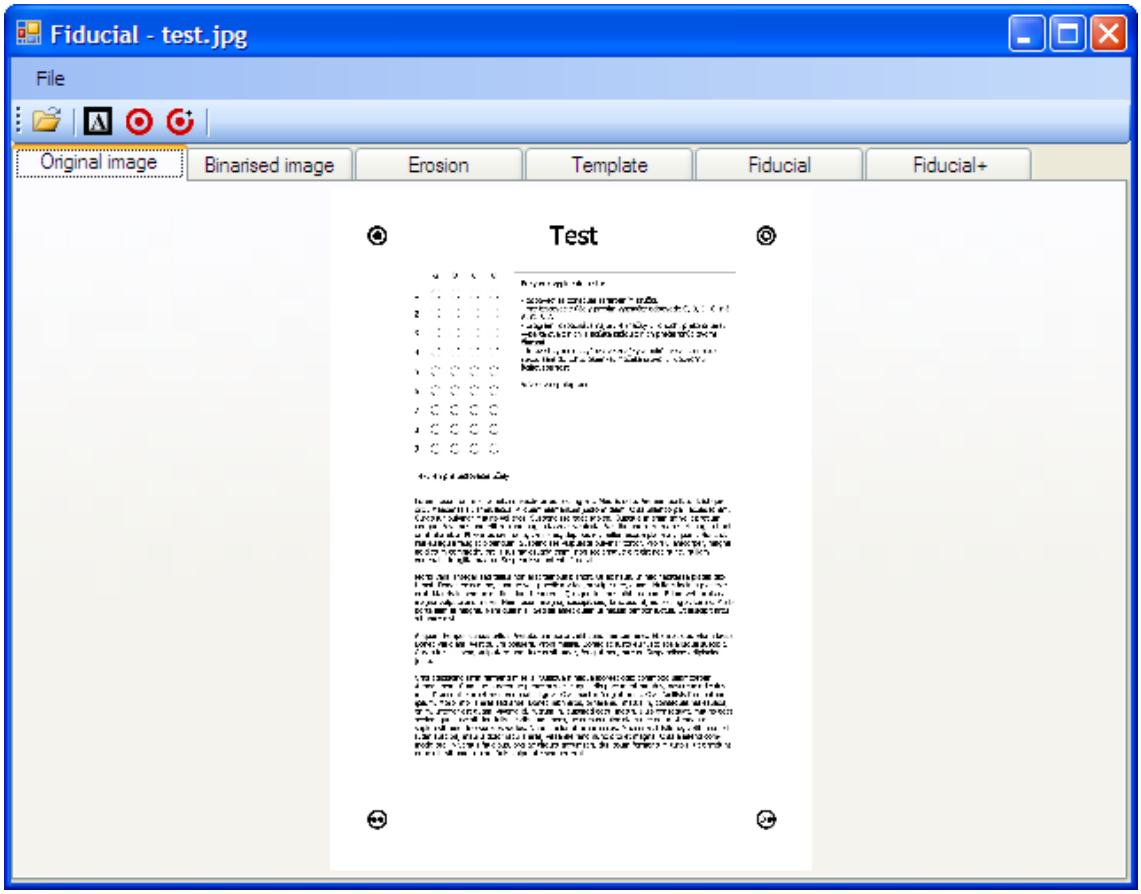
Panel nástrojov

- Open - načítanie a zobrazenie obrázku
- Template matching - spustenie algoritmu pre geometrické značky
- Fiducial matching - spustenie algoritmu pre topologické značky
- Fiducial+ matching - spustenie algoritmu pre vylepšené topologické značky

Panel so záložkami

1. Original image - originálny obrázok
2. Binarised image - binarizovaný a invertovaný obrázok
3. Erosion - erodovaný obrázok (v prípade potreby)
4. Template - nájdené geometrické značky + štatistiky

5. Fiducial - nájdené topologické značky + štatistiky
6. Fiducial+ - nájdené vylepšené topologické značky + štatistiky + vyhodnotenie testu



Obr. 5.8: Ukážka používateľského rozhrania

5.4.3 Dátové štruktúry

Postačia nám dve triedy, jedna pre objekt topologickej značky a druhá pre objekt geometrickej značky. Každá značka si bude uchovávať svoje identifikačné číslo a obdĺžnik, ktorý ju ohraničuje. Ten bude určený súradnicami ľavého horného rohu, šírkou a výškou. Pri geometrických značkách si uchováme navyše koeficient podobnosti.

```
class Fiducial
{
   int Id;
   Rectangle Rectangle;
}
```

```
class Template
{
   int Id;
   Rectangle Rectangle;
   float similarity;
}
```

6 Implementácia

6.1 Technológie

Samotnú implementáciu budeme robiť v jazyku C# a technológii .NET. Aby sme nemuseli programovať základné veci ako je binarizácia a niektoré ďalšie filtre, použijeme knižnicu AForge.NET, ktorá je dostupná pod GPL licenciou.

AForge.NET [25] je C# framework navrhnutý pre vývojárov a výskumníkov na poli počítačového videnia a umelej inteligencie. Poskytuje spracovanie obrazu, neurónové siete, genetické algoritmy, učenie sa a množstvo ďalších užitočných vecí.

Framework sa skladá zo 7 hlavných a niekoľkých prídavných knižníc:

1. AForge.Imaging – spracovanie obrazu a filtre
2. AForge.Neuro – neurónové siete
3. AForge.Genetic – genetické algoritmy
4. AForge.Vision – počítačové videnie
5. AForge.Machine Learning – strojové učenie
6. AForge.Robotics - podpora robotických zostáv
7. AForge.Video - spracovanie videa

Pre účely práce postačuje použiť knižnicu AForge.Imaging.

6.2 Programovanie aplikácie

6.2.1 Popis tried

GrayscaleBT709 - filter, ktorý používa algoritmus BT709 na skonvertovanie farebného obrázku na odtiene šedej. Výpočet prebieha podľa nasledovných koeficientov:

$$Y = 0{,}2125\ R + 0{,}7154\ G + 0{,}0721\ B$$

Threshold - filter, ktorý prevedie šedý obrázok na čiernobiely nastavením globálneho prahu a všetky body s intenzitou väčšou ako prah budú vyhlásené za biele a všetky

body s menšou intenzitou za biele. Filter používa predvolenú hodnotu prahu 128. Nepotrebujeme implementovať adaptívny prah, pretože pri skenovaní papiera máme rovnomerné osvetlenie.

Invert - filter, ktorý invertuje čierne a biele body. Algoritmus označovania súvislých komponentov totiž považuje čierne body za pozadie a biele za komponenty.

Crop - filter, ktorý vystrihne daný obdĺžnik z obrázka.

BlobCounter - connected component labeling algoritmus. Algoritmus je postupný (row-by-row) a môže vrátiť pole obdĺžnikov ohraničujúcich súvislé komponenty alebo pole objektov Blob.

ExhaustiveTemplateMatching - template matching algoritmus, ktorý prikladá vzorku na každú možnú pozíciu v prehľadávanom obrázku a porovnáva podobnosť. Prah podobnosti SimilarityThreshold možno nastaviť z intervalu (0,1>. Predvolená hodnota je 0,9. Algoritmus vráti nájdené vzorky ako pole objektov TemplateMatch.

Erosion - filter erodovania obrázku. Predvolený štrukturálny element je matica 3 x 3 so samými jednotkami.

Dilatation - filter dilatácie obrázku. Predvolený štrukturálny element je matica 3 x 3 so samými jednotkami.

6.2.2 Hlavný program

1. **Inicializácia aplikácie** - inicializácia hlavného formu a načítanie vzoriek (template) do pamäte

```
public MainForm()
{
      InitializeComponent();
      loadTemplates();
}
private void loadTemplates()
{
   try
   {
      Bitmap templateImage=(Bitmap)Bitmap.FromFile("templates/A.jpg");
      AForge.Imaging.Image.FormatImage(ref templateImage);
```

```
            templateImage = binaryFilter.Apply(templateImage);
            templateImages.Add(templateImage);

            ...
        }
        catch {...}
    }
```

2. **Načítanie obrázku** - podporované formáty sú *.jpg, *.png, *.tif, *.bmp, *.gif

```
private void loadImage()
{
    try
    {
        if (openFileDialog.ShowDialog() == DialogResult.OK)
        {
            this.Text = "Fiducial - " +
            System.IO.Path.GetFileName(openFileDialog.FileName);
            sourceImage =
            (Bitmap)Bitmap.FromFile(openFileDialog.FileName);
            AForge.Imaging.Image.FormatImage(ref sourceImage);
            hDPI = sourceImage.Width / A4_PAPER_WIDTH_CM * 2.54;
            vDPI = sourceImage.Height / A4_PAPER_HEIGHT_CM * 2.54;
            sourcePictureBox.Image = sourceImage;
            initializeVariables();
            tabControl.SelectTab(0);
        }
    }
    catch {
        MessageBox.Show("Failed loading the image", "Error",
        MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
```

Ako prvé potrebujeme obrázok načítať do pamäte a normalizovať pixlový formát kvôli zabezpečeniu korektného vstupu pre filtre. To nám zabezpečí funkcia FormatImage(). Obrázok je nedotknutý v prípade, ak je už v jednom z týchto formátov:

- Format24bppRgb - pri farebných obrázkoch
- Format8bppIndexed - pri šedých (grayscale) obrázkoch

Inak je skonvertovaný na formát Format24bppRgb. Následne obrázok zobrazíme používateľovi a inicializujeme premenné programu pomocou metódy initializeVariables().

Spracovanie obrázku - má 4 hlavné časti a bude zahŕňať rozpoznávanie značiek, resp. samotného rozpoznávania testu v závislosti od vybraného algoritmu

```
private void processImage(Bitmap image, int algorhitm)
{
    try
    {
        // 1. konverzia obrázka na binárny
        Bitmap binarisedImage = binaryFilter.Apply(image);
        Bitmap erosionImage = null;
        DateTime startTime = DateTime.Now;
```

```
        // 2. označovanie súvislých komponentov
        Rectangle[] rectangles =
        getConnectedComponentsRectangles(binarisedImage, (int)(vDPI *
        20 / 300), (int)(vDPI * 20 / 300));
        // 3. rozpoznávanie značiek
        searchCandidates(binarisedImage, rectangles, true, algorhitm);
        if (fiducialPlusCount < MAX_FIDUCIALS && algorhitm ==
        FIDUCIAL_PLUS_MATCHING)
        {
            Erosion erosionFilter = new Erosion();
            erosionImage = erosionFilter.Apply(binarisedImage);
            rectangles = getConnectedComponentsRectangles(erosionImage,
            (int)(vDPI * 20 / 300), (int)(vDPI * 20 / 300));
            searchCandidates(erosionImage, rectangles, false,
            algorhitm);
        }
        DateTime stopTime = DateTime.Now;
        if (fiducialPlusCount == 4)
            // 4. rozpoznávanie testu
            evaluateTest(binarisedImage);
        duration = stopTime - startTime;
        showResults(binarisedImage, erosionImage, algorhitm);
    }
    catch {
       MessageBox.Show("Failed processing the image", "Error",
       MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}
```

6.2.2.1 Konverzia obrázka na binárny

Použijeme sekvenciu zloženú z troch filtrov:

```
FiltersSequence binaryFilter = new FiltersSequence(
      new GrayscaleBT709(),
      new Threshold(THRESHOLD),
      new Invert()
);
```

Nastavenie Thresholdu začne zohrávať úlohu ak skenujeme test farebne. Väčšina ľudí totiž píše modrým perom a tak už pri samotnej binarizácii môžeme čiastočne odstrániť čiary, ktoré poškodzujú značky. Nemôžeme sa však na to spoliehať a budeme uvažovať najhorší prípad, keď je test naskenovaný čiernobielo. Ponecháme teda Threshold nastavený na predvolenú hodnotu 128.

6.2.2.2 Označovanie súvislých komponentov

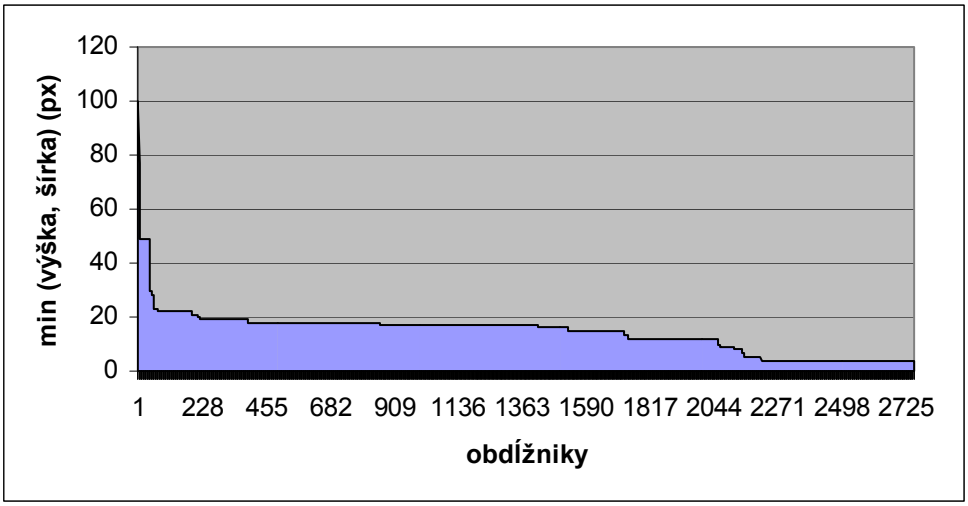
Teraz, keď sme si obrázok predpripravili, použijeme ho ako vstup pre algoritmus označovania súvislých komponentov - trieda BlobCounter. Jeden súvislý komponent budeme nazývať Blob. Implementácia je vykonaná postupným algoritmom (row-by-

row). Biele pixle sú označované ako bloby, čierne pixle sa ignorujú. Algoritmus nám vráti pole obdĺžnikov, ktoré ohraničujú nájdené bloby.

Ešte pred samotným spustením je vhodné nastaviť triedenie obdĺžnikov podľa veľkosti, čím zabezpečíme skúmanie topológie prípadného kandidáta zvonka dovnútra. Algoritmu sa dá nastaviť aj filtrovanie blobov. Blob je vyfiltrovaný práve vtedy, keď je šírka jeho obdĺžnika < MinWidth alebo výška jeho obdĺžnika < MinHeight. Pre filtrovanie potrebujeme nastaviť rovnakú MinWidth aj MinWidth, keďže naše hľadané značky budú mať štvorcové ohraničenie.

```
BlobCounter blobCounter = new BlobCounter();
blobCounter.ObjectsOrder = ObjectsOrder.Size;
blobCounter.FilterBlobs = true;
blobCounter.MinWidth = minWidth;
blobCounter.MinHeight = minHeight;
blobCounter.ProcessImage(image);
```

Filtrovanie nám môže urýchliť celkový čas rozpoznávania, pretože sa nebudú skúmať malé nepodstatné bloby. Potrebujeme ale zistiť ako je to s distribúciou veľkostí obdĺžnikov. Stačí nám preskúmať graf veľkostí miním (výšky, šírky) jednotlivých obdĺžnikov, keďže pri filtrovaní stačí aby bola jedna hodnota menšia ako stanovená hranica.



Obr. 6.1: Rozloženie distribúcie (2751 obdĺžnikov)

Štatistika bola vytvorená s použitím 300dpi na teste s fiktívnym textom Lorem Ipsum, čo predstavuje dobrú reprezentatívnu vzorku. Značky fiducials+ mali veľkosť 100 x 100px. Podľa výsledkov je zjavná prahová hodnota 23px, pričom až 97% obdĺžnikov je pod touto hranicou. Aby sme dali ešte väčšiu toleranciu, zvolíme si prah 20px. V závislosti od dpi môžeme prahovú hodnotu T potom vyjadriť:

$$T = \frac{dpi * 20}{300}$$

6.2.2.3 Rozpoznávanie značiek

Teraz postupne prechádzame cez všetky obdĺžniky. Každý obdĺžnik vystrihneme - trieda Crop a skúmame ako potenciálneho kandidáta na značku.

```
private void searchCandidates(Bitmap image, Rectangle[] rectangles,
Boolean isNormalSearch, int algorithm)
{
   BlobCounter topologyBlobCounter = new BlobCounter();
   topologyBlobCounter.ObjectsOrder = ObjectsOrder.Size;
   foreach (Rectangle rectangle in rectangles)
   {
      if (rectangle.Width > 1 && rectangle.Height > 1)
      {
         Crop cropFilter = new Crop(rectangle);
         Bitmap cropImage = cropFilter.Apply(image);
         topologyBlobCounter.ProcessImage(cropImage);
         Rectangle[] topologyRectangles =
         topologyBlobCounter.GetObjectRectangles();
         switch (algorithm)
         {
            case TEMPLATE_MATCHING:
               ...
            case FIDUCIAL_MATCHING:
               ...
            case FIDUCIAL_PLUS_MATCHING:
               ...
         }
      }
   }
}
```

Každý rozpoznávací algoritmus dostane na vstupe vystrihnutý obrázok ohraničujúci kandidáta na značku, pole obdĺžnikov ohraničujúcich jeho súvislé komponenty a povolenú odchýlku stredov (deviation). Algoritmus vráti, buď identifikačné číslo (ID) značky, alebo -1 v prípade, že sa nejedná o značku.

Značky template - metóda templateMatching()

1. overíme, či je komponent štvorec detekciou kontúry
2. vymedzíme vnútro štvorca
3. template matching - trieda ExhaustiveTemplateMatching

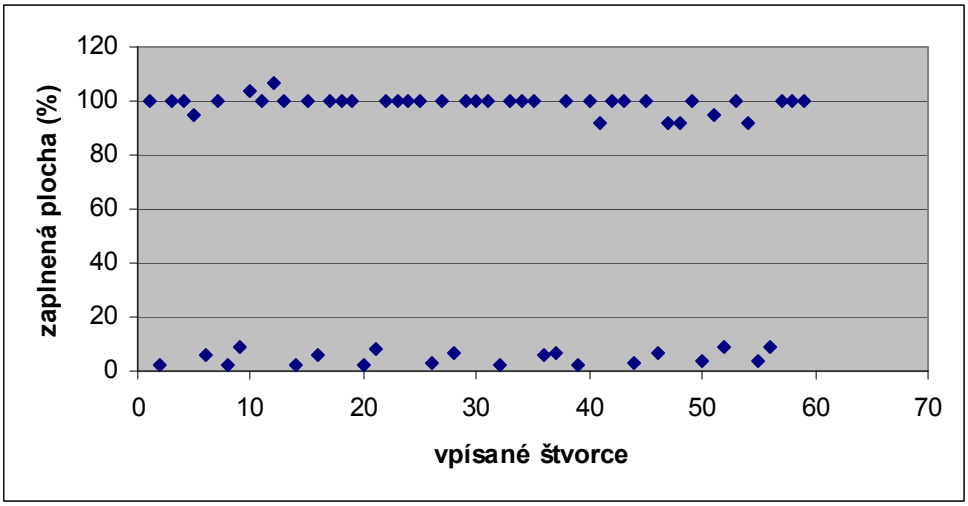
Identifikačné čísla budú rovnaké ako poradie načítania vzoriek do pamäte:

A - značka č.0, B - značka č.1, D - značka č.2, G - značka č.3,

Značky fiducial - metóda fiducialMatching()

1. connected component labeling - trieda BlobCounter
2. overíme počet komponentov, či je 2 alebo 3
3. určíme súradnice stredu vonkajšieho komponentu
4. pre každý ďalší komponent overíme, či je jeho stred identický so stredom vonkajšieho komponentu (sústrednosť kružníc)
5. na rozlíšenie značiek preskúmame na záver posledný komponent, či je kruh alebo medzikružie

Namiesto spočítania zaplnenej plochy celého štvorca a porovnaním s obsahom vpísaného kruhu to spravíme oveľa sofistikovanejšie. Spočítame len zaplnenú plochu vycentrovaného vpísaného štvorca s polovičnou veľkosťou. Tam bude rozdiel medzi kruhom a medzikružím poslednej vrstvy rapídny. Opäť spravíme štatistiku, ktorá nám poskytne značnú voľnosť pri stanovení prahu.



Obr. 6.2: Percentuálne zaplnená plocha (59 vpísaných štvorcov)

Zvolíme si prah 50%. Identifikačné čísla značiek si zvolíme podľa podmienok:

(počet komponentov == 2 a posledný komponent == kruh) => značka č.0

(počet komponentov == 3 a posledný komponent == kruh) => značka č.1

(počet komponentov == 2 a posledný komponent == medzikružie) => značka č.2

(počet komponentov == 3 a posledný komponent == medzikružie) => značka č.3

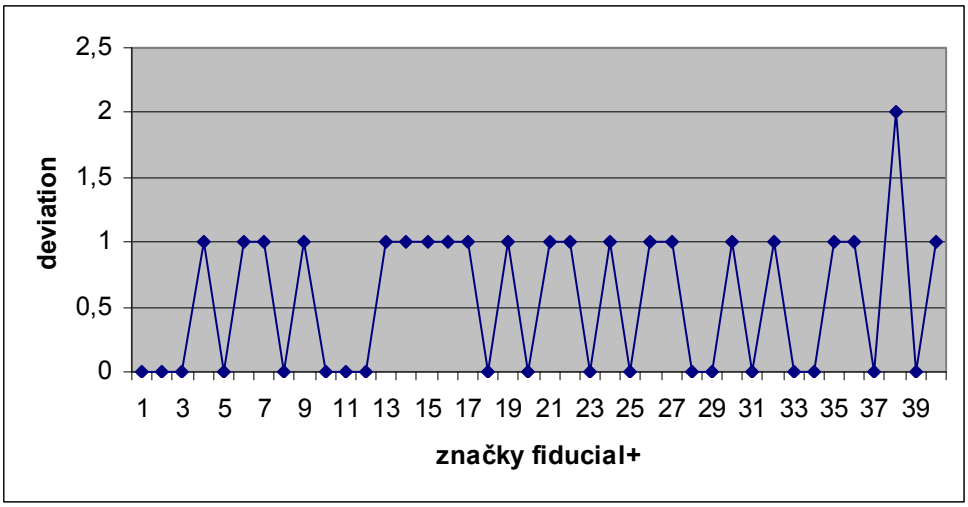
Značky fiducial+ - metóda fiducialPlusMatching()

1. connected component labeling - trieda BlobCounter
2. ak je počet komponentov 2, tak overíme, či sa stred vnútorného komponentu zhoduje s vonkajším
3. ak je počet komponentov 3, tak overíme, či je priemer stredov dvoch vnútorných komponentov identický so stredom vonkajšieho komponentu
4. na rozlíšenie značiek opäť použijeme metódu kontroly komponentu, či je kruh alebo medzikružie. Ak je počet komponentov 2, skontrolujeme len posledný komponent. Ak je počet komponentov 3, skontrolujeme obidva posledné komponenty a ak je jeden z nich medzikružie, jedná sa o značku č.3, inak o značku č.1

Pri vylepšených značkách (fiducial+) implementujeme aj hľadanie poškodených značiek (damage test), ak preskúmanie kandidátov nenašlo všetky značky. Vstupný binarizovaný obrázok erodujeme a spustíme celý náš proces prehľadávania kandidátov odznovu s tým, že nastavíme väčšiu toleranciu pri overovaní stredov. Značky by mali byť dostatočne hrubé na to, aby "prežili" aspoň jednu eróziu.

Potrebujeme ešte ošetriť, ak program nájde viac rovnakých značiek. Vtedy by mal program vyhlásiť, že nenašiel značku vôbec, pretože nevieme jednoznačne určiť, ktorá je tá správna. Ak nájdeme všetky 4 značky, program zastaví skúmanie ďalších kandidátov.

Pri overovaní stredov potrebujeme nastaviť určitú možnú odchýlku (deviation). Pozrime sa na jednotlivé odchýlky stredov značiek fiducial+.



Obr. 6.3: Odchýlky stredov (40 značiek)

Graf bol vytvorený na vzorke 10 naskenovaných testov s nepoškodenými značkami pri rozlíšení 300dpi. Až 97,5% kandidátov malo odchýlku < 2. Možnú odchýlku stredov nastavíme teda pri prvotnom hľadaní na 2 a pri hľadaní poškodených značiek dáme ešte väčšiu toleranciu na 3.

6.2.2.4 Rozpoznávanie testovej časti

Rozpoznávanie testovej časti začneme, len ak boli nájdené všetky 4 značky fiducial+. Teraz môžu ešte nastať komplikujúce situácie v podobe nejakej afinnej transformácie. Zameriame sa na rotáciu a škálovateľnosť. Translácia nám v podstate nič nezmení a skosenie nebudeme uvažovať. Predpokladajme teda, že test je mierne pootočený. Pomocou tangensu vieme tento uhol vypočítať. Teraz by sme mohli test naspäť pootočiť o daný uhol, rozpoznať značky znova a spraviť presné rozpoznávanie matice otázok a odpovedí, keďže poznáme relatívnu vzdialenosť prvého krúžku od prvej značky v horizontálnom aj vertikálnom smere. Tento prístup je však zdĺhavý a preto spravíme prepočet súradníc krúžkov vzhľadom k pootočeniu papiera. Pri škálovateľnosti prepočítame polomer keďže vieme, že pri 300 dpi má krúžok polomer 25px.

Rozpoznávanie testovej časti bude zahŕňať prejdenie matice otázok a odpovedí. Rozpoznávanie prevedieme na pôvodnom obrázku a pre väčšie zvýraznenie odpovedí použijeme dilatáciu obrázku. Pri každej otázke preskúmame možné odpovede a zistíme, ktorý krúžok je zafarbený. Potrebujeme zistiť, či je počet bielych pixlov vo štvorci ohraničujúcom krúžok väčší ako stanovený prah. Prah môžeme stanoviť ako percentuálnu výplň kruhu. Experimentálne ako 80%. Ak máme kruh vpísaný do štvorca rozmerov a x a, potom bude prah T:

$$T = 0{,}8 * \pi * \left(\frac{a}{2}\right)^2$$

Ak je počet bielych pixlov väčší ako prah, rozhodneme, že odpoveď bola označená.

6.3 Časová a priestorová zložitosť

Odhad časovej zložitosti

Nech mxn sú rozmery vstupného obrázka uloženého v pamäti. Potom časová zložitosť jednotlivých častí programu je:

1. Konverzia obrázka na binárny - zahŕňa tri filtre (GrayscaleBT709, Threshold, Invert), pričom každý z nich prejde maticu obrázka raz - $3O(mn)$.
2. Označovanie súvislých komponentov - dva prechody matice obrázka - $2O(mn)$.
3. Rozpoznávanie značiek - nech máme k obdĺžnikov kandidátov na značky, potom označovanie súvislých komponentov na každom z nich bude trvať maximálne $2kO(sirkaMax * vyskaMax)$, kde $sirkaMax$ a $vyskaMax$ sú rozmery najväčšieho obdĺžnika.
4. Rozpoznávanie testovej časti - kontrola všetkých l odpoveďových krúžkov bude trvať $lO(sirka * vyska)$, kde $sirka$ a $vyska$ sú rozmery štvorca opísaného každému krúžku.
5. Erózia (resp. dilatácia) - nech má štrukturálny element rozmery axb, prikladaním elementu na každú pozíciu obrázka mxn dostávame časovú zložitosť $O(mnab)$. Avšak v našom programe používame štrukturálny element s konštantnými rozmermi $3x3$. Časová zložitosť erózie, resp. dilatácie bude $O(9mn) = 9O(mn)$.

Spolu: $3O(mn) + 2O(mn) + 2kO(sirkaMax * vyskaMax) + lO(sirka * vyska) + 9O(mn) =$
$= O(mn)$

Odhad priestorovej zložitosti

Nech mxn sú rozmery vstupného obrázka uloženého v pamäti. Potom priestorová zložitosť jednotlivých častí programu je:

1. Konverzia obrázka na binárny - filtre si neuchovávajú žiadne pomocné polia - $O(1)$.
2. Označovanie súvislých komponentov - v prvom pomocnom poli veľkosti mxn má každý pixel priradené číslo komponentu, do ktorého patrí a druhé pomocné pole veľkosti $(m/2+1)(n/2+1)+1$ mapuje referencie medzi susediacimi komponentmi: $O(mn) + O((m/2+1)(n/2+1)+1) = O(mn)$.
3. Rozpoznávanie značiek - nech máme k obdĺžnikov kandidátov na značky, potom označovanie súvislých komponentov na každom z nich bude trvať maximálne $kO(sirkaMax * vyskaMax)$, kde $sirkaMax$ a $vyskaMax$ sú rozmery najväčšieho obdĺžnika.
4. Rozpoznávanie testovej časti - neuchováva žiadne pomocné pole - $O(1)$.
5. Erózia (resp. dilatácia) - neuchováva žiadne pomocné pole - $O(1)$.

Spolu: $O(1) + O(mn) + kO(sirkaMax * vyskaMax) + O(1) + O(1) = O(mn)$

Záver: Ukázali sme, že časová aj priestorová zložitosť bude závisieť len od rozmerov dokumentu a nie od veľkosti značiek.

6.4 Testovanie

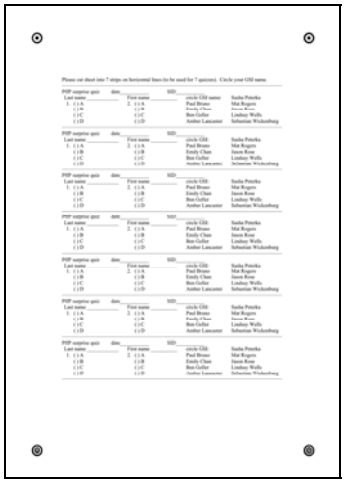
Testovanie prevedieme pri štandardných podmienkach. Veľkosť papiera A4 je 210 x 297 mm, čo nám pri naskenovaní s rozlíšením 300dpi, dá veľkosť obrázka 2480 x 3508 px. Testovanie spustíme na release verzii programu bez zbytočných ladiacich výpisov.

Parametre počítača:

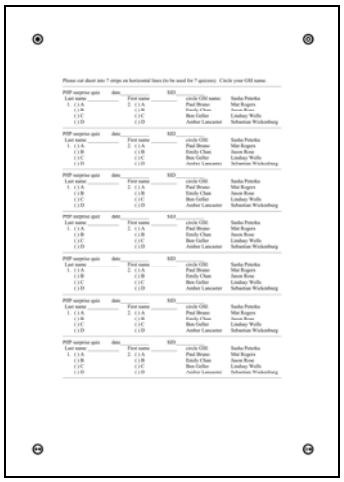
- Intel Pentium M 1,7GHz , 2MB L2 cache
- 2GB DDR2 RAM

6.4.1 Značky

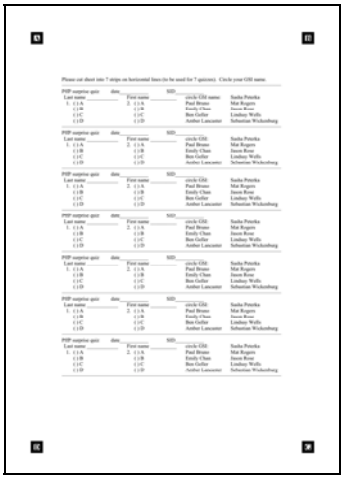
1. Testy vytvorené počítačom pre všetky typy značiek:



1) fiducial

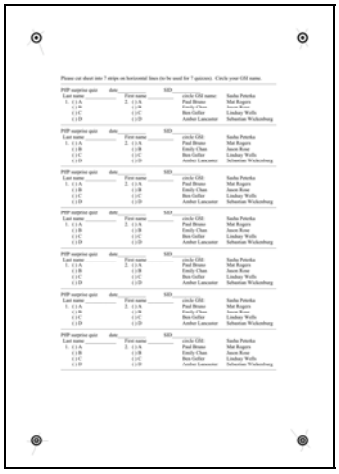


2) fiducial+

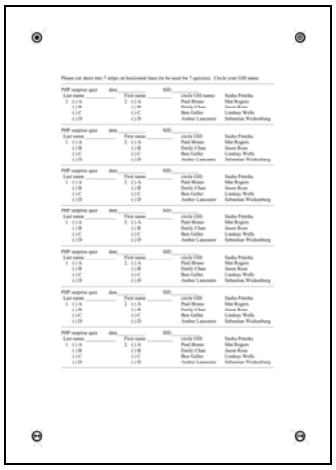


3) template

Na prvých dvoch typoch testov topologické značky ešte poškodíme tak, že ich prešikrtneme tenkými čiarami.



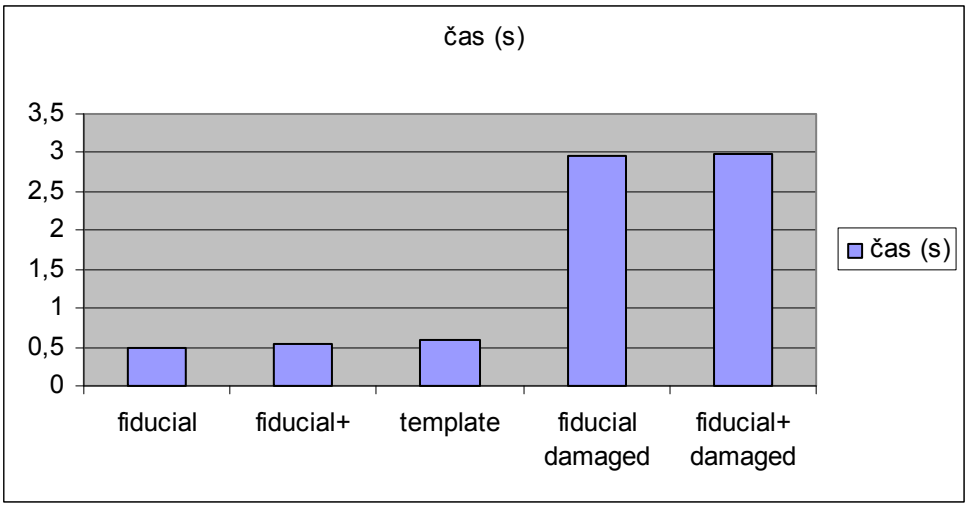
4) fiducial damaged



5) fiducial+ damaged

Výsledky:

značky	čas (s)	úspešnosť
fiducial	0,484	100%
fiducial+	0,531	100%
template	0,593	100%
fiducial damaged	2,953	100%
fiducial+ damaged	2,984	100%



Obr. 6.4: Rýchlosť rozpoznávania značiek

Časový rozdiel pri normálnych a poškodených značkách spôsobuje erózia. Tým sa nám potvrdil aj odhad časovej zložitosti, v ktorom trvá erózia približne 4,5-krát dlhšie ako samotné označovanie súvislých komponentov.

2. Naskenované testy s použitím značiek fiducial+:

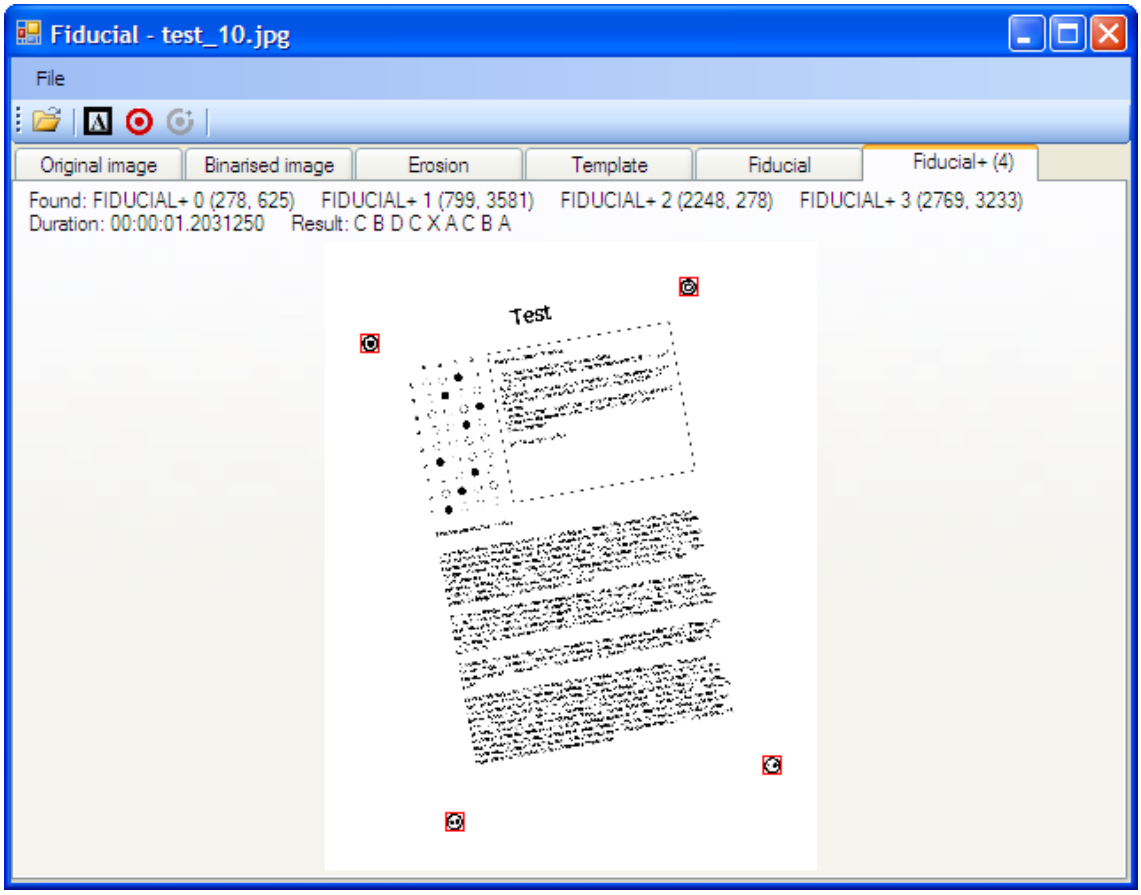
číslo	dpi	rotácia	čas (s)	erózia	fiducials+	úspešnosť
1	200	0	0,359	X	4	100%
2	200	0	0,359	X	4	100%
3	200	0	0,359	X	4	100%
4	200	0	1,5	áno	3	75%
5	200	0	0,359	X	4	100%
6	200	0	0,359	X	4	100%
7	200	0	0,359	X	4	100%
8	200	0	0,359	X	4	100%
9	200	0	0,359	X	4	100%
10	200	0	0,359	X	4	100%
11	200	0	0,359	X	4	100%
12	200	24	1,14	X	4	100%
13	200	7	0,515	X	4	100%
14	200	249	0,593	X	4	100%
15	200	119	1,75	X	4	100%
16	200	306	1,25	X	4	100%
17	200	65	0,656	X	4	100%
18	300	0	0,781	X	4	100%
19	300	0	0,796	X	4	100%
20	300	0	0,796	X	4	100%
21	300	0	0,843	X	4	100%
22	300	0	0,796	X	4	100%
23	300	0	0,781	X	4	100%
24	300	0	0,796	X	4	100%
25	300	0	0,796	X	4	100%
26	300	0	0,796	X	4	100%
27	300	0	0,828	X	4	100%
28	300	22	2,109	X	4	100%
29	300	97	1,515	X	4	100%
30	300	103	2,39	X	4	100%
31	300	52	1,64	X	4	100%
32	300	1	0,859	X	4	100%
33	300	355	0,921	X	4	100%
34	300	27	2,421	X	4	100%
35	300	341	1,203	X	4	100%
36	400	0	1,375	X	4	100%
37	400	0	1,359	X	4	100%
38	400	0	1,375	X	4	100%
39	400	0	1,421	X	4	100%
40	400	0	1,421	X	4	100%
41	400	0	6,515	áno	3	75%
42	400	0	6,39	áno	4	100%
43	400	0	1,375	X	4	100%
44	400	0	6,39	áno	3	75%
45	400	287	5,531	X	4	100%
46	400	300	7,78	X	4	100%
47	400	12	10,31	áno	3	75%
48	400	47	3,15	X	4	100%
49	400	97	2,89	X	4	100%
50	400	33	14,109	áno	4	100%

Obr. 6.5: Testovanie značiek (50 testov)

Podarilo sa nám dosiahnuť úspešnosť rozpoznania všetkých značiek 92%. Pri podrobnejšom preskúmaní nenájdených značiek zistíme, že príčina bola nekvalitným naskenovaním a teda samotný algoritmus považoval značku za viac ako jeden súvislých komponentov. V prípade počarbania vieme značky rozpoznať, len ak sa jedná o tenké čiary.

6.4.2 Vyhodnotenie testu

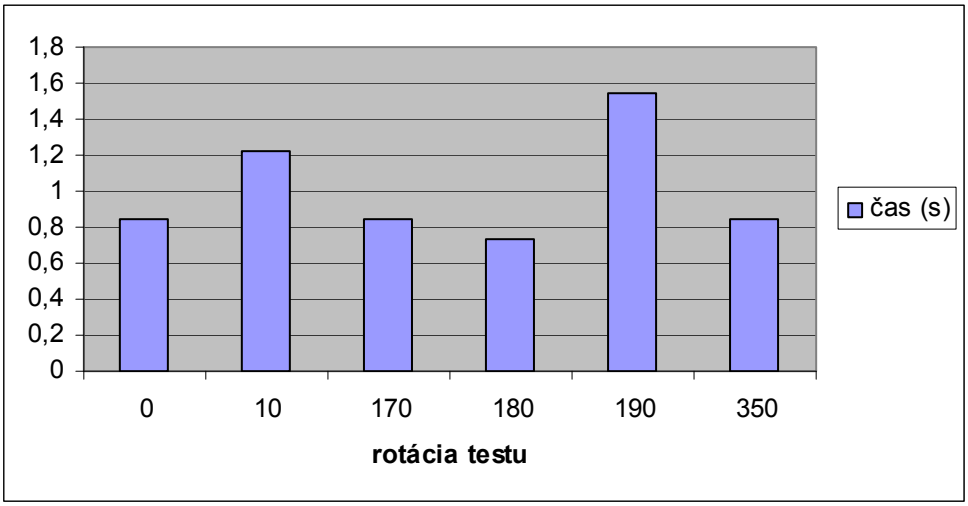
Testovacia sada pre vyhodnotenie odpoveďových políčok bude pozostávať zo 6 rovnakých testov so značkami fiducial+. Test budeme simulovať v pootočeniach 0, 10, 170, 180, 190 a 350 stupňov. Test má 9 otázok a označené odpovede sú C, B, D, C, X, A, C, B, A, pričom X znamená žiadnu neoznačenú odpoveď.



Obr. 6.6: Vyhodnotenie testu

Výsledky:

rotácia	čas (s)	správnosť
0	0,843	100%
10	1,218	100%
170	0,843	100%
180	0,734	100%
190	1,546	100%
350	0,843	100%



Obr. 6.7: Rýchlosť vyhodnotenia testu

7 Zovšeobecnenie počtu značiek

Pre účely optického rozpoznávania testov nám stačia 4 značky. Ak by sme však chceli značky použiť na inú situáciu, kde budeme potrebovať n značiek, kde n > 4, nastáva otázka akým spôsobom generovať ďalšie značky. Sformulujme teda nasledovný matematický problém:

"Máme vygenerovať n rôznych značiek a máme určiť počet kružníc, ktoré budeme potrebovať a hrúbku jednej kružnice, pričom sme obmedzený maximálnou šírkou a výškou značky."

Nech je značka vpísaná do štvorca rozmerov maxWidth x maxWidth. Budeme uvažovať dva prípady - pôvodné topologické značky (sústredné kružnice) a vylepšené topologické značky (dvojica sústredných kružníc).

Značky budeme kódovať dvoma bitmi nasledovne:

- čierne medzikružie - 1
- biele medzikružie - 0

Posledná vrstva sústredných kružníc je špeciálny prípad - kruh. Budeme ju kódovať obdobne a považovať tiež za medzikružie.

- čierny kruh - 1
- biely kruh - 0

7.1 Pôvodné topologické značky

Je zrejmé, že pomocou n medzikruží môžeme zakódovať 2^n značiek. Počet medzikruží na zakódovanie n značiek bude teda $\log_2 n$. Uvažujme minimálnu hrúbku jedného medzikružia k. Potom platí:

$$\frac{\max Width}{2\log_2 n} > k$$

7.2 Vylepšené topologické značky

V tomto prípade bude situácia troška zložitejšia. Bez ujmy na všeobecnosti môžeme vonkajšiu kružnicu zanedbať a počet medzikruží na zakódovanie n značiek bude zahŕňať len dvojicu sústredných kružníc. Pozrime sa na konkrétny príklad: Máme 2 x 2 sústredné kružnice. Pri kódovaní môžu nastať nasledovné možnosti:

11 - 11	10 - 11	01 - 11	00 - 11
11 - 10	10 - 10	01 - 10	00 - 10
11 - 01	10 - 01	01 - 01	00 - 01
11 - 00	10 - 00	01 - 00	00 - 00

10-11 totožné s 11-10

Musíme ešte vylúčiť duplikované dvojice, ktoré majú len vymenené poradie, keďže pri rozpoznávaní dvojice vnútorných kružníc nevieme povedať, ktorá je ktorá. Podčiarknuté možnosti sú teda všetky možnosti. Pomocou 4 medzikruží vieme zakódovať 10 značiek. Robili sme vlastne kombinácie s opakovaním $(2,4)=10$.

Teraz skúsime zovšeobecnenie. Pomocou n medzikruží vieme zakódovať $C'\left(2,2^{\frac{n}{2}}\right)$ značiek.

Veta: Počet medzikruží na zakódovanie n značiek je $2\log_2\left(\frac{-1+\sqrt{1+8n}}{2}\right)$.

Dôkaz: Vieme, že $C'\left(2,2^{\frac{počet}{2}}\right)=n$. Rozpíšeme si to ako kombinačné číslo:

$$\binom{2^{\frac{počet}{2}}+1}{2}=n$$

$$\frac{\left(2^{\frac{počet}{2}}+1\right)!}{\left(2^{\frac{počet}{2}}-1\right)!2!}=n$$

$$\left(2^{\frac{počet}{2}}+1\right)\left(2^{\frac{počet}{2}}\right)=2n$$

Spravíme substitúciu $x = 2^{\frac{počet}{2}}$ a vyriešime kvadratickú rovnicu $x^2 + x - 2n = 0$.

Dostávame korene $x_1 = \frac{-1+\sqrt{1+8n}}{2}, x_2 = \frac{-1-\sqrt{1+8n}}{2}$, pričom koreň x_2 môžeme hneď vylúčiť, lebo bude záporný. Dosadíme do substitúcie:

$$2^{\frac{počet}{2}} = \frac{-1+\sqrt{1+8n}}{2}$$

Kde po vyjadrení počtu medzikruží pri n značkách dostávame:

$$počet = 2\log_2\left(\frac{-1+\sqrt{1+8n}}{2}\right)$$

Uvažujme minimálnu hrúbku jedného medzikružia k. Potom platí:

$$\frac{\max Width}{4\log_2\left(\frac{-1+\sqrt{1+8n}}{2}\right)} > k$$

Dôsledok: Pôvodnými topologickými značkami vieme zakódovať viac značiek na rovnakú plochu a nemá význam uberať sa cestou pridávania n-tice sústredných kružníc, kde n > 2.

8 Záver

Problematika značiek a optického rozpoznávania testov je značne rozsiahla téma. Napriek tomu môžeme konštatovať, že sa nám podarilo splniť stanovené ciele. Spravili sme prehľad existujúcich riešení. Topologické aj geometrické značky a algoritmy na ich rozpoznávanie sme podrobili rozsiahlej analýze a zaoberali sme sa aj značkami pre kódovanie informácií. Po stanovení kritérií sme navrhli vlastné značky a samotný formulár testu.

Zvolili sme si vhodné technológie a overili naše riešenie v praxi. To nám potvrdilo správnosť výberu topologických značiek. Naprogramovali sme funkčnú Windows aplikáciu v jazyku C#. Značky sme otestovali na reálnych testoch formátu papiera A4 a porovnali výsledky. Zaznamenali sme takmer 100%-nú úspešnosť a veľmi rýchle časy behu programu. Program možno nájsť na priloženom CD, ktorého súčasťou je aj ukážková sada testov. Program je po úpravách reálne použiteľný v praxi a môže byť využitý napríklad na akademickej pôde. Značky môžu byť použité pri ľubovoľných problémoch, kde potrebujeme vymedziť určitú plochu.

Pre ďalšie smerovanie uvádzam ako možnosti zlepšenia zvýšiť odolnosť značiek a zdokonaliť návrh testovej časti. Bolo by zaujímavé spraviť generátor, ktorý dostane na vstupe jednotlivé otázky a odpovede a vráti prehľadné rozloženie formulára. Z hľadiska spracovania testov je možné implementovať hromadné načítanie a vyhodnotenie podľa zadanej matice správnych odpovedí. Testy môžu mať viac správnych odpovedí alebo žiadnu, čo potrebujeme takisto zohľadniť.

9 Zoznam použitej literatúry

[1] Efficient computation of adaptive threshold surfaces for image binarization - Ilya Blayvas, Alfred Bruckstein, Ron Kimmel: CS Department, Technion, Haifa 32000, Israel, 2005

[2] Computer Vision - Linda Shapiro, George Stockman, 2000

[3] Feature Extraction in Computer Vision and Image Processing - Mark S. Nixon, Alberto S. Aguado, 2002

[4] Object Recognition from Local Scale-Invariant Features - David G. Lowe, Computer Science, DepartmentUniversity of British Columbia, 1999

[5] Computer Vision and Applications A Guide for Students and Practitioners - Bernd Jahne, Horst Haußecker, 2000

[6] Marker Tracking and HMD Calibration for a video-based Augmented Reality Conferencing Systém - Kato, H., Billinghurst, M. (1999) - In Proceedings of the 2nd International Workshop on Augmented Reality (IWAR 99). October, San Francisco, USA

[7] Virtual Object Manipulation on a Table-Top AR Environment - Kato, H., Billinghurst, M., Poupyrev, I., Imamoto, K., Tachibana, K. (2000) - In proceedings of the International Symposium on Augmented Reality, pp.111-119, (ISAR 2000), Munich, Germany

[8] Designable Visual Markers - Costanza E., Huang J. - Full paper, to appear in ACM CHI2009, April 2009, Boston, MA, USA

[9] D-touch: a Consumer-Grade Tangible Interface Module and Musical Applications - Costanza E., Shelley S. B., Robinson. J. A. - in Proceedings of Conference on Human-Computer Interaction (HCI03), Bath September 2003

[10] A Region Adjacency Tree Approach to the Detection and Design of Fiducials - Costanza, E., Robinson, J. - in Proceedings of Vision, Video and Graphics, 2003, Bath, UK, July 2003

[11] Faster subtree isomorphism - Ron Shamir and Dekel Tsur - J. Algorithms, 33(2):267–280, 1999

[12] Improved Topological Fiducial Tracking in the reacTIVision System - Bencina, R. & Kaltenbrunner, M. & Jordà, S. - Proceedings of the IEEE International Workshop on Projector-Camera Systems (Procams 2005), San Diego (USA)

[13] reacTIVision: A Computer-Vision Framework for Table-Based Tangible Interaction - Kaltenbrunner, M. & Bencina, R. - Proceedings of the first international conference on "Tangible and Embedded Interaction" (TEI07). Baton Rouge, Louisiana.

[14] Using Camera-Equipped Mobile Phones for Interacting with Real-World objects - Michael Rohs, Beat Gfeller In: Alois Ferscha, Horst Hoertner, Gabriele Kotsis (Eds.): Advances in Pervasive Computing. Austrian Computer Society (OCG), ISBN 3-85403-176-9, pp. 265-271, Vienna, Austria, April 2004

[15] Visual Code Widgets for Marker-Based Interaction - Michael Rohs IWSAWC'05: Proceedings of the 25th IEEE International Conference on Distributed Computing Systems – Workshops (ICDCS 2005 Workshops). Columbus, Ohio, USA, June 6-10, 2005

[16] TRIP: a Low-Cost Vision-Based Location System for Ubiquitous Computing - Diego Lopez de Ipina, Paulo R. S. Mendonca, Andy Hopper, 2002

[17] Handbook of machine Vision - Alexander Hornberg, 2006

[18] Direct Least Square Fitting of Ellipses - Andrew Fitzgibbon, Maurizio Pilu, and Robert B. Fisher, 1999

[19] Template matching - http://en.wikipedia.org/wiki/Template_matching

[20] ARToolkit - http://www.hitl.washington.edu/artoolkit/

[21] Remark Office OMR - http://www.gravic.com/remark/officeomr/

[22] QR code - http://www.qrcode.com/

[23] Shotcodes - http://www.shotcode.com/

[24] Microsoft Tag - http://www.microsoft.com/Tag/

[25] AForge.NET Framework - http://www.aforgenet.com/framework/