MASARYKOVA UNIVERZITA
FAKULTA INFORMATIKY

Moderní metody identifikace objektů

BAKALÁŘSKÁ PRÁCE

Petr Šťastný

Brno, 2011

Prohlášení

Prohlašuji, že tato práce je mým původním autorským dílem, které jsem vypracoval samostatně. Všechny zdroje, prameny a literaturu, které jsem při vypracování používal nebo z nich čerpal, v práci řádně cituji s uvedením úplného odkazu na příslušný zdroj.

Poděkování

Chtěl bych poděkovat panu *prof. RNDr. Jiřímu Hřebíčkovi, CSc.* za ochotu a vedení mé bakalářské práce.

Shrnutí

Obsahem bakalářské práce je charakteristika dostupných metod rozpoznávání objektů s ohledem na použití pro aplikace v reálném čase. Práce obsahuje implementaci vybrané moderní metody v podobě vhodného typu umělé neuronové sítě a příslušného učícího algoritmu ve zvoleném softwarovém prostředí včetně zhodnocení dosažených výsledků na základě simulačních experimentů s navrženými objekty.

Klíčová slova

Obrazová scéna, popis objektu, příznakový vektor, rotace objektu, identifikace, umělá neuronová síť, aplikace v reálném čase.

Obsah

1 Úvod

Jednou z nejvíce se rozvíjejících oblastí informačních technologií je v současné době počítačové vidění. Jeho rozvoj je způsoben snahou co nejvíce přiblížit svět počítačů myšlení člověka a procesu jeho učení. Účelem je naučit počítač rozpoznávat objekty v daném prostředí na základě vizuální informace.

Úkolem počítačového vidění je získat potřebná data ze vstupních snímků. Nejprve je nutno analyzovat vstupní dvojrozměrná číselná data a najít kvalitativní informaci. Analýza vstupních dat se provádí pomocí metod pro předzpracování scény, které zahrnují filtraci, detekci hran, segmentaci apod. Pak následuje vyšší úroveň počítačového vidění, kde se vedle tradičních metod stále více prosazují rovněž metody a algoritmy z oblasti umělé inteligence.

Jedním z cílů počítačového vidění je napodobení procesu učení člověka. Tuto úlohu se snaží řešit umělé neuronové sítě. Neuronové sítě jsou vhodné pro řešení problémů, které jsou pro člověka obtížně popsatelné. Největší předností neuronových sítí je jejich schopnost učit se z předložených příkladů. Hlavní rozdíl proti klasickým výpočetním postupům spočívá v tom, že u umělých neuronových sítí nemusíme znát algoritmus řešení daného problému. Postačuje znalost určitého počtu příkladů a jejich řešení.

Cílem mé bakalářské práce je charakteristika dostupných metod rozpoznávání objektů pro aplikace v reálném čase a implementace vhodného typu neuronové sítě pro provedení simulačních experimentů s navrženými objekty ve zvoleném prostředí.

Úvodní kapitoly práce stručně seznamují s problematikou zpracování obrazu. Jedná se o předzpracování obrazu, segmentaci obrazu a metody popisu objektu. Čtvrtá kapitola je zaměřena na metody identifikace objektů se zaměřením na aplikace v reálném čase zejména v robotice. Jsou zde charakterizovány tři vhodné metody, Momentová metoda, Fourierovy deskriptory a umělá neuronová síť MLP (Multi Layer Perceptron). Praktická část práce uvádí návrh a implementaci programového systému s využitím umělé neuronové sítě MLP pro rozpoznávání vytvořených testovacích objektů v libovolné poloze v rovině.

2 Zpracování obrazu

Zpracování obrazu obvykle spočívá v posloupnosti několika kroků [4], [11]:

- Snímání, digitalizace a uložení obrazu
- Předzpracování obrazu
- Segmentace obrazu
- Popis objektu
- Rozpoznání (identifikace) objektu

U aplikací v reálném čase je z důvodu časových nároků logická snaha některé kroky zpracování obrazu vynechat, nebo alespoň minimalizovat. Jedná se zejména o předzpracování obrazu. Z důvodu zaměření této práce na identifikaci objektů jsou popis objektu a identifikace objektu uvedeny v samostatných kapitolách.

2.1 Předzpracování obrazu

Předzpracování obrazu v podstatě slouží k vylepšení obrazu pro potřeby jeho dalšího zpracování. Jedná se zejména o potlačení šumu, který vzniká hlavně při digitalizaci obrazu, nebo o zvýraznění některých parametrů obrazu. Mezi nejdůležitější metody předzpracování obrazu patří [5], [13]:

- Převod na stupně šedi
- Jas a kontrast
- Ekvalizace histogramu
- Ostření obrazu
- Filtrace (průměrováním, mediánem, rotující maskou, Gaussovým filtrem aj.)

2.2 Segmentace obrazu

Proces segmentace spočívá v rozdělení obrazové scény na samostatné části, které představují objekty a pozadí. Výsledkem segmentace jsou disjunktní oblasti. Mezi nejdůležitější metody segmentace obrazu patří [12], [14]:

- Prahování
- Barvení objektu
- Detekce hran
- Spojování hran
- Vyplňování objektů

Z důvodu použití u Momentové metody je blíže popsáno barvení objektu. Rovněž z důvodu použití při lokalizaci objektů je uvedena Sobelova detekce hran.

2.2.1 Barvení objektu

U této metody se obraz prochází po řádcích [11], [13]. Každému nenulovému obrazovému elementu $f(i, j)$ se přiřadí hodnota podle hodnoty jeho sousedních elementů, pokud sousední elementy existují. Algoritmus pro barvení objektů lze popsat následujícími kroky:

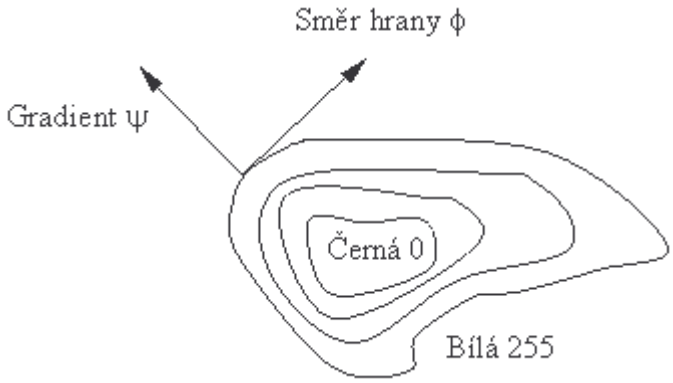
1. *<u>první průchod</u>:* Obraz se prochází po řádcích dokud není nalezen obrazový element $f(i, j)$. Každému nenulovému obrazovému elementu $f(i, j)$ se přiřadí hodnota podle hodnoty jeho sousedních elementů, pokud sousední elementy existují (poloha sousedů je dána maskou na obr. 2.1). Všechny sousední elementy dané maskou již byly v předchozích krocích obarveny.
   - Jsou-li všechny sousední elementy částí pozadí (mají nulovou hodnotu), přiřadí se obrazovému elementu dosud nepřidělená hodnota (barva).
   - Má-li právě jeden ze sousedních elementů nenulovou hodnotu, přiřadí se obarvovanému elementu hodnota tohoto sousedního elementu.
   - Je-li nenulových více sousedních obrazových elementů, přiřadí se obarvovanému elementu hodnota kteréhokoli nenulového obrazového elementu ze zkoumaného okolí. Jestliže nebyly hodnoty sousedních elementů různé (tzv. kolize barev), zaznamená se ekvivalentní dvojice do tabulky ekvivalence barev.
2. *<u>druhý průchod:</u>* Po průchodu celého obrazu jsou všechny obrazové elementy náležející oblastem obarveny, některé oblasti jsou však obarveny více barvami (kolize barev). Proto se obraz projde po řádcích ještě jednou a podle informace o ekvivalenci barev se přebarví obrazové elementy dané oblasti. Každé oblasti odpovídá označení (obarvení) jedinou, v jiné oblasti se nevyskytující hodnotou (barvou).

(i – 1, j – 1)	(i, j – 1)	(i + 1, j – 1)
(i – 1, j)	(i, j)	(i + 1, j)
(i – 1, j + 1)	(i, j + 1)	(i + 1, j + 1)

Obr. 2.1: Maska barvení pro 8-okolí

2.2.2 Detekce hran

Změna obrazové funkce může být popsána *gradientem* [12], [14]. Gradient ∇ je směr největšího růstu obrazové funkce.



Obr. 2.2: Orientace směru hrany a gradientu

Směr gradientu (obr. 2.2) udává směr maximálního růstu funkce (od černé po bílou). Hrany jsou kolmice na směr gradientu.

Dělení gradientních opererátorů :

- Operátory aproximující derivace pomocí diferencí. Operátory invariantní vůči rotaci se realizují jedinou konvoluční maskou.
- Operátory odhadující první derivaci používají několik masek. Směr gradientu se odhaduje hledáním masky, která odpovídá největší velikosti gradientu.
- Operátory založené na hledání hran v místech, kde druhá derivace obrazové funkce prochází nulou *(zero crossing)*.
- Operátory snažící se aproximovat obrazovou funkci jednoduchým parametrickým modelem, např. polynomem dvou proměnných.

Sobelova detekce hran [12], [13], [14] zvýrazňuje všechny hrany obsažené v obraze bez ohledu na směr. Algoritmus je aplikován jako vektorový součet dvou směrových hranových operátorů. Výsledný obraz je z původního transformován tak, že místa s konstantní hodnotou jasu jsou transformována do černých ploch, naopak místa s měnícím se jasem jsou transformována do bílých míst.

Konvoluční masky Sobelova operátoru jsou :

$$h_1 = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \qquad h_2 = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \tag{2.1}$$

Pomocí h_1 je možné detekovat vertikální hrany a pomocí jádra h_2 horizontální hrany v obraze.

3 Popis objektu

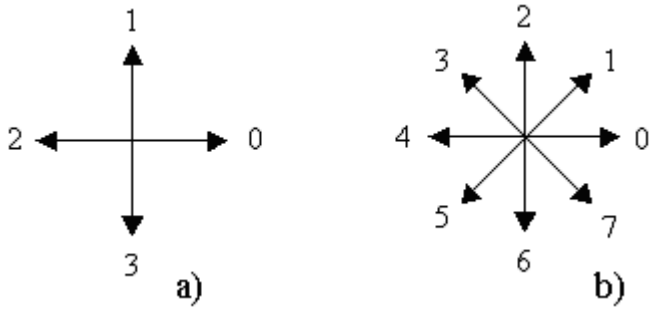
Rozpoznávání vyžaduje exaktní popis objektu tak, aby tento popis mohl být předložen pro identifikaci objektu. Významné je, zda navržený způsob reprezentace vede k příznakovému nebo strukturálnímu popisu [7], [5], [4]. V této práci pracuji s příznakovým popisem. Získání vektoru příznaků jednotlivých objektů je jedním z nejdůležitějších kroků celého postupu zpracování obrazu. Uvedené příznakové metody pro popis objektů:

- Řetězcové kódy
- Diferenciální řetězcové kódy
- Ramena objektu

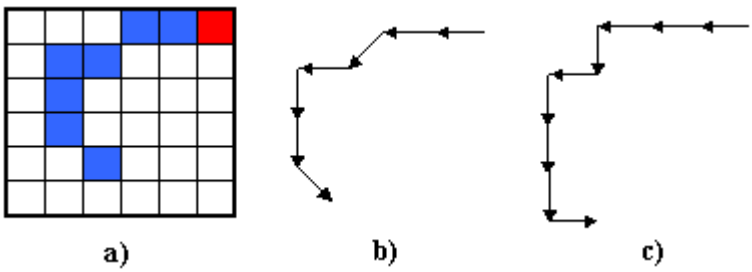
3.1 Řetězcové kódy

Freemanovy řetězcové kódy [4], [6] se používají k popisu hranic objektů. Hranice je určena počátečním bodem a posloupností symbolů, odpovídajících úsečkám jednotkové délky v několika předem stanovených směrech. K popisu hranice se používají dvě varianty řetězcových kódů:

- 4 směry (obr. 3.1 a, obr. 3.2 a,c)
- 8 směrů (obr. 3.1 b, obr. 3.2 a,b)



Obr. 3.1: Řetězcové kódy



Obr. 3.2: Příklad řetězcového kódu

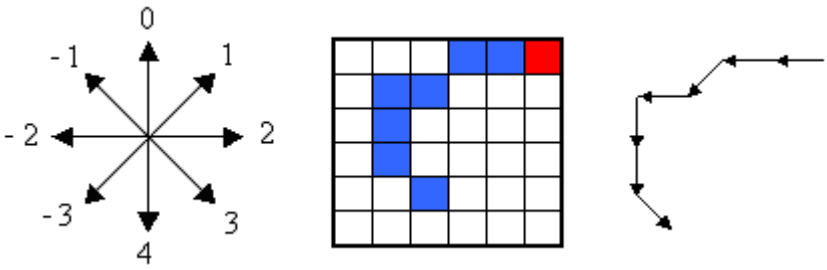
Příklad (obr. 3.2):

- Řetězcový kód pro 4 směry: 2,2,2,3,2,3,3,3,0
- Zápis pomocí exponentů pro 4 směry: $2^3, 3^1, 2^1, 3^3, 0$
- Řetězcový kód pro 8 směrů: 4,4,5,4,6,6,7
- Zápis pomocí exponentů pro 8 směrů: $4^2, 5^1, 4^1, 6^2, 7^1$

Z uvedeného příkladu je zřejmé, že řetězcový kód pro 8 směrů je kratší než řetězcový kód pro 4 směry. Freemanův řetězcový kód je invariantní vůči posunutí, nesplňuje však požadavek invariance vůči rotaci.

3.2 Diferenciální řetězcové kódy

Diferenciální řetězcový kód (obr. 3.3) je modifikací Freemanova řetězcového kódu. Vznikne jako rozdíl dvou následujících kódů [6]. Hladké křivky mají obvykle hodnoty 0, +1, -1 .



Obr. 3.3: Diferenciální řetězcový kód

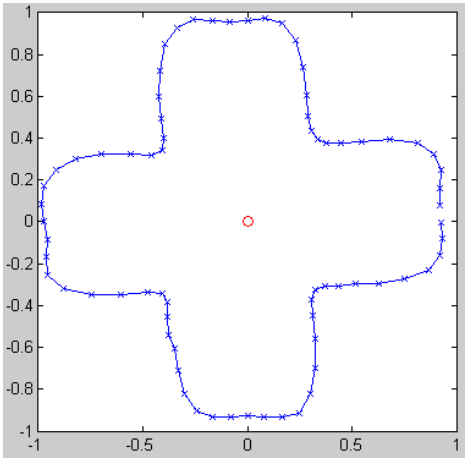
Příklad (obr. 3.3):

- Zápis diferenciálního kódu: 0, 0,-1, 1,-2, 0,-1

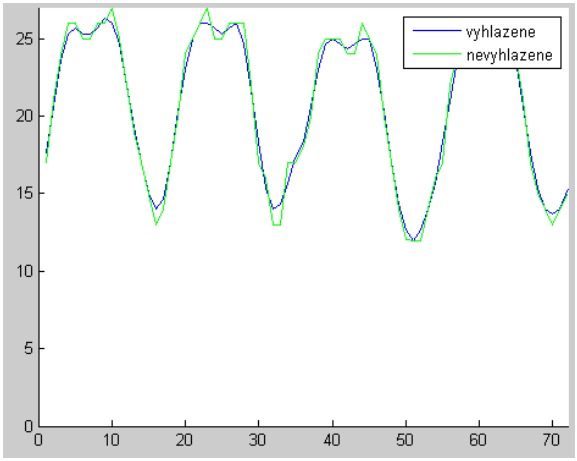
Předností Diferenciálního řetězcového kódu je, že je na rozdíl od předchozího Freemanova řetězcového kódu invariantní nejen vůči posunutí, ale i vůči rotaci.

3.3 Ramena objektu

Ve vytvořené aplikaci je použit příznakový popis pomocí metody Ramena objektu. Tato metoda popisu objektu je založena na výpočtu polárních souřadnic bodů obrysové křivky daného objektu. Implementovaná metoda vychází z metody, která byla vyvinuta na VUT v Brně [2]. Její algoritmus je uveden v praktické části práce. Daný objekt je zde jednoznačně určen vektorem číselných charakteristik, tzv. příznakovým vektorem. Příklad objektu je na obr. 3.4 a jeho příznakového vektoru na obr. 3.5.



Obr. 3.4: Objekt kříž



Obr. 3.5: Příznakový vektor objektu kříž

Příznakový vektor, vypočtený pomocí metody Ramena objektu, poměrně dobře charakterizuje vnější hranu objektu a tím i kvalitní popis objektu. Nedostatkem tohoto algoritmu je, že nedetekuje vnitřní hrany a přerušení (mezi těžištěm a koncovým bodem ramene z něho vycházejících) v objektech. Tvarově shodné objekty lišící se pouze vnitřní hranou tak nemusí být správně klasifikovány. Tento nedostatek je možné odstranit např. rozšířením vektoru o délky ramen prvního průsečíku – nejprve by se ale musel upravit algoritmus vyhledávání hran tak, aby dokázal správně spojit vnitřní a vnější hranu jednoho objektu do dvojice. Takto vzniklý vektor by se dal předkládat např. dvěma neuronovým sítím zároveň a objekt určit až na základě identifikovaných vnějších a vnitřních obrysů.

Ukončením popisu dojde ke změně reprezentace obrazových dat ze segmentovaných obrazů na geometrickou reprezentaci. To s sebou mimo jiné nese výrazný úbytek zpracovávaných dat, což je výhodné pro aplikace v reálném čase.

4 Metody identifikace objektů

4.1 Momentová metoda

Metoda [4], [8] je založena na výpočtu sedmi momentových příznaků objektu. Pro výpočet momentů je nutné znát pozici všech pixelů uvnitř uzavřené hranice. Proto je nutné použít algoritmus vyplňování uzavřené hranice [10] nebo algoritmus barvení (kap. 2.2.1). Momentové vyjádření objektů je invariantní vůči natočení, velikosti a posunutí objektu.

Pro dvourozměrnou funkci *f(x,y)* je obecný moment řádu *(p+q)* definován vztahem:

$$m_{pq} = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} x^p y^q f(x,y)dxdy \tag{4.1}$$

Pro zajištění nezávislosti momentu na poloze a natočení daného objektu se posune souřadný systém do jeho těžiště (obr. 4.1). Centrální moment lze pak zapsat vztahem:

$$\mu_{pq} = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} (x-x_t)^p (y-y_t)^q f(x,y) \tag{4.2}$$

kde x_t, a y_t představují souřadnice těžiště, pro něž platí:

$$x_t = \frac{m_{10}}{m_{00}} \qquad\qquad y_t = \frac{m_{01}}{m_{00}} \tag{4.3}$$

Centrální momenty do řádu 3 invariantní k posunu a rotaci objektu lze zapsat vztahy:

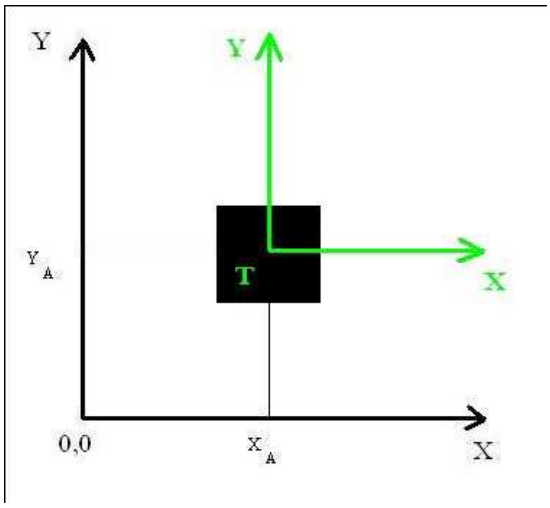
$$\mu_{00} = m_{00}, \quad \mu_{11} = m_{11} - y_t m_{10} \tag{4.4}$$

$$\mu_{10} = 0, \quad \mu_{30} = m_{30} - 3x_t m_{20} + 2m_{10}x_t^{\,2} \tag{4.5}$$

$$\mu_{01} = 0, \quad \mu_{12} = m_{12} - 2y_t m_{11} - x_t m_{02} + 2y_t^{\,2} m_{10} \tag{4.6}$$

$$\mu_{20} = m_{20} - x_t m_{10}, \quad \mu_{21} = m_{21} - 2x_t m_{11} - y_t m_{20} + 2x_t^{\,2} m_{01} \tag{4.7}$$

$$\mu_{02} = m_{02} - x_t m_{10}, \quad \mu_{03} = m_{03} - 3y_t m_{02} + 2y_t^{\,2} m_{01} \tag{4.8}$$



Obr. 4.1: Posun souřadného systému do těžiště

Algoritmus pro Momentovou metodu lze popsat kroky:

- Procházení obrazové scény postupně po jednotlivých bodech
- Určení výchozího bodu hledaného objektu v obrazové scéně
- Vyplnění daného objektu pomocí algoritmu barvení
- Uložení obarveného objektu do paměti
- Pokud splňuje daný objekt podmínku minimální velikosti plochy, pokračování dalším krokem, jinak návrat na začátek
- Výpočet příslušných momentů pro daný objekt
- Identifikace daného objektu
- Ukončení algoritmu

4.2 Fourierovy deskriptory

Fourierovy deskriptory [1], [2] vychází z Fourierovy transformace. Pokud se omezíme na konečný počet spektrálních složek, pak definované Fourierovy deskriptory umožňují formální popis dvourozměrných předmětů, jejichž tvar je vyjádřen uzavřenou křivkou s konečným počtem číselných charakteristik. Díky zavedené normalizaci a základním vlastnostem Fourierových řad má reprezentace Fourierovými deskriptory tyto vlastnosti:

- Nalezené Fourierovy deskriptory mohou sloužit jako příznakový popis dvourozměrných předmětů. Pro takto formálně popsané objekty lze využít pro jejich klasifikaci známých metod rozpoznávání příznakově popsaných objektů.

- Fourierovy deskriptory jsou invariantní vůči posunutí objektu a změně měřítka, v uvedené podobě nejsou invariantní vůči rotaci.
- Lze nalézt vztahy mezi geometrickými vlastnostmi objektu a jeho spektrem.
- Formální popis Fourierových deskriptorů je nezávislý na fyzikálním původu klasifikovaných signálů. Lze jej použít ve všech případech, kdy hranice dvojrozměrného průmětu objektu je reprezentována uzavřenou křivkou.
- Pro výpočet Fourierových deskriptorů je možné použít algoritmus rychlé Fourierovy transformace (FFT).
- Z uložených Fourierových deskriptorů lze křivku znovu rekonstruovat. Při této rekonstrukci dochází k vyhlazení křivky omezením vyšších harmonických složek.

4.2.1 Výpočet Fourierových deskriptorů

Odvození vztahů (4.9) a (4.10) pro Fourierovy deskriptory je uvedeno v [2].

$$a_n = -\frac{1}{n\pi}\sum_{k=1}^{m}\Delta\gamma_k \sin\left(\frac{2\pi n S_k}{L}\right) \tag{4.9}$$

$$b_n = -\frac{1}{n\pi}\sum_{k=1}^{m}\Delta\gamma_k \cos\left(\frac{2\pi n S_k}{L}\right) \tag{4.10}$$

kde: n – je počet Fourierových deskriptorů
m – je počet bodů hranice
$\Delta\gamma$ – je změna směrnice od předchozího bodu
S_k – je délka oblouku od startovacího bodu do vrcholu s indexem k
L – je obvod objektu

Rovnice (4.9) a (4.10) umožňují na základě znalosti hraničních bodů křivky a změny směrnice $\Delta\gamma_i$ vyčíslit Fourierovy deskriptory podle jednoduchých vztahů.

Transformace koeficientů a_n a b_n:

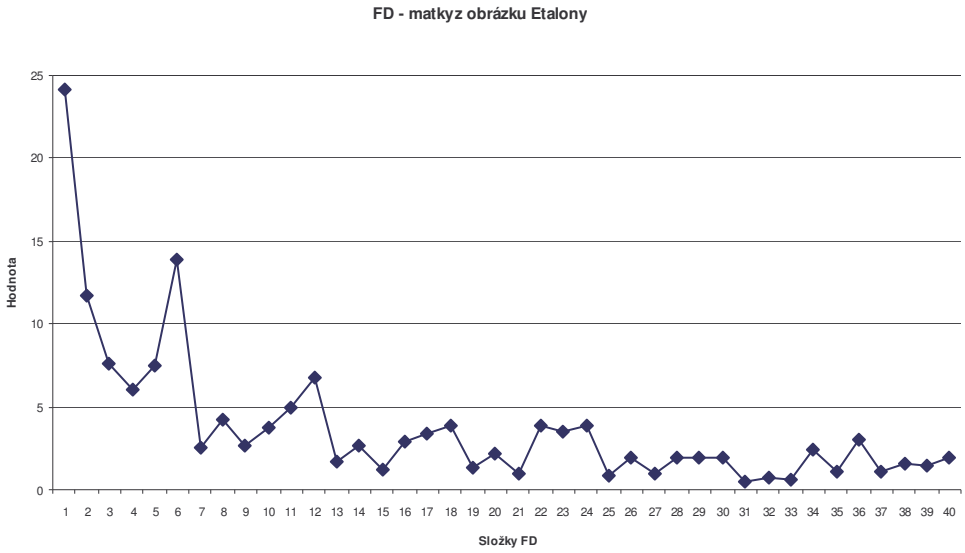
$$r_n = \sqrt{(a_n)^2 + (b_n)^2} \tag{4.11}$$

Normalizace pro zajištění nezávislosti na měřítku:

$$w_n = \frac{r_n}{r_1} \tag{4.12}$$

Pro popis objektů v obrazové scéně stačí 15 až 20 Fourierových deskriptorů, vyšší složky se již významně nemění. Na obr, 4.2 je zobrazeno 40 Fourierových deskriptorů. Je zřejmé, že hodnota

Fourierových deskriptorů s velikostí složky klesá. V některých případech je vhodné vynechat první dva Fourierovy deskriptory, bývají nejvíce zatíženy chybou.



Obr. 4.2: Fourierovy deskriptory pro objekt šestihran

4.3 Umělé neuronové sítě

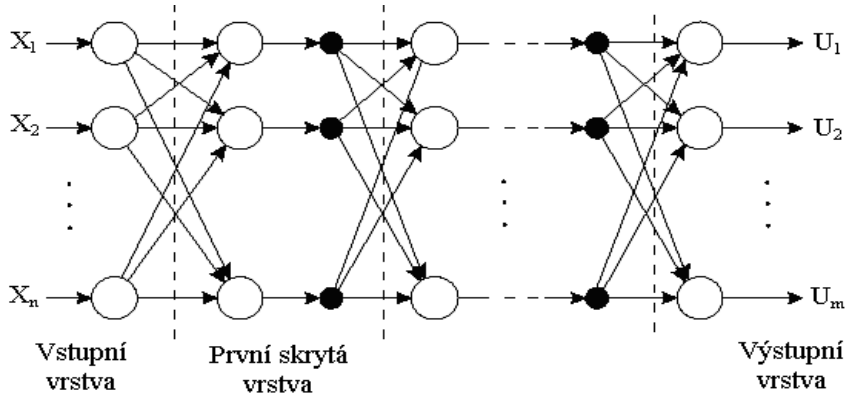
Mezi umělými neuronovými sítěmi a konvenčními výpočetními metodami existuje zásadní rozdíl ve způsobu práce. Konvenční systémy pracují na základě daného algoritmu, podle nějž zpracovávají jednotlivé dílčí operace. Na rozdíl o nich neuronové sítě uskutečňuji vysoký počet dílčích operací současně a pracují bez algoritmu. Činnost neuronové sítě je založena na procesu učení, při němž se síť postupně co nejlépe adaptuje na řešení daného problému [9], [15]. Pomocí umělých neuronových sítí lze možno řešit poměrně složité úlohy, mezi něž patří také identifikace objektů.

Umělá neuronová síť pracuje v zásadě ve dvou fázích - adaptivní a aktivní. V adaptivní fázi se síť učí, v aktivní vykonává naučenou činnost. Při učení dochází v neuronové síti ke změnám, síť se adaptuje na řešení daného problému. Učení se v umělé neuronové síti realizuje nastavováním vah mezi uzly. V praxi se váhám přisoudí počáteční hodnoty. Pak se do sítě přivede trénovací vstup. Síť pak poskytne výstup, odezvu [9].

Učení neuronové sítě může probíhat s učitelem nebo bez učitele. Pří učení s učitelem se umělá neuronová síť učí srovnáváním aktuálního výstupu s výstupem požadovaným a nastavováním vah tak, aby se snížil rozdíl mezi skutečným a žádaným výstupem. Mezi metody učení s učitelem patří algoritmus Back-propagation [15].

4.3.1 Vícevrstvá perceptronová síť

Vícevrstvá perceptronová síť (obr. 4.3), označovaná jako MLP (Multi Layer Perceptron), je acyklická dopředná síť. Neurony lze disjunktně rozdělit do vrstev tak, že výstupy každého neuronu jedné vrstvy jsou napojeny na vstupy každého neuronu vrstvy následující. Neexistují žádné vazby mezi neurony nesousedních vrstev ani mezi neurony stejné vrstvy. Každý neuron má právě tolik vstupů, kolik je neuronů v nižší vrstvě. Vstupní vrstva slouží pouze k distribuci vstupních hodnot do první skryté vrstvy. Síť s jednou skrytou vrstvou a jednou vrstvou výstupní se označuje jako síť *dvouvrstvá,* síť se dvěma skrytými vrstvami jako *třívrstvá,* atd. [15], [9].



Obr. 4.3: Vícevrstvá perceptronová síť

Jako učící algoritmus pro neuronovou síť MLP byl použit algoritmus *Back-propagation*, který je vhodný pro učení vrstvených sítí. Učení probíhá metodou učení s učitelem. Podle způsobu, jakým se vypočítávají váhy, lze algoritmus označit za iterační. Síť se dostává z počátečního nenaučeného stavu do stavu úplného naučení. Energetická funkce je minimalizována na základě gradientu, proto je také tento algoritmus zařazován mezi *gradientní* metody učení [9].

4.4.2 Algoritmus Back-propagation

Princip algoritmu spočívá v testování, zda neuronová síť odpovídá na vstupní vektor přesně podle trénovací množiny. V případě, že síť nereaguje podle požadavků, je nutné měnit váhové koeficienty tak dlouho, dokud nezačne reagovat správně.

Back-propagation je iterační proces, ve kterém se síť dostává z počátečního nenaučeného stavu do stavu plného naučení. Algoritmus lze stručně popsat následujícími kroky [15]:

1. *Inicializace*

   Nastaví všechny váhy v síti na náhodné malé hodnoty. Doporučený rozsah je z intervalu $\langle -0.3, 0.3 \rangle$.

2. *Předložení vzoru*

Z trénovací množiny se vybere vzor a vloží na vstupy sítě. Dále po vrstvách směrem od vstupní k výstupní se počítají výstupy jednotlivých neuronů dle vztahu:

$$u = \sigma\left(\sum_{i=0}^{n} x_i w_i\right) \tag{4.13}$$

kde $\sigma(\varphi) = \frac{1}{1+e^{-\varphi}}$ je aktivační funkce.

3. *Srovnání*

Vypočte se energie (chyba) sítě dle vztahu:

$E = \frac{1}{2}\sum_{i=1}^{n}(u_i - d_i)^2$, kde n je počet neuronů výstupní vrstvy, u_i je skutečná odezva i-tého neuronu ve výstupní vrstvě na vstupní vektor a d_i je očekávaný výstup i-tého neuronu ve výstupní vrstvě. Tato energie bude využita jako přírůstek k celkové energii počítané přes všechny vzory. Dále se vypočte chyba pro výstupní vrstvu:

$$\delta_i^u = (d_i - u_i^u)(1 - u_i^u)u_i^u \tag{4.14}$$

4. *Zpětné šíření chyby*

Pro všechny neurony ve vrstvě l se vypočtou váhy:

$$\Delta w_{ij}^l(t) = \eta \delta_i^l(t) u_j^{l-1}(t) + \mu \Delta w_{ij}^l(t-1) \tag{4.15}$$

$$\Delta \theta_i^l(t) = \eta \delta_i^l(t) + \mu \Delta \theta_i^l(t-1) \tag{4.16}$$

kde η je koeficient učení a μ je koeficient vlivu změny vah z předchozího kroku. Oba dva koeficienty nabývají hodnot z intervalu $\langle 0,1 \rangle$. Pokud je $\mu = 0$, pak minulé nastavení váhy nemá vliv na nové nastavení a hodnota změny synaptické váhy je dána pouze aktuálním krokem. Čím větší je koeficient učení η, tím větší budou změny v neuronové síti a naopak. Pak se podle vztahu:

$$\delta_i^{h-1} = y_i^{h-1}\left(1 - u_i^{h-1}\right)\sum_{k=1}^{n} w_{ki}^h \delta_k^h \tag{4.17}$$

chyba šíří zpětně do vrstvy, která je blíže vstupům. Na závěr tohoto kroku se modifikují váhy:

$$w_{ij}^l(t+1) = w_{ij}^l(t) + \Delta w_{ij}^l(t) \tag{4.18}$$

$$\theta_i^l(t+1) = \theta_i^l(t) + \Delta \theta_i^l(t) \tag{4.19}$$

Krok 4 se opakuje pro všechny vrstvy sítě. Začíná se vrstvou výstupní a pak následují skryté vrstvy. V případě, že se zpracovává skrytá vrstva, která je nejblíže vstupní vrstvě, nahradí se u_j^{l-1} ve vztahu (6.3) odpovídající vstupní hodnotou, tedy x_j. Pak platí:

$$\Delta w_{ij}^{l}(t) = \eta \delta_i^{l}(t) x_j(t) + \mu \Delta w_{ij}^{l}(t-1) \tag{4.20}$$

5. *Konec výběru vzorů z trénovací množiny*
   Jestliže jsme předložili síti všechny vzory z trénovací množiny, pak se pokračuje krokem 6, jinak návrat na krok 2.
6. *Konec učení*
   Jestliže byla energie neuronové sítě po posledním výpočtu menší než zadané kritérium, učení se ukončí, jinak se pokračuje krokem 2.

Během učení neuronové sítě dochází k poklesu energetické funkce. Pro dobře naučenou síť konverguje chyba po nekonečném počtu učících cyklů k nule.

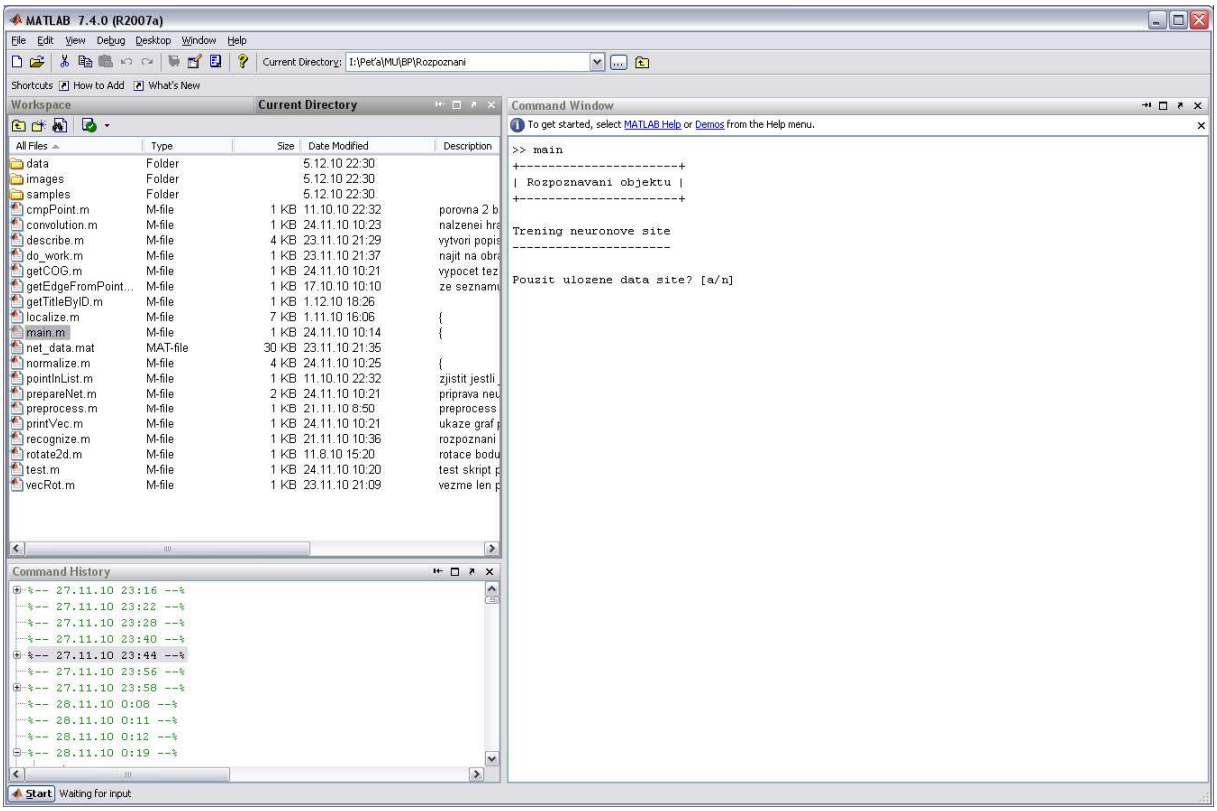
Algoritmus *Back-Propagation* lze dále vylepšit zavedením proměnného kroku učení η. Tímto způsobem se dá učení zrychlit. Síť se učí tím rychleji, čím je krok učení větší. Nesmí však přesáhnout jistou hranici, jinak se učení stane nestabilním. Na začátku je krok učení větší, síť nejdříve najde „hrubé“ hodnoty vah a neuronů, a postupně se zpřesňováním vah dochází ke snižování kroku učení v závislosti na typu výpočtové křivky [9].

5 Praktická část

Cílem praktické části je návrh a implementace programového systému s využitím umělé neuronové sítě MLP pro rozpoznávání daných objektů v libovolné poloze v rovině. Řešení zahrnuje implementaci algoritmů pro popis objektu na základě určeného těžiště a pro zajištění invariance vůči rotaci objektu.

5.1 Poznámky k implementaci

Programový systém pro rozpoznávání objektů byl vytvořen v kódu vývojového prostředí MATLAB od firmy Mathworks prostřednictvím zdrojových souborů, tzv. *m-files* (viz obr. 5.1) s využitím modulu *Neural Network toolbox* pro rozpoznávání objektů pomocí umělé neuronové sítě.



Obr. 5.1: Prostředí MATLAB s pracovním adresářem

Vytvořená aplikace je spustitelná pomocí souboru *main.m* (viz obr. 5.1) v příkazovém okně (Command Window) prostředí MATLAB. Ukázka zdrojového kódu hlavního souboru *main.m* :

```
%{
    Program pro identifikaci objektů

    - lokalizuje objekty v obrazové scéně
    - natočí objekty
    - rozpozná objekty pomocí příznakového vektoru

    Petr Šťastný, 2010
%}
```

```
function main(demo)
    global DEMO_MODE step range stepSeek;

    if nargin == 0
        demo = false;
    end

    %konfigurace

    DEMO_MODE = demo;

    step = 4;               % krok: <--->*<--->*
    range = 2;              % prohledavana oblast
    stepSeek = 2 * step;    % krok pri prohledavani

    %start
    fprintf('+----------------------+\r| Rozpoznavani objektu |\r+---------
    ------------+\r\r');

    %file_path = input('Zadej cestu k souboru: ');
    file_path = 'images\all.png';

    fprintf('Trening neuronove site\r----------------------\r\r');
    classifier = prepareNet('samples\files.txt');

    fprintf('\r\rRozpoznani\r----------\r\r');
    elements = do_work(file_path, classifier);
end
```

Získání obrazu

Ve vytvořené aplikaci je zdrojem obrazu soubor, což je vhodné hlavně při testování. Obraz je získán jako tři matice pixelů daného rozlišení, pro každou základní složku barvy RGB (červená, zelená a modrá) zvlášť. Pomocí modulu *Image Acquisition Toolbox,* příp. také modulu *Image Processing Toolbox,* lze získat obraz také z připojené kamery.

Preprocessing

V tomto kroku je barevný obraz převeden z RGB do stupňů šedi, což ve výsledku urychluje celý proces, protože se nepracuje s třemi velkými maticemi, ale jen s jednou.

Detekce hran

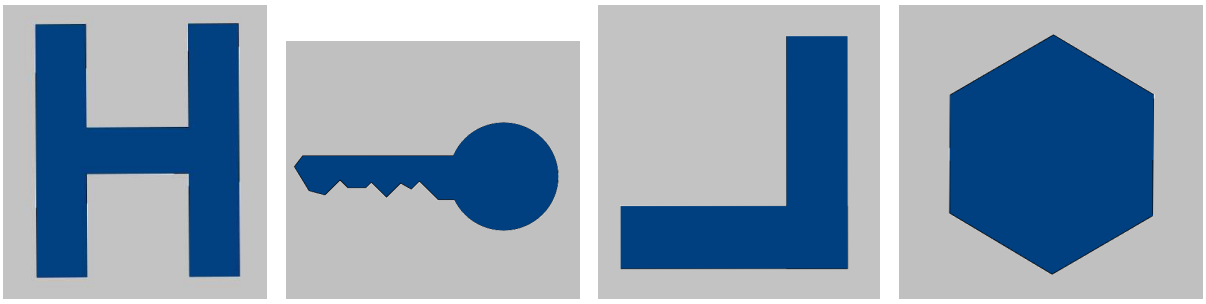
Pro hranovou detekci byl použit Sobelův operátor, který reaguje na změny jasu obrazu. Proto je vhodné zajistit, aby objekty, které jsou předkládány k identifikaci měly kontrastní pozadí vůči jejich barvě.

Postprocessing

Variantně lze doplnit krok, v němž je použit zdrojový obraz v RGB a pomocí nastaveného RGB prahu se z obrazu, který vznikne v kroku detekce hran, odstraní oblasti, které nejsou důležité. Například u světlých objektů je možné odstranit tmavá místa.

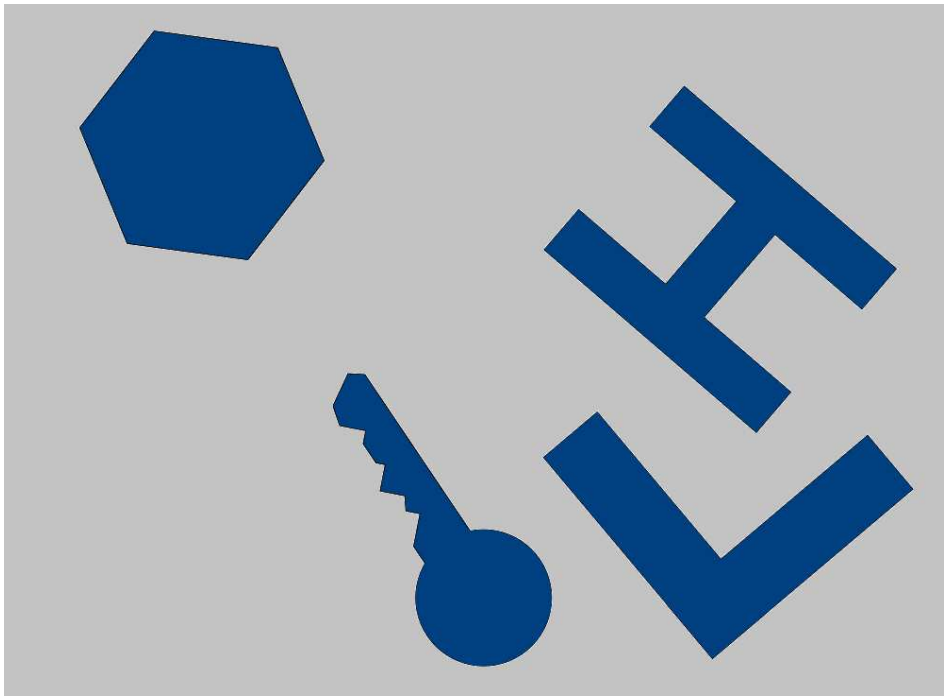
5.2 Testovací objekty

Pro účely testování byly vytvořeny čtyři různé dvojrozměrné objekty: H-profil, klíč, L-profil, šestihran (obr. 5.2). Tyto objekty představují etalony pro naučení umělé neuronové sítě.



Obr. 5.2: Objekty pro testování

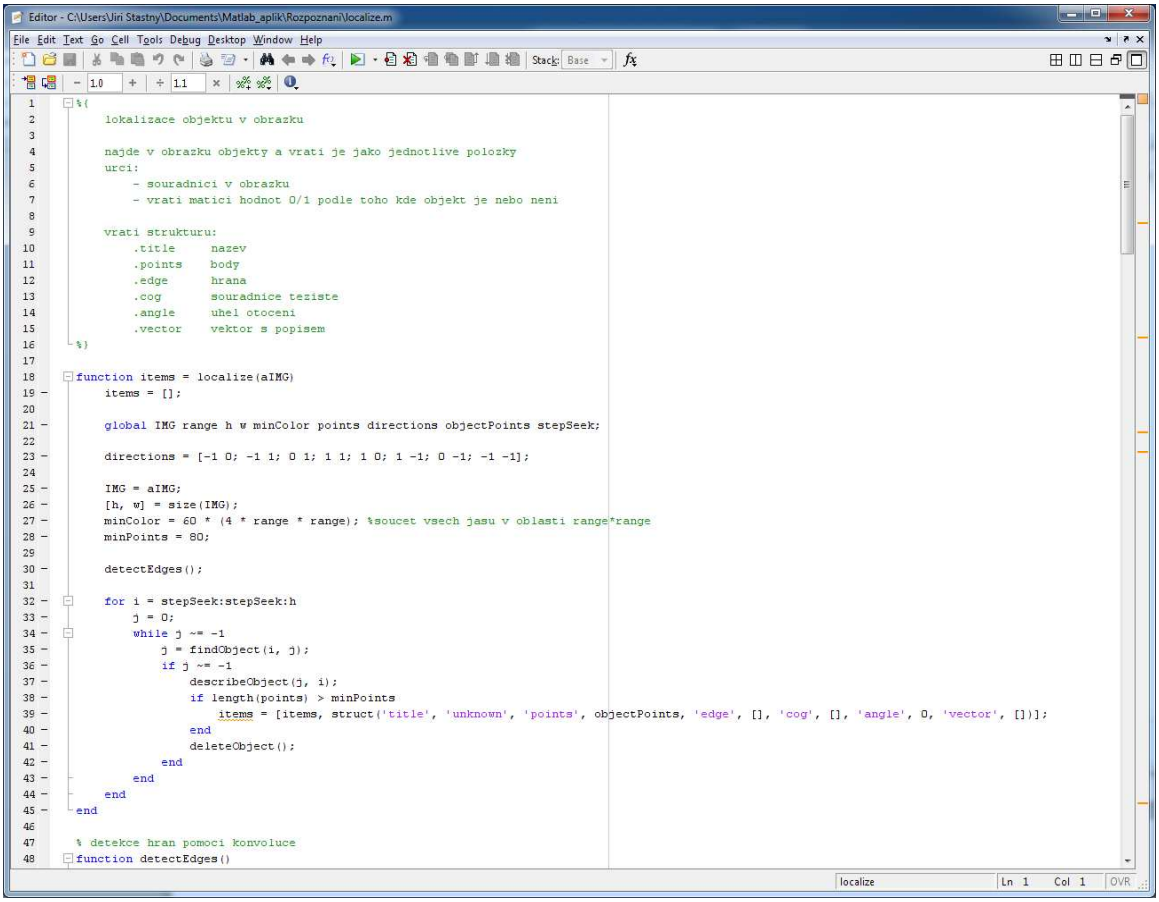
Z uvedených objektů rozmístěných v obecné poloze byly vytvořeny testovací obrazové scény pro rozpoznávání neuronovou sítí. Ukázka testovací obrazové scény je na obr. 5.3.



Obr. 5.3: Obrazová scéna pro testování

5.3 Lokalizace a popis objektu

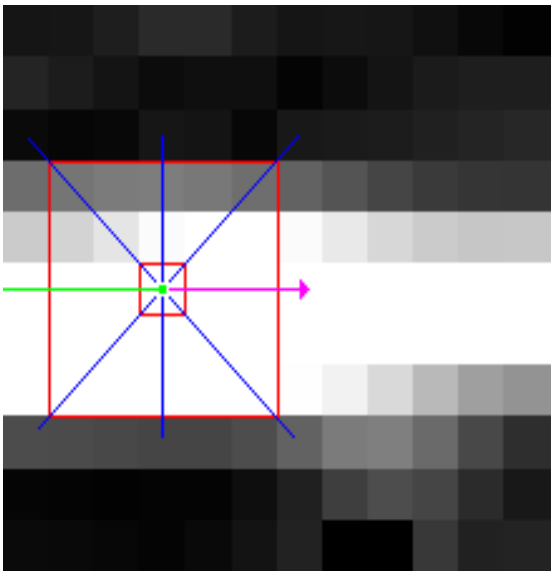
Úkolem lokalizace objektu (obr. 5.4) je určení jeho polohy v testované obrazové scéně. Na základě souřadnic lokalizovaných objektů je pak proveden výpočet těžiště každého objektu.



Obr. 5.4: Okno souboru pro lokalizaci objektu v obrazové scéně

Po aplikaci hranového detektoru vznikne obrazová matice, ve které lze hrany objektů zjistit pomocí vysokých hodnot jednotlivých prvků matice. Aby bylo možné z této obrazové matice vyčlenit jednotlivé objekty, je potřeba najít jednotlivé oblasti, kde je součet hodnot na prvcích obrazové matice větší, než předem stanovený práh. Předpokládá se také, že na jedné obrazové scéně může být více objektů.

Detekce hran pak probíhá tím způsobem, že se nejdříve vyhledává bod, kde je větší součet jasu v nastaveném okolí (oblast je v obr. 5.5 vyznačena velkým červeným čtvercem) než nastavený práh. V dalším kroku se od tohoto bodu postupuje do všech osmi možných směrů a zapisují se body (středy oblastí), kde je opět celkový jas v oblasti vyšší než práh.



Obr. 5.5: Detail detekce hrany

Z nalezených bodů se opět postupuje do všech osmi možných směrů s tím, že prioritní je směr, ze kterého se algoritmus do daného bodu dostal (zelená – vstupní krok, fialová – následující krok). Ostatní směry jsou testovány až v případě, že není v prvním směru nalezen dostatečný jas. Oblasti, které algoritmus již vyhodnotil jsou v obrazové matici nulovány, aby nebyly znovu vyhodnocovány. Jakmile algoritmus nenajde další bod, je detekce ukončena a body (hodnoty 0 nebo 1) jsou předány v matici k dalšímu zpracování. Dále je vyhledána další oblast s vysokým jasem, která může představovat jiný objekt.

Prohledáváním a vyhodnocováním větších oblastí se získá menší výsledná matice s detekovanou hranou objektu. Poměr zmenšení je dán tím, jak velké jsou nastavené vyhodnocovací oblasti.

Popis objektů

Objekty jsou popsány pomocí ramen, která vycházejí z těžiště a končí na vnější hraně objektu. Proto je nutné nejprve najít těžiště objektu. Těžiště se pak vypočte pouze z bodů, které určují hranici objektu.

Postup výpočtu těžiště:

- inicializace proměnných pro součet souřadnic x a y
- pro každý prvek matice s hodnotou 1 se přičte do sumarizačních proměnných x a y index prvku k_i resp. l_i a proměnná c se zvýší o 1
- výpočet souřadnic bodu těžiště se vypočte pomocí vztahů (5.1) a (5.2)

$$x_t = \frac{\sum_{1}^{n} k_i}{c} \tag{5.1}$$

$$y_t = \frac{\sum_{1}^{n} l_i}{c} \tag{5.2}$$

Z bodu těžiště jsou pak spuštěna ramena do všech směrů. Jakmile projde rameno posledním prvkem matice objektu, kde je nenulová hodnota, zapíše se vzdálenost.

Algoritmus získání příznakového vektoru z ramen objektu:

I. Pokud nejsou nalezeny všechny objekty obrazové scény, najdi další objekt a přejdi na následující krok II, jinak je algoritmus ukončen.

II. Prodlužuj rameno z vypočteného těžiště daného objektu ve zvoleném úhlu, dokud není dosažena hrana objektu a ulož délku ramene. Ve vytvořené aplikaci je použito pro každý objekt 72 ramen s úhlovým inkrementem 5°.

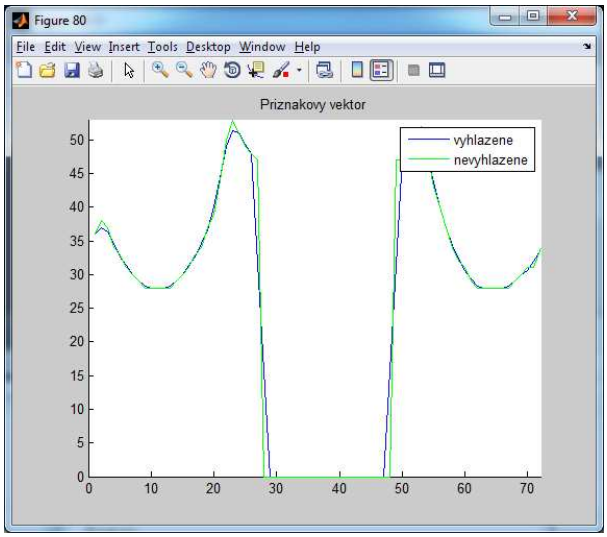
III. Pokud není u daného objektu dosaženo 360° při celkovém úhlovém otočení ramene, otoč rameno o úhlový inkrement a přejdi na krok II, jinak přejdi na krok IV.

IV. Proveď vyhlazení příznakového vektoru. Vyhlazení příznakového vektoru se provede nahrazením jednotlivých složek vektoru aritmetickým průměrem z n sousedních ramen. Ve vytvořené aplikaci je nastavena hodnota $n = 3$, tj. aktuální rameno +1 rameno zleva +1 rameno zprava. Tímto způsobem je provedeno vyhlazení obrysové hrany daného objektu, která může obsahovat chyby způsobené nedokonalostí snímku.

V. Proveď vydělení všech získaných příznaků hodnotou nejdelšího ramene. Pro získání invariance vůči změně měřítka proveď transformaci vypočtených podílů do intervalu $< 0, 1 >$.

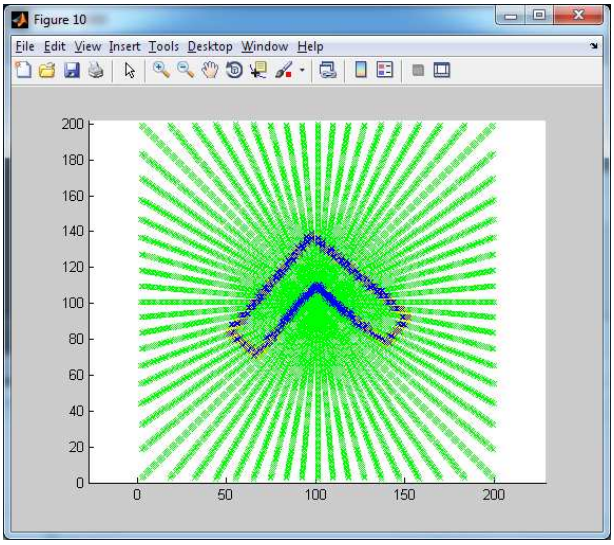
VI. Přejdi na krok I.

Ve vytvořené aplikaci byl zvolen krok ramen 5°, což představuje 72 ramen, aby byl popsán celý objekt. Tato ramena jsou normována do intervalu <0, 1>. Vyhlazení celého vektoru se provede tím způsobem, že je místo každého ramene vypočten průměr z $(n - 1)/2$ sousedních ramen (viz obr. 5.6, kde modrá křivka je nevyhlazená a zelená vyhlazená). Ve vytvořené aplikaci bylo zvoleno $n = 3$, takže se použije jedno rameno po směru a druhé proti směru hodin.



Obr. 5.6: Příznakový vektor pro objekt L-profil

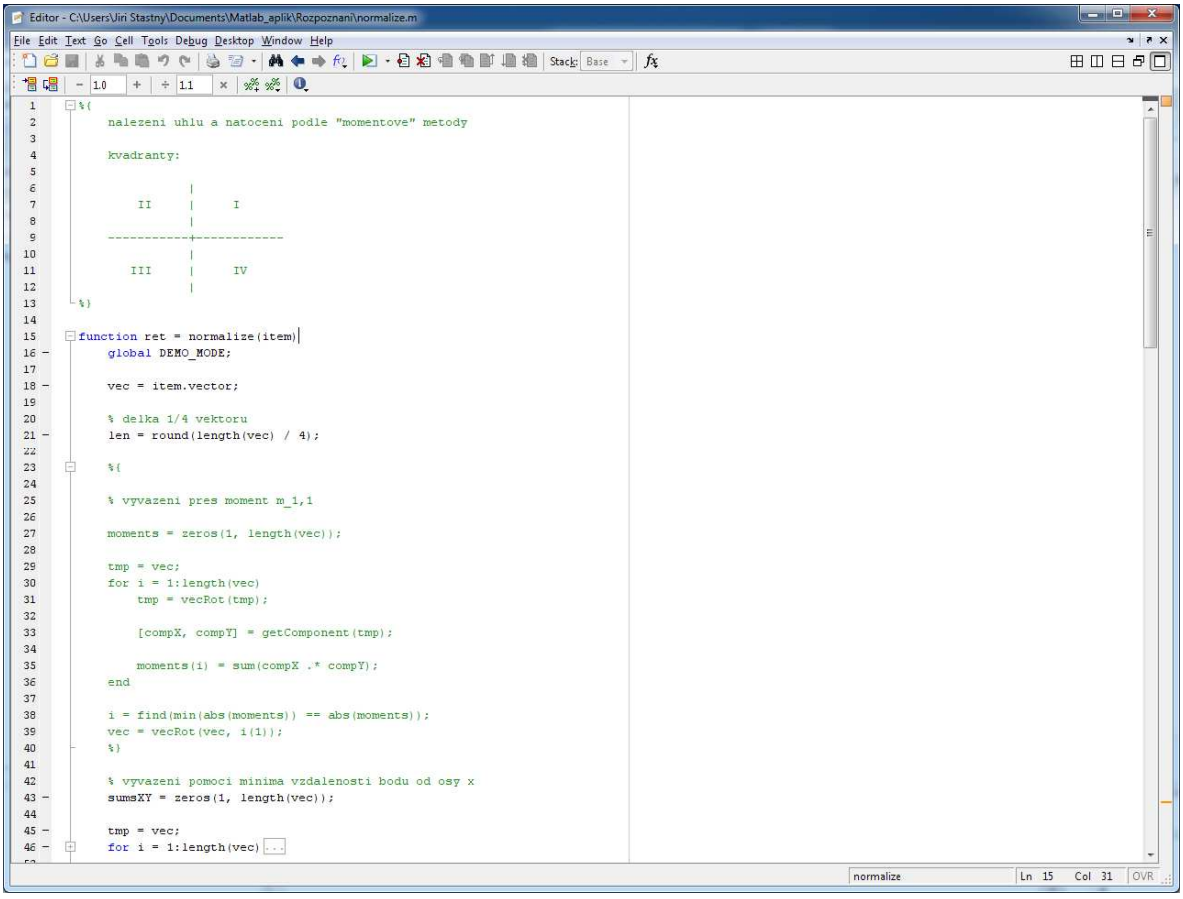
Pro popis jednotlivých objektů byla použita metoda Ramena objektu, která vygeneruje pro každý objekt příslušný příznakový vektor (obr. 5.6). V této metodě odpovídají jednotlivé složky příznakového vektoru délkám ramen vektorů v polárním souřadnicovém systému s počátkem v těžišti právě prohledávaného objektu (obr. 5.7).



Obr. 5.7: Určení příznakového vektoru z ramen objektu

Pro identifikaci objektu v obecné poloze v rovině byla provedena rotace (vyvážení) příznakového vektoru na základě Momentové metody. Rotaci vektoru lze provádět přesouváním prvků ze začátku vektoru posunem doleva a zařazením odebraného prvku na konec (nebo opačně).

Z vektoru lze zjistit složku *x* a složku *y* souřadnic jednotlivých bodů díky tomu, že pozice prvku ve vektoru určuje násobek úhlu mezi jednotlivými body. Takže při počtu 72 prvků vektoru, je krok úhlu 5°. Dále je možné z vektoru určit souřadnice bodů, které spadají do jednotlivých kvadrantů souřadnicového systému tak, že se vektor rozdělí na čtvrtiny (obr. 5.8).



Obr. 5.8: Okno souboru pro určení natočení objektu

Kroky prováděné při rotaci objektu (obr. 5.9):

1. Nalezení polohy, kdy je součet *x* složek a *y* složek v absolutní hodnotě minimální

   $$min\left(\sum|(x_i)|+\sum|(y_i)|\right)$$

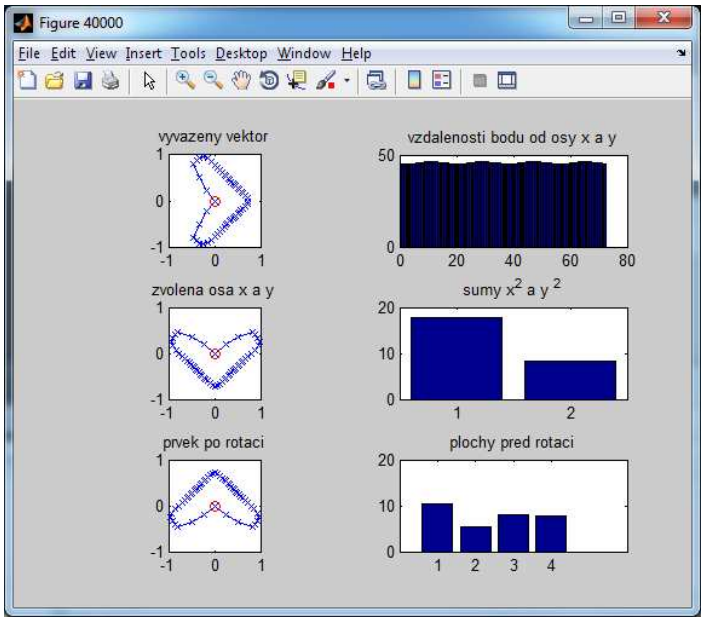
   rotace vektoru o tolik prvků, jaký je index *i* minimální hodnoty

2. Volba osy *x* a *y* podle součtu druhé mocniny složky *x* a složky *y*

   $$\sum x_i^2 > \sum y_i^2$$

   pokud tato podmínka platí, provede se rotace vektoru o polovinu jeho délky.

3. Volba směru osy *x* a *y* podle toho, kde je větší rozdíl (nesymetrie) objektu. V případě potřeby se provede rotace opět o polovinu délky vektoru.



Obr. 5.9: Rotace objektu L-profil Momentovou metodou

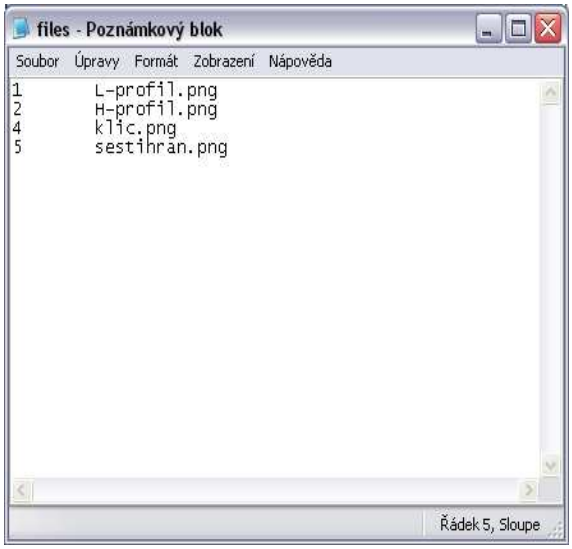
5.4 Simulační experimenty

V této kapitole jsou uvedeny výsledky simulačních experimentů v prostředí MATLAB s testovacími objekty (obr. 5.2) rozmístěnými v obecné poloze. Pro rozpoznávání jednotlivých objektů byla použita neuronová síť MLP s využitím modulu *Neural Network toolbox*.

Jednotlivé objekty byly identifikovány pomocí dopředné, třívrstvé neuronové sítě MLP. Počet neuronů v jednotlivých vrstvách byl volen experimentálně. Ve výstupní vrstvě odpovídá počet neuronů počtu učených vzorů. Síť byla učena pomocí algoritmu *Backpropagation*.

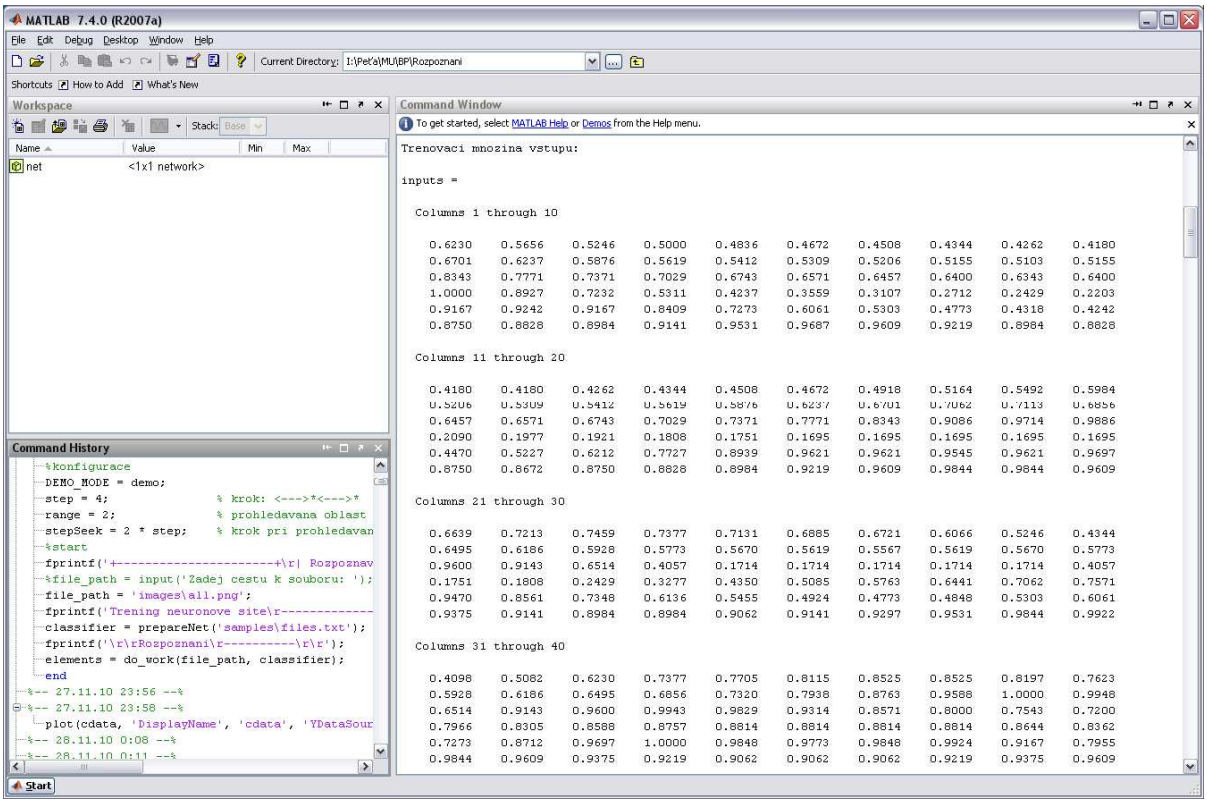
Invariance vůči rotaci je volitelná. V nastavení procesu rozpoznávání je možné zvolit, jestli se budou objekty předkládat neuronové síti v každé možné rotaci (posunem vektoru – přenášením prvního prvku vektoru na jeho konec), nebo jen v poloze, ve které byly nalezeny ve vstupním obraze.

Jednotlivým testovacím objektům jsou pro fázi učení přiděleny číselné identifikátory (obr. 5.10), pomocí nichž je pak ve fázi rozpoznávání vyhodnocena úspěšnost identifikace.



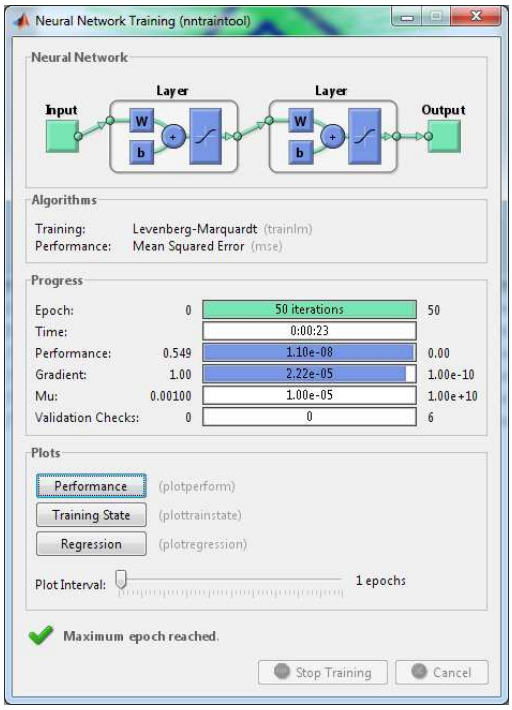
Obr. 5.10: Identifikátory jednotlivých objektů

5.4.1 Fáze učení



Obr. 5.11: Ukázka „trénovací“ množiny pro neuronovou síť

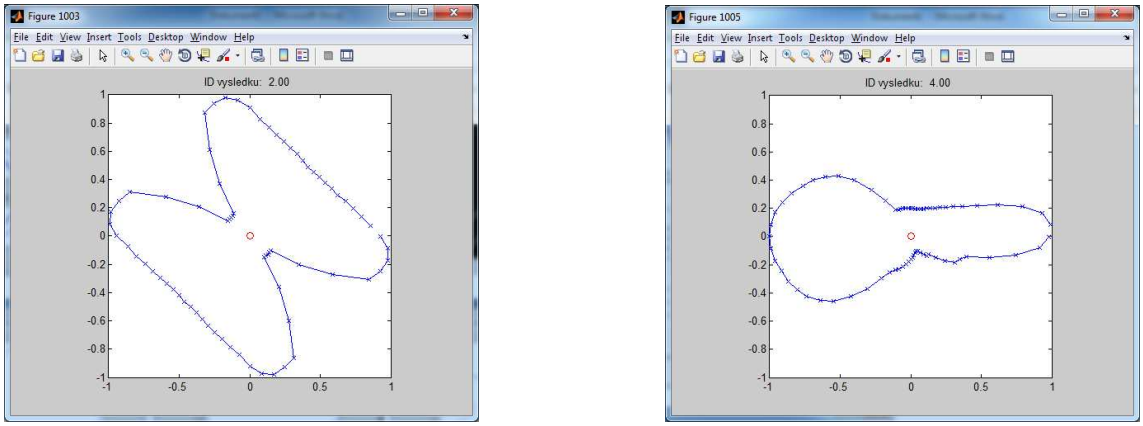
Použitá neuronová síť MLP je v první fázi učena pomocí trénovací množiny (obr. 5.11), tvořené příznakovými vektory etalonů jednotlivých objektů (H-profil, klíč, L-profil, šestihran). Obr. 5.12 představuje ukázku učení třívrstvé neuronové sítě MLP v *Neural Network toolboxu* prostředí MATLAB.



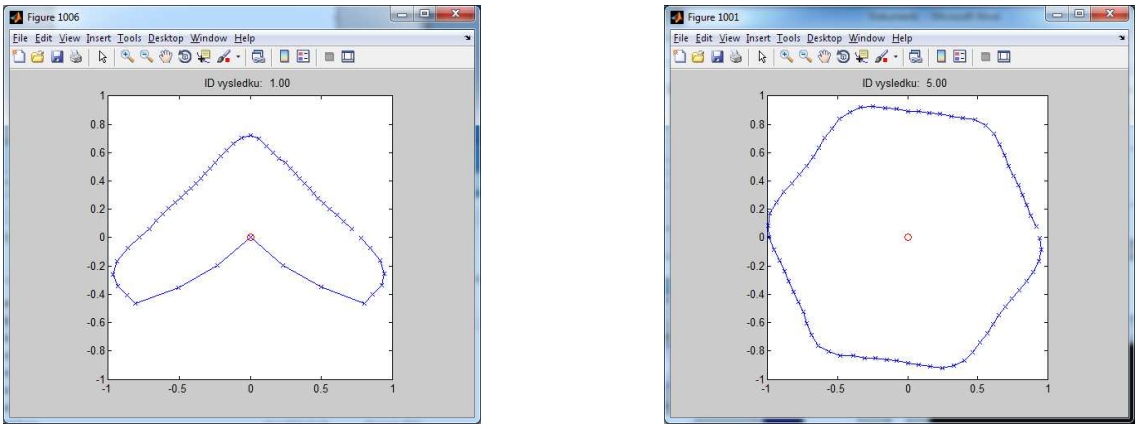
Obr. 5.12: Učení neuronové sítě MLP

5.4.2 Fáze rozpoznávání

Ve výkonné fázi provedla neuronová síť MLP při opakovaných pokusech úspěšné rozpoznání všech čtyř naučených objektů (obr. 5.13, obr. 5.14) v obrazové scéně libovolně rozmístěných objektů (obr. 5.3). Zvolená metoda popisu a neuronová síť jsou schopny při předložení objektu ve správné pozici a na správném pozadí rozeznat každý naučený objekt bezchybně.



Obr.5.13: Identifikace objektů H-profil a klíč neuronovou sítí



Obr. 5.14: Identifikace objektů L-profil a šestihran neuronovou sítí

6 Závěr

Práce obsahuje seznámení s problematikou zpracování obrazu, jmenovitě se jedná o předzpracování obrazu, segmentaci obrazu a metody popisu objektu. Podrobněji je uvedena metoda Ramena objektu, která byla implementována ve vytvořené aplikaci. Další část práce uvádí charakteristiku dostupných metod identifikace objektů se zaměřením na aplikace v reálném čase zejména v robotice. Jsou uvedeny tři vhodné metody: Momentová metoda, Fourierovy deskriptory a umělá neuronová síť MLP s učícím algoritmem Back-propagation. Použití Fourierových deskriptorů pro aplikace v reálném čase je možné pouze bez možnosti identifikace natočených objektů. Momentová metoda je sice vůči natočení invariantní, ale za cenu menší rychlosti identifikace.

V souladu se zadáním zahrnuje praktická část práce návrh testovacích objektů a obrazových scén, implementaci programového systému s využitím neuronové sítě MLP pro identifikaci objektů v libovolné poloze v rovině a simulační experimenty s navrženými objekty ve zvoleném prostředí MATLAB. Neuronová síť MLP opakovaně při všech simulačních pokusech úspěšně identifikovala všechny předložené objekty v libovolné poloze. Na základě provedených simulačních experimentů lze neuronovou síť MLP doporučit pro identifikaci objektů v aplikacích probíhajících v reálném čase.

Možným pokračováním této práce by mohlo být testování schopnosti neuronové sítě identifikovat poškozené objekty (snímky zatížené šumem), případně testování dalších typů neuronových sítí se sledováním konkrétních hodnot času identifikace.

Literatura

[1] BOEF, J., BELIN, P.: Fourier Descriptors, [online]. 1999. Dostupné z URL: <http://www.sig.enst.fr/tsi/enseignement/ressources/mti/descript_fourier/>.

[2] BOGR, J.: Rozpoznávání obrazů metodou Fourierových deskriptorů. [Disertační práce], *VUT*, Brno, 1984.

[3] DEMEL, J.: Grafy a jejich aplikace. *Academia*, Praha, 2002.

[4] HLAVÁČ, V., ŠONKA, M.: Počítačové vidění. *Grada*, Praha, 1993.

[5] JELÍNEK, I., SOCHOR, J.: Algoritmy počítačové grafiky. *ČSVTS*, Praha, 1989.

[6] JEŽEK, B.:Počítačová grafika II. *UHK*, Hradec Králové, [online]. 2002 Dostupné z URL: <http://lide.uhk.cz/home/fim/ucitel/fujezeb1/www/>.

[7] KEPKA, J., PSUTKA, J.: Kombinace příznakových a strukturálních metod rozpoznávání. *ZČU*, Plzeň, 1994.

[8] KOTEK, Z., MAŘÍK, V. a kol.: Metody rozpoznávání a jejich aplikace. *Academia*, Praha, 1993.

[9] NOVÁK, M. a kol.: Umělé neuronové sítě. Teorie a aplikace. *C. H. Beck*, Praha, 1998.

[10] PELIKÁN, J.: Vyplňování souvislé oblasti. *MFF UK*, Praha, [online]. 2000. Dostupné z URL: <http//sun3.ms.mff.cuni.cz/~pepca/>.

[11] SKÁLA, V.: Algoritmy počítačové grafiky I–III. *ZČU*, Plzeň, 1992.

[12] SOCHOR, J.: Počítačová grafika. Brno, *MUNI FI*, [online]. 2002. Dostupné z URL: <http://www.fi.muni.cz/usr/sochor/>.

[13] SOCHOR, J., ŽÁRA, J., BENEŠ, B.: Algoritmy počítačové grafiky. *ČVUT*, Praha, 1998.

[14] SVITÁK, R.: Detekce hran v obraze. *ZU FAV*, Plzeň, [online]. 2001. Dostupné z URL: <http://herakles.zcu.cz/~rsvitak/school/zvi_rsvitak.pdf>.

[15] ŠNOREK, M., JIŘINA, M.: Neuronové sítě a neuropočítače. *ČVUT*, Praha, 1998.

Přílohy

Příloha A

Ukázka zdrojového kódu pro výpočet těžiště objektu (soubor *getCOG.m*):

```
% Výpočet těžiště objektu

function ret = getCOG(item)


    points = item.edge;

    m = length(find(points));

    [wy, wx] = size(points);

    sx = 0;
    sy = 0;
    for i = 1:wy
        for j = 1:wx
            if points(i, j)
                sx = sx + j;
                sy = sy + i;
            end
        end
    end

    cogx = sx / m;
    cogy = sy / m;

    %presun do teziste

    sx = max(0, round((wx / 2) - cogx));
    sy = max(0, round((wy / 2) - cogy));

    cogx = round(cogx + sx);
    cogy = round(cogy + sy);

    pointsEdge = logical(zeros(wy, wx));
    for i = 1:wy - sy
        for j = 1:wx - sx
            pointsEdge(i + sy, j + sx) = item.edge(i, j);
        end
    end

    item.edge = pointsEdge;
    item.cog = struct('x', cogx, 'y', cogy);

    ret = item;
end
```

Příloha B

Ukázka zdrojového kódu pro výpočet příznakového vektoru z ramen objektu (soubor *describe.m*):

```
% Vytvoří příznakový vektor

function ret = describe(item)
%{
Algoritmus výpočtu příznakového vektoru z ramen objektu:

1.  Pokud nebyly prohledány všechny objekty, vyber další objekt a přejdi
    na 2, jinak konec.
2.  Ve zvoleném úhlu prodlužuj rameno z těžiště, dokud není dosažena hrana
    objektu a ulož jeho délku. Ve vytvořené aplikaci je nastaveno 72 ramen
    pro každý objekt s úhlovým inkrementem 5°.
3.  Pokud je celkové úhlové otočení ramene menší než 360°, otoč rameno o
    úhlový inkrement a přejdi na 2, jinak přejdi na 4.
4.  Vyhlaď příznakový vektor. Vyhlazení vektoru příznaků znamená nahrazení
    jednotlivých složek vektoru aritmetickým průměrem z n sousedních ramen.
    Ve vytvořené aplikaci je nastavena hodnota 3 (aktuální rameno +1 rameno
    zleva a +1 rameno zprava)
    Tím je zajištěno vyhlazení obrysové křivky, která může obsahovat chyby
    způsobené nedokonalostí snímku.
5.  Zjisti velikost nejdelšího ramene a touto hodnotou vyděl všechny
    příznaky.
    Transformuj výsledné podíly do daného intervalu.
    Tím je zajištěna invariance vůči změně měřítka.
6.  Přejdi na 1.
%}
    global DEMO_MODE;

    [wy, wx] = size(item.edge);

    if DEMO_MODE
        figure(10);
        clf;
        hold('on');
        axis([0,wx,0,wy], 'equal');
        plot(item.cog.x, item.cog.y, 'xr');

        for i = 1:wx
            for j = 1:wy
                if item.edge(j, i)
                    plot(i, j, 'oy');
                end
            end
        end
    end

    step = 5;
    retVec = [];
    vec = [0, 1];

    % rotovat o uhel
    for i = 0:step:360 - step
        ang = i * pi / 180;
        tmp = rotate2d(vec, ang);

        x = item.cog.x;
```

```
    y = item.cog.y;

    len = 0;

    xxg = [];
    yyg = [];
    xxb = [];
    yyb = [];

    % zvetsovat rameno
    while x >= 3 && y >= 3 && x <= wx - 3 && y <= wy - 3

        x = x + tmp(1);
        y = y + tmp(2);

        minY = round(y) - 1;
        maxY = round(y) + 1;

        minX = round(x) - 1;
        maxX = round(x) + 1;

        if find(item.edge(minY:maxY, minX:maxX))
        %if item.edge(round(y), round(x))
            len = sqrt((x - item.cog.x)^2 + (y - item.cog.y)^2);
            if DEMO_MODE
                xxb = [xxb, x];
                yyb = [yyb, y];
            end
        else
            if DEMO_MODE
                xxg = [xxg, x];
                yyg = [yyg, y];
            end
        end


    end
    retVec = [retVec, len];

    if DEMO_MODE
        figure(10);
        plot(xxg, yyg, 'xg', xxb, yyb, 'xb');
    end
end

%vyhlazeni vektoru
tmp1 = retVec;
s = 3;
for i = 1:length(retVec)
    retVec(i) = sum(getVecPart(tmp1, i, s)) / s;
end

if DEMO_MODE
    figure(80);
    clf;
    hold('on');
    plot(retVec, 'b');
    plot(tmp1, 'g');
    axis([0, 360/step, 0, max(tmp1)]);
    title('Priznakovy vektor');
```

```
        legend('vyhlazene', 'nevyhlazene');
        pause(0.5);
    end

    % normalizace do intervalu 0 az 1
    item.vector = retVec / max(retVec);

    ret = item;
end

% vrati size prvku okolo prvku p
function ret = getVecPart(vec, p, size)
    l = length(vec);
    d = round((size - 1) / 2);
    ret = zeros(1, (d * 2) + 1);
    i = 1;
    for j = p - d:p + d
        if j < 1
            ret(i) = vec(l + j);
        elseif j > l
            ret(i) = vec(j - l);
        else
            ret(i) = vec(j);
        end
        i = i + 1;
    end
end
```

Příloha C

Ukázka zdrojového kódu pro lokalizaci objektu v obraze (soubor *localize.m*):

```
%{
    lokalizace objektu v obraze

    najde v obraze objekty a vrati je jako jednotlive polozky
    urci:
        - souradnici v obrazku
        - vrati matici hodnot 0/1 podle toho kde objekt je nebo neni

    vrati strukturu:
        .title     nazev
        .points    body
        .edge      hrana
        .cog       souradnice teziste
        .angle     uhel otoceni
        .vector    vektor s popisem
%}

function items = localize(aIMG)
    items = [];

    global IMG range h w minColor points directions objectPoints stepSeek;

    directions = [-1 0; -1 1; 0 1; 1 1; 1 0; 1 -1; 0 -1; -1 -1];

    IMG = aIMG;
    [h, w] = size(IMG);
    minColor = 60 * (4 * range * range); %soucet vsech jasu v oblasti
range*range
    minPoints = 80;

    detectEdges();

    for i = stepSeek:stepSeek:h
        j = 0;
        while j ~= -1
            j = findObject(i, j);
            if j ~= -1
                describeObject(j, i);
                if length(points) > minPoints
                    items = [items, struct('title', 'unknown', 'points',
objectPoints, 'edge', [], 'cog', [], 'angle', 0, 'vector', [])];
                end
                deleteObject();
            end
        end
    end
end

% detekce hran pomoci konvoluce
function detectEdges()

    global IMG;

    C = [   -1  0   0   0   -1;
            0  -1   0   -1  0;
```

```
            0   0   8   0   0;
            0   -1  0   -1  0
            -1  0   0   0   -1];

    %{
    C = [
            -1   0   -1;
            0   4   0;
            -1  0   -1
            ];
    %}

    IMG = convolution(IMG, C);

    %{
    C = [0.1 0.1 0.1;
         0.1 0.2 0.1;
         0.1 0.1 0.1;];

    convolution(rIMG, C);
    %}
end

% najit na radku hranu
function ret = findObject(y, xstart)
    global cIMG IMG stepSeek w DEMO_MODE;

    cIMG = IMG;

    x = max(xstart, 1);

    for i = x:stepSeek:w

        %{
        if DEMO_MODE
            figure(50);
            clf;
            cIMG(y, i) = 255;
            imshow(cIMG);
            title('Probiha detekce...');
            pause(0.01);
        end
        %}

        if edgeInRange(i, y)
            ret = i;
            return
        end
    end
    ret = -1;
end

% najit hlavni body na objektu ktery ma hranu na souradnici x, y
function describeObject(x, y)
    global newPoints points objectPoints point_cnt;

    point_cnt = 0;

    newPoints = [struct('x', x, 'y', y, 'dir', 5)];
```

```
    points = [];
    objectPoints = newPoints;

    while ~isempty(newPoints)
        p = newPoints(1);
        newPoints(1) = [];
        points = [points, p];

        findNextPoint(p);
    end

    fprintf('Vyhodnoceno\t%i bodu\rDetekovano\t%i bodu\r\r',
length(points), length(objectPoints));
end

%vymazat z obrazu body patrici objektu
function deleteObject()
    global IMG objectPoints range;
    for i = 1:length(objectPoints)
        p = objectPoints(i);
        for j = -range:range
            for k = -range:range
                IMG(p.y + j, p.x + k) = 0;
            end
        end
    end
end

% prohledat okoli bodu do vsech osmi smeru
function findNextPoint(p)
    global directions;

    %nejprve zkusit smer ze ktereho prichazime
    dx = directions(p.dir, 1);
    dy = directions(p.dir, 2);

    [tmp, waiting, processed] = createPoint(p, dx, dy);

    if ~waiting && ~processed
        if edgeInRange(tmp.x, tmp.y)
            addPoint(tmp, true);
            return
        else
            addPoint(tmp, false);
        end
    end

    % do vsech smeru
    for i = 1:8
        if i ~= p.dir
            dx = directions(i, 1);
            dy = directions(i, 2);

            [tmp, waiting, processed] = createPoint(p, dx, dy, i);

            if ~waiting && ~processed
                if edgeInRange(tmp.x, tmp.y)
                    addPoint(tmp, true);
                else
```

```
                    addPoint(tmp, false);
                end
            end
        end
    end

end

function [ret, waiting, processed] = createPoint(p, dx, dy, dir)
    global step newPoints points h w;

    ret = struct('x', max(1, min(w, p.x + (dx * step))), 'y', max(1, min(h,
p.y + (dy * step))), 'dir', p.dir);

    waiting = pointInList(ret, newPoints);
    processed = pointInList(ret, points);

    if nargin == 4
        ret.dir = dir;
    end
end

%prida bod do seznamu, pokud je to potreba
function addPoint(p, hasEdge)
    global points newPoints objectPoints cIMG DEMO_MODE point_cnt;

    if hasEdge
        objectPoints = [objectPoints, p];
        newPoints = [newPoints, p];
    else
        points = [points, p];
    end

    point_cnt = point_cnt + 1;

    if DEMO_MODE
        if hasEdge
            cIMG(p.y, p.x) = 255;
        else
            cIMG(p.y, p.x) = cIMG(p.y, p.x) + 150;
        end
        if point_cnt > 30
            figure(50);
            s = sprintf('Cekajicich bodu: %i, zpracovanych bodu: %i',
length(newPoints), length(points));
            imshow(cIMG);
            title(s);
            pause(0.01);
            point_cnt = 0;
        end
    end
end

% zjisit, jestli je v okruhu bodu x,y nejaka hrana
function ret = edgeInRange(x, y)
    global IMG range w h minColor;

    sum = uint16(0);
    for i = y - range:y + range
```

```
        for j = x - range:x + range
            if i >= 1 && j >= 1 && i <= h && j <= w
                sum = sum + uint16(IMG(i, j));
            end
        end
    end

    % jeste je moznost scitat jas vsech pixelu v rozsahu a podle toho urcit
    % jestli se jedna o bod telesa nebo ne
    ret = (sum >= minColor);
end

% --------------------------------------------------------------------

function [x, y] = getCenter(points)
    sx = 0;
    sy = 0;
    len = length(points);
    for i = 1:len
        sx = sx + points(i).x;
        sy = sy + points(i).y;
    end
    x = sx / len;
    y = sy / len;
end
```

Příloha D

Ukázka zdrojového kódu pro učení neuronové sítě MLP (soubor *prepareNet.m*):

```
% Příprava neuronové sítě
% Načteni etalonu a předloženi neuronové síti

function ret = prepareNet(sourceFile)

    netFileName = 'net_data.mat';
    if exist(netFileName,'file') & input('Nacist data? [a/n] ', 's') == 'a'
        load(netFileName, '-mat', 'ret');
        return
    end

    f = fopen(sourceFile, 'r');
    imgIDs = fscanf(f, '%u\t%*s');

    imgNames = cell(length(imgIDs), 1);

    frewind(f);
    fl = fgetl(f);
    i = 1;
    while ischar(fl)
        imgNames{i} = char(sscanf(fl, '%*u\t%s')');
        fl = fgetl(f);
        i = i + 1;
    end
    fclose(f);

    fPath = fileparts(sourceFile);

    IDs = zeros(length(imgIDs), length(imgIDs));

    inputs = [];
    for i = 1:length(imgIDs)
        % nacte etalon a ziska popisovy vektor

        fName = imgNames{i};

        IDs(i, imgIDs(i)) = 1;

        fprintf('\rNacitam obrazek: %s\r', strcat(fPath, '\', fName));

        IMG = imread(strcat(fPath, '\', fName));
        pIMG = preprocess(IMG);
        data = localize(pIMG);

        if ~isempty(data)

            item = getEdgeFromPoints(data(1));
            item = getCOG(item);
            item = describe(item);
            item = normalize(item);

            inputs = [inputs; item.vector];
        else
            disp('Na obrazku nebylo nic nalezeno');
```

```
        end
    end

    disp('Trenovaci mnozina vstupu:');
    inputs
    disp('Trenovaci mnozina vystupu:');
    IDs'


    %ret = newff(inputs', IDs', [20]);
    %ret = newff(inputs', IDs', [20 10], {'tansig', 'satlin'});


    [r c] = size(inputs);
    ret = newff([zeros(72, 1), ones(72, 1)], [20 r]);

    ret.trainParam.epochs = 50;
    ret = train(ret, inputs', IDs');

    if input('Ulozit sit? [a/n] ', 's') == 'a'
        save(netFileName, 'ret', '-mat');
    end

end
```