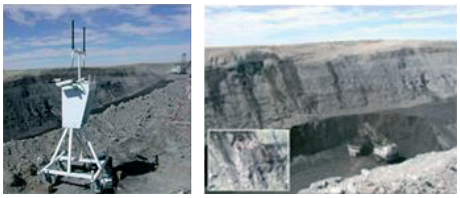
Detekcia pohybu v reálnom čase pomocou fuzzy logiky

Dnes sa pomerne často stretávame s potrebou spracovania videa s cieľom ochrany objektu, bezpečnosti na verejných priestoroch alebo na analýzu obrazu na iné ciele. Jeden z prvých krokov pri spracovaní videa často býva detekcia pohybu. Dosiahnutie vysokej kvality detekcie pohybu pri použití štandardnej webovej kamery je už ťažšia úloha, zvlášť pri potrebe funkčnosti v reálnom čase. V tomto článku je prezentovaný spôsob realizácie kvalitnej detekcie pohybu v reálnom čase pri použití štandardnej webovej kamery.

Úvod

Kamerové systémy spočiatku slúžili len ako vzdialená kontrola objektu príslušníkmi bezpečnostných služieb. Neskôr po nástupe výkonnejších počítačov je snaha niektoré kontroly automatizovať, respektíve tvoriť umelú inteligenciu na základe kamerového systému. Ako príklad môžeme uviesť automatický kamerový varovný systém, ktorý upozorňuje pracovníkov lomu pred zosuvom horniny (obr. 1). Ide o veľmi presné zariadenie citlivé na pohyb horniny. Požiadavka na vývoj tohto zariadenia vznikla v USA v roku 1995 po mnohých smrteľných nehodách pri ťažbe hornín. V súčasnosti táto jednoduchá detekcia pohybu zachraňuje životy.



Obr.1 Detekcia zosuvu horniny ako súčasť varovného systému pri ťažbe hornín

Najjednoduchší spôsob detekcie pohybu možno realizovať pomocou zmien jasu dvoch po sebe idúcich obrázkov.

$$\Delta(x,y) = \left| I_{f1}(x,y) - I_{f2}(x,y) \right| \qquad (1)$$

Pohyb môžeme definovať ako veľkosť zmeny jasu, ktorá je väčšia ako určitá nami zvolená prahová úroveň δ. Pritom prahová úroveň δ sa volí väčšinou empiricky. Pri takejto jednoduchej realizácii je problémom nízka kvalita detegovaného pohybu, pretože šum v obraze bude tiež klasifikovaný ako pohyb.

Obr.2 Jednoduchá detekcia s vysokou prahovou úrovňou (vľavo) a s nízkou prahovou úrovňou (vpravo)

Kvalitnejšiu detekciu môžeme získať kvalitnejším vstupným obrazom (kamerou) alebo predspracovaním obrazu, kde použijeme dolnopriepustný filter. Výsledný efekt aj napriek tomu často nestačí, a tak musíme mať algoritmus na detekciu pohybu zložitejší. Na druhej strane zložitý algoritmus bez jeho optimalizácie býva náročný z hľadiska strojového času, čím sa nevyhneme ďalším problémom pri práci v reálnom čase (obr. 3).

Obr.3 Detekcia pohybu pri rôznych rýchlostiach pohybu ruky

Z uvedených obrázkov vidieť, že čím je pohyb ruky rýchlejší, tým sú kontúry ruky hrubšie. Je to spôsobené tým, že spracovanie dvoch obrázkov trvá nejaký čas t, počas ktorého ruka vykoná dráhu s. V konečnom dôsledku čím je väčšia dráha s do ďalšieho spracovania dvoch obrázkov, tým budú väčšie rozdiely medzi danými obrázkami, a teda kontúry ruky budú hrubšie. To znamená, že pre zabezpečenie vyššej kvality musí algoritmus detekcie pohybu pracovať v reálnom čase.

1. Navrhnutý algoritmus

V experimente bol použitý menej kvalitný obraz (použila sa štandardná webová kamera), ktorý sa predspracoval odfiltrovaním šumu. Na takýto obraz sa aplikovala fuzzy logika s empiricky navrhnutými fuzzy pravidlami. Algoritmus je zložitý a pomocou optimalizácie ho možno implementovať aj do aplikácií, kde je požiadavka na prácu v reálnom čase.

2. Predspracovanie obrazu

Platí všeobecné pravidlo, že čím je vstupný obraz kvalitnejší, tým menšie nároky sú na jeho ďalšie spracovanie. V prípade kamery s nižšou kvalitou obrazu (štandardné webové kamery) je výhodné použiť dolnopriepustný filter. V našom prípade bol použitý filter 4. stupňa.

$$y(k) = a_1 \cdot x + a_2 \cdot x(k-1) + a \cdot (k-2) - \\ - b_1 \cdot y(k-1) - b_2 \cdot y(k-2) \qquad (2)$$

Kde

$$a_1 = \frac{1}{1 + r \cdot c + c^2} \qquad a_2 = 2 \cdot a_1 \qquad a_3 = a_1$$

$$b_1 = 2 \cdot a_1 \cdot \left(1 - c^2\right) \qquad b_2 = a_1 \cdot \left(1 - r \cdot c + c^2\right)$$

$$c = \frac{1}{\tan\left(\frac{\pi \cdot f}{s}\right)} \qquad f = \left\langle 0 \quad \frac{s}{2} \right\rangle \qquad s = \left\langle 0{,}1 \quad \sqrt{2} \right\rangle$$

Pričom f je hraničná frekvencia a s je veľkosť rezu.

3. Fuzzy pravidlá

Funkcia príslušnosti k fuzzy množine „veľký“ a funkcia príslušnosti k množine „malý“ vyjadrujú vzťahy (3) a (4).

$$\text{velky}(\Delta(x,y)) = \begin{cases} 0 & \text{pre } \Delta(x,y) < a \\ \dfrac{\Delta(x,y)-a}{b-a} & \text{pre } a \le \Delta(x,y) \le b \\ 1 & \text{pre } \Delta(x,y) > b \end{cases} \quad (3)$$

$$\text{maly}(\Delta(x,y)) = \begin{cases} 1 & \text{pre } \Delta(x,y) < a \\ \dfrac{1-(\Delta(x,y)-a)}{b-a} & \text{pre } a \le \Delta(x,y) \le b \\ 0 & \text{pre } \Delta(x,y) > b \end{cases} \quad (4)$$

Nech SP je množina veľkosti zmien v jase susedných pixlov (v jednotkovom okolí) k bodu (x,y).

$$SP = \{sp_1, sp_2, sp_3, sp_4, sp_5, sp_6, sp_7, sp_8\} \quad (5)$$

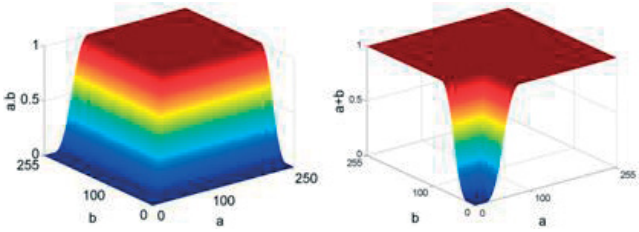
Navrhnuté fuzzy pravidlá:

- Ak $\Delta(x,y)$ je veľký AND tretí najväčší prvok z množiny SP je veľký, potom (x,y) je dynamický pixel, ktorý označíme ako $M_{dyn}(x,y)$.
- Ak $\Delta(x,y)$ je malý OR ($\Delta(x,y)$ je veľký AND šiesty najmenší prvok z množiny SP je malý), potom (x,y) je statický pixel, ktorý označíme ako $M_{stat}(x,y)$.

Logické operátory sú definované takto.

$$a \ \ AND \ \ b = \min\{a,b\} \quad (6)$$

$$a \ \ OR \ \ b = \max\{a,b\} \quad (7)$$



Obr.4 Graf fuzzy operácie AND (vľavo) a OR (vpravo) ako minimum a maximum

Mieru istoty pohybu $Dyn(x,y)$ môžeme vypočítať podľa vzťahu (8)

$$Dyn(x,y) = \frac{M_{dyn}(x,y)}{M_{dyn}(x,y) + M_{stat}(x.y)} \quad (8)$$

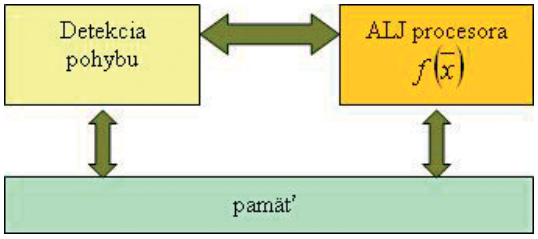
Pre konečné stanovenie, či je daný pixel (x,y) dynamický, určíme podľa empiricky získanej prahovej úrovne Γ. V našom prípade sme stanovili túto prahovú úroveň na hodnotu $\Gamma = 0{,}01$.

$$bDyn = \begin{matrix} 0 & \text{pre } Dyn(x,y) < \Gamma \\ 1 & \text{pre } Dyn(x,y) > \Gamma \end{matrix} \quad (9)$$

4. Optimalizácia implementácie

Každý algoritmus sa skladá z premenných, ktoré sú uložené v pamäti, takisto takmer každý algoritmus využíva matematické operácie, ktoré realizuje aritmeticko-logická jednotka (ALJ) procesora (obr. 5).

Optimalizácia spočíva v efektívnosti algoritmu, ale aj minimalizácii používania ALJ procesora, zvlášť v častiach algoritmu, ktoré sa často využívajú. Ako príklad si ukážeme experiment, kde pre každý prvok v poli *_array* budeme počítať hodnotu podľa dvojparametrickej funkcie $f(i,j)$. Pre presnejšie meranie výkonnosti algoritmu budeme uvedený



Obr.5 Algoritmus detekcie pohybu ako každý algoritmus využíva pamäť aj procesor

algoritmus opakovať 10 000-krát. Experiment bol realizovaný v jazyku C#. Treba pripomenúť, že pre presné meranie výkonu musí byť program spustený samostatne a v režime „release".

```
for (int k = 0; k < 10000; k++)
    for (int i = 0; i < 255; i++)
        for (int j = 0; j < 255; j++)
            _array[i, j] = f(i,j);
```

Ak bude v danom algoritme použitý vzťah (10), potom použitý hardvér (Pentium s mikroprocesorom 1,6 GHz a pamäťou RAM 512MB) potreboval 10,8756384 sekúnd na jeho vykonanie.

$$f(i,j) = |i-j| \quad (10)$$

Ak však bude použitý vzťah (11), potom použitý hardvér potreboval 7,9814768 sekúnd na jeho vykonanie. Z uvedeného vyplýva, že ALJ procesora potrebuje na vykonanie vzťahu (11) kratší čas ako pri vzťahu (10).

$$f(i,j) = \frac{i}{i+j+1} \quad (11)$$

V ďalšom pokuse prepíšeme funkciu $f(i,j)$ zo vzťahu (10) do tabuľkovej reprezentácie (12), pričom v tabuľke budú už len vypočítané hodnoty funkcie s danými parametrami (i,j).

$$f(i,j) = \begin{bmatrix} f(1,1) & . & f(1,n) \\ . & & . \\ f(n,1) & . & f(n,n) \end{bmatrix} \quad (12)$$

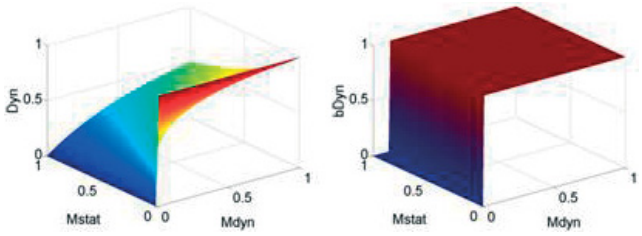
Ľubovoľne zložitá dvojparametrická funkcia $f(i,j)$ sa môže prepísať do tabuľkovej reprezentácie (pri zadefinovaní určitej postačujúcej presnosti), pričom zložitosť algoritmu ostáva rovnaká. V našom prípade budeme pri použití uvedeného hardvéru potrebovať na vykonanie spomínaného algoritmu 6,75972 sekúnd.

Ak danú dvojrozmernú tabuľku implementujeme do programu ako jednorozmernú tabuľku $f(k \cdot i + j)$ kde k je počet prvkov v jednom riadku, potom dosiahneme pri nasledujúcom algoritme ešte vyšší výkon.

```
for (int k = 0; k < 10000; k++)
    for (int i = 0; i < 255; i++)
        for (int j = 0; j < 255; j++)
            _array[i*255+j] = f(i*255+j);
```

Uvedený algoritmus trval pri danom hardvérovom vybavení 3,4950256 sekúnd. Experimenty boli realizované aj v jazyku C++, výsledky však boli rovnaké ako v prípade jazyka C#.

V našom prípade sa často využívajú fuzzy operátory AND a OR. Môžeme ich však nahradiť poľami obsahujúcimi priamo výsledkami. Podobne môžeme nahradiť aj vzťah (1). Vzťah (9) je pomerne náročný a opakuje sa pre každý pixel. Podobným spôsobom bude aj pre tento vzťah implementovaná spomínaná tabuľka s poľami obsahujúcimi priamo výsledky.

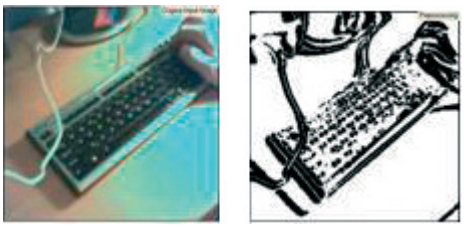


Obr.6 Graf funkcie (vľavo) a graf funkcie

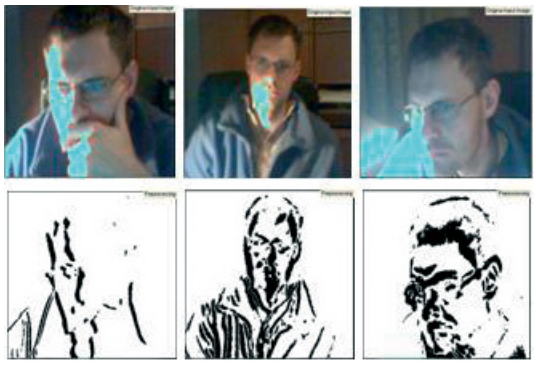
Celý algoritmus teda možno nahradiť uvedenými poľami, čím ALJ procesora úplne vynecháme.

5. Experiment

Experiment bol vytvorený v programovacom jazyku C#, v ktorom bola implementovaná popísaná metóda. Zaujímavá situácia nastáva pri statických objektoch a pohybujúcej sa kamere (obr. 7), kde získame



Obr.7 Detekcia pohybu pomocou fuzzy logiky, kde je scéna statická a kamera pohyblivá. Boli použité parametre: $a = 20$, $b = 80$, $\Gamma = 0{,}2$



Obr.8 Detekcia pohybu pomocou fuzzy logiky, kde je kamera statická a objekty sa pohybujú, prvý stĺpec dole s parametrami $a = 20$, $b = 80$, $\Gamma = 0{,}2$, druhy stĺpec $a = 5$, $b = 10$, $\Gamma = 0{,}05$ a tretí s parametrami $a = 2$, $b = 8$, $\Gamma = 0{,}05$

pomerne kvalitné obrysy objektov (jeden z najrýchlejších a najjednoduchších spôsobov detekcie hrán v obraze).

Ďalší experiment bol pri statickej kamere, kde sa sledoval pohybujúci sa objekt. Na obr. 8 možno vidieť pomerne vysokú citlivosť na pohyb (ľavý stĺpec), pomerne kvalitná detekcia je aj pri zníženej kvalite obrazu (pravý stĺpec), respektíve pri rýchlom pohybe (stredný stĺpec).

Záver

Cieľom bolo navrhnúť zložitý algoritmus detekcie pohybu s minimálnymi nárokmi na matematickú náročnosť, schopný pracovať v reálnom čase. Pri použití kvalitnej kamery ako snímača vstupných dát a vynechaní filtrácie v predspracovaní dát bude navrhnutý algoritmus bez jedinej aritmeticko-logickej operácie, čím aritmeticko-logickú jednotku procesora vynecháme. Celý algoritmus bude teda založený len na práci s pamäťou.

Článok vznikol v rámci riešenia projektu VEGA 1/0690/09.

Literatúra

[1] Osama, Masoud: A Method For Human Action Recognition. Department of Computer Science and Engineering, University of Minnesota.

[2] Mojzeš, Matej: Optická detekcia pohybu pomocou fuzzy logiky. Internet: http://www.matejmojzes.sk/storage/SOFC/Fuzzy_Motion_Detection.pdf

[3] McHugh, Edward L.: Video motion detection for real time hazard warning in surface mines. Internet: http://www.cdc.gov/NIOSH/mining/pubs/pdfs/vmdfr.pdf

Ing. Peter Benický
prof. Ing. Ladislav Jurišica, PhD.

Slovenská technická univerzita v Bratislave
Fakulta elektrotechniky a informatiky
Ústav riadenia a priemyselnej informatiky
Ilkovičová 3, 812 19 Bratislava
e-mail: peter@benicky.info
ladislav.jurisica@stuba.sk