Rozpoznávanie chodcov na báze AOP[1]

Andrej Lúčny

MicroStep-MIS, Čavojského 1, Bratislava 841 04 a KAI FMFI UK Mlynská Dolina, Bratislava, 812 19
andy@microstep-mis.com

Abstrakt

Príspevok referuje o našej účasti na konkrétnom projekte na rozpoznávanie chodcov ktorý viedol DENSO Research Lab. Zaoberáme sa rekonštrukciou 3D scény obsahujúcej chodcov na základe videa z kamery umiestnenej v aute. Používame na to špecifický spôsob programovania - istý druh agentovo-orientovaného programovania (AOP), ktoré vychádza jednak z Brooksovej sumbsumpčnej architektúry, jednak je filozoficky blízky Minského societnému modelu mysle.

1 Úvod

Tvorcovia filmu A.I. urobili zaujímavú chybu, keď dobe, v ktorej ľudstvo disponuje robotmi, ktoré učia panny milostnému umeniu, prisúdili automobily, ktoré treba riadiť pomocou volantu (dôkaz je na obr. 1). Považujeme za nepochybné, že raz budeme disponovať plne automatickými automobilmi, v ktorých sa ľudia budú iba viezť a tú druhú spomínanú aktivitu si zase zrejme plne ponecháme na svojich pleciach. Samozrejme, dnešnej technológii chýba na plnú automatiku auta výkon i know-how. Avšak automobilový priemysel je natoľko masový, že každé drobné zlepšenie prináša obrovskú návratnosť investícií vynaložených na jeho vývoj. Najmä pokiaľ je jeho nasedenie na jeden kus auta relatívne lacné.

Obr. 1. *A.I.: umelý milenec versus automobil*

Jedným zo senzorov, ktoré sa v ostatnom čase presunuli do kategórie lacných je digitálna kamera. Na základe toho faktu DENSO CORPORATION (zduženie japonských automobiliek) poverilo svoj Research lab vyskúmať do roku 2010 možnosti použitia jednej obyčajnej digitálnej kamery na rozpoznávanie scény zo stojaceho i idúceho auta so zámerom vytvoriť varovný systém, ktorý by vodiča upozorňoval napríklad, že mu nejaký chodec môže skončiť pod kolesami. Podobné iniciatívy majú aj ostatní automobiloví giganti.

Iniciatívna DENSO bola pre nás výnimočná tým, že MicroStep-MIS získal možnosť sa do nej na krátky čas zapojiť. Naša úloha nám bola šitá na mieru – mali sme vyskúšať rozpoznávanie chodcov z idúceho alebo stojaceho auta s dôrazom na možnosť ich čiastočného aj úplného zákrytu – pomocou systému zloženého z heterogénnych agentov. Zostavili sme preto tím zložený z ľudí FMFI UK Bratislava, FIIT a FEI STU Bratislava a MicroStep-MIS (Miroslav Baláž, Oto Kužma, Jana Lachová, Andrej Lúčny, Michal Malý, Pavel Petrovič, Martin Šperka, Rastislav Šramek a Peter Tóth) a pustili sa do práce. Pritom sme ako technologický framework použili agent-space [8][9], čo je jedna z architektúr agentovo-orientovaného programovania.

2 Agentovo-orientované programovanie

Pod AOP máme na mysli spôsob tvorby software, ktorý sa opiera o poznatky získané v oblasti multiagentových systémov (MAS). Zatiaľ čo MAS sa zameriava na distribuované systémy (typická aplikácia je napríklad hranie tímového robotického futbalu), pod AOP chápeme použitie rovnakej modularity pre tvorbu software, ktorý nemusí mať distribuovanú povahu (napríklad riadiaci systém jedného mobilného robota). Pri AOP organizujeme software do samostatne aktívnych modulov, ktoré – hoci sa nachádzajú na jednom mieste - vzájomne komunikujú ako keby boli priestorovo distribuované: posielajú si správy a to buď priamo jeden druhému, alebo sprostredkovane skrz pomyselný priestor, v ktorom sa nachádzajú. Tak ako objektovo orientované programovanie (OOP) do modulov pridáva k dátam kód, AOP k tomu pridáva ešte vlastné riadenie (program counter). Kým kód modulov OOP (t.j. metódy v tzv. objektoch) sa vykonávajú v rôznych vláknach podľa toho kto ich zavolá, kód modulov AOP (tzv. agentov) sa vykonáva každý vo vlastnom vlákne. Kým objekty sú odkázané čakať na chvíľu keď ich niekto zavolá, agenty – obrazne povedané - žijú svojím vlastným životom.

[1] Tento výskum podporoval DENSO Research Lab.

Hovoríme, že zatiaľ čo objekty sú reaktívne, agenty sú tzv. proaktívne (obr. 2).

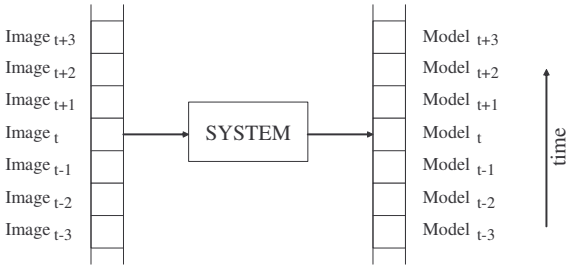
Obr. 2. *Moduly a ich aktivita: pasívna štruktúra je ako stena, od ktorej možeme utrpieť úraz len tak, že my sa zraníme na nej. Objekt je reaktívny ako hrable: dokážu nás zraniť, ale musíme na ne najprv stúpiť. Zato však agent je proaktívny ako pes, ktorý nás môže zraniť bez toho, že by sme to nejako iniciovali.*

Z implementačného hľadiska agenta stvoríme najjednoduchšie v nejakej objektovo orientovanej virtuálnej mašine tým, že objektu priradíme jeho vlastné vlákno a dáme si záväzok nevolať jeho metódy z iných objektov. Navyše musíme implementovať posielanie správ medzi agentami, využijúc spravidla sychronizačné mechanizmy danej platformy. Tým, že sa kód agenta vykonáva v jedinom vlákne a nie ako séria tzv. spätných volaní (callback), má za následok, že tento kód je jednoduchší a ľahšie zrozumiteľný. Na druhej strane je však často menej efektívny. Tento fakt naznačuje, že AOP uľahčuje prácu vývojára na úkor práce počítača a je preto vhodné pre úlohy vyznačujúce sa extrémnou komplexnosťou problému.

V prípade architektúry agent-space je úplne potlačená priama komunikácia medzi agentami, čiže dva agenty sa môžu dohovárať len nepriamo cez priestor v ktorom sa nachádzajú. Robia tak tým, že do priestoru ukladajú pomenované dáta, tzv. bloky a na základe ich mien ich zase čítajú (tieto dáta niektorí nazývajú značky – stigmy a od toho tento spôsob výmeny dát stigmergickou komunikáciou). Keď sa detaily pravidiel na čítanie a zapisovanie blokov urobia dostatočne šikovne (čo je práve podstatou architektúry agent-space), je vždy možné aby do komunikácie medzi dvomi agentami, vstupoval tretí, monitoroval túto komunikáciu, zastavil ju, alebo dokonca zmenil. Tým sa vytvára priestor pre aplikáciu všetkých trikov zavedených Brooksovou subsumpčnou architektúrou [2], ako sú supresia, inhibícia a inkrementálny vývoj zdola nahor. Zároveň je ale možné pri návrhu systému zložených z takýchto agentov používať filozofiu typickú pre Minského societný model mysle [11]. Organizáciu softwaru vyjadrenú v agent-space možno teda prirovnávať k určitým predstavám o organizáciu mozgu, resp. mysle. Navyše je v rámci nej možné bojovať s klasickými problémami subsumpčnej architektúry ako je neprístupnosť vnútorného stavu a to tým, že si agenty do priestoru zapisujú aj všetky interné údaje, ktoré by normálne v sebe dlhodobo prechovávali.

Systém skonštruovaný na princípoch agent-space pozostáva teda z mnohých, v princípe heterogénnych agentov, ktoré manipulujú s blokmi v priestore okolo nich. Aby sa zabránilo live-lock-u je potrebné vykonávanie kódu agenta blokovať. Je vhodné na to používať dva mechanizmy a to časovač (budiaci agenta v pravidelných intervaloch) a spúšťač (budiaci agenta, keď sa zmení obsah určitých blokov). Pritom agenty nie sú budené v rovnakých časových periódach, takže môžeme využiť, že jeden dokáže reagovať na situáciu vo svojom priestore rýchlejšie (a trebárs menej šikovne) a druhý pomalšie (a šikovnejšie). Vyzerá to tak, že prvý rýchlo zapíše určitý blok a po čase mu druhý jeho hodnotu prepíše svojou. Avšak z pohľadu agentov, ktoré túto hodnotu čítajú je jedno od koho hodnota pochádza, pre nich je to proste ďalší variant hodnoty bloku, ktorý ich zaujíma. Čitateľov i zapisovateľov blokov možno ľubovolne pridávať a odoberať, každý zapisovateľ sa môže rozhodnúť zapísať alebo ostať ticho a podobne. Každý blok v princípe realizuje dátový tok od mnohých k mnohým. Systém tak možno usporiadať spôsobom, že v rôznych stavoch priestoru sú aktívne rôzne agenty (pod aktívnym rozumieme takého agenta, ktorý nielen počíta, ale aj zapisuje výsledok). Aby určité hodnoty neboli v blokoch prechovávané do nekonečna a nebolo treba pridávať do systému agenty, ktoré by ich vymazávali, umožňuje architektúra agent-space zapisovať hodnoty s obmedzenou časovou platnosťou. Je to vhodné hlavne pre systémy bežiace v reálnom čase.

Na rozdiel od subsumpčnej architektúry, kde majú jednotliví zapisovatelia hodnoť stanovenú jednoznačnú prioritu, v agent-space je možné simulovať, že si vzájomne konkurujú: jeden prepisuje hodnotu zapísanú druhým a naopak. Konkurencia sa pritom ukazuje ako zaujímavý tvorivý princíp.



Obr. 3. Rozpoznávanie v rélanom čase

3 Rozpoznávanie chodcov

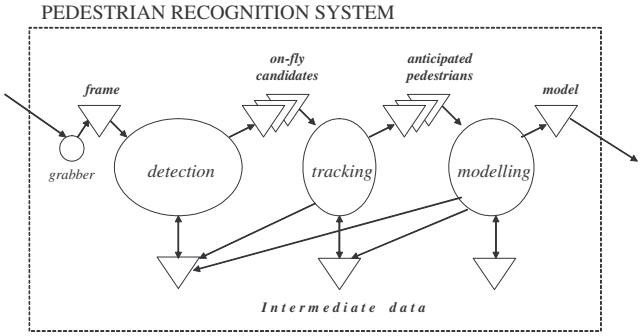
Na problém rozpoznávania chodcov sme sa na základe zvolenej architektúry systému pozerali od začiatku ako na problém odohrávajúci sa v reálnom čase (obr. 3). To je dosť originálny pohľad, lebo spravidla sa na tento problém, nazerá ako na pipe-line pozostávajúci z postupného aplikovania určitých metód na obraz a len dodatočne sa mu pridáva časový rozmer, respektíve je premietnutý do časovo súbežného toku medzivýsledkov v rámci jednej metódy. V týchto tradičných architektúrach však spravidla nie je možné, aby vyššie metódy spracovania obrazu ovplyvňovali nižšie metódy v nasledovných časových okamihoch. Je fakt, že aj bez pohnutia scény a nás samotných vieme po otvorení očí rozpoznať kompletnú scénu. To však neznamená, že v našej hlave prebehne len jeden krok spracovania a potom sa už nič nepohne. Naopak, scénu rozpoznávame - síce rýchlo ale – postupne.

Podobne za výsledok rozpoznávania sa spravidla považuje označenie rozpoznávaného objektu na obraze. Uvedomovali sme si, že takto by len veľmi ťažko mohol systém rozpoznávania robiť úvahy typu „chodci nelietajú a pritom sa tento potenciálny chodec vznáša vo výške niekoľkých metrov - asi bude reklamný balón“.

```
<recognition>
  <object type='pedestrian' key='132'>
    <rectangle2D x1='522' y1='290' x2='551' y2='376'/>
    <speed2D dx='0' dy=0'/>
    <position3D x='2.12' y='0' z='8.54'/>
    <size3D dy='1.67'/>
    <speed3D dx='0' dy='0' dz='0'/>
  </object>
</recognition>
```

Obr. 4. *Vstup a výstup do rozpoznávania chodcov*

Chodca je ľahšie rozpoznať, keď si za cieľ stanovíme vedieť o ňom oveľa viac. Požadovali sme teda výstup vo forme modelu, ktorý by o chodcovi vedel všetky 2D i 3D údaje (obr. 4) - natoľko úplné, že je už hračkou pridať predikciu, či nejaký chodec smeruje pod kolesá alebo nie. Celý systém sme rozdelili na tri časti: detekciu, sledovanie a modelovanie (obr. 5).



Obr. 5. *Celkový pohľad na náš prístup k rozp. chodcov*

Úlohou detekcie je spracúvať nasnímaný obraz (obr. 6) z bloku *frame*, ktorý je v pravidelnom časovom intervale aktualizovaný novým snímkom z kamery, čo zabezpečuje agent *grabber.* (Vzhľadom na nízky výkon súčasných počítačov sme frekvenciu aktualizácie pre naše potreby znížili 1000 krát.)

Obr. 6. *Príklad vstupného snímku.*

Detekcia produkuje množinu okamžitých kandidátov, t.j. takých objektov, ktoré sa jednou z metód detekcie dajú rozpoznať priamo na súčasnom snímku alebo porovnaním s predchádzajúcim (obr. 7). Pritom sa počíta s tým, že tieto metódy môžu byť veľmi nespoľahlivé: jednak nemusia rozpoznať objekt, jednak ho nemusia rozpoznať správne. Úlohou detekcie je

taktiež vysporiadať sa so základným spracovaním obrazu pre detekciu ako je rotácia do vodorovnej roviny, výpočet úbežného bodu, kalibrácia výpočtu 3D súradníc (kedže máme len jednu kameru vieme vypočítať hĺbku len pre objekty, ktoré sa nachádzajú v známej výške, napríklad na zemi), segmentácia farebného, či vyhranenie čiernobieleho obrazu a podobne. V rámci detekcie sa taktiež eliminuje pohyb auta a sleduje sa pohyb objektov na obraze voči predchádzajúcim snímkom.

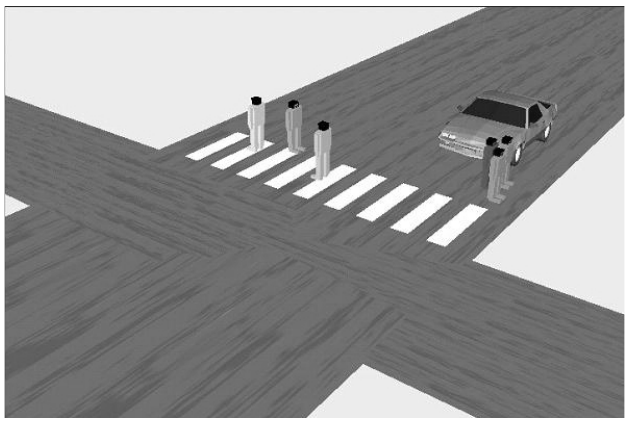
Obr. 7. *Príklad okamžitých kandidátov.*

Ďalšou fázou je sledovanie, ktoré z okamžitých kandidátov vygenerovaných detekciou konštruuje predstavu o dlhodobom pohybe skutočných chodcov (obr. 8), pričom to robí na základe toho ako sa chodec môže pohybovať a ktoré parametre sa na ňom pritom nemenia. Ak si sledovanie raz urobí hypotézu, že určitý objekt na obraze je chodec pohybujúci sa určitým smerom a rýchlosťou, snaží sa v budúcich okamžitých kandidátoch nachádzať jeho budúci obraz a konštruovať jeho kontinuitu v čase.

Obr. 8. *Príklad anticipovaných chodcov*

Na základe toho môže spätne ovplyvňovať detekciu aby sa zameriavala na miesta, kde je tento chodec očakávaný; napríklad môže v danej oblasti zahustiť semiačka segmentácie. Ak sa aj napriek tomu nepodarí chodca zdetekovať , zrejme sa stal obeťou zákrytu iným objektom, napríklad autom, ktoré stojí pred kamerou. Ak však v predpokladanom mieste detekujeme nejakú časť chodca, napríklad hlavu, sledovanie vie urobiť predpoklad, že to bude hlava toho strateného chodca a že tam niekde musí byť celý. Ak sa nepodarí ani to, vieme predpokladať, akým smerom a rýchlosťou chodec pôjde a kde a kedy sa približne vynorí. Očakávanie sa tak stáva dôležitým pre samotné vnímanie chodcov, preto o nich hovoríme ako o anticipovaných.

Poslednou fázou je modelovanie, ktoré skompletizuje informáciu o anticipovaných chodcoch do modelu, z ktorého je okrem iného možné vygenerovať 3D scénu a znázorniť ju z ľubovoľného stanovišťa (obr. 9).

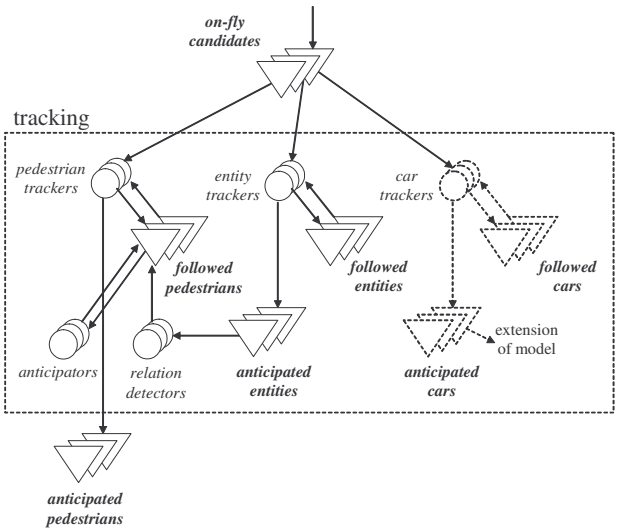


Obr. 9. *Príklad znázornenia výstupného modelu*

4 Sledovanie pomocou agentov

V ďalšom výklade sa obmedzíme na sledovanie, v ktorom sa na dosiahnutých výsledkoch najviac podpísala agentová povaha riešenia. Vychádzali sme z predpokladu, že ani človek nie je schopný sledovať neobmedzený počet chodcov. Pre jednoduchosť sme predpokladali, že máme len tri typy okamžitých kandidátov: pravdepodobného chodca, pravdepodobnú hlavu a prípadne pravdepodobné auto. (Získame ich dosť stupídnymi metódami, napr. hlavu pomocou troch neurónových sietí hľadajúcich hlavy troch rôznych veľkostí v troch vodorovných pásoch na obraze, chodca vyhodnotením segmentov v obdĺžnikoch očakávanej veľkosti podľa jeho z-súradnice za predpokladu, že stojí

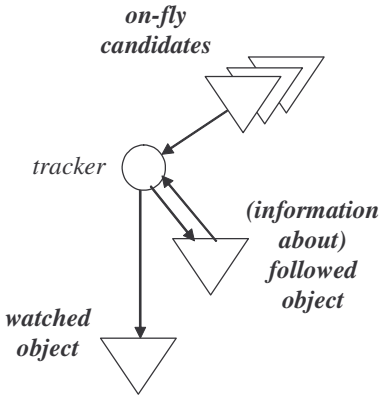
na zemi a pod.). Medzi kandidátmi máme teda aj načisto falošných a niektorí skutoční chodci či hlavy tam môžu zase chýbať. Budú však chýbať skôr chvíľami než trvale. Ako teraz tento zmätku interpretovať aby zodpovedal reálnej scéne s chodcami, ktorí sa pohybujú pre nich typickým spôsobom? Jednou z možností je využiť agentovú povahu riešenia a zaviesť do systému konečný počet rovnakých agentov, ktoré sa snažia ztotožniť s nejakým reálnym chodcom (obr. 9).



Obr. 9. *Agentová povaha sledovania*

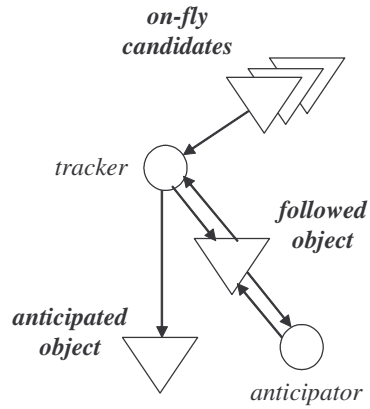
Na začiatku tento sa agent – volajme ho *tracker* – snaží medzi okamžitými kandidátmi nájsť ľubovoľného svojho typu, napríklad *tracker* hlavy hlavu (obr. 10). Akonáhle sa mu to podarí, zapamätá si čo sleduje - je to ako keby si skopíroval blok so zvoleným kandidátom do vlastnej kópie (túto kópiu drží v bloku v priestore, hoci by si ju mohol pamätať v internej pamäti – je to tak kvôli vyššie zmienenému problému neprístupnosti vnútorného stavu). V ďalších časových okamihoch (t.j. pri pracovaní ďalších vstupných snímkov) sa medzi okamžitými kandidátmi snaží nájsť niečo podobné ako má zapamätané v kópii (na základe 2D i 3D polohy, 3D veľkosti, 2D smeru pohybu, farby a pod.) Ak sa mu to podarí upraví si príslušne svoju kópiu. Kópia mu teda vraví, kde naposledy videl objekt s ktorým sa ztotožnil. Keď už chvíľu svoj objekt sleduje, nadobúda istotu, že je skutočný a – v prípade že je to objekt zaujímavý pre výsledný model - začne ho dávať na výstup zo sledovania. Ak naopak už dlho nestretol nič podobné svojej kópii, a teda sa mu ju nepodarilo občerstviť, kópia vďaka obmedzenej časovej platnosti expiruje a agent tým pádom opäť začína hľadať s čím by sa ztotožnil.

Pri takomto prístupe samozrejme nič nebráni, aby sa dva agenty ztotožnili s rovnakým objektom. Preto každý agent, ktorý je už ztotožnený s niečím určitý čas, sleduje kópie svojich druhov a cieľavedome likviduje ich obsah, pokiaľ v nich nájde niečo, čo sa príliš podobá na jeho kópiu. Agenty, ktoré sa teda snažia ztotožniť s niečím už sledovaným, teda nebudú v tejto snahe dlhodobo úspešné. Týmto mechanizmom sa medzi agentov zavádza konkurenčný vzťah (pre podčiarknutie originálnosti tohto prístupu si uvedomme, že subsumpčnej architektúre by niečo podobné robilo vážne problémy).



Obr. 10. *Agent tracker*

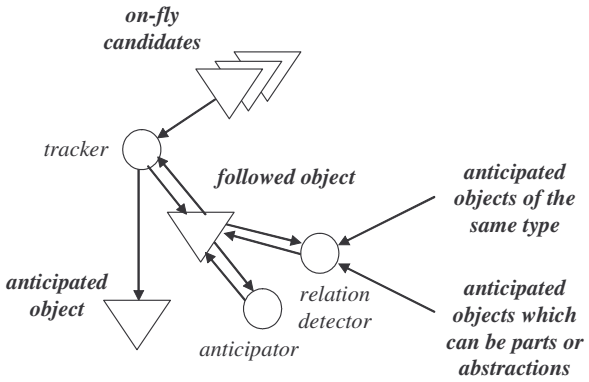
V tomto momente už systém vcelku funguje, pokiaľ býva dĺžka výpadku detekcie len krátka. Ak však chodec zájde za auto, *tracker* ho dlhodobo stratí a teda príde k tomu, že jeho kópia expiruje. Aj keby sme rovno predĺžili jej platnosť, potom, čo chodec vyjde na druhej strane auta, sa už agentovi nemôže podariť pochopiť, že ide o toho istého, lebo nebudú mať podobnú polohu. Toto riešenie teda nie je odolné voči zákrytu.



Obr. 11. *Agent anticipator*

Môžeme však využiť triky inkrementálneho vývoja zdola nahor na to, aby sme ho odolným urobili (obr. 11).

Pridáme každému *trackeru* agenta, ktorý sa bude o jeho kópiu starať na základe predikcie (nazveme ho *anticipator*). Kópiu bude sledovať a bude si počítať smer a rýchlosť jej pohybu. Keď spozoruje, že kópia začína starnúť (preto lebo ju *tracker* nevie s niečím novým ztotožniť), začne ju občerstvovať sám - na základe interpolácie jej doterajšieho pohybu. Takto je dosť pravdepodobné, že keď objekt zo zákrytu vylezie, bude opäť správne ztotožnený.



Obr. 12. *Agent relation detector*

Podobne môžeme využiť fakt, že pokiaľ počas zákrytu určitého objektu, napr. chodca, vidíme aspoň nejakú jeho časť, napr. hlavu, môžeme využiť znalosť o ich doterajšom previazanom pohybe a ešte presnejšiu vedomosť o tom, kde sa zakrytý objekt pohybuje (obrázok 12). Pridáme teda agenta (nazveme ho *relation detector*), ktorý vyhľadáva v kópiách rôznych typov objekty, ktoré sa súbežne hýbu a podobne ako *anticipator* občerstvuje staršie kópie na základe stavu ich čerstvejších súpútnikov. Týmto spôsobom napr. detekcia hlavy pomáha vyrovnať handicapy v detekcii postavy a naopak.

Takýchto trikov je možné robiť viac. Každopádne nepríjemnou súčasťou tejto práce je nastavenie mnohých konštánt, potenciálne závislých od konkrétneho obrazu. V našej implementácii sme tieto konštanty najprv odhadli, potom sme vygenerovaný výsledok ručne opravili a konštanty sme upravili voči ideálnemu výsledku obyčajnou metódou najmenších štvorcov. Podarilo sa tak dostať celkom uspokojivé konštanty, ale tieto by zrejme neboli veľmi úspešné pre iné vstupné video. Od reálneho použitia to má teda dosť ďaleko.

5 Záver

Nami navrhované riešenie rozpoznávania chodcov má teda veľmi ďaleko od realizácie vo forme nejakého funkčného produktu. Nedisponuje dokonca schopnosťou bežať na súčasnom hardwari v reálnom čase. Snaží sa však naznačiť ako by sa na vec malo ísť, vychádzajúc z predpokladu, že táto úloha sa nedá uspokojivo vyriešiť jednou metódou či algoritmom, ale vyžaduje komplexné riešenie. Ako jeden z čiastkových výsledkov sme ukázali, že ak dostávame z obrazu neúplnú a zašumenú informáciu o polohe tiel chodcov a ich hláv, vieme pomerne dobre rekonštruovať 3D scénu s pohybujúcimi sa chodcami. Dúfame, že sme tak svojou trochou prispeli k bezpečnosti automobilovej dopravy a naplneniu snaží tých, ktorý onen dopravný prostriedok nazvali automobilom, t.j. samochodom.

Literatúra

[1] Broggi, A., Bertozzi, M., Fascioli, A., Sechi M.: *Shape-Based Pedestrian Detection.* IEEE Intelligent Vehicles Symposium, IEEE Press, Piscataway, N.J., 2000

[2] R. A. Brooks: *Cambrian Intelligence*, The MIT Press, Cambridge, Mass., 1999.

[3] Brooks, R.: *Robot – The Future of Flesh and Machines.* Penguin Books, London, 2002

[4] Davies, E. R.: *Machine Vision: Theory, Algorithms, Practicalities*., Elsevier, 2005

[5] Gavrila, D.M.: *The Visual Analysis of Human Movement: A Survey.* Computer Vision and Image Understanding, vol. 73, no. 1, Academic Press, 1999

[6] Gavrila, D.M., Groen, F.C.: *3-D Object Recognition from 2-D Images Using Geometric Hashing*. Pattern Recognition Letters, vol. 13, nr. 4, 1992

[7] Kelemen, J.: *The Agent Paradigm.* Computing and Informatics, Vol.22. (2003), pp. 513-519

[8] Lúčny, A.: *Building Complex Systems with Agent-Space Architecture.* Computing and Informatics, Vol. 23 (2004), pp. 1001-1036

[9] Lúčny, A.: Architektúra A*gent-Space.* doctoral theses. FMFI UK Bratislava, 2005

[10] Lúčny, A., Lachová J.: Pedestrian recognition with heterogeneous agents. DENSO Research Lab Report, Bratislava, 2005

[11] Minsky, M.: *The Society of Mind.* Simon&Schuster, New York, 1986

[12] Papageorgiou, C., Evgeniou, T., Poggio, T.*: A Trainable Pedestrian Detection System*. Procs. IEEE Intelligent Vehicles Symposium, 1998

[13] Sonka, M., Hlavac, V., Boyle, R.: *Image processing, Analysis, and Machine Vision*. 2nd. ed. PWS Publishing, 1999