UNIVERZITA KOMENSKÉHO V BRATISLAVE
FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY

FLY ALGORITHM

2011

Robert Maurer

UNIVERZITA KOMENSKÉHO V BRATISLAVE
FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY

c1b0bf54-73a0-4048-a1a0-be55da58bd4a

FLY ALGORITHM

Bakalárska práca

Študijný program : Aplikovaná informatika
Študijný odbor: 9.2.9 Aplikovaná informatika
Školiace pracovisko: Katedra aplikovanej informatiky
Školiteľ: Mgr. Pavel Petrovič, PhD.

Bratislava 2011

Robert Maurer

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Róbert Maurer
Študijný program: aplikovaná informatika (Jednoodborové štúdium, bakalársky I. st., denná forma)
Študijný odbor: 9.2.9. aplikovaná informatika
Typ záverečnej práce: bakalárska
Jazyk záverečnej práce: slovenský

Názov: Fly algorithm

Cieľ: Cieľom je prakticky overiť činnosť algoritmu na spracovanie stereo-videnia, Fly Algorithm. Úlohou študenta bude zmontovať robota, nainštalovať na neho systém pre stereovidenie a spracovať stereo obraz pomocou evolučnej metódy Fly Algorithm. Študent vyhodnotí a analyzuje činnosť algoritmu v reálnych podmienkach. Využité systémy: Tetrix Robotics, Surveyor Stereo Vision System, príp. Gumstix.

Literatúra: Olivier Pauplin, Jean Louchet, Evelyne Lutton, Michel Parent: APPLYING EVOLUTIONARY OPTIMISATION TO ROBOT OBSTACLE AVOIDANCE. Jean Louchet, Maud Guyon, Marie-Jeanne Lesot, Amine Boumaza: Dynamic flies: a new pattern recognition tool applied to stereo sequence processing.

Vedúci: Mgr. Pavel Petrovič, PhD.

Spôsob sprístupnenia elektronickej verzie práce:
bez obmedzenia

Dátum zadania: 01.10.2010

Dátum schválenia: 25.10.2010

doc. RNDr. Mária Markošová, PhD.
garant študijného programu

Maurer
študent

Pavel Petrovič
vedúci

Čestné prehlásenie

Čestne prehlasujem, že túto bakalársku prácu som vypracoval samostatne
s použitím uvedenej literatúry.
Bratislava 1. jún 2011

.....
Robert Maurer

Poďakovanie

Za cenné rady a pripomienky k mojej práci ďakujem svojmu vedúcemu bakalárskej práce Mgr. Pavelovi Petrovičovi, PhD.

Abstrakt

MAURER, Robert: Fly algorithm [Bakalárska práca]. Univerzita Komenského v Bratislave. Fakulta matematiky, fyziky a informatiky; Katedra aplikovanej informatiky. Vedúci bakalárskej práce: Mgr. Pavel Petrovič, PhD. Bratislava: FMFI UK, 2011. 32 s.

Bakalárska práca sa zaoberá fly algorithm. Cieľom tejto práce je zanalyzovať predchádzajúci výskum týkajúci sa tohto algoritmu, zhotoviť vlastnú implementáciu, poskladať robota a otestovať ju v reálnom svete, v našom vlastnom prostredí. Fly algorithm je evolučný algoritmus, ktorý slúži na rýchle spracovanie informácií poskytnutých stereo obrazom, hrubú reprezentáciu trojrozmernej scény. Súčasťou práce je program naprogramovaný v C++, ktorý implementuje daný algoritmus.

Kľúčové slová: Fly algorithm, stereo videnie, robotika, evolučné algoritmy

Abstract

MAURER, Robert: Fly algorithm [Bachelor thesis]. Comenius University in Bratislava. Faculty Of Mathematics Physics And Informatics; Department Of Applied Informatics. Supervisor: Mgr. Pavel Petrovič, PhD. Bratislava: FMFI UK, 2011. 32 s.

This bachelor thesis deals with fly algorithm. The goal of this thesis is to analyse previos research concerning this algorithm, construct own implementation, assemble a robot and test it in the real world, in our own environment. Fly algorithm is evolutionary algorithm, which serves for rapid processing of information provided by stereo image pairs, raw representation of 3D scene. A part of the thesis is program made in C++, which implements the given algorithm.

Keywords: Fly algorithm, stereo vision, robotics, evolution algorithm

Obsah

1 Úvod

Zrak je pre človeka zdrojom 85% informácií. Počítačové videnie v oblasti robotiky nie je priveľmi odlišné a zo všetkých druhov senzorických vstupov robota, práve z počítačového videnia môžeme extrahovať najrozmanitejšiu paletu informácií.

V tejto práci sa budeme venovať jednému konkrétnemu druhu spracovania informácií z obrazu poskytnutého stereo kamerami pomocou fly algorithm.

Cieľom tejto práce je zanalyzovať predchádzajúci výskum týkajúci sa tohto algoritmu, zhotoviť vlastnú implementáciu a otestovať ju v reálnom svete, v našom vlastnom prostredí.

Fly algorithm je evolučný algoritmus, v ktorom je jedna trojrozmerná scéna reprezentovaná takým spôsobom, kde každý jedinec je časťou scény a jedna celá populácia je reprezentácia jednej scény, na rozdiel od iných evolučných algoritmov, s ktorými sa viedli pokusy riešiť tento problém, kde je každý jedinec komplexnou reprezentáciou celej scény. Využitie fly algorithm bolo podľa použitých zdrojov testované v oblasti bezpečnosti dopravy osobným vozidlom a v oblasti robotiky pri vyhýbaní sa prekážkam.

Prvá časť práce, je rozdelená na druhú kapitolu v ktorej je popísaný algoritums a tretiu, v ktorej je popísaný predchádzajúci výskum. Druhá časť je rozdelená na štvrtú a piatu kapitolu, v ktorých sú popísané potrebné prerekvizity pre naprogramovanie a použitie fly algorithm. V tretej časti práce v šiestej kapitole je popísaná implementácia a v siedmej je popísané jej testovanie pomocou kamier.

2 Fly algorithm

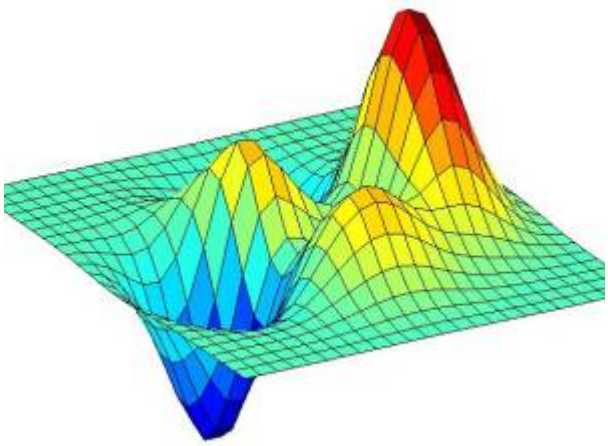
V tejto kapitole opíšeme fly algorithm na základe informácii z [2] a [3].

2.1 Fly algorithm

Jedná sa o evolučný algoritmus určený pre spracovávanie informácií z párového obrazu stereo videnia. Používa sa napríklad pri detekovaní prekážok pred objektom na ktorý sú stereo kamery namontované. V krátkosti ide o vypustenie pomyselných múch, čo sú body v trojrozmernom priestore, do spoločného zorného poľa dvoch kamier. Následne na to sa počíta kde sa muchy premietajú na jednom a druhom obraze kamier. Potom sa každá mucha na základe týchto priemetov ohodnotí a pomocou genetických operátorov sa vyprodukuje ďaľšia generácia múch, pričom za gény sa považujú súradnice múch v priestore. Postupnou iteráciou z generácie na generáciu získame populáciu múch ktoré nám pomôžu detekovať objekty na scéne, keďže: „Darvinova teória predpokladá, že populácia jedincov, charakterizovaná ich génmi, sa bude vyvíjať smerom k lepšej adaptácii na svoje prostredie, podľa zákonov prirodzeného výberu..”[2]

2.2 Účelová funkcia

Je to najpodstatnejšia funkcia akéhokoľvek evolučného algoritmu. V prípade fly algorithm, zaručuje ohodnotenie muchy na základe jej pozície na primetoch, ktorá je určená pozíciou v trojrozmernom priestore, a na základe stereo obrazu, ktorý je momentálne k dispozícii.



Obrázok 1. Príklad fitness landscape(ktorý sa dá obyčajne reprezentovať ako graf).

Zobrazenie možného ohodnotenia v celom priestore si môžeme predstaviť ako fitness landscape (obrázok 1). V našom prípade (ako bude nižšie vysvetlené) má účelová funkcia

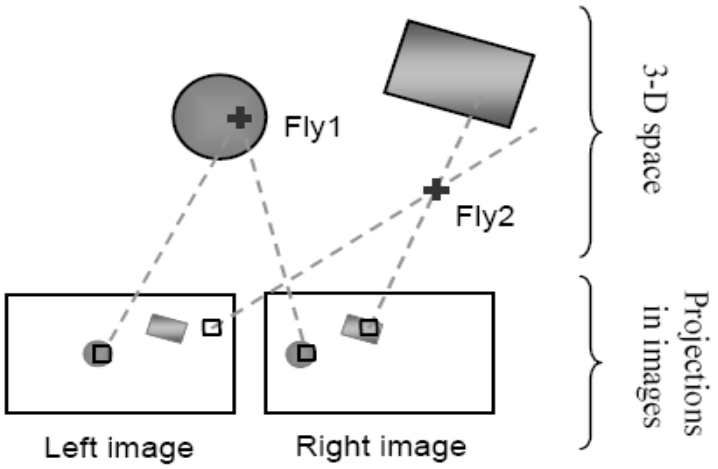
v čitateli zlomku hodnoty zo Sobelovho gradientu pre danú muchu na ľavom aj pravom obrázku, a v menovateli hodnoty farebného rozdielu:

$$fitness = (\,|\, G_l * G_r \,|\, / \sum_{farby\,(i,j)\in N} \sum [L(x_L+i, y_L+j) - R(x_L+i, y_L+j)]^2) \quad (1)$$

Tým pádom, tak ako v každom evolučnom algoritme poskytuje najvyššie ohodnotenie pre tie jedince, ktoré sú z hľadiska algoritmu najužitočnejšie.

2.3 Predpoklad z priemetu

Pri hodnotení múch vychádzame z predpokladu, že mucha ktorá sa nachádza na objekte sa premietne v jednom aj druhom obrázku niekde tam, kde sa premieta aj samotný objekt. V prípade, že sa mucha nachádza vo voľnom priestore okolo objektov, sa určite nepremieta na oboch obrazoch na miesta kam sa premieta ktorýkoľvek jeden objekt.



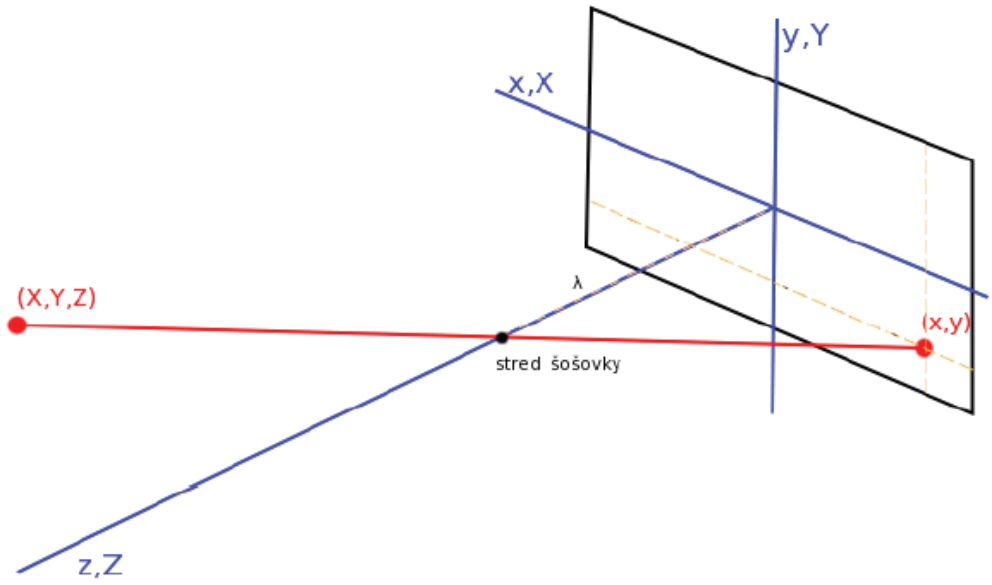
Obrázok 2. Premietanie sa múch (Prevzaté z [2]).

Napríklad na obrázku sa Fly2 nachádza pred objektom, nie na ňom, a tak sa premieta na ľavom obrázku niekam do priestoru a na pravom obrázku sa premieta na kváder. Z tohto predpokladu vyplýva, že mucha ktorá sedí na objekte má svoje okolie na ľavom a pravom obrázku farebne podobnejšie, ako mucha ktorá na nesedí na žiadnom objekte, s čím sa počíta v účelovej funkcii.

2.4 Prepočítavanie súradníc

Na to aby sa dala vypočítať hodnota fitness muchy je potrebné poznať jej súradnice na priemetoch. Na vypočítanie týchto súradníc slúži vzorec, ktorý som našiel v [1] – Návrh

kamerového cestného radaru: „Bod (*X, Y, Z*) vo svetovom súradnicovom systéme sa premietne do bodu (*x, y*) na premietacej rovine xy (napríklad fotografia), čo je vlastne bod (*x, y*, 0) kamerového systému.



Obrázok 3. Závislosť súradníc na priemetni kamery(obdĺžnik) od súradníc v priestore (Prevzaté z [1]).

Súradnice bodu (*x,y*) vieme vypočítať vďaka podobnosti trojuholníkov (obrázok 3), kde môžeme o strede šošovky uvažovať ako o bode (0,0,*λ*) kde *λ* je ohnisková vzdialenosť.”[1]

$$x/\lambda = -(X/Z - \lambda) = X/(\lambda - Z) \quad (2)$$

takže

$$y = \lambda * Y/\lambda - Z \quad (3)$$

a obdobne

$$x = \lambda * X/\lambda - Z \quad (4)$$

2.5 Sobelov gradient

V prípade, že by sme prihliadali iba na farebné okolie múch, by rozsiahle jednofarebné plochy (ako napríklad stena miestnosti za scénou, obloha) poskytovali veľké oblasti s vysokou hodnotou fitness pre muchy. Následkom tohto javu by bolo prežívanie mnohých múch v týchto nepotrebných oblastiach. Preto je vhodné pri vypočítavaní hodnoty fitness použiť aj Sobelov gradient pre okolie muchy na ľavom aj pravom priemete. Tento

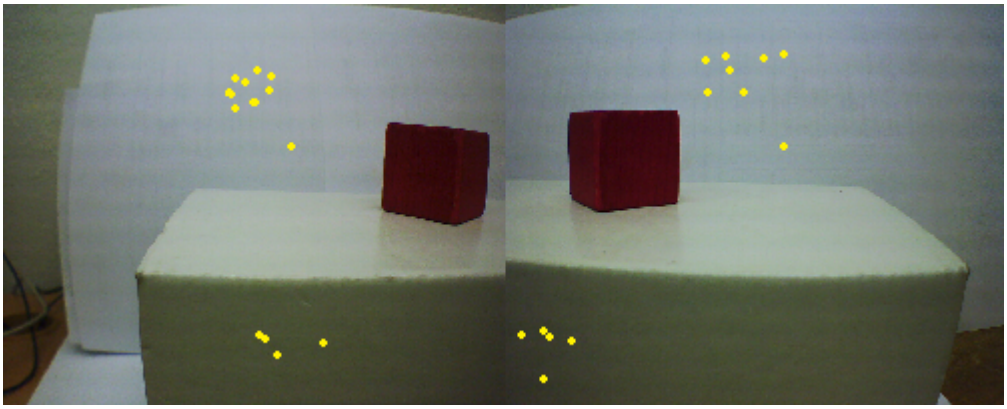
prídavok sústreďuje vrcholy fitness landscape z rozsiahlych jednofarebných oblastí na hrany medzi nimi.

Sobelov gradient pre daný pixel sa počíta tak že sa hodnota tohto pixla a hodnota okolitých pixlov v okolí 3x3 prenásobí maticami 3x3:

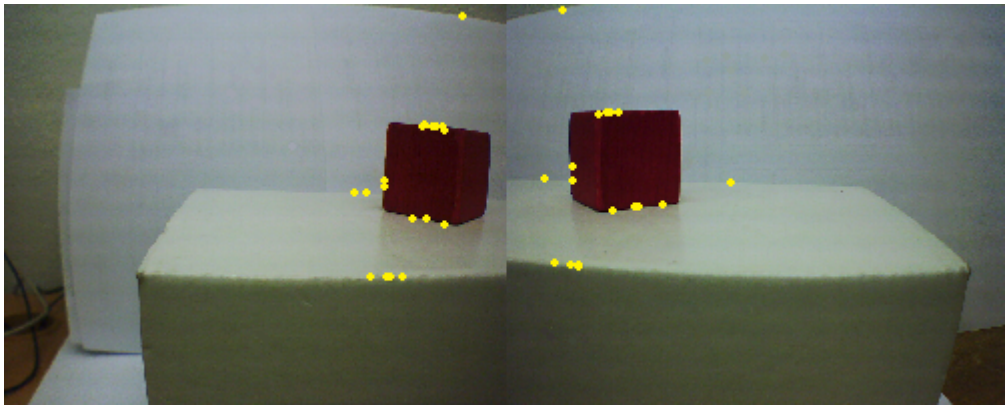
$$G_x = \begin{matrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{matrix} \text{ a } G_y = \begin{matrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{matrix} .$$

Následne sa gradient pre daný pixel získa pomocou vzorca:

$$G = \sqrt{(G_x^2 + G_y^2)} \quad (5)$$



Obrázok 4. Ilustračné sústredenie múch bez Sobelovho gradientu.



Obrázok 5. Ilustračné sústredenie múch so Sobelovym gradientom.

2.6 Genetické a iné operátory

V každej iterácii sa vykonávajú na populácii múch, aby zaručili, že sa vyvinú prirodzeným spôsobom tak aby ich hodnota fitness bola čo najvyššia.

2.6.1 Mutácia

Mutácia je náhodné pozmenenie génov. V našom prípade ide o náhodnú zmenu súradnic muchy v priestore o náhodnú hodnotu v malom rozsahu.

2.6.2 Migrácia

Migrácia je v našom prípade náhodné generovanie múch v zornom poli spoločnom pre jednu aj druhú kameru, čím sa zaručuje preskúmavanie nových oblastí v meniacom sa prostredí.

2.6.3 Kríženie

Kríženie sa vykonáva vytváraním múch na spojniciach medzi dvojicami už žijúcich múch. Miesto na spojnici sa vyberá náhodne.

2.6.4 Selekcia

Na základe fitness jednotlivých múch sa po ohodnotení vyberú najvhodnejšie jedince. Ktoré prežijú do ďalšej generácie.

2.6.5 Operátor zdieľania

V prípade, že na jednom mieste sedí veľa múch treba prinútiť ostatné objavovať okolitý priestor. Na toto slúži operátor zdieľania, ktorý zabije všetky muchy, ktoré sa nachádzajú tam kde už jedna mucha existuje.

3 Predchádzajúci výskum

3.1 Výhoda Parížskeho prístupu k Evolučným algoritmom

Jean Louchet napísal, že: „Parametricko-priestorové volebné prístupy k počítačovému videniu a k rozpoznávaniu vzorov ako napríklad zovšeobecnená Houghova transformácia sú limitované počtom parametrov a veľkosťou prehľadávacieho priestoru. Umelá evolúcia, videná ako trieda parametrických techník preskúmavania priestoru, sa ukázala ako úspešné riešenie na niektoré problémy spracovania obrazu ale často trpí zložitými výpočtami potrebnými na manipuláciu populácie, kde každý jedinec je sám o sebe komplexným opisom scény.”[3] Počítačové videnie je dôležitým aspektom pri návrhu samostatných robotov. Na to aby sa dalo využiť pre prácu s robotom, treba nejakým spôsobom spracovať informácie poskytnuté týmto videním. K spracovaniu sa môže pristupovať ako k riešeniu spätnej rekonštrukcie pravdepodobného modelu scény z obrázkov.

„Aj keď pravdepodobnostné optimalizačné metódy ako Evolučné Algoritmy sú v teórii skvele prispôsobené k riešeniu takýchto problémov, ich použitie v reálnych aplikáciách bývalo relatívne zanedbané pre ich nedostatočnú rýchlosť a vysokú komplexitu algoritmu.”[2] Toto je napísané ohľadom vývinu populácie, v ktorej by bol každý jedinec kompletnou reprezentáciou trojrozmerného priestoru, čo by kládlo omnoho väčšie nároky než súčasné optimalizačné algoritmy aj na veľkosť kódu aj na manipuláciu s pamäťou.

Avšak, technika Parížskej evolúcie uvedená Luttonom na riešenie optimalizačného problému v iterovaných funkčných systémoch, ukázala, že v niektorých prípadoch, rozdelenie reprezentácie objektu na menšie primitíva môže viesť k rýchlemu a efektívnemu riešeniu. Tento prístup k evolučným algoritmom berie každého jedinca ako časť riešenia problému, tým pádom je riešenie prezentované celou populáciou alebo jej veľkou časťou.

Fly algorithm bol vyvinutý na to aby spĺňal tieto kritériá pri riešení problémov počítačového videnia, pomocou účelovej funkcie založenej na operovaní nad pixlami obrazu takým spôsobom, že sa muchy koncentrujú na objekty, ktoré treba detekovať v scéne.

Informácie boli čerpané z [2] a [3].

3.2 Experimenty v reálnom prostredí

3.2.1 Popis

Muchy (čierne body) sa koncentrovali na prekážkach a v regiónoch kde úroveň sivého gradientu bola vysoká, napríklad na okrajoch cesty. Čitateľ účelovej funkcie zabraňoval muchám zachytiť sa v jednoliatych oblastiach (obloha, povrch cesty, atď.). Tri koordináty každej muchy boli známe, populácia múch poskytovala hrubý opis skutočnej trojrozmernej scény.

Obrázok 6. Scéna pred vozidlom (Prevzaté z [2]).

Pomocou fly algorithm ako pomocníka pri jazde autom vyvinuli Pauplin a jeho tým stratégiu na určenie pravdepodobnosti, že sa pred vozidlom nachádza prekážka. Ich cieľom bolo poslať autu príkaz spomaliť alebo úplne zastaviť v prípade, že sa prekážka objaví dostatočne blízko v zornom poli, čo malo zabezpečiť vyhnutie sa čelnej zrážke. Všeobecná idea bola použiť každú muchu ako zdroj "výstražnej hodnoty", vysokej v prípade že:

-mucha je blízko vozidla

-mucha je priamo pred vozidlom

-mucha má vysokú hodnotu fitness.

Muchy neužitočné pre túto špecifickú úlohu boli penalizované znížením fitness, a tým pádom mali vysokú pravdepodobnosť, že budú vyradené mechanizmami algoritmu.

Za také muchy sa považovali:

- muchy ktoré boli viac než dva metre nad povrchom cesty
- muchy ktoré boli nad cestou menej než 10 centimetrov (detekujúcich povrch cesty)
- muchy ktoré sedeli viac než 16 metrov pred vozidlom.

Experimentálna analýza viedla Pauplina a spol. k tomu aby si vybrali nasledujúci jednoduchý vzorec na výpočet výstražnej hodnoty muchy:

$$výstražná\ hodnota(mucha) = F / x^2 * z \quad (6)$$

kde F je hodnota fitness muchy, a x a z sú koordináty muchy v trojrozmernom priestore. Pre $|x| < 0.5$ m považovali $x = 0.5$ m, a pre $z < 1$ m považovali $z = 1$ m. To slúžilo na odfiltrovanie múch, ktoré by mohli mať extrémne vysokú výstražnu hodnotu, vyplývajúcej z príliš nízkej hodnoty koordinátu x alebo z, napriek nízkej hodnote fitness. Prekážky ktoré boli v rozsahu pol metra naľavo, alebo polmetra napravo od stredovej osi vozidla ($|x| < 0.5$ m) považovali za rovnako nebezpečné a následne im prideľovali rovnakú výstražnú hodnotu. Výstražná funkcia bola vymyslená tak aby jej výsledkom boli vysoké hodnoty pre muchy, ktoré mali vhodné všetky tri koeficienty: vysokú F, a nízke x a z. Pre muchy s nízkou hodnotou fitness (čiže pravdepodobne také, ktoré sa nenachádzali na objekte), s veľkou vzdialenosťou od vozidla, alebo také čo boli ďaleko od stredovej osi vozidla, nevykazovali evidenciu bezprostrednej kolízie. Experimenty s x namiesto x^2 neposkytli Pauplinovi a spol. dostatočne uspokojivé výsledky, pretože výstražná funkcia mala tendenciu nadhodnotiť dôležitosť múch nachádzajúcich sa príliš ďaleko od stredovej osi vozidla.

3.2.2 Výsledky

Na vyhodnotenie algoritmu testovaním, použil Pauplin dva stereo páry obrázkov: jeden pár reprezentujúci cestu bez akejkoľvek bezprostrednej prekážky (obrázky 7 a 8), a jeden pár reprezentujúci chodcu prechádzajúceho cez cestu priamo pred vozidlom (obrázky 9 a 10). Obrázok 7 nezobrazuje prípad, v ktorom by bolo potrebné pohotovostne zabrzdiť, pričom obrázok 9 zobrazuje situáciu, ktorá sa viac blíži svojou povahou kolízii. Na získanie

výsledkov boli použité dve CCD kamery a počítač (Pentium 2 GHz). Počet múch mali nastavený na 5000. Na spracovanie jednej generácie potreboval počítač okolo 10 milisekúnd. Update populácie a výpočet výstražných hodnôt bol vykonávaný vo veľkej miere plynule a systém spotreboval 10 až 30 generácií na reakciu v prípade, že sa stalo niečo nové na scéne. Obrázky 7 a 9 ukazujú najlepších 250 múch z výslednej populácie. Muchy sú zakreslené ako čierne krížiky.

Obrázok 7. Scéna bez prekážky (Prevzaté z [2]).

Obrázok 8. Výstražné hodnoty piatich múch (Prevzaté z [2]).

Obrázok 9. Chodec štyri metre pred kamerou (Prevzaté z [2]).

Obrázok 10. Výstražné hodnoty múch (Prevzaté z [2]).

Muchy sa na oboch obrázkoch 7 aj 9 zoskupovali na viditeľných objektoch na scéne (auto, chodec, obrubníky...). Obrázky 8 a 10 sú tie isté pohľady ako obrázky 7 a 9, s tým rozdielom, že sa na nich nachádzajú iba muchy vykreslené na základe veľkosti výstražnej hodnoty. Algoritmus vyhodnotil obrázok 10 vyššou výstražnou hodnotou ako obrázok 8, kde boli tieto hodnoty veľmi blízko k nule.
Všeobecná výstražná hodnota bola definovaná ako priemer výstražných hodnôt jednotlivých múch. V prvom prípade bol priemer 0.09, v druhom bola táto hodnota 0.85. Vysoký rozdiel medzi týmito dvoma hodnotami naznačoval, že môžu byť použité na rozlíšenie medzi týmito dvoma situáciami. Pauplin sa zmienil v [2], že budú potrebné ďalšie experimenty na potvrdenie alebo vylepšenie tohto kritéria.

Informácie boli čerpané z [2].

3.3 Detekovanie mobilných prekážok

Louchet v [3] rozšíril algoritmus na spracovanie postupností párových obrazov braných kamerami v pohybe.

3.3.1 Náhodná metóda

Prístup náhodnej metódy pozostával z udržiavania tej istej populácie vyvýjajúcej sa cez zmeny obrázkov videa. Tým pádom iba obrázky (ciže parametre použité účelovou funkciou) boli postupne modifikované kým populácia múch sa vyvíjala. Keď bol pohyb dostatočne pomalý, použitie posledného kroku urýchľoval konvergenciu významne v porovnaní voči použitiu kompletne novej náhodnej populácie vytvorenej každý obrázok. Táto metóda bola autormi považovaná za úvod ku kolektívnej pamäti priestoru, využívajúc podobnosť medzi za sebou nasledujúcimi scénami. Príliš to nemenilo na jednoduchosti statickej metódy a vyžadovalo si to minimálne zmeny algoritmu. Extra mutácia bola aplikovaná na pozície múch pri každej zmene obrazu. Na vylepšenie detekovania nových objektov objavujúcich sa v zornom poli, zaviedli nový mutačný operátor navyše, imigráciu, ktorá vytvárala nové muchy na náhodných miestach podobne ako procedúra, ktorá inicializovala populáciu: použitá s nízkou pravdepodobnosťou (1-5%) zvýhodňovala objavovanie céleho prehľadávacieho priestoru.

3.3.2 Dopredná metóda

Za účelom spracovávania rýchlejšieho pohybu, Louchet a spol. zaviedli rozšírenie chromozómu, ktorý v tejto verzii obsahoval aj pozíciu muchy aj koordináty rýchlosti v súradnicovom systéme mobilného kamerového systému. Účelovú funkciu a genetické operátory modifkovali tiež vzhľadom na túto zmenu. Na využitie redundancie dát a pravdepodobnú spojitosť rýchlostí vo väčšinne aplikácií, bol najvhodnejší prístup (dopredný prístup) nasledujúci. Genóm každej muchy bola šestica (x, y, z, x', y', z'). Na prvom obraze sekvencie, boli súradnice pozície $x, y, z,$ vyvinuté bežným spôsobom a rýchlosti x',y',z' boli inicializované náhodne. Potom pre nasledujúce obrazy sekvencie, vďaka tomu, že poznali genóm múch (x, y, z, x', y', z') v okamihu t, boli schopní tento genóm extrapolovať v okamihu $t+\Delta t$ ako ($x+x'\Delta t, y+y'\Delta t, z+z'\Delta t, x', y', z'$) a potom vyvíjať komponenty rýchlosti (x', y', z'). Na konci tohto kroku, boli súradnice múch aktualizované na $x+x'\Delta t, y+y'\Delta t, z+z'\Delta t$, takže populácia bola pripravená pre ďalší krok. Táto metóda mohla byť podľa Loucheta považovaná za úvod do rýchlostnej pamäti múch. Nezahŕňala dôležité zmeny v programe, okrem toho, že evolúcia bola aplikovaná skôr na rýchlosti než na pozície.

3.3.3 Spätná metóda

Pre lepšie využitie časovej redundancie zaviedli spätný prístup: vyhodnotiť muchu s genómom, (x, y, z, x', y', z') obsahujúcim pozíciu rýchlosť v okamihu t, vypočítali aj súdržnosť (x, y, z) s momentálnym párom obrázkov aj súdržnosť ($x-x'\Delta t, y-y\Delta t, z-z'\Delta t$) s predchádzajúcim obrázkom. Tento prístup znamenal, že množina párov (stará pozícia, nová pozícia) a tým pádom celá šestica parametrov genómu (x, y, z, x', y', z') je preskúmaná viac systematicky, než s doprednou metódou. Z toho vyplývalo, že algoritmus v tomto prípade museli dôkladne prepracovať.

3.3.4 Zhoda fitness vs. porovnanie úrovne sivej

Statická účelová funkcia zahŕňala iba parametre pozície (x,y,z). Teraz potrebovali definovať dynamickú účelovú funkciu, ktorá berie do úvahy dva po sebe idúce stereo páry obrázkov a 6-členný genóm muchy. Prvá metóda (Zhoda fitness) pozostáva z vypočítania statického fitness múch na starom páre obrázkov, a z vypočítania statického fitness na novej pozícii na novom páre obrázkov, potom dostane mucha vysokú hodnotu fitness len keď sú obe hodnoty vypočítané statickou metódou vysoké. Aby výsledky neboli

ovplyvnené možným rozdielom hodnôt zdieľania, iba pozícia na nových obrázkoch sa používala pri výpočte oboch fitness, a dynamická hodnota fitness bola definovaná ako výsledok novej statickej hodnoty s gausovou funkciou rozdielu medzi starou a novou hodnotou:

$$dynamická\ fitness = fitness_1 exp(-(fitness_1-fitness_2)^2/2s^2) \quad (7)$$

kde parameter *s* umožňoval nastaviť toleranciu rozptylu fitness.

Druhá metóda pozostávala z porovnania úrovne sivej na všetkých štyroch obrázkoch z dvoch po sebe nasledujúcich stereo párov, a pridelila vysokú hodnotu fitness iba muche, ktorej šesť koordinátov na všetkých štyroch obrázkoch poskytuje podobnú úroveň šedej. Louchet vybudoval dynamickú účelovú funkciu ako inverziu sumy rozdielov úrovne sivej na štyroch obrázkoch: starý ľavý vs. nový ľavý, starý pravý vs. nový pravý, starý ľavý vs. starý pravý, nový ľavý vs. nový pravý. Prvé dva rozdiely penalizujovali muchy za nesprávne rýchlosti, ďalšie dva za nesprávne pozície: potrebovali experimentálne zistiť aké koeficienty bolo potrebné zaviesť do vzorca.

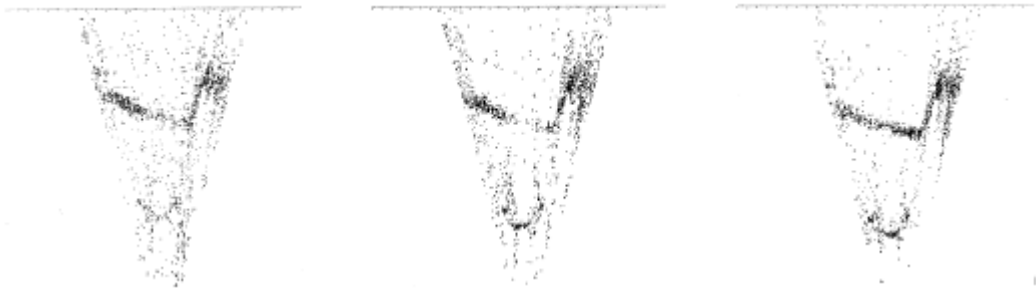
3.3.5 Genetické operátory

Za účelom využitia statickej verzie algoritmu s postupnosťou obrázkov, zaviedli v [3] operátor imigrácie, ktorý pomáha algoritmu brať do úvahy nové objekty, ktoré sa môžu objaviť na scéne a pri spracovávaní statickej scény by nemuseli byť užitočné. V dynamickej verzii, genetické operátory, imigrácia, mutácia a barycentrické kríženie, sú podobné tým ktoré sú v statickej verzii. Imigrácia znova vytvára úplne nových jedincov nachádzajúcich sa v priesečníku zorných polí oboch kamier, s náhodne zvolenými rýchlosťami v danom intervale. Mutácia v tejto verzii ovplyvňuje normu a smer rýchlostného vektora.

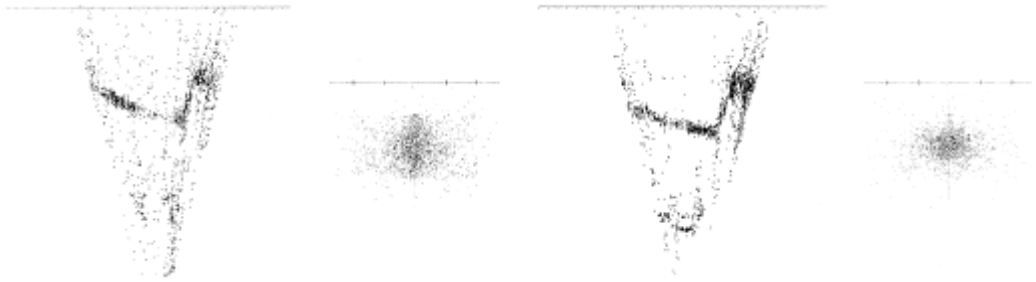
3.3.6 Inicializácia populácie

Algoritmus je v prvom kroku inicializovaný statickou verziou. Počnúc druhým obrazom je potrebné inicializovať aj rýchlosti, čo je ponechané na používateľa. Presná informácia o rýchlosti, umožňuje redukciu rôznorodosti začiatočného rozmiestnenia, zvýhodňuje lokálny prieskum múch a pomáha konvergencii.

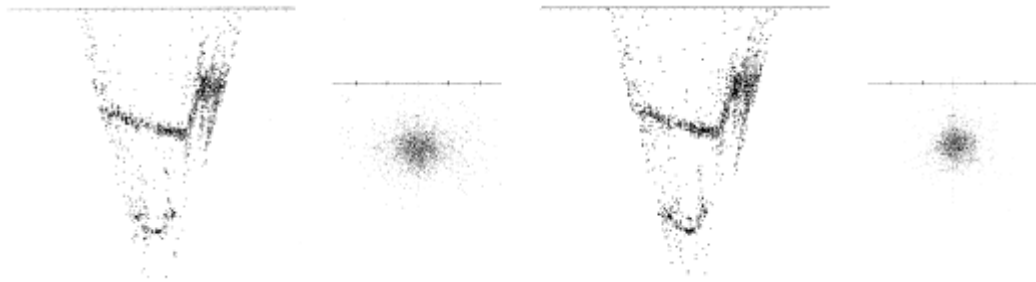
3.3.7 Výsledky



Obrázok 11. Výsledky náhodnej metódy, obrázky 2 4 a 6 zo sekvencie (Prevzaté z [3]).



Obrázok 12. Dopredný prístup, vyhodnocovanie porovnaním fitness, výsledky z obrázkov 2 a 6 zo sekvencie druhý a štvrtý obrázok ukazujú hodnoty rýchlostí (Prevzaté z [3]).



Obrázok 13. Porovnanie úrovne sivej na štyroch obrázkoch: dopredný prístup (vľavo) spätný prístup (vpravo) (Prevzaté z [3]).

Výsledky na obrázkoch 11 – 13 z [3] boli získané z rovnakej sekvencie šiestich stereo párov obrázkov scény v interiéri. Rozsah rýchlosti robota V bol známy, a použitý na inicializáciu z' rýchlosti rovnomerným rozdelením v intervale [-3V/2,3V/2]. Ostatné komponenty x', y' boli inicializované Gausovou distribúciou okolo nuly. Pokiaľ to nie je v popise ináč, obrázky ukazujú výsledky získané zo šiesteho obrázku sekvencie premietnuté na horizontálnu rovinu (*x,z*), s kamerou smerujúcou smerom k vrchu obrázka. Obrázky

rýchlostí (v Obrázkoch 12 a 13) zobrazujú priemet rýchlostných vektorov na horizontálnu plochu (x', z').

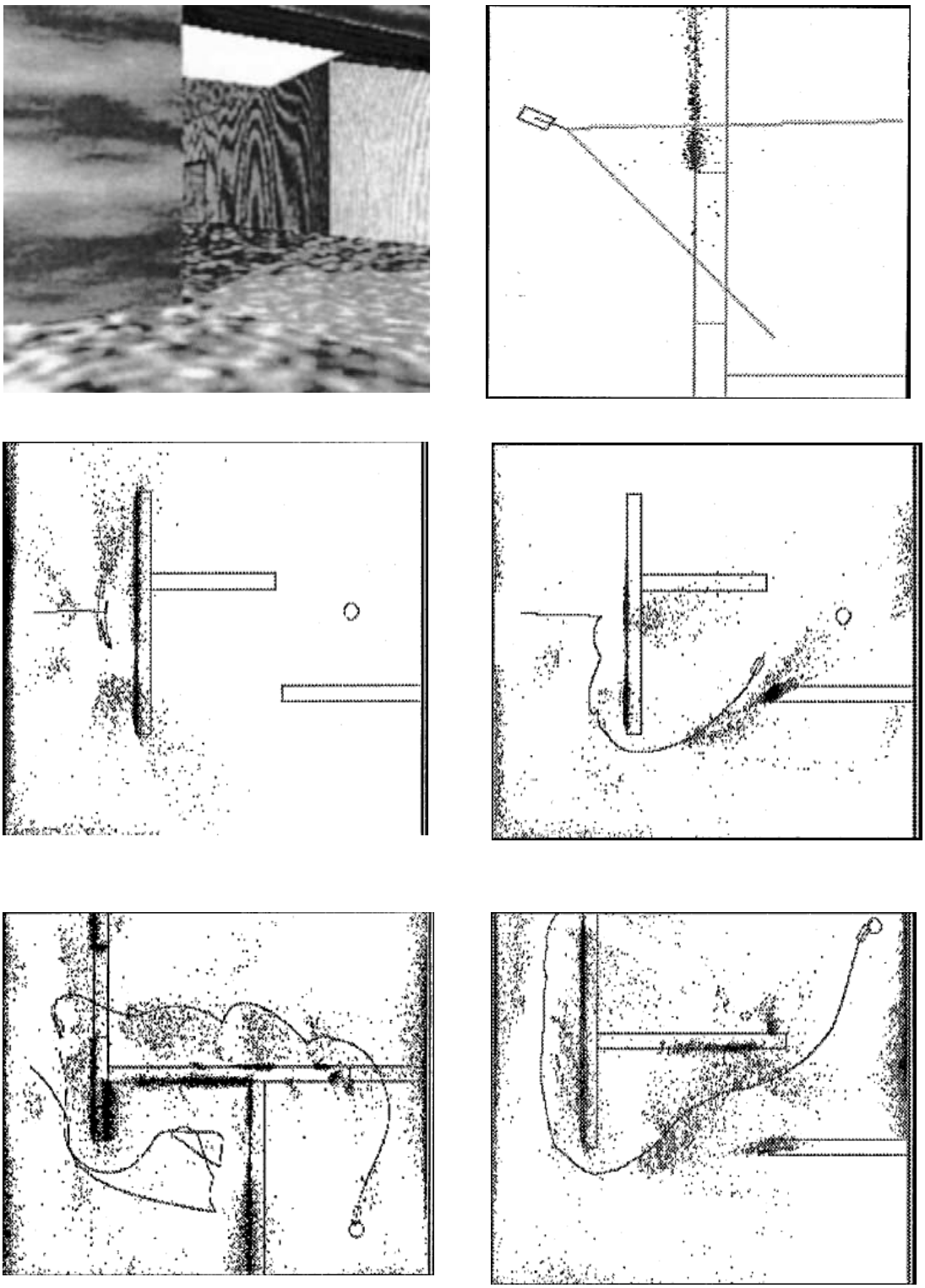
Vzhľadom na pozície múch, boli presvedčivejšie výsledky v prípade, v ktorom je účelová funkcia založená na porovnaní úrovne sivej na štyroch obrázkoch. Spätná metóda poskytovala lepšiu konvergenciu rýchlostí, a tým pádom pravdepodobne lepšie výsledky na dlhých sekvenciách obrázkov, za podmienky že pohyb bol dostatočne plynulý, ale spotrebovala viac výpočtového výkonu.

3.4 Aplikácia fly algorithm na vyhýbanie sa prekážkam v robotike

Spôsob akým je scéna opísaná použitím fly algorithm (v porovnaní s typickejšími algoritmami používanými na spracovanie stereo obrazu v mobilnej robotike) viedol Loucheta k adaptovaniu typickej metódy pre navigáciu robota za účelom využiť výsledky z fly algorithm ako vstupné dáta, a urobili simulátor (Obrázky 14 vľavo hore a vpravo hore). Na udržanie potenciálnej výhody rýchlosti fly algorithm, si zvolili implementovať, rýchlu metódu založenú na vektoroch. Definovali silu zloženú zo sumy príťažlivej sily (cieľ) a odpudivých (muchy)

$$\vec{F} = \vec{F}a + \vec{F}r \quad (8)$$

pôsobiaciu na robota v roli riadenia. Prížažlivá sila $\vec{F}a$ smeruje smerom k cieľu a má konštantnú amplitúdu s výnimkou blízkeho okolia cieľa. Odpudivá sila $\vec{F}r$ je závislá od všetkých múch. Ich vnútorná dvojrozmerná reprezentácia prostredia robota bola inšpirovaná Martinovou a Moravcovou evidenčnou mriežkou pre robota: každé pole obsahuje informáciu obsiahnutu v muchách ktoré sa nachádzajú v ňom a generuje svoj príspevok k odpudivej sile, v proporcii k sume fitness múch. Kompletný simulátor obsahoval: simulátor pohľadu cez stereo kameru (PovRay), fly algorithm, vektorovo založený plánovač popísaný vyššie, a jednoduchý kinetický simulátor robota. Situácie v ktorých sa robot zasekne v potenciálnom lokálnom minime boli riešené použitím dvoch heuristík ktoré vytvorili novú silu vyťahujúcu robota von z potenciálneho minima (náhodná prechádzka, a metódy nasledovania steny)
(Obrázok 14 posledné štyri časti).



Obrázok 14. Pohľad robota v simulátore a výsledky pokusov (Prevzaté z [3]).

3.5 Výsledky výskumu Loucheta a spol.

Evolučné algoritmy sú často považované za pomalé a nevhodné do real-time aplikácií.
Tu sú však výhody evolučných algoritmov a fly algorithm, ktoré Louchet a spol. opísali v závere [3]:
Evolučné sttratégie sú vo všeobecnosti schopné adaptácie, t.j. sú schopné prispôsobiť sa modifikácii fitness počas vykonávania algoritmu. Systém je schopný využiť vstupné dáta v momente keď sa stanú prístupnými, a poskytnúť používateľovi kedykoľvek vhodný aktualizovaný výsledok.

Pri konvenčných algoritmoch pre spracovanie obrazu musí byť celý obrázok načítaný, na to aby algoritmus mohol vypočítať a poskytnúť výsledok. Fly algorithm, ktorý nepotrebuje mať celý obraz načítaný, toleruje aktualizovanie dát počas svojho behu a vždy používa čerstvo aktualizované údaje. Táto vlastnosť v [3] využili použitím CMOS kamerovej technológie, ktorá je schopná asynchrónneho čítania dát.

Výsledky fly algorithm boli aktualizované nepretržite (v miere trvania jednej generácie), čo umožňovalo používateľovi (napr. plánovaču cesty robota) reagovať rýchlejšie na zmeny vonkajšieho prostredia.

Na čas najnáročnejšou procedúrou v evolučných algoritmoch je obyčajne výpočet účelovej funkcie. Parížsky prístup im však umožnil rozptýlenie reprezentácie scény na veľké množstvo extrémne jednoduchých primitív, preto bola účelová funkcia fly algorithm jednoduchá a umožňovala jej rýchly priebeh.

Čo opísali v [3] ako celkom nezvyklé je, že pri spracovaní obrazu je štruktúra programu vo veľkej miere nezávislá od aplikácie, keďže veľká časť znalostí špecifických pre problém je vyjadrená v účelovej funkcii, čo umožňuje jednoduchý prenos do iných aplikácii.

Informácie boli čerpané z [3]

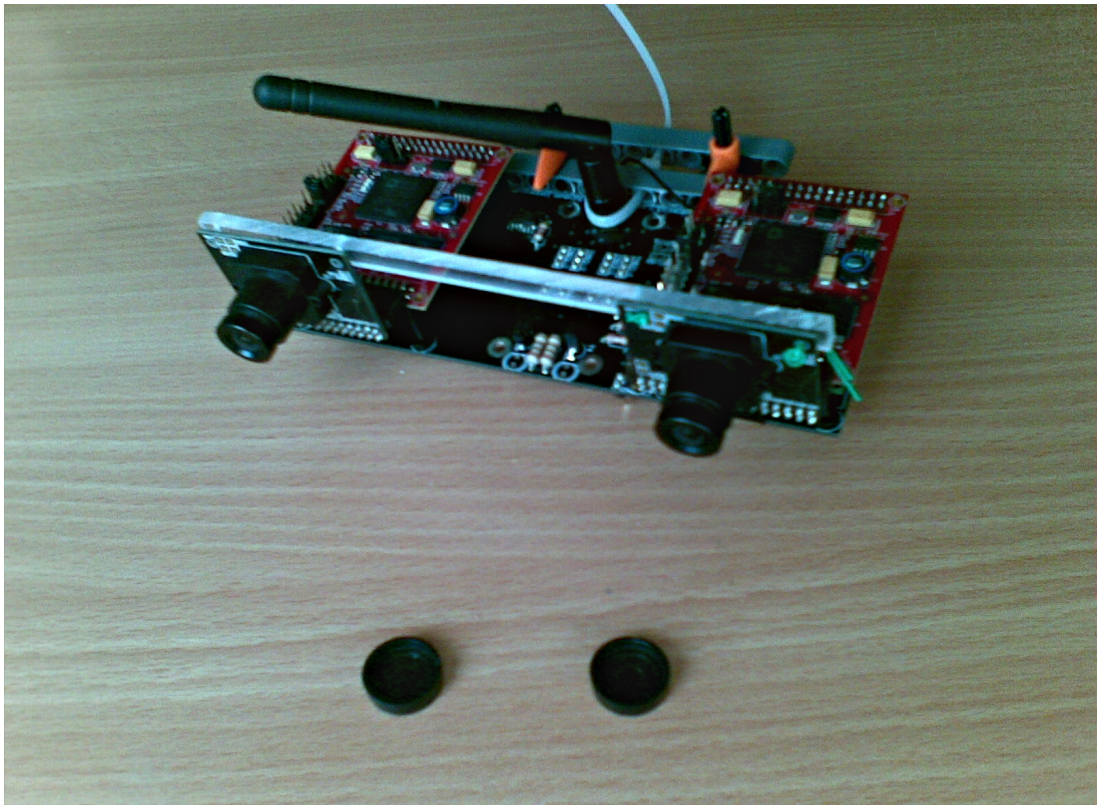
4 OpenCV

OpenCV je opensource knižnica určená primárne k spracovaniu obrazu v reálnom čase. Bola vyvinutá v Inteli a v súčasnosti je podporovaná Willow Garage. Obsahuje cez 500 algoritmov na spracovanie obrazu. Obsahuje moduly pre prácu v C++, Pythone, Androide. V našej práci použijeme jej C++ové moduly (OpenCV verzie 2.2) na rektifikovanie obrazov, konkrétne na vyrovnanie kamerami spôsobeného skreslenia objektívom, zarovnanie podľa y-ovej súradnice a vizualizáciu projektu.

Pre použitie OpenCV s IDE CodeBlocks treba projekt nastaviť nasledovne: v project build options si zvolíme v ľavom menu položku s názvom nášho projektu, ktorá je na vrchu štruktúry menu. Potom si zvolíme záložku linker settings. V okne link libraries pridáme cesty ku všetkým súborom z adresárov bin a lib, ktoré sa nachádzajú v priečinku inštalácie. Potom si zvolíme kartu search directories a v nej si zvolíme kartu compiler kde nalinkujeme cesty k potrebným v hlavnej triede zahrnutým knižniciam.

5 Surveyor stereo vision system (SVS)

Toto je kamerový systém ktorý sme použili pre snímanie scény.



Obrázok 15. Surveyor stereo vision system.

Pre ilustráciu uvádzame kompletný prehľad parametrov systému:

- Two SRV-1 Blackfin camera modules separated by 10.75 cm (4.25"). Each camera module includes:
  - 500MHz Analog Devices Blackfin BF537 Processor (1000 integer MIPS), 32MB SDRAM, 4MB SPI Flash, JTAG, external 32-pin i/o header w/ 2 UARTS, 4 timers (PWM/PPM), SPI, I2C, 16 GPIO
  - Omnivision OV9655 1.3 megapixel sensor with AA format header and interchangeable lens - M12 P0.5 format - 3.6mm f2.0 (90-deg FOV) or optional 2.2mm f2.5 (120-deg FOV)
- Processor-to-processor communications via SPI bus (64MHz)
- Lantronix Matchport WLAN 802.11g radio w/onboard 3dB dipole antenna for Wifi communications
- On-board 3.3V high efficiency switching regulator (Recom R-783.3-1.0) for battery input (4.75 - 18.0 VDC)
- Headers for 8 servos (5V supply provided)
- Dual H-bridge motor driver (Fairchild FAN8200) with 1000mA drive current per motor
- Two switching transistor drivers with 100mA drive current for lights and laser pointers
- Low battery detect circuit

- Headers for 8 servos (5V regulator provided)
- Extended pin headers for full access to **S-32** expansion bus of both processors
- Board dimensions - 60 mm x 150 mm (2.5" x 6.0"), 140g (5 oz)
- 1" x 2" mounting hole pattern for compatibility with SRV-1 robot base (dual and quad motor versions)
- Total power draw - 300mA @ 7.4V (approx 2 watts)
- RoHS compliant

Parametre ktoré sú potrebné pre naprogramovanie algoritmu:

vzdialenosť kamier - 107,5 mm

šírka priemetne - 4,17 mm

výška priemetne - 3,29 mm

ohnisková vzdialenosť - 2,0 mm

Na kamery sa dá pripojiť cez wifi. Jedna je na adrese 169.254.0.10:10001 a druhá je na adrese 169.254.0.10:10002, po zadaní týchto napríklad do prehliadača môžeme v prehliadači pozorovať video z týchto kamier. Cez adresy 169.254.0.10:10001/robot.jpg a 169.254.0.10:10002/robot.jpg sa potom dá zobraziť aktuálna snímka. Z tých adries sa pomocou wgetu dajú tieto snímky stiahnuť, a nechať s nimi pracovať program.

6 Implementácia

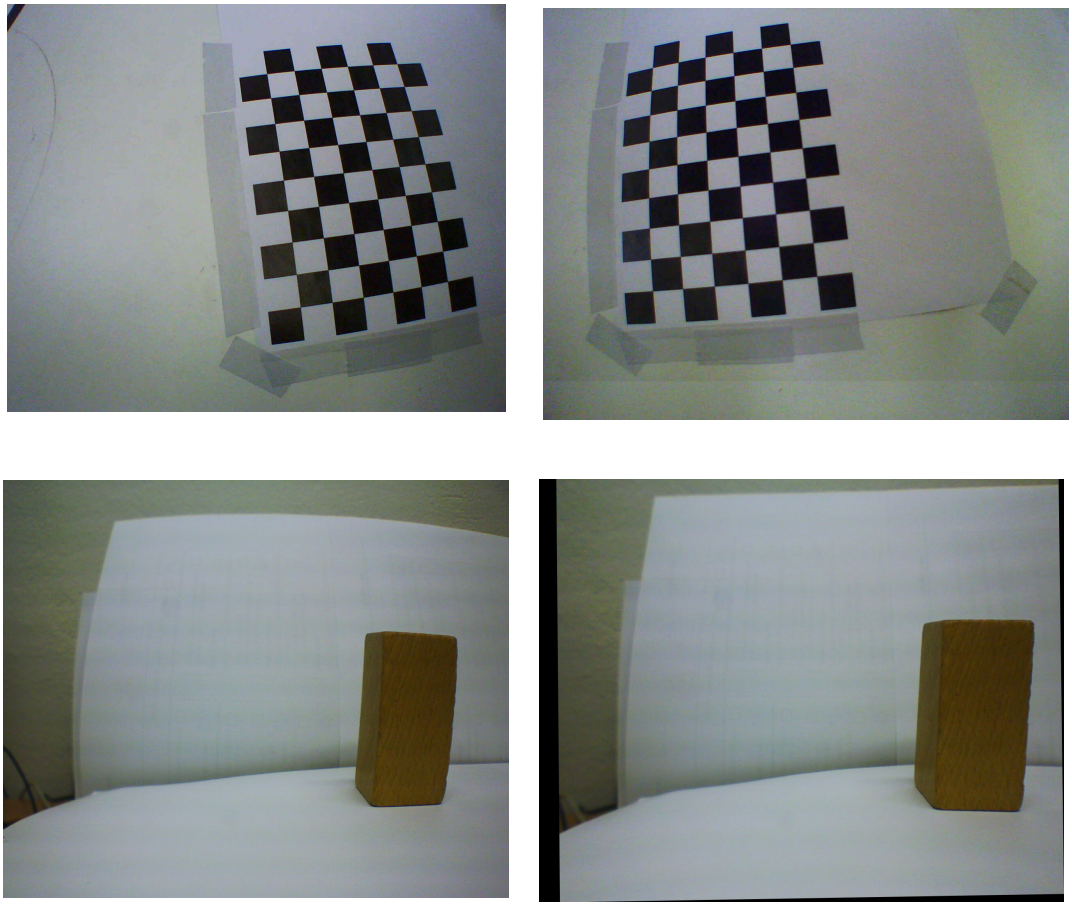
V tejto časti je opísaný program ktorý je naprogramovaný v C++, pomocou knižníc OpenCV.

6.1 Iplimage

Je to dátová štruktúra používaná pri práci s obrazom pomocou OpenCV. V [5] je uvedené, že v jej premennej imageData sú úložené všetky údaje z obrázku v jednorozmernom poli po farebných zložkách v poradí B, G, R. Pre túto prácu bola ešte dôležitá premenná widthStep, v ktorej je uložené koľko takýchto prvkov je v jednom riadku.

6.2 Rektifikácia

Na to aby sa dala účelová funkcia na obrázkoch správne počítať bolo potrebné odstrániť skreslenie spôsobené objektívom a zarovnať obrázky podľa riadka.



Obrázok 16. Kalibračná šachovnica a výsledok rektifikácie.

Na toto bola použitá kalibračná knižnica OpenCV – Calibfilter. Pre konfiguráciu Calibfiltra bolo potrebné získať niekoľko obrázkov s kalibračnou šachovnicou (odfotili sme 12 párov)

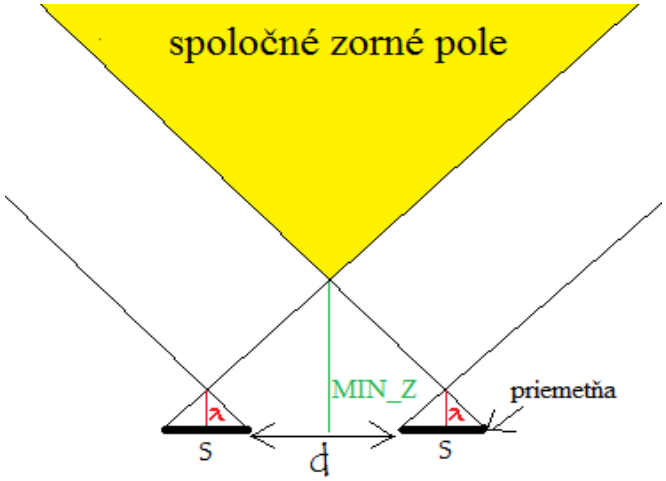
odfotených stereo kamerou. Po nafotení kalibračných obrázkov, bolo potrebné obrázky načítať a nechať na nich zbehnúť vyhľadávanie šachovnice. Calibfilter si rohy týchto šachovníc uložil a pre potrebu konfigurácie Calibfiltra pri ďalšom spustení sme potrebné údaje uložili do súboru pomocou jednej z jeho funkcií. Z tohto súboru si vie Calibfilter načítať všetky potrebné údaje. Na každom obrázku, s ktorým pracuje účelová funkcia najprv prebiehajú procedúry Calibfiltra – undistort a rectify.

6.3 Trieda Fly

Prvé čo sme sa rozhodli naprogramovať bola trieda Fly. K triede sme potrebovali zoznam konštánt, s ktorými sa v triede narába a napríklad v prípade potreby zmeny ohniska alebo iných parametrov kamery stačí prepísať niektorú z nich. Zoznam obsahuje vzdialenosť kamier, ohniskovú vzdialenosť, šírku priemetne, výšku priemetne, šírku priemetne v pixeloch, výšku priemetne v pixeloch, šírku a výšku pixela na priemetni v milimetroch. Na zaručenie toho, že sa mucha nevygeneruje mimo priesečníku zorného poľa dvoch kamier som potreboval minimálnu hodnotu pre súradnicu *z*, vypočítanú z ohniskovej vzdialenosti, vzdialenosti kamier a šírky priemetne nasledujúcim spôsobom:

$$2*(\lambda*(d/2+s))/s \quad (9)$$

λ je ohnisková vzdialenosť, *d* je vzdialenosť kamier a s je šírka priemetne.



Obrázok 17. Nákres kamier.

Okrem toho ohraničenie múch zaručuje konštanta pre maximálne *z*, plus dve funkcie ktoré opíšem nižšie v tejto kapitole. Pri výpočte ohraničenia sa používa ešte ďalších päť konštánt: štyri sú hodnoty zodpovedajúce umiestneniu okrajov priemetní vzhľadom na stred súradnicovej sústavy v danej priemetni, piata je vzdialenosť o ktorú treba posunúť stred súradnicovej sústavy spomedzi kamier tak aby sa nachádzala uprostred priemetne,

všetkých päť sa počíta zo šírky, výšky priemetne a vzdialenosti kamier.

Ďalej sme si k triede definovali dva struct-y: point a pixel, pixel obsahuje dve celé čísla a point dve reálne. Tie používame na zaznamenávanie pozície muchy na ľavom a pravom priemete najprv v milimetroch potom v pixloch.
Samotná trieda muchy obsahuje súradnice v priestore, súradnice na ľavom a pravom priemete, hodnotu fitness a funkcie.

6.3.1 Funkcie triedy Fly

Najprv sme potrebovali zapísať vzorce (3) a (4) na prepočítavanie priemetu. Tie sa nachádzajú vo funkcii suradnicePriemetu ktorá dostane ako vstup súradnice v trojrozmernom priestore a jej výstup sú súradnice na premietni v milimetroch so stredom súradnicovej sústavy v strede priemetne. Následne tieto súradnice sú poslané do ďalšej funkcie suradnicePriemetu, ktorej výstupom sú už potrebné súradnice na priemetoch v pixloch so stredom súradnicovej sústavy v ľavom hornom rohu s y-ovou osou smerom dole. Funkcia vyratajPriemet volaná vždy keď treba vyrátať priemet, zabezpečuje pomocou týchto dvoch funkcií a konštanty POSUN, vyrátanie oboch potrebných priemetov.

Potom sme potrebovali urobiť vytvorenie muchy, pre čo sme najprv potrebovali ešte funkcie na vypočítanie y-ového a x-ového ohraničenia vzhľadom na z-ovú súradnicu: minY, maxY, minXP, maxXL. Na čo používame vzorec, ktorý bol tiež zmieňovaný v [1] a vzniká obrátením rovnice použitej na výpočet súradníc na priemetni:

$$X = x/\lambda * (\lambda - Z) \quad (10)$$

(obdobne pre Y). Keďže Z podľa ktorého sa pýtam funkcie na ohraničenie je vždy vygenerované predtým a za x dosadím súradnicu potrebného okraja priemetne, X nie je problém vypočítať. V konštruktore sa potom pomocou random generátora a týmto funkciám vygenerujú súradnice muchy v rámci potrebných ohraničení.

Pre potreby fly algorithm pribudli v neskoršej fáze vývoja do triedy dve funkcie. Prvá je rozmnoz, ktorá dostane ako vstupný parameter inú muchu, s ktorou sa má vykonať operácia rozmnoženia. Najprv sa náhodne vygeneruje lambda od 0 do 1, ktorá určuje kde presne na spojnici medzi rodičovskými muchami sa má nachádzať potomok. Potom tieto súradnice prebehnú kontrolou, či sú v rámci zorného poľa kamier, pomocou funkcií ktoré sa používajú aj v konštruktore. Súradnice ktoré sú mimo sa nastavia na maximálnu, resp.

minimálnu hodnotu. Potom sa vo funkcii vytvorí nový objekt muchy, ktorý sa vráti ako výstupný parameter. Druhá je mutuj, ktorá pohne náhodne muchou v rozsahu ktorý dostane v parametri. Tiež sa súradnice kontrolojú či sú v zornom poli a tiež sa nastavujú. Po zmene súradnic sa na konci funkcie volá funkcia na vyrátanie nového priemetu.

6.4 Trieda main

Tu je naprogramovaný celý algoritmus. Pri spustení prebehne inicializácia: vynulujú sa potrebné polia, vytvoria sa potrebné okná, načítajú sa kalibračné parametre, preventívne sa zavolá funkcia na odstránenie stiahnutých obrázkov z kamery, pole múch sa naplní prvou generáciou múch na náhodne vygenerovaných miestach, a potom sa pustí hlavný cyklus.

6.4.1 Hlavný cyklus

Dá sa rozdeliť na tieto sekcie: stiahnutie obrázkov z kamier do súboru, načítanie pomocou OpenCV zo súboru, odstránenie súborov obrázkov, rektifikovanie obrázkov, prekonvertovanie dát z obrázkov, množenie, generovanie, mutáciu, účelovú funkciu, vykreslovanie a selekciu.

Sťahovanie obrázkov prebieha pomocou vonkajšej aplikácie wget, ktorá sa pripojí na kamery a uloží stiahnuté obrázky na disk. Po načítaní sa hneď mažú systémovým volaním remove, aby sa dali stiahnuť ďalšie obrázky.

Ak je miesto v poli pre nové muchy, tak prebehne podľa počtu miest generovanie múch množením. Šanca s akou prebehne množenie(kríženie) sa dá nastaviť v programe určitou konštantou. Pri vykonávaní množenia, sa najprv náhodne vygenerujú indexy prvej a druhej muchy v poli a pre tie muchy ktorým zodpovedajú indexy sa zavolá generovanie nového jedinca.

Predpokladá sa, že po prebehnutí množenia je ešte voľné miesto v poli. Ak je tento predpoklad splnený, prebehne generovanie, čo je vlastne vyššie uvedená imigrácia. Generovanie vytvorí taký počet múch na náhodných miestach, aby sa zaplnilo zvyšné voľné miesto v poli. Po prebehnutí generovania je teda pole vždy naplnené počtom múch určených vo flyNum. Pre každú v tejto generácii vytvorenú muchu prebehne ešte mutácia, ktorá posunie muchu náhodne v danom rozsahu (jedná sa o pár milimetrov). Tiež sa

vykonáva pre náhodne vybrané muchy, šanca na mutáciu sa dá nastaviť v ďalšej konštante. Po mutácii nasleduje účelová funkcia, ktorá bude opísaná nižšie v tejto kapitole. Po nej nasleduje vykresľovanie, ktoré prebieha v dvoch oknách. V jednom sa vykresľujú obrázky z kamier a na nich sa vykresľujú pozície múch v priemete, pričom tie ktoré prežijú selekciu sa vykresľujú odlišným spôsobom. V druhom okne sa vykresľujú pozície múch z pohľadu zhora.

V selekcii sa pole múch zoradí podľa ich hodnoty fitness, následne sa vyberie istá časť najvhodnejších múch. Počet vybraných múch sa dá tiež určiť v konštante na začiatku programu. Pri tomto výbere sa kontroluje, či na tom istom mieste nesedí už iná mucha, ak sedí tak práve kontrolovaná selekciou neprejde.

6.4.2 Účelová funkcia

V cykle pridelí hodnoty fitness všetkým muchám v poli. Pre každú si zistí súradnice ľavého a pravého priemetu. Keďže na okrajoch obrázkov vznikajú pri odstraňovaní skreslenia objektívom čierne jednofarebné oblasti, na ktorých by sa mohli muchy chytiť tak sa najprv skontroluje či niektorá súradnica priemetu nie je príliš blízko k niektorému z okrajov. Ak je pri niektorom okraji bližšie ako 50 pixelov tak sa muche automaticky priradí hodnota fitness 0 a cyklus už s touto muchou nič nerobí.

Ak sa mucha nachádza v rámci týchto hraníc tak sa prejde do ďalších dvoch cyklov, ktoré prebehnú deviatimi okolitými pixlami. Keďže sa farebné zložky nachádzajú všetky za sebou v jednorozmernom poli, tak sa index príslušných zložiek počíta vzorcom:

$$(widthStep)*(y+i)+3*(x+j) \quad (11)$$

x a y sú súradnice muchy na priemete, i a j sú iterátory cyklu a widthStep je počet prvkov v jednom riadku, ktorý sa dá získať z premennej v štruktúre Iplimage, v ktorej je obrázok uložený.

Pri prechode týmito pixlami sa zbiera do akumulátorov farebný rozdiel medzi pixlom na ľavom a na pravom obrázku pre každú farebnú zložku, a hodnota x-ovej a y-ovej časti Sobelovho gradientu vzhľadom na prostredný pixel. Po týchto cykloch sa farebné rozdiely pre jednotlivé zložky sčítajú do jednej premennej, takisto sa zložky Sobelovho gradientu sčítajú pre ľavý a pravý obrázok. Potom sa podľa vzorca (1) priradí hodnota fitness do

príslušnej premennej muchy. Ak je hodnota farebného rozdielu nula tak sa počíta tak isto ako keby bola rovná jednej čo zamedzuje deleniu nulou.

6.5 Verzia s výstrahou

Neskôr sme doplnili druhú verziu, ktorej vykonávanie sa dá zapnúť jednou premennou typu boolean v kóde. Bola pridaná, pretože sme si chceli vyskúšať na načítávaní obrázkov s dlhším časovým intervalom vyvodzovanie výstražnej hodnoty – čo by napríklad mohlo robotovi na ktorom by sa nachádzal tento systém poslať signál na zastavenie v prípade, že sa pred ním nachádza prekážka.

Pre tento účel boli pridané:

Premenná warning v triede fly na zapamätanie výstražnej hodnoty muchy.

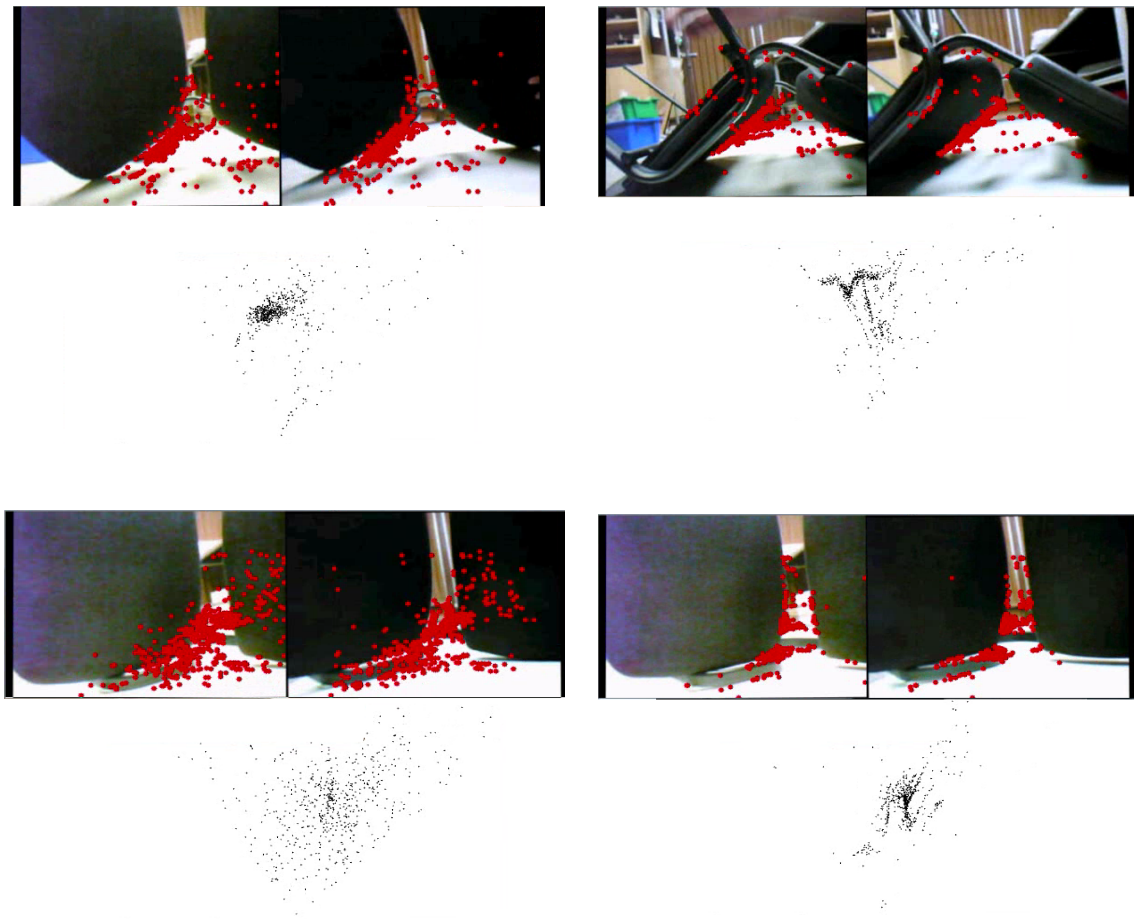
Možnosť inicializovania populácie pre každý obrázok odznova.

Cyklus, ktorý vykonáva evolučný algoritmus dovtedy, kým sa nedosiahne hodnota, ktorá znamená výstrahu, alebo dovtedy kým prešiel nejaký určený počet generácií.

Počítanie výstražnej hodnoty pre muchu. Najprv sme skúsili vzorec (6). Celková výstražná hodnota sa potom dá odvodiť napríklad súčtom hodnôt zo všetkých múch.

Skúšali sme pôvodnú variantu vzorca, aj rôzne iné varianty napríklad namiexto x^2 a z použiť x a z^2. Avšak nestihli sme už počas tejto práce pomocou pokusných objektov na reálnom pozadí nášho interiéru v laboratóriu nájsť hraničnú hodnotu, ktorá by stopercentne zaručila rozlíšenie situácie, v ktorej je objekt položený tesne pred kamerami, od situácie keď je objekt položený ďalej od robota.

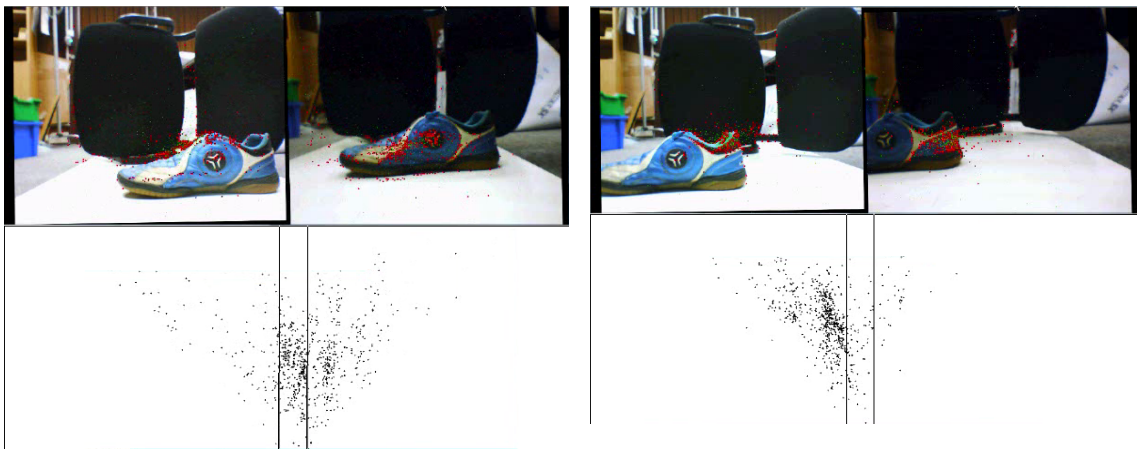
7 Experimenty



Obrázok 18. Snímky z pokusov.

Obrázok 18 demonštruje testovanie fly algorithm v spoločnom laboratóriu robotiky FEI STU a FMFI UK. Červené body vo vrchnej časti obrázkov predstavujú priemety najvhodnejších múch v obraze jednej a druhej kamery, ktoré sú vybrané selekciou. V spodnej časti obrázkov predstavujú biele body tie isté muchy pri pohľade na scénu zhora (s pozíciou kamier na spodku obrázku nasmerovaných hore). Pri pomalom pohybe muchy pomerne dobre sledovali hrany stoličky (obrázok 26), po otočení (obrázok 27) sa prispôsobili novej scéne za jednu generáciu. Pri náhlej zmene scény (rýchle otočenie z obrázk. 27 a 28) sa muchy na chvíľu rozpŕchnu s väčším rozptýlením po scéne aby našli nové miesta, v ktorých dosiahnu vysokú hodnotu fitness. Po troch generáciách (menej než jedna sekunda, obrázok 29) už z veľkej časti relatívne nemenne sedia na najvýhodnejších miestach – hranách stoličky.

Obrázok 19 predstavuje test verzie s výstrahou . V tejto verzii sa menili obrázky vo väčších časových intervaloch a na každom bolo pustených viacero generácií.



Obrázok 19. Testovanie verzie s výstrahou.

Pozitívne výsledky týchto tesov sa prejavili tým, že aj v tejto verzii sa muchy zhromažďovali s najväčšou hustotou tam kde sme očakávali: keď sa pokusný objekt nachádzal viac vpravo muchy sa zhromažďovali viac vpravo a keď sme ho posunuli doľava tak aj muchy sa zhromaždili na ňom viac vľavo. Počas testov sme už neboli schopní nájsť správnu hodnotu vhodnú na rozlíšenie situácie, v ktorej sa pokusný objekt nachádza tesne pred kamerou, od tej v ktorej sa nachádza ďalej od nej, pretože rozpätie celkových výstražných hodnôt sa v týchto situáciach prekrýva.

Počas testovania algoritmu v našich podmienkach sa vyskytol ďalší problém: veľa hrán v pozadí (najmä horizontálne), alebo dva podobné objekty, ktoré sú ďalej než maximálna hranica pokiaľ sa generujú muchy pútajú istý počet vybraných múch. A to takým spôsobm, že na takéto objekty si vie sadnúť mucha tak, že si na jednom obrázku sadne na horizontálnu hranu viac vľavo (príp. na ľavú bedňu) a na druhom obrázku na tú istú horizontálnu hranu sadne viac vpravo (príp. na pravú bedňu). V trojrozmernom priestore sa takáto mucha nachádza pred hranou (bedňami).

Tento problém však neprekážal tomu aby väčšina múch sadala na objekty.

8 Záver

Bakalárska práca sa venovala predchádzajúcemu výskumu ohľadom fly algorithm a vlastnej implementácii a testovaniu v spoločnom laboratóriu robotiky FEI STU a FMFI UK.

Cieľ bakalárskej práce sme splnili. Spracovali sme poskytnuté zdroje a algoritmus sme na základe dostupných informácií úspešne implementovali a s dostupným hardvérom otestovali v reálnej situácii, ktorá bola ekvivalentná inštalácii systému na mobilnom robotovi, systém je preto na tento účel pripravený. Z vizualizácie, ktorú sme mali v programe, sme počas našich pokusov zaznamenali, že muchy nasledujú objekt po scéne, a nájdu ho do niekoľkých generácií (výsledky sú zozbierané v kapitole experimenty).

Naskytli sa problémy týkajúce sa vlastností nášho interiéru, napríklad horizontálnych hrán na ktoré si vedela mucha sadnúť tak, že v priestore v skutočnosti sedela pred hranou (bližšie popísané v kapitole experimenty). Tieto problémy však nemali závažný vplyv na samotné nachádzanie objektov pomocou algoritmu.

Skúšali sme dve verzie algoritmu. V prvej verzii sa na začiatku programu inicializovala populácia a potom sa postupne vyvíjala na základe meniacich sa obrázkov. Táto verzia splnila naše očakávania. V druhej verzii (verzia s výstrahou) sa použila aktualizácia obrázkov s väčším časovým intervalom a na každom obrázku sa generovalo viacero generácií, pre každý sa inicializovala odznova. Táto verzia bola tiež schopná lokalizovať objekty na scéne, rovnako ako prvá verzia. Ďalší plán s druhou verziou zahŕňal počítanie výstražnej hodnoty pre daný vstup. Dokončenie tohto plánu už ostáva pre ďalší výskum, pretože výstražné hodnoty pre situáciu, v ktorej sa nachádzal objekt blízko kamier a situáciu, v ktorej bol pred kamerou voľný priestor, sa počas nášho výskumu výstražné hodnoty prekrývali.

9 Použitá literatúra

[1] Dupaľ, D. 2009. bakalárska práca - Návrh kamerového cestného radaru
Bratislava: Fakulta matematiky, fyziky a informatiky Univerzity Komenského v Bratislave

[2] Pauplin O. et al. 2004 Applying evolutionary optimisation to robot obstacle avoidance. International Symposium on Computational Intelligent and Industrial Applications 2004, Haikou, China.

[3] Louchet J. et al. 2002 Dynamic flies: a new pattern recognition tool applied to stereo sequence processing. Pattern Recognition Letters, Vol. 2 No. 23 pp. 335-345 (2002)

[4] Domovská stránka Surveyor Stereo Vision System 1.6.2011
Dostupné na internete: <http://www.surveyor.com/stereo/stereo_info.html>.

[5] Dokumentácia OpenCV 1.6.2011
Dostupné na internete: <http://opencv.jp/opencv-2.2_org/cpp/>.

10 Príloha

K práci je priložené CD, ktoré obsahuje zdrojové kódy.