Technická univerzita v Košiciach
Fakulta elektrotechniky a informatiky
Katedra kybernetiky a umelej inteligencie

Imitácia pohybu objektu v zornom poli robota Nao s použitím TLD

(Zadanie predmetu Biokybernetika a evolučná robotika)

2. ročník Ing. štúdia
Umelá inteligencia
Zimný semester 2012/2013

Jaroslav Vraštiak
Michal Puheim

Obsah

1 Text zadania

Cieľom zadania je imitácia pohybu objektu v zornom poli robota Nao s použitím TLD. Body zadania:

1) Modifikujte implementáciu TLD pre účely zadania (C alebo Matlab).
2) Vytvorte skript pre nastavenie štartovacej polohy.
3) Naskenujte a vzorkujte trajektóriu pohybu, ktorý ukáže užívateľ.
4) Nastavte začiatok vždy do štartovacej polohy.
5) Vytvorte chromozóm ako reťazec 10 hodnôt súradníc vzoriek pohybu. Použite vzorkovanie po 1 sekunde.
6) Určite vhodnosť ako podobnosť kriviek.
7) Vytvorte prevenciu proti pohybu objektu mimo zorného poľa.

2 Inštalácia a použitie NAOqi SDK

Pri programovaní pre robota Nao sme sa rozhodli použiť jazyk C++. Pre umožnenie kompilácie a spustenia napísaného kódu je potrebné nainštalovať vývojárske nástroje (SDK). V nasledujúcich častiach uvedieme, ktoré operačné systémy sú podporované, následne si popíšeme závislosti, ktoré sú potrebné pre inštaláciu SDK a nakoniec uvedieme aj vlastný postup inštalácie. Pre bližšie informácie viď [1].

2.1 Podporované operačné systémy

Medzi podporované operačné systémy patria:

- **Linux** – verzia Ubuntu 10.04 LTS (Lucid) až 11.04 (Natty),
  - podporuje kompiláciu kódu bežiaceho na PC aj na robotovi.
- **Windows** – verzia Windows XP SP3 a Windows 7,
  - podporuje kompiláciu kódu bežiaceho len na PC.
- **Mac** – verzia Mac OS X 10.6.4 Snow Leopard a Mac OS X 10.7.1 Lion,
  - podporuje kompiláciu kódu bežiaceho len na PC.

Pozn. 1: Kroskompilátor generujúci kód bežiaci na robotovi je dostupný iba pre Linux.

Pozn. 2: V prostredí OS Windows je potrebné kompilovať v 32-bitovom režime.

2.2 Vyžadované závislosti

Pre kompiláciu a vývoj programov na danom operačnom systéme je potrebný kompilátor a vhodné vývojové prostredie (IDE). Pre podporované operačné systémy sú to:

- pre *Linux*:
  - kompilátor – **GCC** verzie 4 alebo vyššej,
  - IDE – aktuálna verzia **QtCreator**,
- pre *Windows*:
  - kompilátor a IDE - **Visual Studio 2008** alebo **Visual Studio 2010**,
- pre *Mac*:
  - kompilátor – **XCode** (verzia závisí od verzie operačného systému),
  - IDE – aktuálna verzia **QtCreator**.

C++ SDK pre robota Nao zároveň umožňuje medzi-platformový systém kompilácie (tzv. „cross-platform build system“), čo znamená, že rovnaký kód je možné skompilovať na ľubovoľnom z podporovaných operačných systémov. Pre naše účely sme si vybrali práve tento prístup, pretože umožňuje najširšie možnosti nasadenia a použitia vytvoreného kódu.

V tomto prípade je okrem inštalácie samotného SDK potrebné nainštalovať aj **qiBuild**, čo je nástroj určený na generovanie medzi-platformových projektov. qiBuild je nadstavbou medzi-platformového kompilačného manažéra CMake. CMake generuje súbory riadiace kompiláciu („GNU makefiles“ alebo „Visual Studio solutions“) pre ľubovoľný operačný systém a teda umožňuje jednoduchú kompiláciu projektov súčasne pre Windows, Mac, Linux aj OpenNAO.[1]

Pre funkčnosť nástroja qiBuild je potrebné nainštalovať tieto závislosti:

- **CMake** (verzia 2.8.3 alebo vyššia):
  - *Linux*:
    - použite balíček poskytovaný pre vašu distribúciu,
  - *Windows* a *Mac*:
    - použite inštalátor z oficiálnej webovej stránky: http://www.cmake.org/cmake/resources/software.html,
    - povoľte inštalátoru pridanie cesty k inštalačnému adresáru nástroja CMake do systémovej premennej PATH,
- **Python** 2.7:
  - pozn. pre *Windows*:
    - Pre použitie skriptov napísaných v jazyku Python je potrebné do systémovej premennej PATH pridať cesty: „*C:\Python27*“ a „*C:\Python27\Scripts*“.

2.3 Postup inštalácie SDK

Postup inštalácie C++ SDK je nasledovný:

1) Inštalácia vyžadovaných závislostí:
   - Viď. časť 2.2.
2) Stiahnutie C++ SDK:
   - Alternatívy archívu:
     1) *naoqi-sdk-1.14.x-[OS].tar.gz*
     2) *naoqi-sdk-1.14.x-win32-[VisualStudioVersion].zip*
   - Sťahovať je možné zo stránky http://users.aldebaran-robotics.com/ pod záložkou *software / download*.
   - Pozn. 1: Pre sťahovanie je potrebné sa prihlásiť. Pre prihlasovacie meno a heslo sa obráťte na členov AI-CIT.
   - Pozn. 2: Je dôležité stiahnuť verziu určenú pre váš operačný systém!
3) Rozbalenie a inštalácia archívu na počítači:
   - Rozbaľte archív do voľného priečinka na počítači.
   - Pre účely tohto návodu predpokladajme, že je to */path/to/naoqi-sdk*.
4) Overenie závislostí pre *qiBuild*:
   - Uistite sa, že máte nainštalovaný *Python* a *CMake*. (Viď. časť 2.2.)
5) Stiahnutie a rozbalenie nástroja *qiBuild*:
   - Stiahnite archív *qibuild-x.zip*.
   - Sťahovať je možné z rovnakého umiestnenia ako v bode 2.
   - Ďalšou možnosťou je získať zdrojové kódy pomocou služby *GitHub*: https://github.com/aldebaran/qibuild.
   - Rozbaľte archív do voľného priečinka v počítači (napr. do */path/to/qibuild*).

6) Inštalácia nástroja *qiBuild*:

   - Pre *Linux*, *Mac*:
     1) Spustite: `./install-qibuild.sh`
     2) Uistite sa, že cesta „*~/.local/bin*“ je v systémovej premennej PATH.
   - Pre *Windows*:
     1) Spustite: `install-qibuild.bat`
     2) Uistite sa, že cesta „*C:\Python27\Scripts*“ je v systémovej premennej PATH.

7) Konfigurácia nástroja *qiBuild*:

   - V termináli spustite príkaz: `qibuild config -wizard`
   - Konfigurátor vás vyzve na zadanie:
     1) cesty k nástroju CMake (v prípade, že nebola automaticky nájdená),
     2) CMake generátora:
        - pre *Linux* a *Mac* zvoľte „*Unix Makefiles*“,
        - pre *Windows* zvoľte „*Visual Studio 9 2008*“ alebo „*Visual Studio 10*“ (nie 64-bit),
     3) vývojového prostredia (IDE):
        - pre *Linux* a *Mac* zvoľte „*QtCreator*“,
        - pre *Windows* zvoľte „*Visual Studio*“.
   - Pozn. 1: Tento konfigurátor je možné spustiť kedykoľvek v budúcnosti odznova.
   - Pozn. 2: Konfiguračný súbor bude uložený v „*~/.config/qi/qibuild.xml*“. Bude využitý všetkými projektami, ktoré v budúcnosti pomocou nástroja qiBuild vytvoríte.

8) Inštalácia plnej verzie OpenCV:

   - Obsahom SDK sú aj knižnice OpenCV, ktoré sú pre väčšinu OS dodávané v plnej verzii.
   - Avšak v prípade použitia *OS Ubuntu* je dodávaná knižnica OpenCV bez podpory GTK (vykresľovania grafiky). V tomto prípade je potrebné nahradiť túto orezanú verziu jej plnohodnotnou oficiálnou verziou. Pre informácie, ako túto náhradu vykonať viď. [2].

2.4 Kompilácia projektu pomocou nástroja qiBuild

Pre kompletný návod, ako vytvárať a kompilovať projekty pomocou nástroja qiBuild, viď. [3]. V rámci tejto časti si uvedieme skrátený návod[1], ako skompilovať zdrojové kódy, ktoré sú súčasťou tohto zadania:

1) Vytvorenie pracovného priečinku:

   - Vytvorte nový priečinok, do ktorého budete ukladať projekty.
   - Otvorte tento priečinok v termináli[2] a zadajte príkaz: `qibuild init`

1 Návod bol otestovaný v prostredí *OS Windows*, ale mal by fungovať rovnako bez ohľadu na použitý operačný systém (*Windows*/*Linux*/*Mac*).

2 V prostredí *OS Windows* je možné spustiť terminál priamo z otvoreného priečinka podržaním klávesy shift a pravým kliknutím vo vnútri priečinka. Z otvoreného kontextového menu je následne možné vybrať možnosť „*Otvoriť príkazový riadok v tomto umiestnení*“ (resp. „*Open command window here*“).

2) Vytvorenie „sady pracovných nástrojov“ („toolchain“):
   - Príkaz pre vytvorenie novej sady nástrojov má tvar:
     `> qitoolchain create <name> <path/to/naoqi-sdk/toolchain.xml> --default`
     pričom jeho parametre sú:
     - `<name>` – ľubovoľný názov sady nástrojov (napr. „*mytoolchain*“).
     - `<path/to/sdk/toolchain.xml>` – cesta k súboru „*toolchain.xml*“ v inštalačnom adresári SDK.
   - Vytvoriť sadu nástrojov „mytoolchain“ je teda možné napr. príkazom:
     `> qitoolchain create mytoolchain 'C:\NaoqiSDK\toolchain.xml' --default`
3) V do pracovného priečinka rozbaľte archív obsahujúci zdrojové kódy zadania.
4) Otvorte koreňový priečinok projektu „*naoimitation*“.
5) Pre kompiláciu projektu otvorte terminál v koreňovom priečinku projektu a zadajte príkazy:
   ```
   > qibuild configure -c mytoolchain
   > qibuild make -c mytoolchain
   ```
6) V prípade úspešnej kompilácie budú spustiteľné súbory uložené v priečinku „*naoimitation\build-mytoolchain\sdk\bin*“.

3 Návrh a implementácia riešenia

Program je postavený ako konzolová aplikácia a je napísaný v jazyku C++. Na vykreslenie obrazových okien a spracovanie obrazu používa knižnicu OpenCV. Ďalej sú použité knižnice umožňujúce prístup k NAOqi[4] (ovládanie robota Nao a získanie senzorických informácii) a knižnica „libopentld“ obsahujúca C++ implementáciu algoritmu TLD[5] (Tracking Learning Detection) umožňujúca sledovanie objektu na obraze. Bližšie informácie:

- TLD – viď [5], [6], [7], [8].
- OpenCV – viď [9].
- C++ Nao SDK – viď [10].

3.1 Cieľ programu

Cieľom programu je naučiť robota Nao napodobniť trajektóriu pohybu objektu, ktorá mu bola dopredu ukázaná. Užívateľ určí referenčnú trajektóriu ako vektor bodov (x, y) na obraze z kamery. Robot, ktorý v ruke drží sledovaný objekt, sa následne ovládaním dvoch servomotorov na pleci („*Shoulder Roll*“ a „*Shoulder Pitch*“) pokúsi náhodnými pohybmi túto trajektóriu napodobniť („nakresliť“). Na túto sériu náhodne vygenerovaných pohybov je následne aplikovaný evolučný algoritmus s cieľom vygenerovať pohyb, ktorý čo najpresnejšie odpovedá referenčnej trajektórii.

Pri tomto procese hraje dôležitú úlohu vzťah medzi súradnicami (x, y) objektu na obraze a polohou servomotorov (*pitch*, *roll*). Platí, že pre každú jednu dvojicu súradníc (x, y) existuje odpovedajúca dvojica polôh servomotorov (*pitch*, *roll*) a naopak. Zároveň je zrejmé, že je veľmi jednoduché nastaviť ruku robota do danej polohy (*pitch*,*roll*), napr. príkazmi:

```
motionproxy.setAngles("RShoulderRoll", new_roll_value, speed);
motionproxy.setAngles("RShoulderPitch", new_pitch_value, speed);
```

Pre danú polohu (*pitch*, *roll*) je následne možné pomocou sledovania objektu použitím TLD určiť aj odpovedajúce súradnice (x, y) na obraze. Teda vo všeobecnosti nie je problém vykonať transformáciu:

$$T: (pitch, roll) \rightarrow (x, y).$$

Avšak v prípade, že máme danú referenčnú trajektóriu ako vektor súradníc (x, y), pre získanie vektora polôh servomotorov (pitch, roll) potrebujeme vykonať opačnú transformáciu:

$$T': (x, y) \rightarrow (pitch, roll).$$

To však bez adekvátneho matematického modelu alebo spätnoväzobného riadenia nedokážeme. Preto hodnoty (*pitch*, *roll*) nastavujeme náhodne (resp. evolúciou) a následne zisťujeme odchýlku od žiadanej hodnoty (x, y). Táto odchýlka je našou kriteriálnou funkciou v procese evolúcie.

3.2 Popis štruktúry programu

Program pozostáva so série rôznych menu, ktoré pomocou ktorých je možné postupne inicializovať všetky potrebné komponenty:

1. Menu – Inicializácia TLD:
   1 – inicializácia TLD pomocou určenia nového objektu a prechod na 2. Menu,
   2 – inicializácia TLD pomocou uloženého modelu objektu a prechod na 2. Menu,
   3 – ukončenie programu.

2. Menu – Učenie TLD a uloženie modelu:
   1 – učenie modelu objektu použitého v TLD,
   2 – uloženie modelu objektu do súboru,
   3 – prechod na 3. Menu,
   4 – ukončenie programu.
3. Menu – Určenie referenčnej trajektórie pohybu:
   1 – určenie novej trajektórie pohybu, možnosť jej uloženia a prechod na 4. Menu,
   2 – načítanie trajektórie zo súboru a prechod na 4. Menu,
   3 – ukončenie programu.
4. Menu – Evolúcia pohybu:
   1 – evolúcia novej populácie,
   2 – pokračovanie v evolúcii predchádzajúcej populácie,
   3 – zobrazenie najlepšieho riešenia,
   4 – zobrazenie najlepších riešení zo všetkých generácií,
   5 – ukončenie programu.

3.3 Získanie obrazu z robota Nao

Pri získavaní obrazu z robota Nao sme spočiatku použili postup spomínaný v [11]. Toto riešenie však z nejakého dôvodu nefungovalo v prípade, že sme takto získaný obraz chceli použiť pri detekcii objektu pomocou TLD. Obraz sa nám napokon podarilo skonvertovať a úspešne použiť pomocou nasledovného postupu:[3]

1) Vytvorenie video proxy na pripojenie k robotovi:
```
AL::ALVideoDeviceProxy *proxy = new AL::ALVideoDeviceProxy(robotIp, 9559);
```
2) Pokus o odhlásenie prípadného neodhláseného klienta z video proxy:
```
try {proxy->unsubscribe(clientName);} catch(...) {;}
```
3) Prihlásenie klienta k video proxy (rozlíšenie – 320*240, paleta – BGR):
```
std::string proxyClient = proxy->subscribe(clientName, AL::kQVGA, AL::kBGRColorSpace, 30);
```
4) Získanie obrazových dát z kamery:
```
AL::ALValue img = proxy->getImageRemote(proxyClient);
```
5) Vytvorenie hlavičky pre obrázok vo formáte `Iplimage`, do ktorého budeme zapisovať dáta:
```
IplImage* imgHeader = cvCreateImageHeader(cvSize(320, 240), 8, 3);
```
6) Priradenie dát k obrázku:
```
cvSetData(imgHeader,(char*)img[6].GetBinary(),imgHeader->width * imgHeader->nChannels);
```
7) Uvoľnenie obrazového zásobníka na robotovi:
```
proxy->releaseImage(proxyClient);
```
8) Konverzia obrázku na formát `cv::Mat`:
```
cv::Mat frame = cv::Mat(imgHeader).clone();
```
9) Uvoľnenie hlavičky obrázku vo formáte `Iplimage`:
```
cvReleaseImageHeader(&imgHeader);
```
10) Odhlásenie klienta z video proxy:
```
proxy->unsubscribe(clientName);
```
11) Odalokovanie pamäte video proxy:
```
delete proxy;
```

V premennej `frame` teraz máme uložený obrázok z robota kompatibilný s knižnicou TLD.

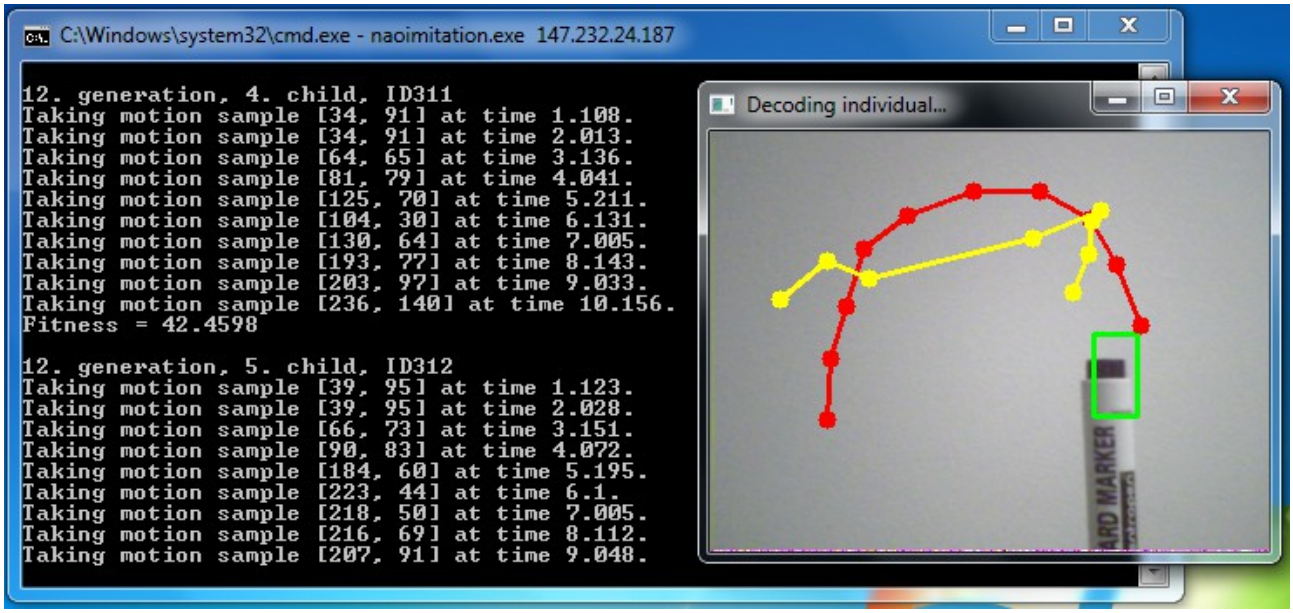
3 Kroky 4) až 9) možno v prípade spracovania videa vykonávať v cykle, resp. v samostatnej funkcii.

3.4 Využitie TLD v programe

TLD je algoritmus umožňujúci sledovanie ľubovoľného objektu vo videu v reálnom čase, ktorý vyvinul Zdeněk Kálal na univerzite v Surrey vo Veľkej Británii[7][8]. Sledovaný objekt je na videu definovaný ohraničujúcim rámcom („*bounding box*“). Sekvencia videa môže byť neohraničená, objekt môže čiastočne meniť svoju veľkosť a vzhľad. Metóda je tiež odolná voči čiastočnému zakrytiu objektu a v prípade jeho zmiznutia zo zorného poľa a opätovného návratu dokáže objekt znovu detegovať.

Pre účely nášho programu sme použili C++ implementáciu algoritmu TLD od Georga Nebehaya[5], pričom sme použili iba tú časť kódu (knižnica „libopentld“), ktorá implementuje triedu TLD. Najdôležitejšie členské premenné a funkcie tejto triedy sú:

- `cv::Rect *currBB;`
  - premenná určuje oblasť ohraničujúcu sledovaný objekt,
- `float currConf;`
  - premenná určuje istotu sledovania,
- `bool learning;`
  - premenná označuje či je učenie zapnuté (aktualizuje sa model objektu) alebo vypnuté,
- `void selectObject(const cv::Mat &img, cv::Rect *bb);`
  - funkcia umožňuje inicializovať detektor na sledovanie nového objektu,
- `void processImage(const cv::Mat &img);`
  - funkcia spracuje vstupný obrázok aktualizuje polohu objektu v premennej `currBB`,
- `void writeToFile(const char *path);`
  - funkcia ukladá aktuálny model sledovaného objektu do súboru,
- `void readFromFile(const char *path);`
  - funkcia umožňuje inicializovať detektor podľa modelu objektu uloženého v súbore.



Obr. 1: TLD je v programe použité ako dekodér, ktorý prevádza hodnoty polôh servomotorov do priestoru súradníc na obraze. Červenou farbou je vyznačená referenčná trajektória, žltou farbou dekódovaná trajektória aktuálneho pohybu.

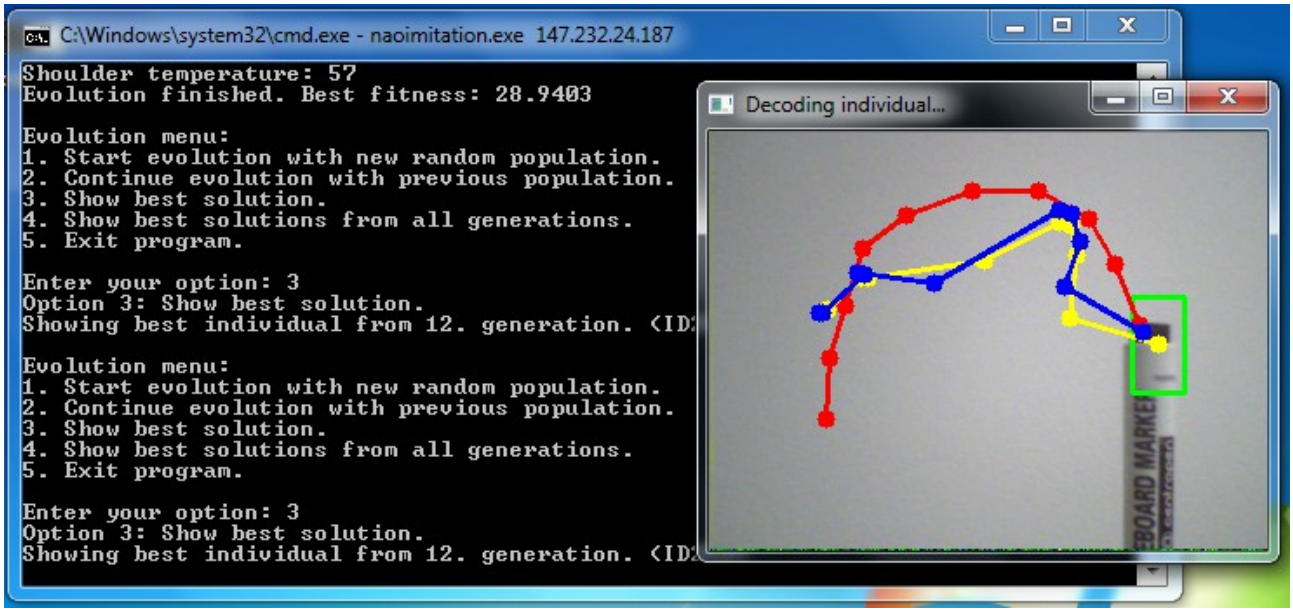
Ako je už naznačené v časti 3.1, v našom programe používame TLD ako dekodér, ktorý umožňuje vykonať transformáciu z priestoru polôh servomotorov (*pitch*, *roll*) do priestoru súradníc (x, y) na obraze. Keďže TLD je potrebné pre funkčnosť ďalších častí programu, je dôležité ho

inicializovať hneď po spustení programu. Hneď v 1. Menu je možné inicializovať TLD vybratím nového objektu, alebo načítaním modelu objektu zo súboru. Následne v 2. Menu je potrebné vykonať adekvátne naučenie modelu, pretože počas evolúcie je učenie detektora vypnuté. Po naučení je tiež výhodné model uložiť pre prípadné neskoršie použitie.

3.5 Popis evolúcie

Pri evolúcii[12] spočiatku generujeme novú populáciu náhodných jedincov, kde každému jedincovi odpovedajú dva reťazce, ktorých dĺžka je totožná s dĺžkou vektora súradníc určujúceho referenčnú trajektóriu. Tieto reťazce sú:

- **genotyp** – vektor polôh servomotorov (*pitch*, *roll*),
  - generuje sa náhodne (resp. krížením a mutáciou v procese evolúcie),
  - nie je možné určiť pomocou neho vhodnosť (nevieme porovnať *trajektóriu* určenú ako vektor súradníc (x_{ref}, y_{ref}) s *genotypom* určeným ako vektor polôh (*pitch*, *roll*)),
- **fenotyp** – vektor súradníc (x, y) na obraze,
  - je prejavom genotypu (tj. zobrazuje transformáciu vektoru polôh servomotorov (*pitch*, *roll*) na vektor súradníc (x, y) na obraze),
  - určuje sa dekódovaním genotypu pomocou TLD,
  - je možné pomocou neho určiť vhodnosť[4] (porovnávame súradnice (x_{ref}, y_{ref}) referenčnej trajektórie so súradnicami (x, y) určenými fenotypom).



Obr. 2: Ukážka jedinca. Legenda: červená – referenčná trajektória, modrá – trajektória určená fenotypom jedinca, žltá – trajektória určená práve spusteným detektorom.

Vhodnosť jedincov je určená ako suma euklidovských vzdialeností medzi referenčnou hodnotou súradníc (x_{ref}, y_{ref}) a dosiahnutou hodnotou súradních (x, y) cez celý vektor fenotypu:

$$f(ind) = \sum_{i=1}^{len(ind)} \sqrt{(x_i^{ref} - x_i)^2 + (y_i^{ref} - y_i)^2}$$

4 V prípade, že TLD počas dekódovania zlyhá, tj. objekt sa dostane mimo obrazu, resp. nebude detegovaný aj napriek tomu, že sa na obraze nachádza, fenotyp nie je možné určiť. Takýto jedinec dostane automaticky najhoršiu možnú vhodnosť.

Cieľom je minimalizácia tejto hodnoty. Náhodne vygenerovaná populácia následne cyklicky prechádza nasledujúcim procesom:

1) **Selekcia** – výber rodičov,
   - Použitá je selekcia orezaním, teda *n* najlepších potomkov sa dostane do množiny rodičov. Počet rodičov *n* je zadaný užívateľom.
2) **Kríženia** a **mutácia** – generovanie potomkov a ich ohodnotenie,
   - Podľa počtu potomkov *p* zadaného užívateľom sú títo postupne vytvorení z rodičov krížením a mutáciou. Rodičia konkrétneho potomka sú vyberaní nasledujúcim spôsobom:
     1) Rodič A je vybraný z množiny rodičov podľa hodnoty fitness. Pri prvom potomkovi je vybratý rodič s najlepšou hodnotou fitness a pri ďalších potomkoch postupne všetci ostatní. V prípade, že zadaný počet potomkov je vyšší, ako počet rodičov, po krížení najhoršieho rodiča algoritmus znova začína pri najlepšom rodičovi.
     2) Rodič B je vybraný z množiny rodičov náhodne. Toto zaručuje, že každý z rodičov má šancu splodiť potomka aj v prípade že počet potomkov je nižší, ako počet rodičov.
   - Použité je dvojbodové kríženie s náhodným umiestnením krížiacich bodov v genotype.
   - Mutácia má na každej pozícii genotypu rovnakú šancu mutovať *pitch* zložku ako aj *roll* zložku. Úroveň zmeny je v prípade výskytu mutácie na danej pozícii náhodne určená z množiny {+5 %, +10 %, –5 %, –10 %}. Pravdepodobnosť mutácie na každej pozícii je zadaná užívateľom.
3) **Akceptácia** – nahradenie najslabších jedincov v populácii potomkami.

4 Zoznam nevyriešených chýb a problémov

Nasleduje zoznam možných problémov v implementácii programu, ktoré z nedostatku času, resp. programátorských skúseností nebolo možné odstrániť:

1) Možný pád programu pri spracovaní snímku pomocou TLD:
   - napr. v prípade, že rozmery oblasti ohraničujúcej objekt prerastú rozmery obrázku,
   - zrejme ide o chybu použitej implementácie TLD,
   - frekvencia výskytu tejto chyby je minimálna.
2) Možné problémy s nesprávnym načítaním súborov:
   - program počíta so správnym formátom použitých súborov (detektor, trajektória, populácia),
   - v prípade manuálnej úpravy súborov je potrebné dodržať pôvodný formát.
3) Možné problémy s nesprávnym ID kamery:
   - v prípade, že kamera použitá na robotovi bola manuálne prepnutá na spodnú kameru, program nebude fungovať správne,
   - je potrebné prepnúť kameru na hornú:
     1) napr. príkazom: `cameraProxy.setParam("kCameraSelectID", 0);`
     2) alebo: `cameraProxy.setParam("kCameraSetDefaultParamsID");`
4) Nesprávna funkcia fail-safe mechanizmu pri použití iného objektu, než bol použitý pri pokusoch.
   - V programe je implementovaná funkcia, ktorá pri pohybe objektu zabraňuje jeho presunu mimo obrazu, pričom bolo nevyhnutné nastaviť parametre fail-safe mechanizmu napevno s ohľadom na tvar a veľkosť sledovaného objektu.
   - V našom prípade sme použili ako sledovaný objekt fixku (prepisovačku).
   - Pri použití iného objektu nemusí fail-safe mechanizmus fungovať správne.
   - Fail-safe mechanizmus (ktorý nie je nevyhnutne potrebný pre funkčnosť programu, avšak urýchluje proces evolúcie zmenšením priestoru prehľadávania) je možné odstrániť úpravou funkcie `positionFix()` v súbore `motion.cpp` z tvaru:

```
void positionFix(double *roll, double *pitch)
{
    *roll = checkRoll(*roll, *pitch);
    *pitch = checkPitch(*roll, *pitch);
}
```

     na tvar:

```
void positionFix(double *roll, double *pitch)
{
    ;
}
```

5 Autori

- Jaroslav Vraštiak
  → Programovanie:
    - ovládanie servomotorov robota a fail-safe mechanizmus, kríženie a mutácia jedinca, evolúcia populácie.
  → Vykonanie pokusov a dohľad nad evolúciou.
  → Tvorba video-dokumentácie.
- Michal Puheim
  → Programovanie:
    - získanie obrazu z robota a prepojenie s knižnicou TLD, dekódovanie fitness jedinca, evolúcia populácie.
  → Tvorba dokumentácie.

6 Video z experimentov

Video z experimentov je dostupné na: *http://www.youtube.com/watch?v=yVN_5vIsT6c*

7 Zdroje

[1] ALDEBARAN ROBOTICS: *C++ SDK Installation*. [online]. [citované 2012-12-28]. Dostupný na internete: <http://www.aldebaran-robotics.com/documentation/dev/cpp/install_guide.html>

[2] ALDEBARAN ROBOTICS: *Using OpenCV: Toolchain OpenCV versus system OpenCV*. [online]. [citované 2012-12-31]. Dostupný na internete: <http://www.aldebaran-robotics.com/documentation/dev/cpp/examples/vision/opencv.html#toolchain-opencv-versus-system-opencv>

[3] ALDEBARAN ROBOTICS: *Using qiBuild with Aldebaran packages*. [online]. [citované 2012-12-31]. Dostupný na internete: <http://www.aldebaran-robotics.com/documentation/dev/cpp/tutos/using_qibuild.html>

[4] ALDEBARAN ROBOTICS: *NAOqi Framework*. [online]. [citované 2012-13-28]. Dostupný na internete: <http://www.aldebaran-robotics.com/documentation/dev/naoqi/index.html>

[5] NEBEHAY, G.: *OpenTLD C++ Implementation*. [online]. [citované 2013-01-02]. Dostupný na internete: <http://gnebehay.github.com/OpenTLD/>

[6] NEBEHAY, G.: *Robust Object Tracking Based on Tracking-Learning-Detection*. Diplomarbeit. Fakultät für Informatik, Technische Universität Wien. 2012. Dostupný na internete: <http://gnebehay.github.com/OpenTLD/gnebehay_thesis_msc.pdf>

[7] KALAL, Z.: *TLD – Tracking Learning Detection*. [online]. [citované 2013-01-02]. Dostupný na internete: <http://info.ee.surrey.ac.uk/Personal/Z.Kalal/tld.html>

[8] KALAL, Z., MATAS, J., MIKOLAJCZYK, K.: *P-N Learning: Bootstrapping Binary Classifiers by Structural Constraints*. Conference on Computer Vision and Pattern Recognition. 2010. Dostupný na internete: <http://info.ee.surrey.ac.uk/Personal/Z.Kalal/Publications/2010_cvpr.pdf>

[9] WILLOWGARAGE: *OpenCV Wiki*. [online]. [citované 2013-01-02]. Dostupný na internete: <http://opencv.willowgarage.com/wiki/>

[10] ALDEBARAN ROBOTICS: *NAO Software 1.14 documentation*. [online]. [citované 2013-01-02]. Dostupný na internete: <http://www.aldebaran-robotics.com/documentation/index.html>

[11] ALDEBARAN ROBOTICS: *Getting an image*. [online]. [citované 2013-01-02]. Dostupný na internete: <http://www.aldebaran-robotics.com/documentation/dev/cpp/examples/vision/getimage/getimage.html>

[12] OBITKO, M.: *Introduction to Genetic Algorithms*. [online]. [citované 2013-01-02]. Dostupný na internete: <http://www.obitko.com/tutorials/genetic-algorithms/>