TECHNICKÁ UNIVERZITA V KOŠICIACH

FAKULTA ELEKTROTECHNIKY A INFORMATIKY

APLIKÁCIA METÓDY TLD NA SLEDOVANIE OBJEKTOV V STEREOVÍZNOM OBRAZE

DIPLOMOVÁ PRÁCA

Príloha B: Systémová príručka

Košice 2013 **Michal PUHEIM**

Obsah

1 Funkcia programu

Program Stereovision umožňuje robotovi Nao sledovať ľubovoľný vybraný objekt a určiť jeho polohu v priestore. Taktiež umožňuje naučiť robota hýbať rukou s cieľom dotknúť sa objektu.

Program je postavený ako konzolová aplikácia, ktorá je schopná v prípade potreby zobraziť grafické okná s obrazom z kamier robota. Hoci bol testovaný len na OS Windows, jeho preloženie na iný systém by nemalo byť problémom.

2 Analýza riešenia

Cieľom navrhnutého systém je využitie metódy TLD pri použití stereovíznej hlavy robota Nao. Takto navrhnutý systém po svojom umožňuje robotovi sledovať ľubovoľne vybraný objekt, a tiež určiť jeho pozíciu v priestore vztiahnutú voči hlave robota v dvoch súradnicových systémoch – karteziánskom a polárnom.

Sledovaný objekt je najprv manuálne označený užívateľom zadaním ohraničujúceho rámca. Pomocou tohto rámca je inicializovaný jeden z TLD systémov a následne je užívateľovi ponúknutá možnosť trénovať tento detektor (napr. pohybom označeného objektu, zmenou jeho vzdialenosti, resp otočenia). Tento trénovací mód nie je nevyhnutný, ale v konečnom dôsledku zlepšuje úspešnosť a presnosť sledovania objektu počas ďalšej činnosti stereovízneho systému.

Po inicializácii počiatočného TLD detektora (a voliteľnom učení) sa tento detektor skopíruje, čím dostaneme dva samostatné TLD systémy, každý pre jednu kameru. Pri návrhu stereovízneho systému teda počítame s dvoma paralelne bežiacimi TLD, kde každý sleduje objekt na jednej z kamier v stereovíznej konfigurácii.

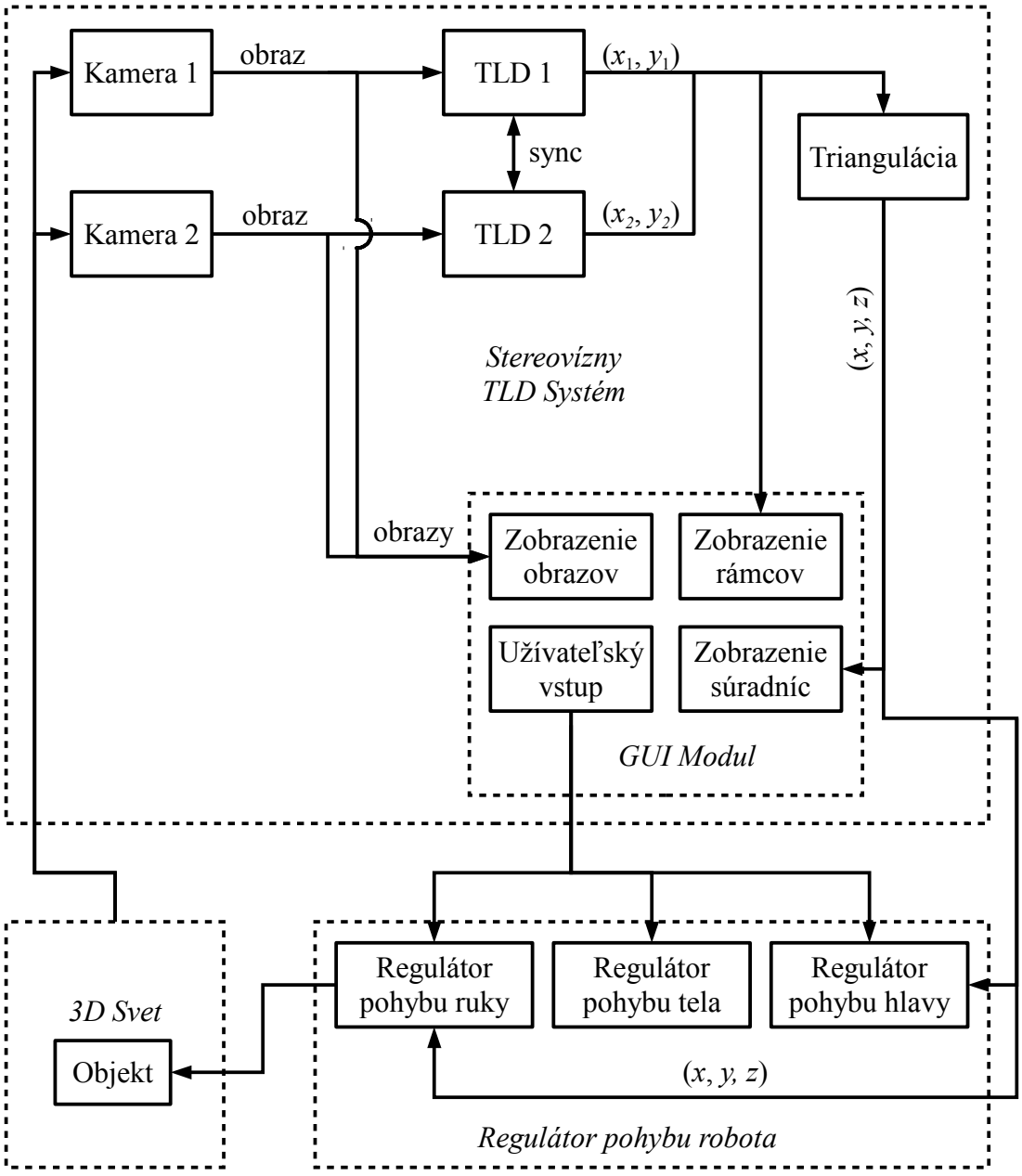
Súčasťou riešenia je špeciálny regulátor pohybu ruky robota. Jeho cieľom je umožniť robotovi dotknúť sa sledovaného predmetu na základe informácií o polohe objektu získanej z výstupov stereovízneho systému. Na tento účel bola použitá dopredná viacvrstvová neurónová sieť.

Podrobný teoretický rozbor riešenia a návrhu systému je uvedený v hlavnej časti diplomovej práce. V ďalších častiach tejto príručky sa budeme venovať rozboru systému z programátorského hľadiska.

3 Popis programu

3.1 Popis riešenia

Systém sa skladá z dvoch základných súčastí. Prvým je stereovízny systém, v rámci ktorého je implementované grafické užívateľské rozhranie. Druhým je regulátor pohybu robota, ktorý zabezpečuje ovládanie servomotorov hlavy, ruky a do určitej miery aj zvyšnej časti tela. Realizácia systému a spôsob prepojenia jeho vnútorných komponentov je zobrazený v schéme na obr. 1.



Obr. 1 Zjednodušená bloková schéma systému.

3.2 Implementácia systému v C++ a použité knižnice

Program je postavený ako konzolová aplikácia a je napísaný v jazyku C++. Tento jazyk bol vybraný z dôvodu najvyššieho výkonu spomedzi všetkých ostatných jazykov, ktoré sú pri programovaní pre robota Nao podporované a tiež z dôvodu jednoduchého prepojenia knižníc potrebných pre jeho funkčnosť. Ďaľšou výhodou je možnosť spustenia programu priamo na matičnej doske robota (po menších úpravách a odstránené grafických rozhraní).

Na vykreslenie obrazových okien a spracovanie obrazu používa knižnicu OpenCV [1]. Ďalej sú použité knižnice zabezpečujúce prístup k NAOqi [2], ktoré umožňujú ovládanie robota Nao a získanie jeho senzorických informácii, a knižnica LibOpenTLD obsahujúca C++ implementáciu algoritmu TLD od Georga Nebehaya [3] umožňujúca sledovanie objektu na obraze. V prípade bližšieho záujmu o použité technológie viď nasledujúce odkazy:

TLD – viď [3], [4], [5], [6], [7] a [8].

OpenCV – viď [1].

Nao C++ SDK – viď [9] a [10].

3.3 Popis tried a údajových štruktúr

Nasledujúci zoznam obsahuje identifikáciu tried a štruktúr, ktoré boli pre potreby programu implementované. Neobsahuje dokumentáciu tried tretích strán použitých v programe (napr. knižníc OpenCV, OpenTLD a pod.). V zozname sú uvedené ich stručné popisy a v zátvorkách odkazy na strany obsahujúce detailnejšiu dokumentáciu. Zoznam je usporiadaný v abecednom poradí:

- `CorrParams` – korekčné parametre použité pri korekcii vzdialenosti. (11)
- `DataSet` – množina trénovacích dát pre účely učenia neurónovej siete. (11)
- `HandController` – regulátor pohybu ruky robota. (9)
- `HeadController` – regulátor pohybu hlavy robota. (12)
- `ImAcquisition` – modul akvizície obrazu. (8)
- `Individual` – jednotlivec v populácii evolučného algoritmu. (12)
- `Layer` – vrstva neurónovej siete. (13)
- `Limits` – hraničné hodnoty pohybu servomotorov ruky a hlavy robota. (13)
- `MinMax` – dvojica hodnôt minimum a maximum. (14)
- `MotionController` – regulátor pohybu robota. (9)
- `NeuralNetwork` – neurónová sieť. (14)

- `Neuron` – neurón, základná jednotka neurónovej siete. (15)
- `Object` – trieda určuje hodnoty parametrov sledovaného objektu. (10)
- `Pattern` – jedna vzorka v množine trénovacích dát. (16)
- `Population` – populácia evolučného algoritmu. (16)
- `Robot` – hlavná trieda programu, robot. (6)
- `Sample` – dvojica vstupov x a výstupov y funkcie $y = f(x)$. (17)
- `StereoTLD` – trieda implementujúca stereovízny TLD systém. (7)
- `Timer` – počítadlo času. (17)

3.3.1 Dokumentácia najvýznamnejších tried programu

Nasleduje dokumentácia najvýznamnejších tried programu usporiadaných podľa miery významnosti.

3.3.1.1 Dokumentácia triedy Robot

Trieda robot je hlavnou triedou programu a zahŕňa všetky ostatné triedy a metódy, ktoré sú potrebné pre fungovanie programu. Najdôležitejšie moduly tejto triedy sú modul akvizície obrazu ImAcquisition, modul stereovízie StereoTLD a modul kontroly pohybu robota MotionController.

Verejné metódy:

- `Robot (const std::string &ip, const std::string &proxyClientName)`
  - konštruktor,
- `void init ()`
  - inicializácia komponentov.

Verejné atribúty:

- `std::string robotIp`
  - IP adresa robota,
- `std::string clientName`
  - názov klienta pre pripojenie do modulu akvizície obrazu,
- `MotionController * motion`
  - modul regulácie pohybu robota,
- `AL::ALMemoryProxy * memory`
  - pamäťový modul, ktorý poskytuje informácie z pamäte robota,
- `ImAcquisition * acquisition`
  - modul akvizície obrazu,
- `StereoTLD * stereotld`

- modul stereovízneho videnia.

3.3.1.2 Dokumentácia triedy StereoTLD

Trieda StereoTLD umožňuje robotovi sledovať ľubovoľne vybraný objekt, a tiež určiť jeho pozíciu v priestore vztiahnutú voči hlave robota v dvoch súradnicových systémoch – karteziánskom a polárnom. Zabezpečuje inicializáciu objektu, učenie detektorov a uloženie modelu objektu do súboru. Tiež implementuje grafické používateľské rozhranie pre prezentáciu vypočítaných atribútov objektu.

Verejné metódy:

- `StereoTLD (ImAcquisition *acquisition)`
  - konštruktor,
- `void initFromBB ()`
  - inicializácia označením nového sledovaného objektu,
- `void initFromFile (const char *path)`
  - inicializácia pomocou modelu objektu uloženého v súbore,
- `void runAndTrainTLD ()`
  - učenie TLD detektorov,
- `void saveToFile (const char *path)`
  - uloženie modelu objektu do súboru,
- `void processImages ()`
  - spracovanie aktuálnych obrazov z kamier a aktualizácia parametrov sledovaného objektu,
- `void initGui ()`
  - inicializácia grafického užívateľského rozhrania,
- `void refreshGui ()`
  - aktualizácia grafického užívateľského rozhrania,
- `void destroyGui ()`
  - vypnutie grafického užívateľského rozhrania,
- `void setSyncTime (double ms)`
  - nastavenie času synchronizácie detektorov
    (ak ms=0, potom je synchronizácia vypnutá)

Verejné atribúty:

- `tld::TLD * leftDetector`
  - ľavý TLD detektor,

- `tld::TLD * rightDetector`
  - prevý TLD detektor,
- `Object * object`
  - sledovaný objekt,
- `bool syncEnabled`
  - parameter určujúci, či je synchronizácia detektorov zapnutá alebo vypnutá.

3.3.1.3 Dokumentácia triedy ImAcquisition

Trieda ImAcquisition zabezpečuje pripojenie programu na video proxy robota a prístup k aktuálnym obrazom z kamier. Zabezpečuje tiež potrebné spracovanie týchto obrazov, ako je napr. konverzia obrazu z natívneho formátu na cv::Mat.

Verejné metódy:

- `ImAcquisition (const std::string &robotIp, const std::string &clientName)`
  - konštruktor,
- `void init (const std::string &robotIp, const std::string &clientName)`
  - inicializácia a pripojenie na robota,
- `void updateImages ()`
  - aktualizácia obrazov,
- `void showVideo ()`
  - zobrazenie grafického okna s videom z kamery robota.

Verejné atribúty:

- `bool initialized`
  - parameter určujúci, či modul bol alebo nebol úspešne inicializovaný,
- `std::string robotIp`
  - IP adresa robota,
- `AL::ALVideoDeviceProxy * proxy`
  - proxy na pripojenie ku kamerám robota (video proxy),
- `std::string proxyClient`
  - názov klienta použitého na pripojenie ku kamerám robota,
- `cv::Mat leftImage`
  - aktuálny obraz z ľavej kamery,
- `cv::Mat rightImage`
  - aktuálny obraz z pravej kamery,

- `double focalDist`
  - ohnisková vzdialenosť kamier,
- `double camDist`
  - vzdialenosť medzi kamerami,
- `double horizontFov`
  - horizontálny zorný uhol,
- `double verticalFov`
  - vertikálny zorný uhol,
- `CorrParams distCorrection`
  - korekčné parametre pre určovanie vzdialenosti objektov.

3.3.1.4 Dokumentácia triedy MotionController

Trieda MotionController zabezpečuje prístup k motion proxy robota a teda umožňuje riadenia jeho pohybu. Tiež zahŕňa priamo implementované metódy pre niektore základné pohyby, ako je napr. pohyb do základnej polohy. Súčasťou sú moduly pre ovládanie pohybu ruky a hlavy robota.

Verejné metódy:

- `MotionController (const std::string &robotIp, AL::ALMemoryProxy *memoryProxy, ImAcquisition *acquisition)`
  - konštruktor,
- `void initialPosition ()`
  - zapnutie motorov robota a pohyb do základnej pozície,
- `void rest ()`
  - pohyb robota do bezpečnej pozície v podrepe a vypnutie motorov.

Verejné atribúty:

- `AL::ALMotionProxy * proxy`
  - pohybová proxy (motion proxy) robota,
- `HandController * handController`
  - regulátor pohybu ruky robota,
- `HeadController * headController`
  - regulátor pohybu hlavy robota.

3.3.1.5 Dokumentácia triedy HandController

Trieda HandController umožňuje ovládanie pohybu ruky robota pomocou neurónovej siete. Zabezpečuje tiež zbieranie dát pre učenie siete a vlastné učenie tejto

siete. Tiež má priamo implementované metódy pre pohyb ruky do niektorých základných polôh.

Verejné metódy:

- `HandController (AL::ALMotionProxy *motionProxy, AL::ALMemoryProxy *memoryProxy, ImAcquisition *acquisition)`
  - konštruktor,
- `void init (const char *path)`
  - inicializácia pomocou konfigurácie neurónovej siete uloženej v súbore,
- `void init (DataSet *trainDataset)`
  - inicializácia neurónovej siete učením na trénovacej množine dát,
- `void trainOnDataset (DataSet *trainDataset)`
  - učenie neurónovej siete na trénovacej množine dát,
- `DataSet * createDataset (StereoTLD *stereotld, HeadController *head)`
  - vytvorenie novej množiny trénovacích dát,
- `void save (const char *path)`
  - uloženie konfigurácie neurónovej siete do súboru,
- `void frontPosition ()`
  - pohyb ruky do štandardnej pozície pred robotom,
- `void initialPosition ()`
  - pohyb ruky do základnej pozície vedľa tela robota,
- `void grabObject (Object *object)`
  - pokus o dotyk objektu.

Verejné atribúty:

- `bool initialized`
  - atribút určuje či je regulátor pohybu ruky inicializovaný alebo nie.

3.3.1.6 Dokumentácia triedy Object

Trieda Object implementuje sledovaný objekt a zabezpečuje aktualizáciu jeho polohy vo forme priestorových súradníc, ako aj prístup k týmto súradniciam.

Verejné metódy:

- `Object (ImAcquisition *acquisition)`
  - konštruktor,
- `void update (cv::Rect *leftBB, cv::Rect *rightBB)`
  - aktualizácia polohy objektu pomocou ohraničujúcich rámcov.

Verejné atribúty:

- `bool valid`
  - určuje, či sú aktuálne súradnice objektu platné, tj. objekt je detegovaný správne pomocou oboch TLD detektorov,
- `bool partial`
  - určuje, či sú aktuálne súradnice objektu aspoň čiastočne platné, tj. objekt je detegovaný aspoň jedným z dvoch TLD detektorov,
- `double x`
- `double y`
- `double z`
  - karteziánske súradnice objektu (x, y, z) vztiahnuté k hlave robota,
- `double dist`
- `double pitch`
- `double yaw`
  - polárne súradnice objektu (*dist*, *pitch*, *yaw*) vztiahnuté k hlave robota, kde *dist* je vzdialenosť, *pitch* je vertikálny uhol a *yaw* je horizontálny uhol.

3.3.2 Dokumentácia ostatných tried

Nasleduje dokumentácia ďalších menej významných tried programu uvedených v abecednom poradí.

3.3.2.1 Dokumentácia štruktúry CorrParams

Korekčné parametre.

Verejné atribúty:

- `double a0`
- `double a1`
- `double a2`
  - korekčné parametre funkcie $f(x) = a_2x^2 + a_1 x + a_0$.

3.3.2.2 Dokumentácia triedy DataSet

Dátová množina určená na trénovanie neurónovej siete.

Verejné metódy:

- `DataSet (int patternCount, int inputCount, int outputCount)`
  - konštruktor – novej dátovej množiny definovanej vstupnými parametrami,
- `DataSet (const char *path)`
  - konštruktor – načíta dátovú množinu uloženú v súbore,
- `void saveToFile (const char *path)`

- uloženie dátovej množiny do súboru.

Verejné atribúty:

- `int patternCount`
  - počet prvkov dátovej množiny,
- `int inputCount`
  - počet vstupov jedného prvku dátovej množiny,
- `int outputCount`
  - počet výstupov jedného prvku dátovej množiny,
- `Pattern * input`
  - vstupy,
- `Pattern * output`
  - výstupy,

3.3.2.3 Dokumentácia triedy HeadController

Regulátor pohybu hlavy robota. Umožňuje otáčať hlavu s cieľom sledovať objekt.

Verejné metódy:

- `HeadController (AL::ALMotionProxy *motionProxy)`
  - konštruktor,
- `void initialPosition ()`
  - presun hlavy do základnej polohy,
- `void trackObject (Object *object)`
  - posun hlavy s cieľom vycentrovať objekt v strede zorného poľa.

3.3.2.4 Dokumentácia triedy Individual

Jednotlivec v populácii evolučného algoritmu používaného na hľadanie parametrov funkcie pre korekciu vzdialenosti.

Verejné metódy:

- `Individual (int lenght)`
  - konštruktor – vytvára nového jedinca s náhodným genotypom danej dĺžky,
- `Individual (Individual parent1, Individual parent2, int mutation)`
  - konštruktor – vytvára nového jedinca krížením genotypov dvoch pôvodných jedincov a následnou mutáciou takto vytvoreného genotypu,
- `void evaluate (std::vector< Sample > *dataVector)`
  - ohodnotenie jedinca, určenie jeho vhodnosti.

Verejné atribúty:

- `unsigned int id`
  - unikátne ID jedinca,
- `float fitness`
  - vhodnosť jedinca,
- `int chromosomeLength`
  - dĺžka chromozómu (genotypu) jedinca,
- `std::vector< float > chromosome`
  - chromozóm (genotyp) jedinca.

3.3.2.5 Dokumentácia triedy Layer

Jedna vrstva neurónovej siete.

Verejné metódy:

- `Layer (int size)`
  - konštruktor vstupnej vrstvy,
- `Layer (int size, Layer *previous)`
  - konštruktor nasledovnej vrstvy,
- `void calculateOutput ()`
  - výpočet výstupu neurónov vo vrstve.

Verejné atribúty:

- `Layer * previous`
  - odkaz na predchádzajúcu vrstvu,
- `Neuron ** neuron`
  - zoznam neurónov vo vrstve,
- `int neuronCount`
  - počet neurónov vo vrstve.

3.3.2.6 Dokumentácia štruktúry Limits

Hraničné hodnoty vybraných kĺbov ruky a hlavy robota.

Verejné atribúty:

- `MinMax shoulderPitch`
- `MinMax shoulderRoll`
- `MinMax elbowYaw`
- `MinMax elbowRoll`
- `MinMax wristYaw`
- `MinMax viewVertical`
- `MinMax viewHorizontal`

- `MinMax distance`

3.3.2.7 Dokumentácia štruktúry MinMax

Dvojica hodnôt – minimum a maximum.

Verejné atribúty:

- `double max`
- `double min`

3.3.2.8 Dokumentácia triedy NeuralNetwork

Neurónová sieť. Umožňuje mapovať konkrétne vstupy ku konkrétnym výstupom. Učenie siete je možné pomocou trénovacej dátovej množiny. Presnosť mapovania siete je tiež možné testovať použitím testovacích vzoriek, alebo celých testovacích množín.

Verejné metódy:

- `NeuralNetwork (int layerCount, int *neuronCount)`
  - konštruktor novej neurónovej siete,
- `NeuralNetwork (const char *path)`
  - konštruktor neurónovej siete určenej konfiguráciou v súbore,
- `void run (float *input)`
  - výpočet výstupov neurónovej siete na základe daných vstupov,
- `void calculateOutput (float *input, float *output)`
  - výpočet výstupov neurónovej siete na základe vstupov v poli `input` a uloženie výstupov v poli `output`.
- `void trainOnSample (float *input, float *output, float gamma)`
  - učenie neurónovej siete na základe zadaných vstupných a výstupných hodnôt v poliach `input` a `output` a zadaného parametra učenia `gamma`.
- `void train (DataSet *trainDataset, float gamma)`
  - učenie neurónovej siete na celej množine trénovacích dát s daným parametrom učenia.
- `float testOnSample (float *input, float *output)`
  - ohodnotenie výkonu neurónovej siete na základe zadaných vstupných a výstupných hodnôt v poliach `input` a `output`; navratová hodnota je v intervale <0, 1> a určuje presnosť mapovania vstupov voči výstupom,
- `float test (DataSet *testDataset)`
  - ohodnotenie neurónovej siete na celej množine trénovacích dát; navratová hodnota je v intervale <0, 1> a určuje priemernú presnosť mapovania vstupov voči výstupom v celej trénovacej množine,

- `void saveToFile (const char *path)`
  - uloženie konfigurácie neurónovej siete (tj, hodnôt synaptických váh) do súboru.

Verejné atribúty:

- `Layer ** layer`
  - vrstvy neurónovej siete,
- `int layerCount`
  - počet vrstiev neurónovej siete.

3.3.2.9 Dokumentácia triedy Neuron

Implementácia umelého neurónu – základnej jednotky neurónovej siete.

Verejné metódy:

- `Neuron ()`
  - konštruktor vstupného neurónu,
- `Neuron (Neuron **previous, int previousCount)`
  - konštruktor nasledovného neurónu; parametrom je zoznam a počet neurónov, s ktorými je konštruovaný prepojený na vstupe,
- `void calculateOutput ()`
  - výpočet výstupu neurónu (cez vstupnú, aktivačnú a výstupnú funkciu),

Verejné atribúty:

- `float input`
  - vstup neurónu,
- `float activation`
  - aktivačná hodnota neurónu,
- `float output`
  - výstup neurónu,
- `float delta`
  - chybový signál generovaný neurónom,
- `int previousCount`
  - počet predchádzajúcich neurónov, na ktoré je daný neurón pripojený,
- `Neuron ** previousNeuron`
  - zoznam predchádzajúcich neurónov, na ktoré je daný neurón pripojený,
- `float * weight`
  - váhy medzi daným neurónom a predchádzajúcimi neurónmi,

- `float biasWeight`
  - prahová váha.

3.3.2.10 Dokumentácia štruktúry Pattern

Vzorka dát (vstupov alebo výstupov).

Verejné atribúty:

- `float * value`
  - pole s hodnotami danej vzorky.

3.3.2.11 Dokumentácia triedy Population

Populácia evolučného algoritmu používaného na hľadanie parametrov (a_0, a_1, a_2) funkcie pre korekciu vzdialenosti $y = f(x) = a_2x^2 + a_1 x + a_0$. Na určenie týchto parametrov je potrebné udať množinu vstupov x a odpovedajúcich výstupov y vo forme ukazovateľa na pole štruktúr `Sample` (viď 3.3.2.12).

Verejné metódy:

- `Population (int individualLength, Sample *data, int dataCount)`
  - konštruktor; inicializácia populácie jednotlivcov s danou dĺžkou chromozómu (genotypu) ; ako parameter je tiež potrebné zadať vzorku dát (vstupov a odpovedajúcich výstupov, viď 3.3.2.12) pomocou ktorých sa bude určovať vhodnosť jedinca a tiež počet tých to dát.
- `void evolve ()`
  - evolúcia populácie podľa predvolených parametrov,
- `void evolve (float fitnessGoal, double timeLimit)`
  - evolúcia populácie podľa zadaných parametrov; parametrami sú cieľová hodnota vhodnosti (čím nižšia, tým lepšie riešenie – 0 znamená globálne optimálne riešenie, tj. najlepšie) a maximálny čas evolúcie v milisekondách.

Verejné atribúty:

- `int size`
  - veľkosť populácie,
- `int generation`
  - aktuálna generácia,
- `std::vector< Individual > individual`
  - vektor jedincov v populácii.

3.3.2.12 Dokumentácia štruktúry Sample

Dvojica vstupu *x* a odpovedajúceho výstupu *y*.

Verejné atribúty:

- `float x`
- `float y`

3.3.2.13 Dokumentácia triedy Timer

Časomiera. Umožňuje meranie času, ktorý uplynul od spustenia časomiery.

Verejné metódy:

- `Timer (double updateTime)`
  - Konštruktor; spustenie časomiery; vstupný parameter určujúci čas aktualizácie je zadaný v milisekundách,
- `void reset ()`
  - reštart časomiery, tj. nastavenie uplynutého času na 0,
- `void reset (double newUpdateTime)`
  - reštart časomiery, tj. nastavenie uplynutého času na 0 a súčasná zmena času aktualizácie (udaná v milisekundách),
- `bool isTime ()`
  - v prípade, že ešte nenastal čas aktualizácie, metóda vracia hodnotu `false`;
    v prípade, že už nastal čas aktualizácie, metóda vracia hodnotu `true` a reštartuje časomieru,
- `double getElapsedTime ()`
  - metóda vracia čas uplynutý od spustenia (alebo posledného reštartu) časomiery v milisekundách.

3.4 Vstupné a výstupné súbory programu

Systém pracuje celkovo so štyrmi druhmi vstupno-výstupných súborov. Sú to:

- model objektu uložený v súbore *„model.tld“*,
- konfigurácia váh neurónovej siete uložená v súbore *„handnn.net“*,
- množina dát určených na trénovanie neurónovej siete v súbore *„traindata.pat“*,
- konfigurácia systému akvizície obrazu uložená v súbore *„acquisition.cfg“*.

Taktiež je použitý jeden dočasný súbor, ktorým je *„temp.tld“*. Tento je použitý na krátkodobé uloženie modelu detektora TLD pri synchronizácii dvojice detektorov použitej v stereovíznom module.

4 Preklad programu

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, uvedieme postup ich inštalácie a nakoniec aj samotný postup prekladu (kompilácie) programu. Pre bližšie informácie viď [10].

4.1 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.

Poznámky:

- Kroskompilátor generujúci kód bežiaci na robotovi je dostupný iba pre GNU/Linux.
- V prostredí OS Windows je potrebné kompilovať v 32-bitovom režime.

4.2 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 [10].

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*“.

4.3 Postup inštalácie SDK

Postup inštalácie C++ SDK je nasledovný:

1) Inštalácia vyžadovaných závislostí:
   - Viď. časť 4.2.
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 z umiestnenia http://users.aldebaran-robotics.com/ pod záložkou *software / download*.
   - Pre MS Windows je možné použiť archív uložený na priloženom CD v umiestnení: *CD\4. Ďalšie\naoqi-sdk-1.14.2-win32-vs2010.zip*
   - Pozn. 1: Pre umožnenie sťahovania z uvedenej stránky je potrebné sa prihlásiť. Pre prihlasovacie údaje sa obráťte na správcu robota Nao, na ktorom plánujete program používať, resp. na zákaznícku podporu firmy Aldebaran-Robotics.
   - 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 4.2.)
5) Stiahnutie a rozbalenie nástroja *qiBuild*:
   - Stiahnite archív *qibuild-x.zip*.
   - Sťahovať je možné z rovnakého umiestnenia ako v bode 2, resp. z priloženého CD v umiestnení: *CD\4. Ďalšie\qibuild-1.14.2.zip*
   - 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. [11].

4.4 Kompilácia projektu pomocou nástroja qiBuild

Pre kompletný návod, ako vytvárať a kompilovať projekty pomocou nástroja qiBuild, viď. [12]. V rámci tejto časti si uvedieme skrátený návod[1], ako skompilovať zdrojové kódy, ktoré sú súčasťou tohto programu:

1) Vytvorenie pracovného priečinku:
   - Vytvorte nový priečinok, do ktorého budete ukladať projekty.
   - Otvorte tento priečinok v termináli[2] a zadajte príkaz: `qibuild init`
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 programu.
4) Otvorte koreňový priečinok projektu „*stereovision*“.
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 „*stereovision\build-mytoolchain\sdk\bin*“.

[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*“).

5 Zoznam známych chýb v implementácii

Nasleduje zoznam známych chýb a problémov, ktoré sú súčasťou implementácie a z nedostatku času ich nebolo možné odstrániť. Hoci nejde o závažne chyby, v určitých prípadoch môžu spôsobovať problémy. Ich odstránenie je možné a plánované v prípadných budúcich verziách programu:

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 súborov „model.tld“, „handnn.net“, „traindata.pat“ a „acquisition.cfg“,
   - v prípade manuálnej úpravy súborov je potrebné dodržať pôvodný formát.
   - funkcie zabezpečujúce prácu zo súbormi je potrebné ošetriť pre prípad chýb vo formáte súborov.
3) Možné problémy s nesprávnym ID kamery:
   - v prípade, že pozícia kamier bola prehodená iným programom, program nebude fungovať správne,
   - je potrebné implementovať automatické prepnutie na predvolené hodnoty v prípade potreby. Je možné použiť napr. príkaz:
     `cameraProxy.setParam("kCameraSetDefaultParamsID");`
4) Problém s ukladaním konfigurácie neurónovej siete do súboru a načítaním zo súboru:
   - pri implementácii ukladania/načítania do/zo súboru sa pozabudlo na prahové hodnoty váh:
     - pri ukladaní do súboru sa neukladajú prahové hodnoty váh,
     - pri načítaní zo súboru sa nenačítavajú prahové hodnoty váh,
   - je potrebné upraviť funkcie načítania/uloženia neurónovej siete.

6 Zoznam použitej literatúry

[1] WILLOWGARAGE: OpenCV Wiki. [online]. [Citované 2013-04-26]. Dostupné na internete: <http://opencv.willowgarage.com/wiki/>

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

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

[4] NEBEHAY, George: *Robust Object Tracking Based on Tracking-Learning-Detection*. Diplomarbeit. Fakultät für Informatik, Technische Universität Wien. 2012.

[5] KÁLAL, Zdeněk: *TLD*. [Online]. University of Surrey. Guildford. UK. 2011. [Citované 22. 4. 2013]. Dostupné na internete: <http://personal.ee.surrey.ac.uk/Personal/Z.Kalal/tld.html>

[6] KÁLAL, Zdeněk – MATAS, Jíří – MIKOLAJCZYK, Krystian: *P-N Learning: Bootstrapping Binary Classifiers by Structural Constraints*. Conference on Computer Vision and Pattern Recognition. 2010.

[7] KÁLAL, Zdeněk – MATAS, Jíří – MIKOLAJCZYK, Krystian: *Online learning of robust object detectors during unstable tracking*. 3rd On-line Learning for Computer Vision Workshop 2009, Kyoto, Japan, IEEE CS, 2009.

[8] KÁLAL, Zdeněk – MATAS, Jíří – MIKOLAJCZYK, Krystian: *Forward-Backward Error: Automatic Detection of Tracking Failures*. International Conference on Pattern Recognition, Istambul, Turkey, 23-26 August, 2010.

[9] ALDEBARAN ROBOTICS: *Nao Documentation v1.14.2*. [Online]. [Citované 22. 4. 2013]. Dostupné na internete: <http://www.aldebaran-robotics.com/documentation/index.html>

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

[11] 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>

[12] 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>