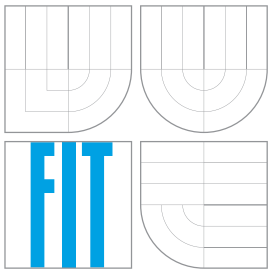
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

DETEKCE RUKY UŽIVATELE V HLOUBKOVÝCH DATECH

USER HAND DETECTION IN DEPTH DATA

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

PAVOL MALÍK

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. VÍTĚZSLAV BERAN, Ph.D.

BRNO 2016

Zadání bakalářské práce/18160/2015/xmalik14

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačové grafiky a multimédií Akademický rok 2015/2016

Zadání bakalářské práce

Řešitel: **Malík Pavol**
Obor: Informační technologie
Téma: **Detekce ruky uživatele v hloubkových datech**
User Hand Detection in Depth Data
Kategorie: Zpracování obrazu

Pokyny:

1. Seznamte se s postupy detekce, segmentace a výpočtu 3D pózy objektů v hloubkových datech. Nastudujte algoritmy pro detekci a odhad pózy ruky uživatele.
2. Navrhněte metodu, která v hloubkových datech ze senzoru, který je umístěn nad uživatelem, bude detekovat a odhadovat pozici ruky.
3. Připravte a anotujte datovou sadu pro vývoj, trénování a testování systému.
4. Implementujte navrženou metodu s využitím existujících knihoven a proveďte experimenty a vyhodnocení stability a přesnosti metody v různých podmínkách.
5. Připravte demonstrační aplikaci.
6. Vytvořte plakát a krátké demonstrační video reprezentující Vaše řešení.

Literatura:

- Bradski G. R., Kaehler A. *Learning OpenCV: Computer Vision with the OpenCV Library.* O'Reilly Media, Inc. 2008.
- M. Sonka, V. Hlaváč, R. Boyle: Image Processing, Analysis, and Machine Vision, CL-Engineering, ISBN-13: 978-0495082521, 2007.
- Dále dle pokynu vedoucího.

Pro udělení zápočtu za první semestr je požadováno:

- Body 1, 2, 3 a částečně bod 4.

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Beran Vítězslav, Ing., Ph.D.**, UPGM FIT VUT
Datum zadání: 1. listopadu 2015
Datum odevzdání: 18. května 2016

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav počítačové grafiky a multimédií
612 66 Brno, Božetěchova 2

L.S.

doc. Dr. Ing. Jan Černocký
vedoucí ústavu

Abstrakt

Cieľom tejto práce je detekovať ruku užívateľa nad interaktívnym stolom a zobraziť jej pozíciu voči stolu. Na vypracovanie bol zvolený prístup pomocou hĺbkových dát, použitím senzora Kinect. Detekcia je realizovaná použitím registračnej metódy SAC-IA, pracujúcej na princípe šablónového porovnávania. Ďalej práca pojednáva o existujúcich riešeniach detekcie rúk, ako aj postupe použitom vo výslednom programe. Bola definovaná dátová sada, obsahujúca vzory rúk a testovacie scény, na ktorých bol klasifikátor vyhodnotený. Záverečné testovanie prebehlo na viac ako sto objektoch a úspešnosť klasifikátora dosiahla 82 %.

Abstract

The aim of this thesis is to detect hand of user above the interactive table and show its position with respect to the table. Used sensor was Kinect so the depth data approach was selected. The detection is realized using registration method SAC-IA, which works on the template matching principle. Thesis deals with existing solutions of hand detection as well as procedure used in this work. Dataset was defined, containing hand templates and testing scenes, from which the classifier was evaluated. Final testing was performed on more than a hundred objects and the success rate reached 82 %.

Klíčová slova

Kinect, 3D, PCL, ROS, interaktívny stôl, detekcia objektov, detekcia rúk, mračno bodov, registrácia, porovnávanie šablónou, hĺbkové dáta

Keywords

Kinect, 3D, PCL, ROS, interactive table, object detection, hand detection, pointcloud, registration, template matching, depth data

Citace

MALÍK, Pavol. *Detekce ruky uživatele v hloubkových datech.* Brno, 2016. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Beran Vítězslav.

Detekce ruky uživatele v hloubkových datech

Prohlášení

Prehlasujem, že som túto bakalársku prácu vypracoval samostatne pod vedením pána Ing. Vítězslava Berana, Ph.D.

.....
Pavol Malík
18. května 2016

Poděkování

Moje poďakovanie patrí Ing. Vítězslavovi Beranovi, Ph.D., za podporu, odborné vedenie, konzultácie a poskytnuté rady pri tejto bakalárskej práci. Ďalej by som chcel poďakovať Ing. Michalovi Kapinusovi, ako aj ostatným z ÚPGM, ktorí mi poskytli pomoc pri práci v robotickom laboratóriu.

Obsah

Kapitola 1

Úvod

Počítačové videnie má dnes nespočetné množstvo využití. Vďaka moderným kamerovým systémom či senzorom je možné zachytiť každý snímaný detail či už v 2D alebo 3D. To, čo bolo kedysi cenovo nedostupné sa dnes stáva bežným – kamery s vysokým rozlíšením kompaktnej veľkosti sú štandardným vybavením aktuálnych mobilných zariadení. Široké spektrum aplikácií bežne využíva tieto kamery, či už na analýzu dopravy, detekciu rôznych objektov, v biometrii, medicíne, pri sledovaní podozrivých osôb alebo v hernom priemysle.

Už od počiatkov vzniku počítača sa s ním ľudia pokúšali komunikovať inak ako prostredníctvom klávesnice, myši, či neskôr rôznych ovládačov. Vynájdením Kinectu a jemu podobných senzorov nastala revolúcia najmä v hernom priemysle. Pomocou gest tela môžu ľudia ovládať rôzne aplikácie, a to bez použitia ovládačov. Ovládanie robotov, prehrávačov, televízie gestami rúk – nielen to umožňuje dnešné počítačové videnie.

Výsledná aplikácia si kladie za cieľ detekovať ruku užívateľa nad interaktívnym stolom a následne vykresliť pozíciu tejto ruky voči stolu.

Táto práca má za úlohu opísať jednotlivé postupy pri rozpoznávaní rúk a práci s interaktívnym stolom. V ďalšej kapitole (2) sú predstavené rôzne prístupy k tejto problematike, informácie o nástrojoch a zariadeniach, ako aj algoritmoch, ktoré boli použité pri vývoji výslednej aplikácie. Potom je všeobecne opísaný návrh riešenia (3), rozdelený do podproblémov a ich riešení. Samotná realizácia (4) je už venovaná konkrétnemu riešeniu týchto podproblémov. Jednak opisuje použité nástroje a techniky, ako aj implementáciu jednotlivých častí. Záverom sa venuje opisom testov a výslednému vyhodnoteniu produktu.

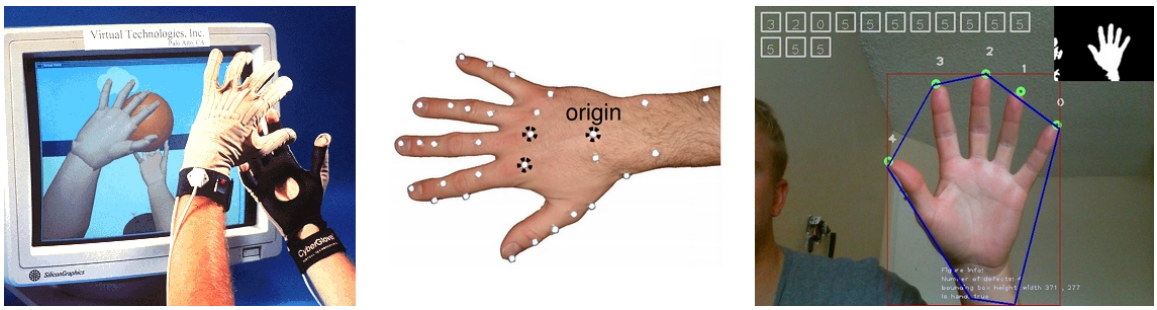
Kapitola 2

Teória

Táto kapitola sa venuje teoretickým znalostiam potrebným k pochopeniu jednotlivých pojmov, bez ktorých by sme sa v tejto práci nezaobišli. Zozačiatku opisuje existujúce riešenia nielen z historického, ale aj súčasného hľadiska. Ďalej sú v nej opísané knižnice, nástroje a algoritmy, relevantné pre túto prácu.

2.1 Existujúce riešenia

Existuje množstvo rôznych prístupov pre rozpoznanie gest rúk, pričom každý z nich má svoje pre a proti. Ako literatúra [20] uvádza, medzi tie najstaršie sa radí „káblová“ technológia, kedy užívateľ musel byť doslova spojený s počítačom pomocou káblov – jeho pohyb bol tak obmedzený dĺžkou káblov. Elektronické rukavice patrili tiež k tomuto systému. Jednalo sa o rukavice vybavené senzormi, ktoré poskytovali informácie o pozícii ruky a orientácii prstov. Tieto, taktiež nazývané dátové rukavice, síce poskytovali dobré výsledky, no boli dosť nákladné pre použitie v bežných aplikáciách.



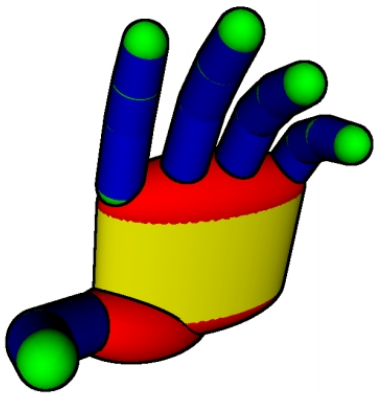
Obrázek 2.1: Jednotlivé metódy – zľava: dátové rukavice[1], optické senzory[2], spracovanie obrazu[3]

Neskôr ich nahradili optické senzory, rozmiestnené na ruke, ktoré vysielali infračervené svetlo. Toto svetlo sa odrážalo od obrazovky a poskytlo tak informácie o pozícii ruky a končekoch prstov. Tieto systémy tiež poskytovali dobré výsledky, ale vyžadovali komplexnú

[1] prebraté z `http://bit.ly/1Tgavfh`
[2] prebraté z `http://bit.ly/1sfRsar`
[3] prebraté z `http://bit.ly/1mOrrMu`

konfiguráciu. Techniky založené na spracovaní obrazu sa stali ďalším následníkom. Na základe prvkov ako textúr, farieb či tvarov dokázali rozpoznať gestá rúk, no výsledky sa odlišovali, pretože existuje množstvo odtieňov a textúr kože a tiež pod rôznym svetlom sa mení farba textúry, čo vedie k zhoršeniu požadovanej detekcie.

Nástupom nových technológií, ako napr. stereo hĺbkových senzorov, ktoré kombinujú výhody vyššie uvedených metód (farba + hĺbka), sa detekcia značne zlepšila. Je možné teda rozpoznať tvary rôznych farieb, a to bez obmedzovania sa na voľnosť pohybu, či používania rôznych pomôcok. Navyše, oproti tradičným 3D skenrom sú rýchlejšie a cenovo dostupnejšie [16]. Aj pri senzoroch tohto typu existujú rôzne metódy rozpoznávania rúk. Jedny predpokladajú existenciu telo v snímanej scéne, z ktorého potom určia polohu ruky [15, 21], čo je však nevhodné pre prípad, kedy by sa v obraze vyskytovala len ruka. Dalším prístupom je vnímanie ruky z anatomického hľadiska – zohľadňujú sa obmedzenia pohybu prstov [17].



Obrázek 2.2: Ruka vnímaná ako anatomický model[4]

2.2 Kinect

Kinect, pôvodne navrhnutý pre Xbox 360 a Xbox One herné konzoly, je senzor vyvinutý firmou Microsoft. V súčasnosti existujú dva druhy Kinectu, a to Kinect v1 a Kinect v2. Druhá verzia sa oproti tej prvej líši hlavne vo väčšom rozlíšení, či už farebnej alebo hĺbkovej kamery, ale tiež v iných aspektoch.

Najdôležitejšou súčasťou Kinectu v2 je jeho hĺbková infračervená (IR) kamera. RGB kamera zachytáva farebnú informáciu s rozlíšením 1920x1080 pixelov, pričom IR kamera sa používa pre získavanie hĺbkovych máp v reálnom čase a ponúka rozlíšenie 512x424 pixelov. Ďalšími parametrami sú frekvencia snímkov až do 30 Hz a zorné pole v rozsahu 70 stupňov horizontálne a 60 stupňov vertikálne. Microsoft udáva operačnú vzdialenosť od 0,5 do 4,5 metra [16].

Pôvodným cieľom Kinectu bolo umožniť užívateľom Xboxu ovládať hry gestami tela a taktiež ich „oslobodiť“ o rôzne pomocné ovládače (napr. Wii), teda spríjemniť celkový

[4]prebraté z `http://bit.ly/1XdeJWb`

pocit z daného okamihu. Situácia sa však zmenila a dnes je možné využiť tento senzor na účely vývoja aplikácií.

Spoločnosť Microsoft vyvinula adaptér[5] pre Kinect v2, vďaka ktorému ho je možné pripojiť priamo k PC cez USB3 rozhranie. Oficiálnou aplikáciou pre použitie Kinectu na operačnom systéme Windows je Microsoft SDK 2.0, ktorej alternatívou je libfreenect[6], poskytujúci ovládače pre spustenie na systéme GNU/Linux.

2.3 ROS

Robot Operating System je kolekcia frameworkov pre vývoj aplikácií robotov a slúži hlavne na komunikáciu medzi rôznymi aplikáciami. Odporúčanou a plne podporovanou platformou je Ubuntu, avšak je možné použiť aj iné platformy, v ktorých je ale ROS len tzv. „experimentálny“, tj. môžu nastať neočakávané stavy [18, 3]. Bolo vydaných viacero distribúcií ROSu[7] a wiki stránky poskytujú množstvo tutoriálov pre začínajúcich, ale aj pokročilých[8], ktoré sú písané pre jazyky C++ (roscpp) a Python (rospy).

Hlavnou časťou ROSu je komunikácia prostredníctvom *Topics*, ktoré umožňujú posielanie správ medzi *Nodes*. Jedná sa o komunikáciu typu *publish/subscribe*, pričom uzly[9], ktoré majú záujem o dáta sa nazývajú *Subscribers* a tie, ktoré dáta generujú (a odosielajú na daný *Topic*) sa nazývajú *Publishers* [4].

Kolekcia nástrojov a knižníc *IAI Kinect2* slúži ako rozhranie pre ROS a Kinect v2. Obsahuje *Kinect2 Bridge*, ktorý „premosťuje“ knižnicu libfreenect a ROS a ponúka tri *Topics* – HD (`/kinect2/hd/`, 1920x1080), QHD (`/kinect2/qhd/`, 960x540) a SD (`/kinect2/sd/`, 512x424), na ktoré Kinect publikuje obrazové dáta. Stránky [2] uvádzajú, že dáta publikované posledným *Topic* (SD) poskytuje senzor sám, tj. nie je nutná dodatočná kalibrácia. Rozlíšenie u SD sa zhoduje s rozlíšením IR kamery Kinectu.

2.3.1 ROSBAG

Balíček `rosbag` obsahuje nástroje, slúžiace hlavne pre záznam a prehrávanie na ROS *Topics* z/do súborov `.bag`. Pomocou nástroja `record` je teda možné odoberať[10] všetky správy publikované na danom *Topic* a tie zaznamenať do BAG súborov, spolu s informáciou o čase. Nástroj `play` číta obsah týchto súborov a umožňuje ich prehratie podľa času [6]. Následne je možná vizualizácia v nástroji `rviz`, nastavením príslušného *Topic*, z ktorého sa bude prehrávať.

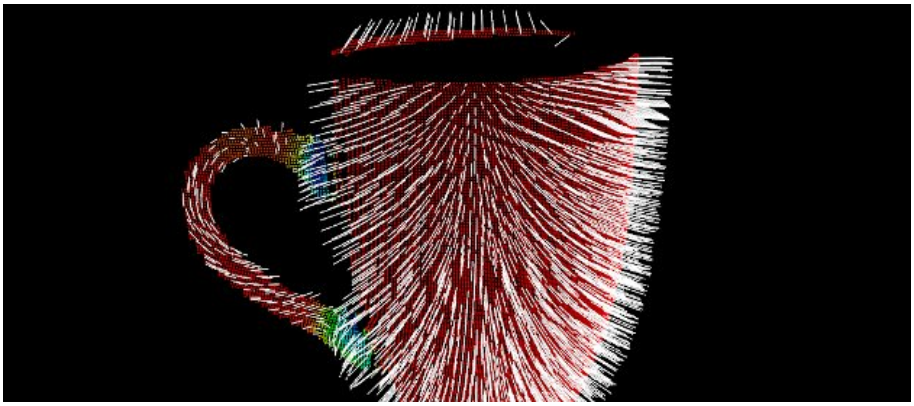
2.4 PCL

Point Cloud Library je plne šablónová, moderná C++ knižnica pre 3D spracovanie *pointcloudov*[11]. Zahŕňa množstvo funkcií ako: filtrácia, segmentácia, registrácia, transformácia, vizualizácia atď. Oficiálne stránky ponúkajú škálu ukážok a návodov[12] pre detekciu objektov, segmentáciu povrchov, orezanie *pointcloudov* a pod.

[5] `http://bit.ly/1uMsmuW`
[6] `https://github.com/OpenKinect/libfreenect2`
[7] `http://wiki.ros.org/Distributions`
[8] `http://wiki.ros.org/ROS/Tutorials#Core_ROS_Tutorials`
[9] Nodes, p. `http://wiki.ros.org/Nodes`
[10] z angl. *subscribe*
[11] množina bodov v 3D priestore, tiež mračno bodov
[12] `http://pointclouds.org/documentation/tutorials/`

Jednou z dôležitých tried, ktoré knižnica poskytuje, je komplexný nástroj PCLVisualizer, v ktorom je možné si jednotlivé *pointcloudy* jednoducho zobraziť, pridávať a vymazávať rôzne primitíva, útvary, súradnicové systémy atď.

PCL ďalej obsahuje štruktúry určené pre použitie so systémom ROS[13], taktiež rôzne metódy na konverziu *pointcloudov* a komunikáciu s ROS. Knižnica je zatiaľ plne implementovaná len v jazyku C++, no existuje napr. *python-pcl*[14], ktorá však poskytuje len obmedzené množstvo knižníc PCL.



Obrázek 2.3: Ukážka z nástroja PCLVisualizer[15]

2.5 Eigen

Eigen je open-source C++ knižnica, ktorá disponuje šablónami a funkciami pre lineárnu algebru, vektorové a maticové operácie. Podporuje matice všetkých veľkostí a všetky štandardné numerické typy. Túto knižnicu využíva množstvo projektov, vrátane ROS a PCL, ktoré ju používa ako matematický backend [14].

2.6 OpenCV

Knižnica OpenCV je vytvorená pre rôzne programovacie jazyky ako napr. C++, Java, Python a pracuje pod rôznymi operačnými systémami (Linux, Windows, Android atď.). Je napísaná v C++ pod BSD licenciou a obsahuje množstvo algoritmov, určených pre počítačové videnie či strojové učenie, pričom sa hlavne zameriava na chod aplikácií v reálnom čase [11]. Zároveň umožňuje vykresľovanie rôznych primitív na obrazovku.

2.7 Filtrácia

Pointcloudy získané zo senzorov, podobným Kinectu, často vyžadujú určitý druh filtrácie. Väčšinou je to kvôli nepresnosti senzoru, ale môže sa jednať aj o konkretizáciu požadovanej oblasti, s ktorou sa následne pracuje. Po redukcii bodov *pointcloudu* nie je nutné spracovávať

[13] `http://wiki.ros.org/pcl/Overview`
[14] `https://github.com/strawlab/python-pcl`
[15] prebraté z `http://pointclouds.org/documentation/overview/visualization.php`

také veľké množstvo bodov ako pred ňou. Niekedy však môže ísť o opačný problém – ak je napr. potrebné vyhladiť hrany *pointcloudu*, resp. doplniť určité body, aby rysy objektu boli zreteľnejšie.

2.7.1 *Passthrough* filter

Passthrough filter funguje na princípe odstránenia bodov, ktoré neležia v danom intervale. Tento filter dokáže odstrániť nežiadúce body, no problémom je, že pracuje len podľa danej osi, tj. napr. objekt, ktorý je potrebné takýmto spôsobom orezať, by nemal byť otočený voči počiatočnému súradnicovému systému. [8]

2.7.2 Vzorkovanie

Pri vzorkovaní dochádza ku zmene počtu bodov *pointcloudu*, pričom sa môže jednať buď o redukciu (podvzorkovanie) alebo pridávanie (nadvzorkovanie) bodov.

Hlavným cieľom podvzorkovania je redukcia bodov *pointcloudu*. Deje sa pomocou voxelizácie, kedy je *pointcloud* rozdelený na niekoľko kociek o zadanej veľkosti[16] a následne dochádza ku spracovaniu všetkých bodov v každej kocke, takže zostane len jeden [8].

Pri nadvzorkovaní sú pridané body *pointcloudu* čo umožňuje napr. „odhad“ určitých bodov potrebných pre získanie pôvodného povrchu objektu [8]. V tejto práci nadvzorkovanie nebolo použité, preto ani nebude ďalej opísane.

2.7.3 Redukcia šumu

Ďalším typom filtrácie je redukcia šumu, kedy dochádza k odstráneniu nepotrebných bodov, nazývaných *outliery*. Prítomnosť takýchto bodov je spôsobená nepresnosťou senzoru a môže byť príčinou chýb vznikajúcich pri výpočtoch. Existujú rôzne druhy filtrov tohto typu, ako napr. filtre založené na štatistickej analýze vzdialeností medzi susednými bodmi či filtre založené na vzdialenosti od daného bodu (*radius-based*). [8]

2.8 *k*-d stromy

k-rozmerné stromy[17] sú dátové štruktúry, slúžiace pre uloženie bodov a ich organizáciu v *k*-rozmernom priestore. Sú špeciálnym prípadom binárnych stromov, v ktorých každý uzol predstavuje *k*-rozmerný bod. Každý nelistový uzol rozdeľuje priestor na dve časti – polroviny. Body naľavo od polroviny sú reprezentované ľavým podstromom a body napravo pravým podstromom.

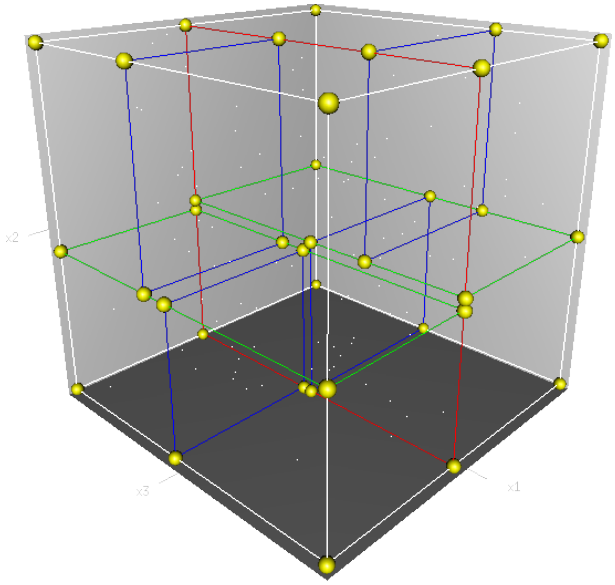
Každý uzol je spojený s jednou z *k*-dimenzií a smer rozdelenia je určený rovinou kolmou na os tejto dimenzie. Napr. pre os x: všetky body v podstrome, s hodnotou x väčšou ako uzol, budú patriť do pravého podstromu, zvyšné budú patriť do ľavého podstromu. [7]

(Trojrozmerné) *k*-d stromy (obr. 2.4) sú teda vhodným kandidátom pre reprezentáciu rozloženia bodov v *pointcloude* a pre vyhľadanie najbližších susedov jednotlivých bodov.

2.8.1 Hľadanie najbližšieho suseda

Vyhľadať najbližšie susedné body je potrebné z hľadiska aspektov, opísaných v ďalších kapitolách. Strom je pre túto operáciu ideálnou štruktúrou, vďaka rýchlej eliminácii veľkých

[16] tiež označovanej ako veľkosť listu
[17] z angl. *k-dimensional trees*



Obrázek 2.4: Trojrozmerný k-d strom – červená rovina rozdeľuje celú bunku (biela) na dve podbunky, z ktorej každá je rozdelená zelenými rovinami. Nakoniec sú všetky štyri rozdelené modrými rovinami. Keďže už nedochádza k ďalšiemu deleniu, výsledných osem buniek je listových.[18]

častí prehľadávaného priestoru. Algoritmus 1 sa zameriava na nájdenie bodu v strome, ktorý je najbližší k danému bodu. [7]

2.9 Reprezentácie príznakov

Pri detekcii objektov je nutné nájsť spôsob, ako od seba odlíšiť dva objekty. V priestore môžu byť body – z ktorých sú objekty zložené – definované ich súradnicami (x, y, z), prípadne farbou. To však nezaručuje ich unikátnosť, a preto je potrebné ich opísať tak, aby boli odlíšiteľné, resp. rozpoznateľné [22].

Príznak bodu je charakteristika, ktorá pomáha v jeho odlíšení od ostatných bodov. Normály sú príkladom veľmi jednoduchých príznakov, pretože zakódovávajú informáciu o okolí bodu. Pri výpočte sa počíta so susednými bodmi, ktoré udávajú, aký je okolitý povrch. Na to, aby bol príznak optimálny, musí spĺňať nasledujúce podmienky:

- ***robustnosť voči transformáciám*** – rigidné transformácie (nemenia vzdialenosť medzi bodmi) ako translácie či rotácie nemôžu ovplyvniť príznak
- ***robustnosť voči šumu*** – jemný šum by nemal mať veľký vplyv na výpočet príznaku
- ***nemennosť voči rozlíšeniu*** – po zmene vzorkovacej hustoty (napr. podvzorkovaní) by výsledok mal byť identický alebo podobný

K týmto účelom slúžia deskriptory, ktoré sú komplexnejšie a dokážu podrobnejšie reprezentovať geometriu povrchu objektov. PCL ponúka veľké množstvo rôznych deskriptorov.

[18] prebraté z `https://en.wikipedia.org/wiki/K-d_tree`

Algoritmus 1 Hľadanie najbližšieho suseda [19]

```
1:  procedure NNS(q: point, n: node, p: ref point, w: ref distance)
2:      if n.left= n.right=NULL  {leaf case}
3:          w' := ||q − n.point||;
4:          if w' < w
5:              w := w';
6:              p := n.point;
7:      else
8:          if q(n.axis) ≤ n.value
9:              search_first :=left;
10:         else
11:             search_first :=right;
12:         if search_first =left
13:             if q(n.axis) − w ≤ n.value
14:                 nns(q, n.left, p, w);
15:             if q(n.axis) + w > n.value
16:                 nns(q, n.right, p, w);
17:         else
18:             if q(n.axis) + w > n.value
19:                 nns(q, n.right, p, w);
20:             if q(n.axis) − w ≤ n.value
21:                 nns(q, n.left, p, w);
22: Počiatočné volanie: NNS(q, root, p, infinity)
```

Niektoré pracujú s rozdielmi medzi uhlami normál bodu a jeho susedných bodov, iné používajú vzdialenosť medzi bodmi. Každé deskriptory majú svoje pre a proti, každé sú vhodné pre určité prípady. [8, 22]

2.9.1 Lokálne deskriptory

Lokálne deskriptory sú vypočítané pre jednotlivé vstupné body, opisujú lokálnu geometriu v okolí daného bodu. Nepočítajú s objektom ako s celkom, ale len s jeho časťami. Zvyčajne je na programátorovi, aby zvolil, ktoré body chce použiť pre výpočet – kľúčové body. Tieto body je možné potom určiť buď výberom všetkých bodov po podvzorkovaní, alebo pomocou tzv. detektorov. [8]

2.9.2 Globálne deskriptory

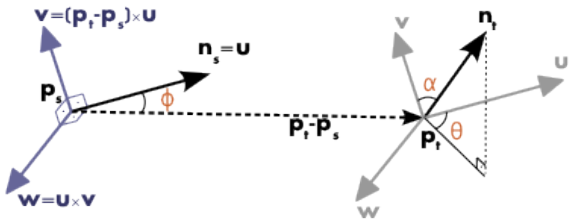
Globálne deskriptory kódujú geometriu objektu ako celok. Analogicky k lokálnym deskriptorom, nie sú počítané zvlášť pre jednotlivé body ale pre celý zhluk, ktorý predstavuje objekt. Z tohto dôvodu je potrebné najskôr vykonať segmentáciu pre získanie možných kandidátov. [8]

2.9.3 FPFH deskriptory

Deskriptory Point Feature Histogram sú založené na geometrii povrchu. Reprezentujú relatívnu orientáciu normál, rovnako ako vzdialenosti medzi pármi bodov [10]. Pre každý bod

sú určené jeho susedné body a analyzované rozdiely mezdi smermi normál v okolí bodu. Z tohto dôvodu je dôležitý správny výpočet normál [8].

Najskôr algoritmus páruje všetky susedné body v okolí, pričom pre každý pár je vypočítaný súradnicový rámec[19] z ich normál. Pomocou tohto rámca môže byť rozdiel medzi normálami zakódovaný do 3 uhlových premenných (obr. 2.5). Ako už vyplýva z názvu, PFH pracuje s histogramom, preto sú tieto premenné uložené spolu so vzdialenosťou bodov v euklidovskom priestore a následne premietnuté do histogramu, keď už boli všetky páry spočítané. Výsledný deskriptor je potom konkatenáciou histogramov každej premennej. [8]



Obrázek 2.5: Súradnicový rámec pre pár bodov – rozdiel normál daný troma uhlami[20]

PFH deskriptory vykazujú presné výsledky, avšak sú príliš výpočtovo náročné – pre *pointcloud* zložený z n bodov je zložitosť $O(nk^2)$, kde k udáva počet susedov pre každý bod v *pointcloude* [22]. Pre aplikácie bežiace v reálnom čase sú tieto deskriptory nevhodné. Z tohto dôvodu boli vytvorené FPFH (Fast PFH) deskriptory (obr. 2.6), ktoré sú odvodené z PFH deskriptorov. FPFH redukujú výpočetnú zložitosť algoritmu na $O(nk)$, pričom si zachovávajú rovnakú presnosť ako PFH. Je to kvôli využitiu len priamych spojení medzi bodom a jeho susedmi, čím sa ostatné spojenia medzi susedmi odstránia. Dôsledkom tohto je výsledný histogram označovaný ako SPFH (Simplified PFH). Na vyváženie straty zvyšných spojení je vykonaný ďalší krok po výpočte všetkých histogramov: SPF histogramy susedov daného bodu sú spojené s jeho histogramom a váhované podľa vzdialenosti. To má za následok získanie informácie o povrchu pre body dvojnásobne vzdialené než použitý polomer. Nakoniec sú tri histogramy konkatenované pre vytvorenie výsledného deskriptoru. [8]

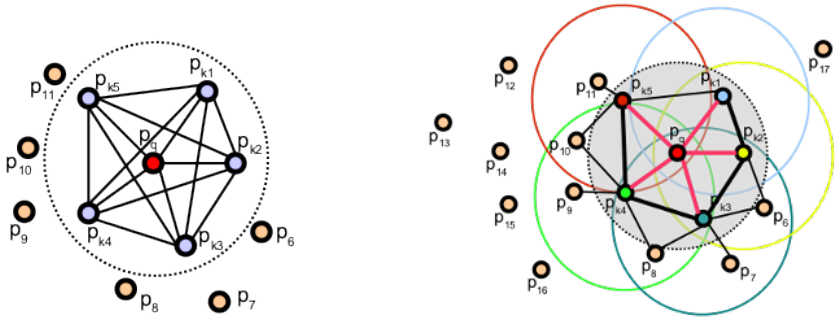
2.10 RANSAC

Random Sample Consensus (zhoda náhodných vzoriek) je iteračná metóda pre výpočet parametrov matematického modelu z množiny dát obsahujúcich *outliery*. Tiež môže byť interpretovaná ako metóda detekcie *outlierov*. [5]

Pri prirovnávaní určitých modelov k danému *pointcloudu* je RANSAC vhodným kandidátom, vďaka jeho jednoduchej implementácii a robustnosti. Pracuje na princípe výberu náhodných vzoriek a ich verifikácii voči modelom [12]. Modelmi sa myslia útvary, jednoduché na výpočet ako rovina, valec, guľa, ihlan, kužeľ a pod. Pomocou RANSACu je teda možné detekovať rovinu v priestore, a to použitím troch nerovnakých bodov, ktoré neležia na jednej priamke, čo definuje algoritmus 2.

Posledným krokom je vyhodnotenie „skóre“ podobnosti s modelom. Každá množina

[19] tri pravouhlé súradnicové osi
[20] prebraté z `http://pointclouds.org/documentation/tutorials/pfh_estimation.php`



a) b)

Obrázek 2.6: Pri PFH deskriptoroch (a)[21] sú body navzájom pospájané, pričom pri FPFH (b)[22] je každý bod spojený len s jeho priamym susedom.

Algoritmus 2 RANSAC pre detekciu roviny v priestore [22]

1: náhodne vyber tri nerovnaké body, ktoré neležia na jednej priamke $\{p_i, p_j, p_k\}$ z *pointcloudu* P;
2: vypočítaj koeficienty modelu z troch bodov $(ax + by + cz + d = 0)$;
3: vypočítaj vzdialenosti všetkých bodov $p \in P$ od modelu roviny (a, b, c, d);
4: urči počet bodov $p^* \in P$, ktorých vzdialenosť d od roviny patrí do intervalu $0 \leq |d| \leq |d_t|$, kde d_t predstavuje prah definovaný užívateľom;

bodov p^* je uložená a celý algoritmus sa opakuje po žiadaný počet iterácií. Po ukončení algoritmu je vybraná tá množina, ktorá obsahuje najväčší počet bodov [22]. Ako literatúra [13] uvádza, čím menšie množstvo *inlierov*[23], tým väčší počet náhodných vzoriek je potrebných pre spoľahlivosť algoritmu, čo má za následok dlhšiu dobu behu.

2.11 Obálka

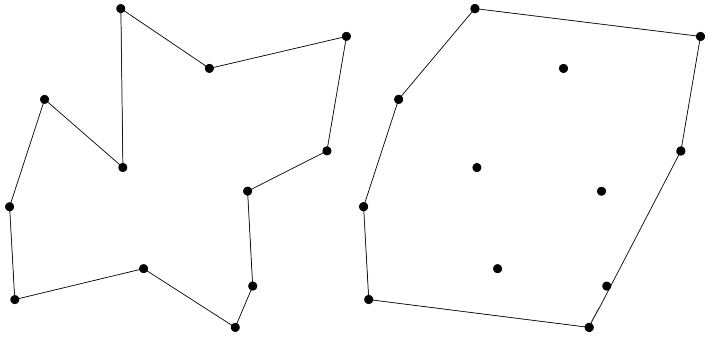
Obálka vytvára hranicu okolo danej množiny bodov, nad ktorou je zostrojená. Existujú dva typy, ktoré je možné vytvoriť, a to:

- ***konvexná obálka*** – obklopuje všetky body danej množiny takým spôsobom, že všetky body ležia vnútri tejto obálky (p. obr. 2.7b)
- ***konkávna obálka*** – väčšinou zaberá menší priestor než konvexná (obr. 2.7a) [8]

2.12 Euklidovské zhlukovanie

Metóda zhlukovania rozdeľuje *pointcloud* na menšie časti, takže celkový čas jeho spracovania je značne redukovaný. Jednoduchou metódou by bolo použitie 3D rozdelenia priestoru s bunkami pevnej šírky, avšak tento prístup je použiteľný len v prípade rovnomerného rozdelenia priestoru. Väčšinou sa však jedná o situácie, kedy zhluky môžu nadobúdať rôzne veľkosti, a preto je nutné použiť zložitejší algoritmus.

[21] prebraté z `http://pointclouds.org/documentation/tutorials/pfh_estimation.php`
[22] prebraté z `http://pointclouds.org/documentation/tutorials/fpfh_estimation.php`
[23] *inlier* – vzorka, ktorá vyhovuje zadanému modelu



Obrázek 2.7: Typy 2D obálok: konkávna (a) a konvexná (b) [24]

Ak je daný *pointcloud* P obsahujúci rovinu (napr. stôl), na ktorej ležia objekty (zhluky bodov), je potrebné odlíšiť jeden zhluk od druhého. Nech $O_i = \{p_i \in P\}$ je zhluk odlišný od $O_j = \{p_j \in P\}$, ak platí:

$$\min||p_i - p_j||_2 \geq d_{th}, \tag{2.1}$$

kde d_{th} je prah maximálnej vzdialenosti. Rovnica vyššie (2.1) udáva, že ak je minimálna vzdialenosť medzi dvomi odlišnými množinami bodov $p_i, p_j \in P$ väčšia ako daná hodnota vzdialenosti, potom p_i patrí zhluku O_i a p_j patrí zhluku O_j. Táto minimálna vzdialenosť môže byť určená využitím algoritmu vyhľadania najbližšieho suseda pomocou k-d stromu. Zhlukovanie je teda možné opísať nasledujúcim algoritmom [22]:

Algoritmus 3 Euklidovské zhlukovanie [22]

1: vytvor k-d strom reprezentujúci vstupný *pointcloud* P;
2: nastav prázdny zoznam zhlukov C a frontu bodov, ktoré sa budú kontrolovať Q;
3: **foreach** bod $p_i \in P$
4: pridaj p_i do aktuálnej fronty Q;
5: **foreach** bod $p_i \in Q$
6: hľadaj množinu P_k^i susedných bodov bodu p_i v guli s polomerom $r < d_{th}$;
7: **foreach** susedný bod $p_k^i \in P_k^i$
8: **if not** bol už bod spracovaný *// ak bod ešte nebol spracovaný*
9: pridaj ho do Q;
10: keď už bol zoznam všetkých bodov v Q spracovaný, pridaj Q do zoznamu zhlukov C a resetuj Q na prázdny zoznam
11: algoritmus končí, keď všetky body $p_i \in P$ boli spracované a sú teraz súčasťou zoznamu zhlukov C

2.13 SAC-IA

Sample Consensus Initial Alignment je metóda, slúžiaca na zarovnanie viacerých obrazov (*pointcloudov*) a následné vyhodnotenie ich podobnosti. Je založená na princípe zhody vzoriek. Jej predchodcom bola metóda Greedy Initial Alignment, ktorá bola síce robustná, avšak výpočetne náročná, z dôvodu kontroly všetkých zodpovedajúcich párov. Oproti tomu,

[24] `http://bit.ly/1V5ujmA`

SAC metóda vzorkuje veľké množstvo korešpondujúcich kandidátov, ktoré veľmi rýchlo hodnotí podľa nasledujúcich krokov:

1. Vyber s vzoriek bodov z P tak, aby vzdialenosti medzi jednotlivými pármi boli väčšie ako definovaná minimálna vzdialenosť d_{min}.
2. Pre každú vzorku bodov nájdi zoznam bodov v Q, ktorých histogramy sú podobné histogramom vzoriek bodov. Z týchto (bodov) vyber jeden náhodne, ktorý bude považovaný za zodpovedajúci.
3. Vypočítaj rigidnú transformáciu definovanú vzorkami bodov a ich zodpovedajúcimi bodmi a vypočítaj metrickú chybu pre *pointcloud*, ktorá vypočíta kvalitu transformácie. [23]

2.14 Transformácia bodov medzi súradnicovými systémami

Ak je bod v 3D priestore definovaný troma súradnicami (x, y, z) voči určitému súradnicovému systému, je niekedy potrebné získať jeho súradnice vzhľadom k inému súradnicovému systému. Príkladom môže byť prevod zo súradnicového systému kamery do súradnicového systému stola. K tomuto účelu je potrebné najskôr opísať transformačnú maticu.

2.14.1 Transformačná matica

Lineárne transformácie, ako translácia, rotácia a zmena mierky, možu byť reprezentované 4x4 maticou. Translácia je reprezentovaná prvými troma hodnotami v poslednom riadku, 3x3 matica nachádzajúca sa v ľavom hornom rohu predstavuje rotáciu, a zmena mierky je zakódovaná do prvých troch koeficientov, ležiacich na diagonále. [9]

Transformačnú maticu (obr. 2.8) je teda možné vidieť ako súradnicový systém (obr. 2.9), pričom jednotlivé osi sú reprezentované riadkami 3x3 matice umiestnenej vľavo hore v transformačnej matici. Treba poznamenať, že tieto osi sú jednotkové vektory. Posledný riadok udáva pozíciu tohto systému vzhľadom ku globálnemu súradnicovému systému.

$$\begin{bmatrix} c_{00} & c_{01} & c_{02} & c_{03} \\ c_{10} & c_{11} & c_{12} & c_{13} \\ c_{20} & c_{21} & c_{22} & c_{23} \\ c_{30} & c_{31} & c_{32} & c_{33} \end{bmatrix} \begin{matrix} \rightarrow \text{ os } x \\ \rightarrow \text{ os } y \\ \rightarrow \text{ os } z \\ \rightarrow \text{ translácia} \end{matrix}$$

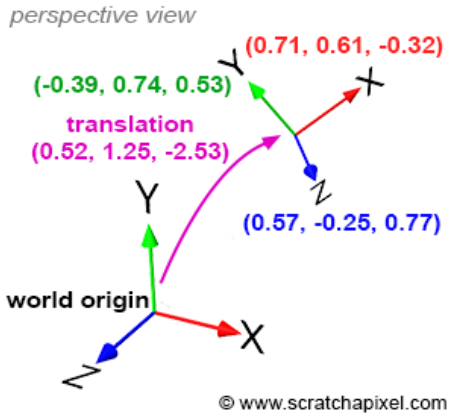
Obrázek 2.8: Transformačná matica[25]

2.14.2 Globálny a lokálny súradnicový systém

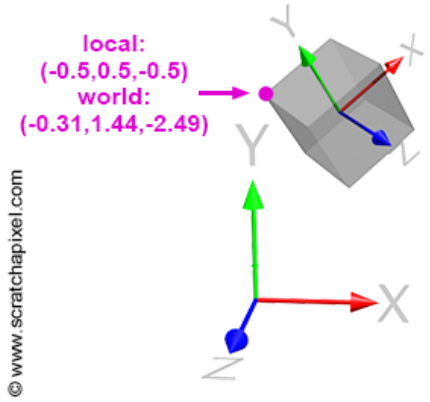
Globálny súradnicový systém je hlavným systémom v danom priestore/scéne a jeho počiatok má súradnice $[0, 0, 0]$. Súradnice všetkých bodov sú teda určované vzhľadom k tomuto systému. Pre zjednodušenie výpočtov je často potrebné definovať tieto body vzhľadom k lokálnemu súradnicovému systému.

Pohyb 3D objektu v scéne môže byť reprezentovaný 4x4 transformačnou maticou, ktorú je tiež možné vidieť ako lokálny súradnicový systém tohto objektu. V prípade na obrázku 2.10 sú globálne súradnice bodu iné ako lokálne, pričom sa jedná o ten istý bod. [9]

[25] prebraté z `http://bit.ly/1TSpZVi`



Obrázek 2.9: Translácia súradnicového systému a jednotlivé osi sú definované vzhľadom ku globálnemu systému[26]



Obrázek 2.10: Definovaný bod má iné globálne súradnice ako lokálne, určené vzhľadom k objektu[26]

Pre výpočet globálnych súradníc bodu je potrebné vynásobiť jeho pôvodné súradnice lokálno-globálnou maticou (definuje súradnicový systém vzhľadom ku globálnemu súradnicovému systému):

$$P_{globálny} = P_{lokálny} * M, \tag{2.2}$$

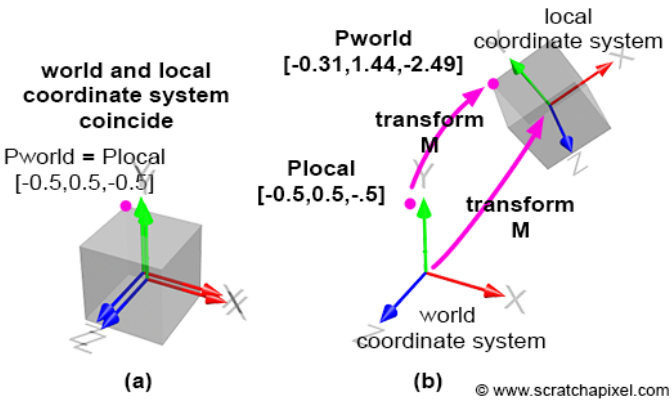
kde $P_{globálny}$ predstavuje globálne súradnice bodu, $P_{lokálny}$ lokálne súradnice bodu a M lokálno-globálnu transformačnú maticu.

Analogicky pri prevode z globálneho do lokálneho systému je potrebné násobiť globálno-lokálnou maticou:

$$P_{lokálny} = P_{globálny} * M^{-1}, \tag{2.3}$$

kde $P_{lokálny}$ predstavuje lokálne súradnice bodu, $P_{globálny}$ globálne súradnice bodu a M^{-1} maticu, inverznú k matici M, teda globálno-lokálnu transformačnú maticu.

[26] prebraté z `http://bit.ly/1TSpZVi`
[27] prebraté z `http://bit.ly/1TSpZVi`



Obrázek 2.11: Transformácia medzi dvoma súradnicovými systémami. V prípade a) je lokálny a globálny súradnicový systém zhodný (bod P má rovnaké súradnice), v prípade b) sú systémy odlišné – odlišné súradnice bodu P.[27]

2.15 ROC

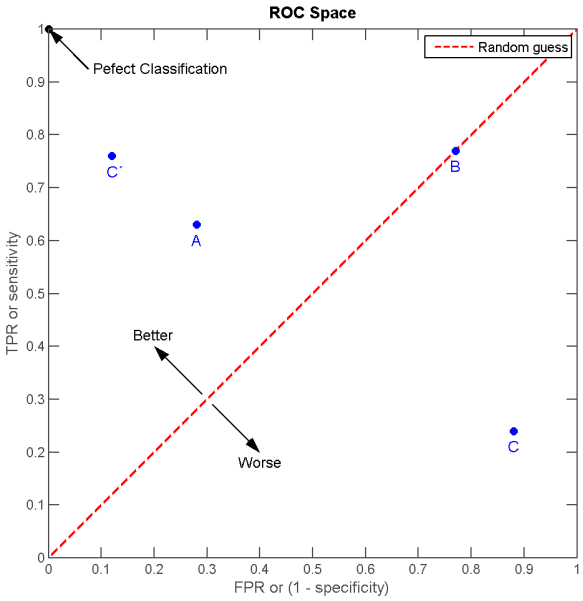
Receiver Operating Characteristic je krivka, ktorá zobrazuje výkonnosť binárneho klasifikátora. Základom je výpočet štyroch hodnôt, z ktorých je možné určiť viacero charakteristík klasifikátora. Ak sú jeho objekty označené buď ako pozitívne (p) alebo negatívne (n), potom existujú štyri možné hodnoty: [1]

- ***TP*** – *True Positive* – značí predpokladaný objekt p, ktorý bol označený ako p
- ***FP*** – *False Positive* – značí objekt n, ktorý bol označený ako p
- ***TN*** – *True Negative* – značí objekt n, ktorý bol označený ako n
- ***FN*** – *False Negative* – značí objekt p, ktorý bol označený ako n

Príkladom môže byť ROC krivka (obr. 2.12), ktorej osi x a y sú označené ako TPR a FPR. Zobrazené sú tiež korešpondujúce body (prahy), tvoriace túto krivku. TPR udáva, koľko bolo správne označených pozitívnych vzoriek. FPR zasa znamená počet nesprávne označených negatívnych vzoriek. Keďže TPR je ekvivalentom citlivosti a FPR je rovný 1–špecifickosť, sa graf ROC nazýva citlivosť vs. (1–špecifickosť) [1].

Najlepší výsledok je potom reprezentovaný bodom v ľavom hornom rohu $(0, 1)$, ktorý by znamenal 100% citlivosť a 100% špecifickosť. Tento bod je nazývaný „perfektná klasifikácia“[28]. Ak sú body v grafe rozložené pozdĺž diagonály, jedná sa o *náhodnú klasifikáciu*[29]. Úspešnosť klasifikátora je daná hodnotou, nazývanou AUC, čo znamená *Area under the curve*, tj. obsah plochy pod ROC krivkou [1]. Ďalším pojmom je *ground truth*, čo znamená že binárne hodnoty klasifikátora zadáva užívateľ, teda on rozhodne čo je a čo nie je hľadaný objekt.

[28] z angl. *perfect classification*
[29] z angl. *random guess*
[30] `http://bit.ly/1In2v72`



Obrázek 2.12: ROC krivka: jednotkový graf zobrazujúci pomer FPR (x) ku TPR (y), ďalej diagonálu deliacu dobré výsledky od zlých a bod označujúci „perfektnú klasifikáciu“, ktorý sa nachádza v ľavom hornom rohu grafu $(0, 1)$.[30]

Kapitola 3

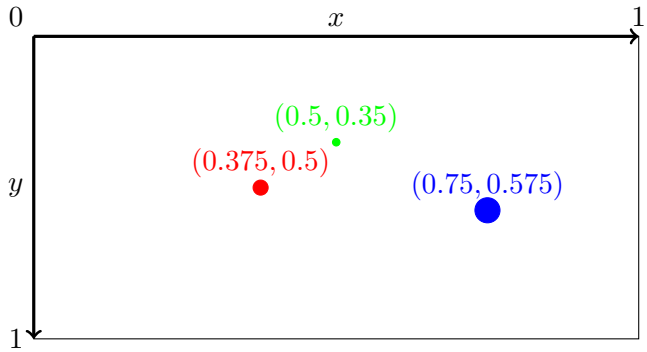
Návrh riešenia

Kapitola opisuje čo je cieľom výslednej aplikácie, jednotlivé požiadavky na aplikáciu a čiastkové kroky pri postupe riešenia vyjadrené vývojovým diagramom.

3.1 Cieľ aplikácie

Výsledná aplikácia by mala rozpoznať ruku/ruky užívateľa na interaktívnom stole, a následne zobraziť jej pozíciu voči stolu. Vstupom sú dáta získané zo senzoru Kinect, ktorý je umiestnený nad stolom, tj. sníma sa zhora. Zobrazený stôl (obr. 3.1) je reprezentovaný obdĺžnikom a detekovaná ruka/ruky kruhom, ktorého farba sa prípadne mení podľa daného gesta. Vzdialenosť ruky od plochy stola má vplyv na veľkosť kruhu – čím vyššie sa ruka nachádza nad stolom, tým vačší je kruh a naopak.

Súradnicový systém stola je jednotkový, tj. súradnice ruky, nachádzajúcej sa v oblasti nad plochou stola, nadobúdajú hodnoty v rozmedzí $\langle 0, 1 \rangle$. Počiatok tohto systému predstavuje ľavý horný roh stola (z pohľadu Kinectu), čo sa dá využiť napr. na prepočet rozlíšenia obrazovky, a následnú simuláciu kurzora myši. To už však nie je cieľom tejto aplikácie.

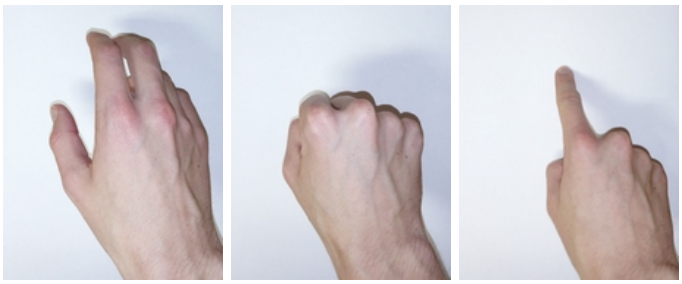


Obrázek 3.1: Návrh vykreslenia: body reprezentujú detekované ruky, ich veľkosť sa líši podľa vzdialenosti od plochy stola.

Pre detekciu rúk sú navrhované nasledovné gestá:

- ***Voľná ruka*** – ruka je v uvoľnenej pozícii, tak ako ukazuje obr. 3.2a. Vykresľuje sa červenou farbou.

- ***Päsť*** – gesto zovretej päste (obr. 3.2b). Zelená farba.
- ***Ukazujúci prst*** – ukazovák smerujúci k ploche stola (obr. 3.2c), ruka je jemne naklonená voči stolu. Modrá farba.



a) b) c)

Obrázek 3.2: Definované gestá

3.2 Návrh postupu práce

Pri návrhu postupu práce budem vychádzať z predošlej kapitoly (2). V tejto kapitole budú všeobecne rozobrané jednotlivé kroky (s využitím už opísaných algoritmov), ktoré sú potrebné k chodu aplikácie. Navrhovaný postup ukazuje vývojový diagram na obrázku 3.3.

Program je rozdelený na dve časti, a to offline a online. Offline vetva obsahuje dôležité kroky, ktoré je nutné vykonať ešte predtým ako program „pobeží“ online. Následne je spustená vetva online, v ktorej program iteruje postupným načítavaním a spracovávaním jednotlivých *pointcloudov* (sekvencie) až dokiaľ nedôjde ku koncu behu samotného programu, a to buď predčasným ukončením, alebo spracovaním všetkých pointcloudov na vstupe.

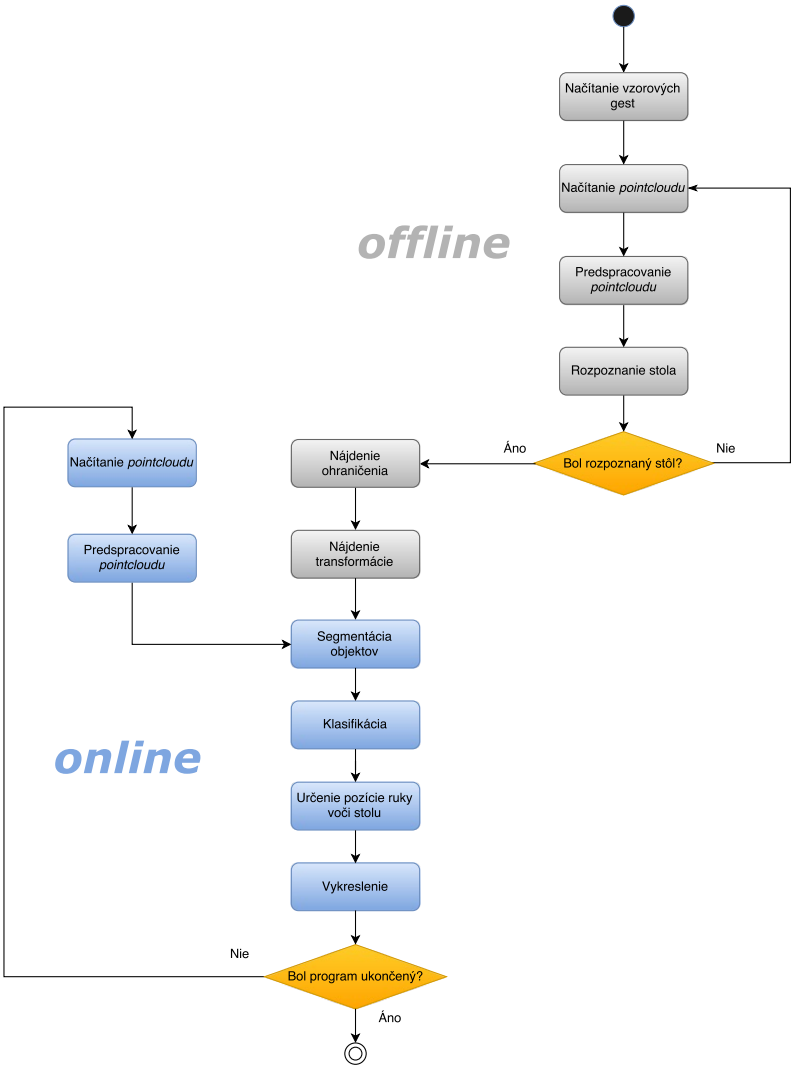
3.2.1 Výpočet deskriptorov vzorových rúk

Ešte predtým ako sa načíta sekvencia vstupných scén, začína aplikácia načítaním vzorových gest vo forme *pointcloudov*, a to len jedenkrát za celý beh programu. Pre každý vzor sú určené kľúčové body, z ktorých sú následne vypočítané deskriptory pomocou FPFH (kapitola 2.9.3). Tie budú neskôr využité pre porovnávanie a klasifikáciu. Ďalej je pre každý vzor definovaný kontrolný bod, ktorý reprezentuje konkrétne gesto a neskôr sa použije pre konečné vykreslenie.

3.2.2 Predspracovanie

Po načítaní vstupnej scény (*pointcloudu*), je potrebná fáza určitého predspracovania (offline aj online), kedy dochádza k „zahodeniu“ *outlierov*, čo uľahčuje ďalšiu prácu s *pointcloudom*. Nielenže dochádza k redukcii bodov (čím sa skráti doba behu aplikácie), ale taktiež je možné konkretizovať hľadanú oblasť a odľahčiť tak program o ďalšie spracovania.

V offline časti sa predspracovaním myslí hrubé orezanie *pointcloudu*, čo dopomáha k lepšej detekcii plochy stola (vo väčšine prípadov), keďže prítomných plôch v scéne môže byť viac.



Obrázek 3.3: Vývojový diagram postupu práce

Po prechode do časti online je predspracovanie konkretizované využitím už nájdených hraníc stola (opísané nižšie), čo môže mať teoreticky za následok ešte väčšie zvýšenie rýchlosti behu programu.

3.2.3 Rozpoznanie stola

Keď sú už odstránené jednotlivé *outliery*, môže program prejsť k detekcii plochy stola. Tá je realizovaná pomocou metódy RANSAC, ktorá nájde najväčšiu rovinu v predspracovanej scéne. To, ako bude stôl detekovaný, bude mať vplyv na ďalšie operácie v programe. V prípade, že nedošlo k rozpoznaniu, dochádza k načítaniu ďalšieho *pointcloudu* a program sa znova pokúsi o detekciu.

3.2.4 Nájdenie ohraničenia

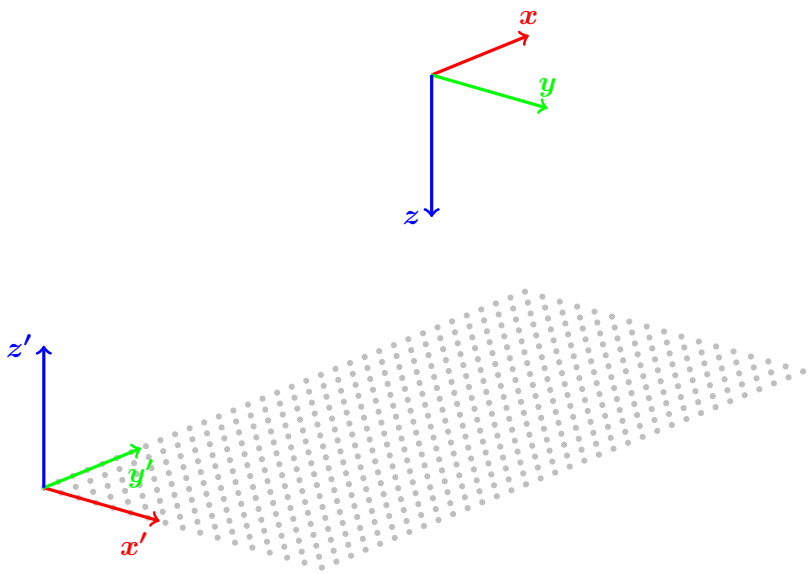
Predošlá operácia by mala vrátiť nájdenú plochu bez prípadných objektov, nachádzajúcich sa na nej. Kvôli spracovaniu ďalších načítaných *pointcloudov* (hlavne vo vetve online) je dôležité nájsť hranice stola a vymedziť oblasť, v ktorej bude program ďalej pracovať. Ideálne je teda použitie konvexnej obálky zostrojenej z bodov reprezentujúcich plochu stola, a určenie výšky nad touto obálkou, ktorá predstavuje hranicu pracovnej oblasti.

3.2.5 Nájdenie transformácie

Posledným krokom v offline vetve je nájdenie transformácie medzi súradnicovým systémom Kinectu a stola, čo je dôležitým prvkom z hľadiska určenia pozície objektov (nachádzajúcich sa nad stolom) voči stolu.

Podľa problematiky, opísanej v kapitole 2.14, program získa transformačnú maticu stola, konkrétne jeho rohu (obr. 3.4), od ktorého sa počítajú súradnice rúk (kontrolných bodov). S touto transformačnou maticou program ďalej pracuje v online vetve.

Rovnako ako načítanie vzorových gest či nájdenie ohraničenia, aj transformácia rohu stola je vypočítaná len jedenkrát, a to z dôvodu predpokladu nemennej pozície stola v sekvencii na vstupe. Ak sú všetky vyššie spomínané úkony splnené, môže aplikácia „prejsť“ do online vetvy.



Obrázek 3.4: Súradnicový systém Kinectu (x, y, z) a súradnicový systém rohu stola (x', y', z'). Šedé body reprezentujú *pointcloud* stola.

3.2.6 Segmentácia objektov

Pointcloud, s ktorým sa pracuje v offline vetve, môže obsahovať ruku. Preto program pokračuje v online vetve, konkrétne segmentáciou objektov na stole.

Po detekcii stola a následnej aplikácii konvexnej obálky by mala vymedzená pracovná oblasť obsahovať už len objekty, ktoré sa nachádzajú na stole. Použitím euklidovského zhlukovania sú tieto objekty rozdelené na jednotlivé zhluky (tzv. kandidátov), s ktorými

sa ďalej pracuje. Podobne ako pri *pointcloudoch*, reprezentujúcich vzorové gestá, aj tu sú pre každý zhluk určené kľúčové body, z ktorých sú vypočítané deskriptory opäť pomocou FPFH.

Ku segmentácii dochádza v každom ďalšom načítanom *pointcloude*, keďže objekty môžu meniť svoju polohu.

3.2.7 Klasifikácia

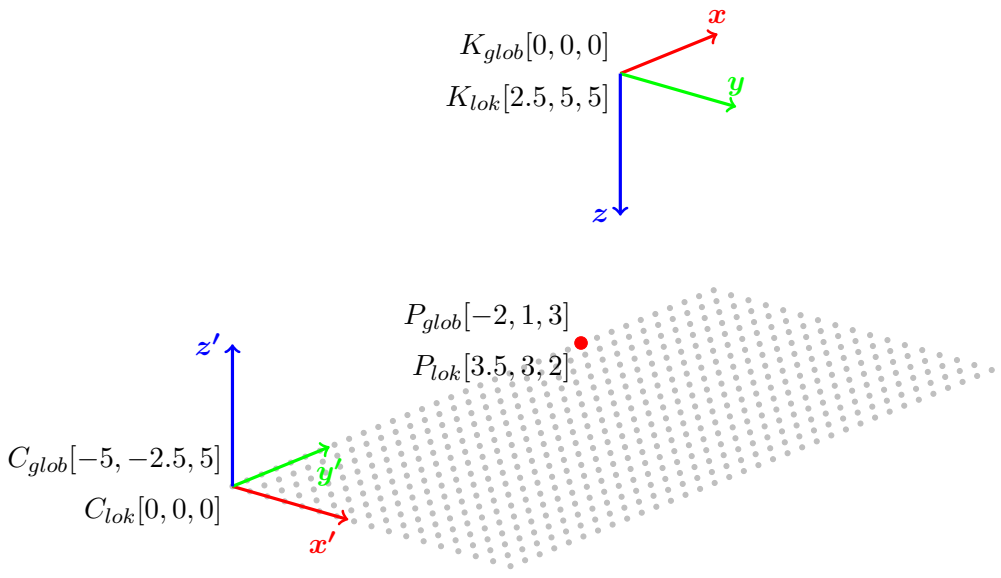
Po vykonaní predchádzajúcich krokov, môže program pristúpiť k jadru celej aplikácie – detekcii rúk. Pomocou registračnej metódy SAC-IA je každý kandidát porovnaný s každým vzorom. Klasifikátor potom vyhodnocuje skóre podobnosti deskriptorov. Vybrané gesto, ktoré získalo najlepšie skóre spomedzi všetkých, bude reprezentovať daného kandidáta. Vybraní sú len tí kandidáti, ktorých skóre dosiahlo určitú hranicu (prah), kedy je možné povedať, že sa jedná o ruku.

Klasifikátor ďalej určí transformáciu potrebnú pre zarovnanie vzoru na kandidáta. Na začiatku offline vetvy boli určené kontrolné body pre vzorové *poincloudy*, ktoré sa teraz transformujú spolu s nimi.

3.2.8 Určenie pozície ruky

Po úspešnej detekcii ruky je potrebné zistiť jej polohu voči ploche stola. V predchádzajúcich krokoch bola zistená transformačná matica rohu stola, ktorú je teraz možné využiť pre výpočet lokálnych súradníc kontrolného bodu.

Aplikáciou rovnice 2.3, s využitím získanej transformačnej matice, program získa súradnice bodu vzhľadom k rohu stola. Na obrázku 3.5 je znázornený príklad súradníc systémov – globálne súradnice, určené vzhľadom na Kinect, sú označené indexom *glob* a lokálne súradnice, podľa rohu stola, sú označené indexom *lok*.



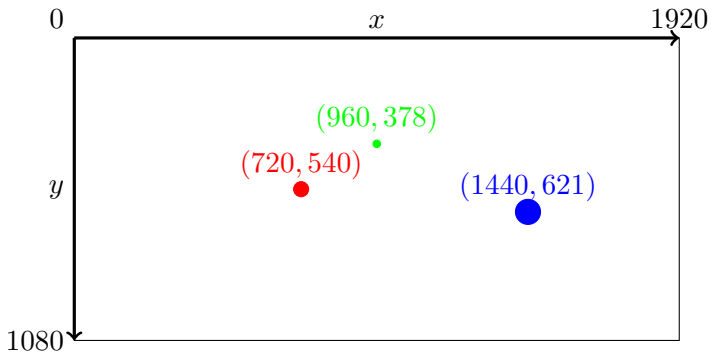
Obrázek 3.5: Kinect (K), predstavujúci globálny súradnicový systém (x, y, z) a roh stola (C) – lokálny systém (x', y', z'). Červený bod (P) predstavuje kontrolný bod ruky. Šedé body reprezentujú *pointcloud* stola.

3.2.9 Vykreslenie

Záverečnou fázou celej aplikácie je vykreslenie stola a jednotlivých bodov. Stôl je reprezentovaný obdĺžnikom, ktorého pomer strán sa zhoduje s reálnym pomerom strán stola. Oblasť vykresľovania bodov je tiež prispôsobená hraniciam obdĺžnika.

Ako už bolo spomenuté v kapitole 3.1, lokálne súradnice bodov sa pohybujú v rozmedzí $\langle 0, 1 \rangle$, čo sa dá prepočítať na rozlíšenie obrazovky prenásobením týchto súradníc šírkou, resp. výškou rozlíšenia. Os x súradnicového systému rohu stola, znázorneného na obrázku 3.4, rastie smerom nadol (z pohľadu Kinectu). Väčšinou sa však za počiatok obrazovky pokladá jej ľavý horný roh, pričom x rastie smerom doprava. Z tohto dôvodu je teda nutné súradnice x a y vymeniť.

Na nasledujúcom obrázku je znázornený príklad prepočtu súradníc kontrolných bodov z obr. 3.1 pri rozlíšení obrazovky 1920x1080 pixelov:



Obrázek 3.6: Ukážka prepočtu súradníc bodov pri rozlíšení 1920x1080: jednotkové súradnice sú vynásobené rozlíšením obrazovky $(x * 1920, y * 1080)$

Po vykonaní predchádzajúcich krokov sa pokračuje už len v online vetve, dokiaľ nie je aplikácia ukončená, alebo ak už bola spracovaná celá sekvencia *pointcloudov*.

Kapitola 4

Realizácia

V kapitole návrh riešenia (3) boli všeobecne opísané postupy, ktoré je potrebné konkretizovať s ohľadom na použité nástroje, knižnice ap. Najprv opisuje použitý interaktívny stôl, na ktorom bola vytvorená dátová sada. Ďalej sú v implementácii opísané jednotlivé časti programu a prolémy, ktoré sa pri práci objavili. Záverečná kapitola obsahuje kroky, vykonané pri testoch, a ich celkové vyhodnotenie.

4.1 Interaktívny stôl

Jednou z neoddeliteľných súčastí mojej aplikácie je využiť stôl ako určitú pracovnú plochu, s ktorou môže užívateľ interagovať, tj. pokladať na ňu rôzne objekty, ako aj ruky, ktoré má aplikácia rozpoznať. Taktiež musí užívateľovi poskytovať určitú spätnú väzbu – napr. vo forme projekcie obrazu priamo na plochu stola. Touto kombináciou je možné považovať stôl za interaktívny.

Stôl, disponujúci spomínanými vlastnosťami, sa nachádza v robotickom laboratóriu na FIT VUT (4.1a). Nad stolom (s ktorým je možné voľne pohybovať) sa nachádza Kinect v2, spolu s projektorom, pripevnený k hliníkovej konštrukcii (4.1b). Tento interaktívny stôl bol využitý na realizáciu a dosiahnutie cieľa aplikácie.

4.2 Dátová sada

Základom celej aplikácie je detekcia rúk užívateľa. Na rozpoznanie definovaných gest (obr. 3.2), bolo preto potrebné zostaviť dátovú sadu, ktorá pozostáva zo:

- ***Sady vzorov***, ktorá definuje vzory gest rúk.
- ***Testovacej sady***, skladajúcej sa z celých scén obsahujúcich rôzne predmety, no hlavne ruky v rôznych pozíciách, na ktoré sa aplikujú dané vzory.

Obe sady boli vytvorené pomocou Kinectu na interaktívnom stole. Na získanie dát z Kinectu bola použitá knižnica *IAI Kinect2* (kapitola 2.3), konkrétne *Topic* `/kinect2/sd/points` poskytujúci *pointcloudy* o rozlíšení 512x424. Tento *Topic* bol využitý kvôli tomu, že v rovnakom rozlíšení pracuje aj Kinect – dáta sú teda reálnejšieho charakteru. Je nutné poznamenať, že program pracuje s *pointcloudami*, ktorých body nesú informáciu len o ich polohe (nie o farbe – objasnené nižšie). Nasnímané scény boli uložené do BAG súborov pre jednoduchší prístup k nim a pre prehľadnosť. Kvôli kompatibilite s PCL

a individuálnemu prístupu v ďalších krokoch však bolo následne nutné BAG súbory skonvertovať do súborov s príponou `.pcd`[1], s ktorými už knižnica dokáže pracovať. Na konverziu bol použitý nástroj `bag_to_pcd` balíka `pcl_ros` systému ROS.

a) b)

Obrázek 4.1: Interaktívny stôl v robotickom laboratóriu (a). Detailný záber na projektor a Kinect (b).

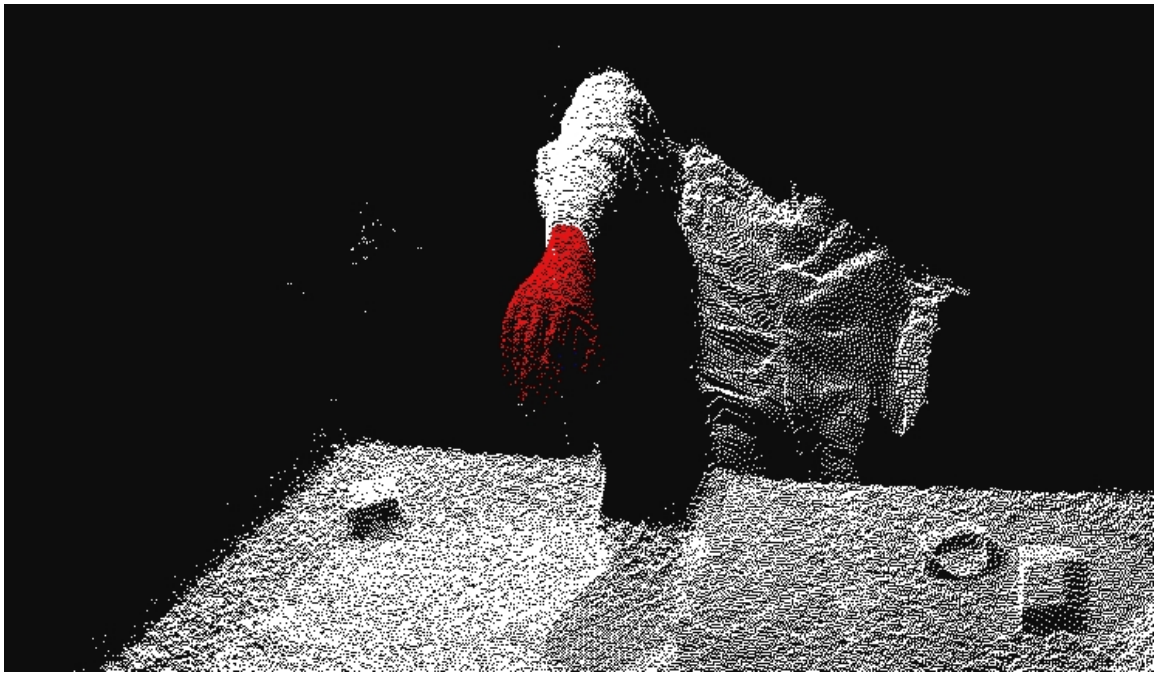
4.2.1 Príprava sady vzorov

Pôvodné *pointcloudy* obsahovali veľa zbytočných objektov, ktoré bolo najskôr potrebné odstrániť. Toto predspracovanie si vyžadovalo manuálny prístup k jednotlivým *pointcloudom* – zistenie kvality, orezanie, odstránenie šumu, prípadne celých *pointcloudov* (ktoré neboli vhodné ako vzory). Gestá rúk boli nasnímané z viacerých uhlov, pretože ruka môže byť rôzne orientovaná a tiež mať viacero podobných tvarov.

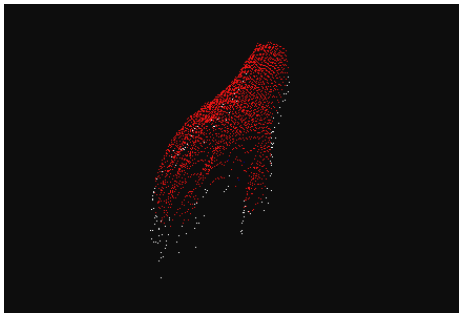
Kľúčovou časťou tejto prípravy bolo vystrihnutie potrebnej časti, tj. ruky. Nastavením požadovaných hraníc *passthrough* filtra[2] (trieda `pcl::PassThrough`) bolo možné túto akciu realizovať. Po odstránení nepotrebných bodov ostali už len ruky, ktoré však obsahovali šum. Využitím metód triedy `pcl::StatisticalOutlierRemoval` bol šum čiastočne odstránený. Priebežne boli *pointcloudy* kontrolované, či nedošlo aj k odstráneniu charakteristických znakov rúk, čo by mohlo viesť k znehodnoteniu vzorového gesta.

Vzorová sada je teda zložená z troch gest, pričom pre každé gesto bolo takto vytvorených približne deväť *pointcloudov*.

[1] Point Cloud Data, `http://pointclouds.org/documentation/tutorials/pcd_file_format.php`
[2] `http://pointclouds.org/documentation/tutorials/passthrough.php`



Obrázek 4.2: Červená časť je výsledkom aplikácie *passthrough* filtra. Zobrazené pomocou nástroja PCLVisualizer.



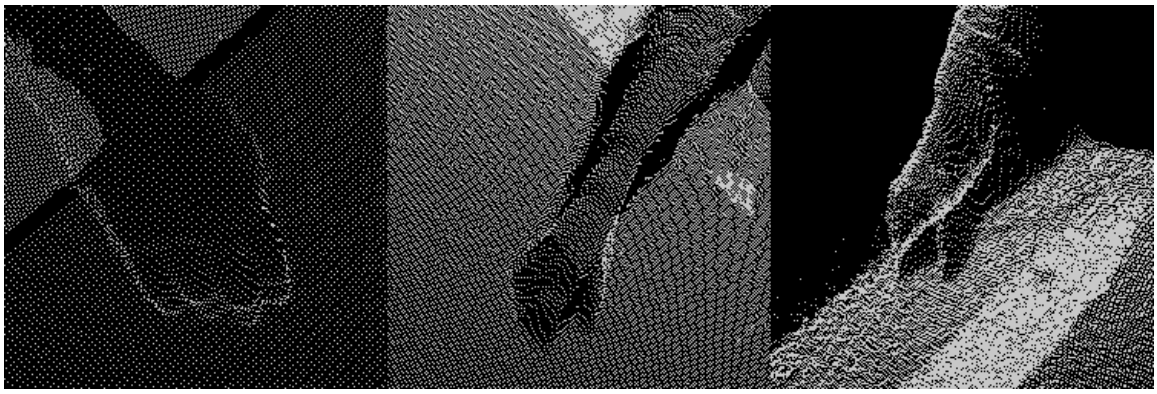
Obrázek 4.3: *Pointcloud* po odstránení šumu je označený červenou farbou, biele body boli vybrané ako *outliery*

4.2.2 Príprava testovacej sady

Pri testovacej sade bolo nutné rozdeliť jednotlivé scény z hľadiska rozpoznateľnosti. Bolo teda opäť potrebné postupovať manuálne a kontrolovať tak každý *pointcloud*. U jednoducho rozpoznateľných sa javia črty ruky jasnejšie a zreteľnejšie ako u menej rozpoznateľných scén. Rozloženie prstov u daných gest je tiež jasne badateľné. Pri menej rozpoznateľných nie sú už črty ruky tak výrazné, dochádza tu k šumu. Ťažko rozpoznateľné scény obsahujú ruky, ktoré pripomínajú zhluk bodov, črty nejasné, nemožno celkom rozhodnúť o aké gesto sa jedná.

Testovacia sada bola navyše rozdelená na vyhovujúce a nevyhovujúce prípady, kedy sa

ruka nachádzala napr. mimo zóny stola alebo v nevhodnej výške.



Obrázek 4.4: Ukážky testovacích scén. Zľava: jednoducho, menej, ťažko rozpoznateľné.

4.3 Implementácia

V tejto časti bude opísaná implementácia jednotlivých krokov podľa kapitoly 3.2, ako aj problémy, na ktoré som pri implementácii narazil. Budem sa teda riadiť diagramom znázorneným na obrázku 3.3.

Pre implementáciu programu bol zvolený jazyk C++ a knižnica PCL, ktorá obsahuje veľké množstvo užitočných tried a funkcií, ktoré sú aplikovateľné z hľadiska práce s *pointcloudami*. Knižnica Eigen bola použitá pre výpočty s maticami a vektormi, a v neposlednom rade knižnica OpenCV, ktorú som použil pre záverečnú vizualizáciu. Cieľovou platformou je GNU/Linux, konkrétne distribúcia Ubuntu 14.04, kvôli plnej kompatibilite so systémom ROS.

4.3.1 Opis programu

Program sa skladá z troch súborov typu `.cpp`. Prvým z nich je `feature_cloud.cpp`, definujúci triedu `FeatureCloud`. Táto trieda poskytuje metódy pre výpočet normál, deskriptorov a kontrolných bodov jednotlivých *pointcloudov*. Ďalšou triedou je `TemplateAlignment`, ktorá sa nachádza v súbore `template_alignment.cpp` a slúži pre registráciu pomocou metódy SAC-IA a jej vyhodnotenie (opísané ďalej). Obe tieto triedy boli prebraté z návodu[3] a doplnené o ďalšie metódy.

Jadrom aplikácie je súbor `hand_recognizer.cpp`, ktorý využíva spomínané triedy a obsahuje funkcie, potrebné pre chod celej aplikácie. V nasledujúcich kapitolách sa budem na tieto súbory a ich funkcie odkazovať.

Aplikácii je možné predať veľké množstvo argumentov, pričom požadované sú len dva súbory obsahujúce zoznam ciest k súborom typu `.pcd`. Ostatné argumenty majú prednastavené implicitné hodnoty. Prvý súbor by mal definovať potrebné vzorové gestá a druhý sekvenciu celých scén, ktoré sa budú spracovávať. Ďalej je možné zadávať voliteľné argumenty modifikujúce použité algoritmy. Ukážka súborov a opis argumentov je súčasťou príloh tejto práce.

[3] `http://pointclouds.org/documentation/tutorials/template_alignment.php`

4.3.2 Spracovanie vzorov ruky

Program začína postupným načítaním vzorov zo súboru definovaného v prvom argumente. To zabezpečuje funkcia `loadObjectTemplates()`, ktorá každý súbor predá metóde `loadInputCloud()` triedy `FeatureCloud` a tá automaticky „spracuje“ daný *pointcloud*.

Najskôr je potrebné určiť kľúčové body. Ako uvádza literatúra [10], pre extrakciu je vhodnejšie použiť rovnomerné podvzorkovanie než detektory kľúčových bodov, keďže pri nich sa určenie bodov môže líšiť od použitých deskriptorov. Z tohto dôvodu používam voxelizáciu (`pcl::VoxelGrid`) o veľkosti listu 5mm.

Ďalším krokom je výpočet normál a deskriptorov, čo zabezpečujú triedy `pcl::NormalEstimationOMP` a `pcl::FPFHEstimationOMP`. Skratka OMP značí OpenMP[4], čo je knižnica umožňujúca využitie viacerých vlákien, a tým urýchlenie celkového procesu. Ako už bolo spomenuté, body sú dané len ich súradnicami (štruktúra `pcl::PointXYZ`), pretože FPFH deskriptory pracujú s geometriou povrchu, nie farbou. Následne sú pomocou k-d stromu vyhľadané najbližšie susedné body, ktoré ležia v zadanom okruhu – z nich sú vypočítane normály, ktoré sa použijú ako vstup pre výpočet deskriptorov.

Keďže vzorových *pointcloudov* je približne tridsať, rozhodol som sa, pre uľahčenie manuálneho nastavovania jednotlivých kontrolných bodov, použiť efektívnejšie riešenie. Najprv som použitím metód triedy `pcl::MomentOfInertiaEstimation` zostrojil *bounding box*[5] okolo každého *pointcloudu* a následne som určil kontrolný bod ako stred diagonály prednej steny boxu (špičky prstov ruky).

Týmto spôsobom sú teda určené kľúčové body a vypočítané normály, deskriptory a kontrolné body vzorov.

4.3.3 Preprocessing

Ešte pred fázou predspracovania sú z druhého súboru načítané a uložené všetky pointcloudy reprezentujúce scény. Tie sa spracúvajú po jednom, pričom samotný *preprocessing* má na starosti funkcia `preprocessPointcloud()`, ktorá pomocou *passthrough* filtra odstráni body, nachádzajúce sa mimo vymedzených hraníc. Pomocou argumentov je možné tieto hranice meniť (celkovo šesť argumentov – minimum a maximum pre každú os).

4.3.4 Detekcia stola

To čo zostalo z predošlej fázy sa predá funkcii `removeTable()`. Ako už názov napovedá, funkcia odstraňuje plochu stola, no ešte predtým ju musí detekovať. Detekcia je zabezpečená nastavením príslušných parametrov resp. atribútov, ako počet iterácií a prah (kapitola 2.10), ako aj modelu (rovina) do objektu triedy `pcl::SACSegmentation`. Následne je detekovaná rovina ohraničená 2D konvexnou obálkou. Použitím metód triedy `pcl::ExtractPolygonalPrismData` je vytvorený mnohosten, zostrojený nad určenými hranicami, ktorý je uložený pre ďalšie potreby. Mnohostenu sú opäť nastavené hranice výšky, čím je vytvorená pracovná oblasť, spomínaná v kapitole 3.2.4. V prípade, že nedošlo k detekcii, program načíta a spracuje ďalší *pointcloud* zo súboru a znova vykoná detekciu.

Výstupom funkcie `removeTable()` je *pointcloud* obsahujúci objekty, nachádzajúce sa vo vymedzenej oblasti, a plocha stola potrebná pre výpočet transformácií.

[4] `http://openmp.org/wp/`

[5] obklopujúca/ohraničujúca schránka

4.3.5 Získanie transformácie stola

Pre získanie súradnicového systému rohu stola som sa rozhodol opäť použiť *bounding box* (zostrojený okolo plochy stola) obdobným spôsobom, aký je opísaný vyššie (4.3.2). Keďže vrcholov boxu bolo osem, využil som z nich len vrchné štyri, pre ktoré som vypočítal transformačné matice. Získal som tak súradnicové systémy pre jednotlivé rohy stola.

4.3.6 Segmentácia zhlukov

Po odstránení plochy stola zostal jeden *pointcloud* zložený z objektov na stole. Pre prístup k jednotlivým zhlukom je potrebné ich najprv oddeliť, takže z nich vzniknú samostatné *pointcloudy*.

Objekty ako celok sú teda reprezentované k-d stromom, určeným pre prehľadávanie bodov. Strom sa použije ako vstup triedy, pomocou ktorej je zabezpečené euklidovské zhlukovanie (`pcl::EuclideanClusterExtraction`), spolu s ďalšími parametrami – minimálna a maximálna veľkosť zhlukov (v bodoch) či tolerancia vzdialenosti medzi bodmi. Tieto parametre je potrebné správne nastaviť, pretože sa môže stať, že objekt bude rozdelený na niekoľko menších zhlukov, alebo naopak, bude viacero objektov tvoriť jeden celok. Opäť možnosť modifikácie prostredníctvom argumentov.

4.3.7 Detekcia ruky

Po rozdelení celku na samostatných kandidátov, je každý kandidát podrobený voxelizácii s rovnakou veľkosťou listu ako tomu bolo u vzorov. Ďalej je každý zhluk reprezentovaný objektom triedy `FeatureCloud`, kde sú vypočítané jeho normály a deskriptory, podobne ako v kapitole 4.3.2.

Následne je možné pristúpiť ku samotnej detekcii použitím registrácie, ktorú poskytuje trieda `TemplateAlignment`. Najskôr sú uložené všetky vzorové gestá, s ktorými sa bude daný kandidát porovnávať. Porovnávanie však musí prebiehať na základe niečoho, čím budú práve vypočítané príznaky, resp. normály a deskriptory. Z tohto dôvodu som tiež zvolil rovnaké podvzorkovanie porovnávaných *pointcloudov*.

Ďalej sú nastavené parametre: minimálna a maximálna tolerancia vzdialenosti medzi bodmi a počet iterácií, po ktorý algoritmus pobeží. Všetky predošlé parametre sa predajú metódam triedy `pcl::SampleConsensusInitialAlignment`, ktorá následne vykoná zarovnanie *poincloudov* metódou `align()`.

Po zarovnaní je volaná vyhodnocovacia funkcia, ktorá určí kvalitu zarovnania. Vyhodnocuje sa výpočtom skóre metódou `pcl::Registration::getFitnessScore()`, ktorej sa ako parameter predá vyššie spomínaná maximálna tolerancia vzdialenosti. Výsledné skóre je v podstate metrická chyba, spomínaná v treťom kroku v kapitole 2.13. Je počítaná ako priemerná vzdialenosť, pomocou k-d stromu a algoritmu hľadania najbližšieho suseda, pričom do priemeru sú započítané len vzdialenosti menšie ako predaný parameter.[6] Možno teda povedať, že čím menšia je táto vzdialenosť, tým podobnejšie sú porovnávané *pointcloudy*. Týmto spôsobom sú zarovnané a vyhodnotené postupne všetky vzorové gestá, z ktorých sa pre daného kandidáta vyberie to, ktoré dosiahlo najlepšie skóre. Ďalším problémom bolo nastaviť hranicu skóre pre ruku (viac v kapitole 4.4).

[6] vychádzal som z `http://docs.pointclouds.org/trunk/registration_8hpp_source.html#l00132`

4.3.8 Vykreslenie

V záverečnej fáze je potrebné zobraziť detekované ruky. Pôvodne som chcel použiť nástroj PCLVisualizer pre výslednú vizualizáciu, avšak z dôvodu interaktivity by musel bežať na pozadí, pričom by sa spracovávali a prekresľovali *pointcloudy*. Celý proces spracovania bol najprv aplikovaný len na jeden *pointcloud*, ktorý sa následne vykreslil. Vypočítané skóre zostalo nezmenené aj po viacnásobnom spustení programu s rovnakými parametrami (a vstupnými *pointcloudami*). Problém nastal, ak PCLVisualizer bežal na pozadí popri spracovávaní – výsledky skóre sa menili každým spustením programu. Nepomohlo ani použitie vlákien, ani ochrana kritických sekcií kódu[7]. Na oficiálnom fóre PCL bol nakoniec problém použitia vlákien v kombinácii s PCLVisualizerom[8] potvrdený a následne pridaný aj do programovej dokumentácie[9]. Čo však konkrétne spôsobovalo zmenu skóre, zostáva záhadou.

Rovnaký problém som zaznamenal aj pri súčasnom vykresľovaní pomocou knižnice OpenCV. Výsledky skóre vypočítané klasifikátorom sa javili ako náhodné (kapitola 4.4), preto som sa rozhodol pre vykreslenie až po spracovaní všetkých scén.

Do obrázkov veľkosti 640x480 pixelov je na stred vykreslený obdĺžnik, ktorého šírka tvorí 70 % šírky obrázka, pričom výška je dopočítaná podľa pomeru strán stola. V prípade, že je stôl orientovaný zvislo (z pohľadu Kinectu), dopočíta sa šírka stola a výška potom tvorí 70 % výšky obrázka.

V ďalšom kroku sa vykresľujú kontrolné body reprezentujúce ruky. Najskôr je však potrebné prepočítať ich súradnice vzhľadom na ľavý horný roh stola. Funkcia `transformWorldToLocal()` invertuje už vypočítanú transformačnú maticu daného rohu a potom ňou vynásobí (transformuje) kontrolný bod, čím sú získané jeho lokálne súradnice. Následne sú zamenené súradnice x a y.

Po zarovnaní vzoru na kandidáta môže dôjsť k tomu, že kontrolný bod bude ležať mimo pracovnej oblasti stola. Preto sú všetky súradnice, väčšie (menšie) ako hranice tejto oblasti, orezané na minimálnu, resp. maximálnu hodnotu hraníc. Ďalej sú tieto súradnice vynásobené pomerom veľkostí strán obdĺžnika a stola, čím sú konvertované na body obrazovky – tie sú potom prevedené na jednotkové súradnice. Nakoniec sú pre body určené farby podľa gesta a nastaví sa ich veľkosť podľa vzdialenosti od stola. Všetky obrázky sú uložené do formátu PNG.

4.4 Testy a vyhodnotenie

Testovanie prebiehalo na definovanej testovacej sade (kapitola 4.2.2), pričom priebežne bola upravovaná a doplňaná o ďalšie scény, resp. objekty. Keďže niektoré objekty v scénach sa môžu zdať podobné, jedná sa o iné objekty, pretože pri šume, ktorý nastáva, dochádza ku zmene bodov objektu. Preto aj tieto objekty boli zahrnuté do testovacej sady. Počet scén, kedy sa ruka nachádzala mimo pracovnej oblasti, bol redukovaný. Kvôli detekcii stola a následnému vymedzeniu hraníc, nebolo potrebné tento typ scén až tak testovať.

4.4.1 Nájdenie správnych hodnôt parametrov

Vstupných parametrov ovplyvňujúcich celkovú detekciu je naozaj veľa, preto som im musel nastaviť správne hodnoty. Na začiatku som pracoval s dvadsaťjeden testovacími objektami,

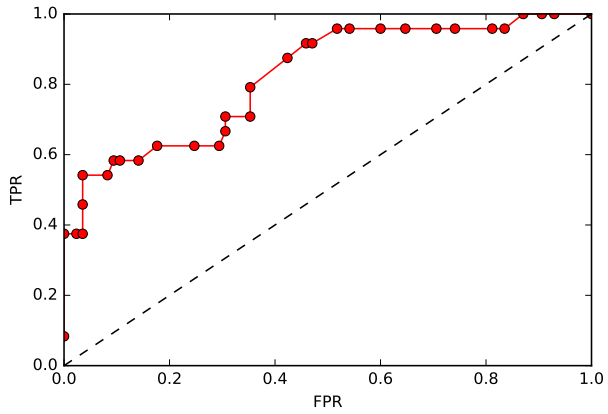
[7] vyskúšané podľa `http://bit.ly/1rXraJQ`
[8] `http://www.pcl-users.org/Updating-PCLVisualizer-from-another-thread-td4021570.html`
[9] `http://docs.pointclouds.org/trunk/classpcl_1_1visualization_1_1_p_c_l_visualizer.html`

Obrázek 4.5: Ukážka testovaných objektov

z čoho šesť bolo rúk. Ako už bolo spomenuté, veľkosť listu pri podvzorkovaní porovnávaných *pointcloudov* (vzory + objekty) bola nastavená narovnako – 5mm. Pri nastavení menšej hodnoty dochádzalo ku rozdielnosti skóre testovaných objektov, pri väčšej hodnote algoritmus vykazoval viacmenej rovnaké výsledky. Parameter voxelizácie, spolu s ostatnými parametrami, som určil pomocou ROC kriviek, pričom som menil vždy len jednu hodnotu daného parametru. Vstupom vyhodnotenia ROC bolo skóre, určené klasifikátorom a binárne hodnoty, udávajúce či sa jedná alebo nejedná o ruku (*ground truth*). Nakoniec som zvolil takú hodnotu pre parameter, ktorá dosiahla najvyššiu hodnotu AUC.

4.4.2 Vyhodnotenie klasifikátora

Hlavným problémom bolo určenie správnej hranice (prahu) pre vypočítané skóre, rozhodujúce o rozpoznaní ruky. Vstupná testová sada nakoniec obsahovala vyše sto objektov (vrátane rúk), z ktorých skóre bol určený prah opäť pomocou ROC krivky (obr. 4.6). Na záver bola vypočítaná hodnota AUC tejto krivky, udávajúca úspešnosť klasifikátora približne 82 %.



Obrázek 4.6: Výsledná ROC krivka, znázorňujúca jednotlivé prahy skóre. Ako najlepší bol určený bod, najbližší k ľavému hornému rohu.

Testovanie detekcie gest zobrazuje tabuľka v prílohe C. Bolo použitých celkovo desať

zhlukov rúk, pričom z každého gesta približne tri. Pre všetky objekty vzorových gest boli zaznamenané skóre u jednotlivých zhlukov do stĺpcov $R1$ až $R10$. Následne bola pre každý stĺpec vybraná priemerná hodnota skóre a zvýraznené boli bunky, obsahujúce lepšie (menšie) skóre ako táto hodnota. Úspešnosť bola vypočítaná ako podiel počtu zvýraznených buniek a celkového počtu buniek pri danej triede. Najväčšiu úspešnosť dosahovala trieda číslo tri, reprezentujúca gesto „päsť“, a to približne 68 %. Druhá trieda (gesto „ukazujúci prst“) dosiahla približne 56 % a na poslednom mieste skončila prvá trieda (gesto „voľná ruka“), dosahujúca necelých 34 %.

4.4.3 Záverečné zhodnotenie

Celá práca bola testovaná a vyvíjaná na počítači s viacjadrovou architektúrou a operačným systémom GNU/Linux distribúcie Ubuntu 14.04. Pre správnu funkciu programu je potrebné mať nainštalované spomínané knižnice, ktoré som v tejto práci použil.

Pre program a definované argumenty bola nastavená nájdená hranica skóre. Pri tejto hranici program „vyradí“ prípadných kandidátov a keďže úspešnosť nie je stopercentná, sú prípady kedy očakávaná ruka nebola detekovaná, prípadne bol označený iný objekt ako ruka. Vyhodnotenie môže ovplyvniť aj nesprávne určený polomer pri výpočte normál, kedy povrch objektu nebude správne definovaný.

Čo sa týka rýchlosti behu programu, tá zavisí od viacerých faktorov. Najväčšou „brzdou“ programu je registrácia pomocou metódy SAC-IA, pretože porovnáva každý vzor s každým objektom a rýchlosť potom závisí od počtu týchto objektov. Dôležitú úlohu tu tiež zohráva počet nastavených iterácií. Pri nastavení nízkeho počtu môže dôjsť k nepresnému vyhodnoteniu skóre, avšak pri vysokom počte iterácií zasa dochádza ku značnému spomaleniu celkového procesu. Z dôvodu pomalého vyhodnocovania nie je aplikácia vhodná pre beh v reálnom čase, napriek použitým urýchleniam (normály a deskriptory – knižnica OpenMP atď.).

Kapitola 5

Záver

Cieľom tejto práce bolo vytvorenie aplikácie, ktorá by umožňovala detekciu rúk užívateľa nad interaktívnym stolom a následné zobrazenie týchto dvoch aspektov na plochu obrazovky, resp. uloženie do obrazového formátu. Bol zvolený prístup pomocou hĺbkových dát, tj. zachytené snímky sú vo forme trojrozmerného obrazu. Hlavnú úlohu tu teda zohráva hĺbkový senzor Kinect, ktorý vyjadruje moderný prístup počítačového videnia spojený s cenovou dostupnosťou.

Čitateľ bol oboznámený s problematikou rozpoznávania gest rúk a s tým spojenou prácou na interaktívnom stole. Boli opísané jednotlivé metódy, nástroje a algoritmy, potrebné k dosiahnutiu cieľa aplikácie. V teoretickej časti boli tiež uvedené existujúce prístupy, akými je možné ruku detekovať, následne kapitola návrh bola zameraná na postup, ako tieto nástroje využiť pre konečnú detekciu. V záverečnej kapitole boli potom postupy konkretizované s ohľadom na implementáciu a nakoniec bol algoritmus otestovaný a vyhodnotený.

Bola vytvorená dátová sada pre účely aplikácie a definovanie s následným testovaním jednotlivých gest. V priebehu vývoja programu bola dopĺňaná, resp. upravovaná pre konkrétne oblasti testov. Metóda dokáže detekovať rovinu a jednotlivé objekty, nachádzajúce sa v oblasti nad stolom. Potom vyhodnocuje podobnosť týchto objektov so všetkými definovanými vzormi na základe ich rysov. Podobnosť, resp. rozdielnosť je udaná hodnotou skóre, ktorú klasifikátor určuje a následne rozhodne či išlo o ruku. Ďalej je vypočítaná poloha ruky voči stolu a detekovaná ruka je potom zobrazená bodom vykresleným do obrazového formátu spolu so stolom, reprezentovaným obdĺžnikom.

Metóda miestami zlyháva, čo má za následok označenie iného objektu ako ruky, alebo naopak. Napriek tomu vyšla úspešnosť klasifikácie na viac ako 80 %. Ako „bonus“ dokáže metóda určiť o aké gesto sa jedná, no to už nemožno hovoriť o takej úspešnosti. Program je možné modifikovať prostredníctvom rôznych hodnôt argumentov príkazového riadka, avšak tie sú závislé od použitých vstupných dát. Sada vzorov môže byť doplnená o ďalšie *pointcloudy*, ktoré napr. predstavujú už definované či ďalšie gestá. Týmto spôsobom je možné experimentovať s rôznymi kombináciami a program „vyladiť“ podľa vlastných potrieb. Ako už bolo spomínané, vykreslené body a ich jednotkové súradnice je možné prepočítať a použiť napr. na simuláciu myši.

Keďže je program vyvíjaný primárne pomocou C++, PCL a Kinectu, ktoré sú kompatibilné so systémom ROS, v budúcnosti by mohol nájsť využitie napr. pri interakcii s robotmi, podporujúcimi Kinect (prípadne iný senzor) či zariadenia, ktoré pracujú so systémom ROS.

Literatura

[1] Receiver operating characteristic - Wikipedia. [online], Naposledy cit. 10.5.2016.
URL https://en.wikipedia.org/wiki/Receiver_operating_characteristic

[2] code-iai/iai_kinect2: Tools for using the Kinect One (Kinect v2) in ROS. [online], Naposledy cit. 13.12.2015.
URL https://github.com/code-iai/iai_kinect2/tree/master/kinect2_bridge

[3] ROS/Installation - ROS Wiki. [online], Naposledy cit. 15.12.2015.
URL http://wiki.ros.org/ROS/Installation

[4] Topics - ROS Wiki. [online], Naposledy cit. 15.12.2015.
URL http://wiki.ros.org/Topics

[5] RANSAC - Wikipedia. [online], Naposledy cit. 1.5.2016.
URL https://en.wikipedia.org/wiki/RANSAC

[6] rosbag/Commandline - ROS Wiki. [online], Naposledy cit. 25.4.2016.
URL http://wiki.ros.org/rosbag/Commandline

[7] k-d tree - Wikipedia. [online], Naposledy cit. 27.4.2016.
URL https://en.wikipedia.org/wiki/K-d_tree

[8] PCL/OpenNI tutorial 4: 3D object recognition (descriptors) - Robótica - ULE. [online], Naposledy cit. 28.4.2016.
URL
http://robotica.unileon.es/mediawiki/index.php/PCL/OpenNI_tutorial_4:_3D_object_recognition_(descriptors)

[9] Computing the Pixel Coordinates of a 3D Point (Mathematics of Computing the 2D Coordinates of a 3D Point). [online], Naposledy cit. 30.4.2016.
URL http://bit.ly/1TSpZVi

[10] Aldoma, A.; Marton, Z. C.; Tombari, F.; aj.: Tutorial: Point Cloud Library: Three-Dimensional Object Recognition and 6 DOF Pose Estimation. *IEEE Robotics Automation Magazine*, ročník 19, č. 3, sept 2012: s. 80–91, ISSN 1070-9932, doi:10.1109/MRA.2012.2206675.

[11] Bradski, D. G. R.; Kaehler, A.: *Learning Opencv, 1st Edition*. O'Reilly Media, Inc., první vydání, 2008, ISBN 9780596516130.

[12] Choi, S.; Kim, T.; Yu, W.: Performance Evaluation of RANSAC Family. In *Proceedings of the British Machine Vision Conference*, BMVA Press, 2009, ISBN 1-901725-39-1, s. 81.1–81.12, doi:10.5244/C.23.81.

[13] Cupec, R.; Grbic, R.; Nyarko, E. K.; aj.: Detection of Planar Surfaces Based on RANSAC and LAD Plane Fitting. In *Proceedings of the 4th European Conference on Mobile Robots, ECMR'09, September 23-25, 2009, Mlini/Dubrovnik, Croatia*, 2009, s. 37–42.

[14] Guennebaud, G.; Jacob, B.; aj.: Eigen v3. [online], 2010.
URL http://eigen.tuxfamily.org

[15] Kurakin, A.; Zhang, Z.; Liu, Z.: A real time system for dynamic hand gesture recognition with a depth sensor. In *Signal Processing Conference (EUSIPCO), 2012 Proceedings of the 20th European*, Srpen 2012, ISSN 2219-5491, s. 1975–1979.

[16] Lachat, E.; Macher, H.; Mittet, M.-A.; aj.: FIRST EXPERIENCES WITH KINECT V2 SENSOR FOR CLOSE RANGE 3D MODELLING. *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, ročník XL-5/W4, 2015: s. 93–100, doi:10.5194/isprsarchives-XL-5-W4-93-2015.
URL http://www.int-arch-photogramm-remote-sens-spatial-inf-sci.net/XL-5-W4/93/2015/

[17] Makris, A.; Argyros, A.: Model-based 3D Hand Tracking with on-line Shape Adaptation. In *Proceedings of the British Machine Vision Conference (BMVC)*, editace M. W. J. Xianghua Xie; G. K. L. Tam, BMVA Press, Září 2015, ISBN 1-901725-53-7, s. 77.1–77.12, doi:10.5244/C.29.77.
URL https://dx.doi.org/10.5244/C.29.77

[18] Martinez, A.; Fernández, E.: *Learning ROS for Robotics Programming*. Packt Publishing, Září 2013, ISBN 1782161449.
URL http://www.packtpub.com/learning-ros-for-robotics-programming/book

[19] Nguyen, T.: Lecture 13+: Nearest Neighbor Search, Naposledy cit. 1.5.2016.
URL https://amath.colorado.edu/faculty/martinss/Teaching/APPM5720_2016s/week11_wed_kd_tree.pdf

[20] Panwar, M.: Hand gesture recognition based on shape parameters. In *Computing, Communication and Applications (ICCCA), 2012 International Conference on*, Únor 2012, s. 1–6, doi:10.1109/ICCCA.2012.6179213.

[21] Pedersoli, F.; Benini, S.; Adami, N.; aj.: XKin: an open source framework for hand pose and gesture recognition using kinect. *The Visual Computer*, ročník 30, č. 10, 2014: s. 1107–1122, ISSN 1432-2315, doi:10.1007/s00371-014-0921-x.
URL http://dx.doi.org/10.1007/s00371-014-0921-x

[22] Rusu, R. B.: *Semantic 3D Object Maps for Everyday Manipulation in Human Living Environments*. Dizertační práce, Computer Science department, Technische Universitaet Muenchen, Germany, October 2009.

[23] Rusu, R. B.; Blodow, N.; Beetz, M.: Fast Point Feature Histograms (FPFH) for 3D Registration. In *Proceedings of the 2009 IEEE International Conference on Robotics and Automation*, ICRA'09, Piscataway, NJ, USA: IEEE Press, 2009, ISBN 978-1-4244-2788-8, s. 1848–1853.
URL http://dl.acm.org/citation.cfm?id=1703435.1703733

Přílohy

Seznam příloh

Příloha A

Obsah CD

Prílohou tejto práce sú tiež zdrojové súbory programu, vzorová sada rúk a testovacia sekvencia scén, spolu s ďalšími samostatnými scénami na neprepisovateľnom pamäťovom médiu CD. Ďalej disk obsahuje manuál k prekladu a spusteniu aplikácie, ako aj programovú dokumentáciu. Je tu priložený aj plagát, demonštračné video aplikácie a technická správa tejto práce.

Příloha B

Opis argumentov a ukážka súborov

Program je možné spustiť s viacerými argumentmi, pričom potrebné sú len prvé dva. Zvyšné argumenty majú nastavené implicitné hodnoty.

Preprocessing

Fáza hrubého orezania *pointcloudu*. Interval by mal byť zadaný tak, aby v sa v ňom nachádzal stôl.

`--min_x` a `--max_x` interval osi x pre hrubé orezanie pomocou *passthrough* filtra
`--min_y` a `--max_y` interval osi y pre hrubé orezanie pomocou *passthrough* filtra
`--min_z` a `--max_z` interval osi z pre hrubé orezanie pomocou *passthrough* filtra – rozdiel musí byť väčší ako hodnota 0.3 (30cm)

Detekcia stola

Fáza detekcie stola pomocou metódy RANSAC, použitím modelu roviny.

`--max_i` maximálny počet iterácií
`--thresh` prah vzdialenosti modelu

Vymedzenie oblasti

Vymedzenie pracovnej oblasti po aplikácii konvexnej obálky na plochu stola.

`--height` výška pracovnej oblasti od stola

Euklidovské zhlukovanie

Fáza segmentácie zhlukov na stole.

`--cluster` tolerancia zhlukov – vzdialenosť medzi bodmi
`--min` minimálna veľkosť zhluku v bodoch
`--max` maximálna veľkosť zhluku v bodoch

Zarovnanie

Fáza registrácie pomocou SAC-IA.

`--iter` počet iterácií
`--min_d` minimálna vzdialenosť medzi porovnávanými bodmi
`--max_d` maximálna vzdialenosť medzi porovnávanými bodmi – horná hranica skóre, udaná ako štvorcová vzdialenosť (jedna strana štvorca)

Vyhodnotenie

Fáza vyhodnotenia metódy SAC-IA.

`--score` hranica skóre, udávajúca rozdiel medzi rukami a ostatnými objektmi – objekty, ktorých skóre je menšie ako nastavená hranica budú vyhodnotené ako ruka

Vykreslenie

Fáza vykreslenia obrázkov do súborov. Z prepínačov je možné zadať len jeden.

`-s` prepínač, udávajúci, či sa majú do obrázkov vypisovať aj skóre
`-c` prepínač, udávajúci, či sa majú do obrázkov vypisovať aj jednotkové súradnice

Vstupné súbory

Povinnými argumentmi sú dva vstupné súbory obsahujúce cesty k *pointcloudom*. Prvým argumentom musí byť súbor reprezentujúci vzorové objekty a druhým súbor, reprezentujúci scény (sekvenciu). Riadky v súboroch je možné „zakomentovať“ symbolom `#`.

```
gestures/1/1.pcd
gestures/1/2.pcd
gestures/2/1.pcd
#gestures/2/1.pcd

gestures/3/1.pcd
```

Kód B.1: Ukážka súboru so vzorovými gestami

```
scenes/scene1.pcd
scenes/scene2.pcd
scenes/scene3.pcd

#scenes/scene4.pcd
#scenes/scene5.pcd
```

Kód B.2: Ukážka súboru so sekvenciou scén

Příloha C

Vyhodnotenie gest rúk

Trieda	Objekt	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10
#1	1	0,000045	0,000091	0,000122	0,000086	0,000093	0,000118	0,000107	0,000138	0,000063	0,000093
	2	0,000043	0,000078	0,000087	0,000074	0,000099	0,000098	0,000108	0,000108	0,000101	0,000107
	3	0,000076	0,000126	0,000101	0,000070	0,000113	0,000083	0,000143	0,000082	0,000114	0,000126
	4	0,000097	0,000095	0,000125	0,000106	0,000103	0,000108	0,000106	0,000089	0,000088	0,000121
	5	0,000088	0,000099	0,000113	0,000092	0,000135	0,000103	0,000114	0,000107	0,000113	0,000109
	6	0,000096	0,000075	0,000095	0,000140	0,000076	0,000069	0,000117	0,000102	0,000113	0,000097
	7	0,000118	0,000116	0,000108	0,000112	0,000080	0,000087	0,000111	0,000099	0,000093	0,000090
	8	0,000061	0,000115	0,000060	0,000074	0,000079	0,000087	0,000101	0,000106	0,000117	0,000097
	9	0,000064	0,000088	0,000093	0,000104	0,000092	0,000100	0,000078	0,000082	0,000111	0,000062
	10	0,000069	0,000092	0,000101	0,000097	0,000072	0,000079	0,000082	0,000106	0,000081	0,000084
	11	0,000070	0,000079	0,000082	0,000121	0,000099	0,000099	0,000106	0,000104	0,000106	0,000076
#2	1	0,000108	0,000105	0,000088	0,000084	0,000076	0,000111	0,000033	0,000085	0,000139	0,000105
	2	0,000087	0,000077	0,000088	0,000092	0,000097	0,000101	0,000049	0,000076	0,000100	0,000109
	3	0,000040	0,000061	0,000097	0,000099	0,000100	0,000081	0,000067	0,000085	0,000111	0,000058
	4	0,000036	0,000076	0,000056	0,000076	0,000059	0,000055	0,000063	0,000061	0,000095	0,000085
	5	0,000097	0,000063	0,000074	0,000062	0,000085	0,000142	0,000087	0,000129	0,000097	0,000128
	6	0,000073	0,000059	0,000077	0,000056	0,000067	0,000098	0,000058	0,000116	0,000134	0,000098
	7	0,000072	0,000080	0,000100	0,000109	0,000092	0,000082	0,000042	0,000099	0,000088	0,000135
#3	1	0,000054	0,000036	0,000087	0,000075	0,000065	0,000089	0,000095	0,000060	0,000083	0,000046
	2	0,000085	0,000067	0,000094	0,000084	0,000078	0,000093	0,000071	0,000065	0,000053	0,000058
	3	0,000099	0,000063	0,000070	0,000068	0,000070	0,000131	0,000052	0,000062	0,000021	0,000106
	4	0,000100	0,000091	0,000100	0,000085	0,000109	0,000074	0,000075	0,000068	0,000064	0,000089
	5	0,000091	0,000043	0,000123	0,000092	0,000086	0,000059	0,000072	0,000096	0,000060	0,000039
	6	0,000025	0,000063	0,000081	0,000069	0,000087	0,000102	0,000111	0,000079	0,000105	0,000087
	7	0,000031	0,000079	0,000074	0,000070	0,000097	0,000071	0,000085	0,000085	0,000031	0,000086
	8	0,000090	0,000119	0,000077	0,000094	0,000089	0,000051	0,000105	0,000063	0,000085	0,000108
	9	0,000080	0,000056	0,000050	0,000083	0,000100	0,000119	0,000087	0,000090	0,000085	0,000060

Obrázek C.1: Tabuľka znázorňuje skóre pre desať testovaných objektov (rúk). Pre každý stĺpec $R1$ až $R10$ je vypočítaná priemerná hodnota a zvýraznené bunky reprezentujú lepšie skóre (ako daný priemer).

Ruka	Zodpovedajúca trieda
R1	#1
R2	#1
R3	#1
R4	#2
R5	#2
R6	#2
R7	#2
R8	#3
R9	#3
R10	#3

Obrázek C.2: Tabuľka znázorňuje, ktoré triedy (C.1) zodpovedajú ktorým objektom rúk