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

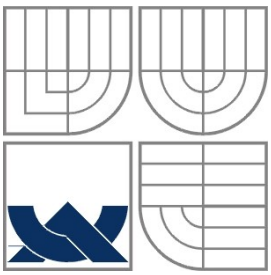
DETEKCE OBJEKTŮ NA DESCE PRACOVNÍHO STOLU

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

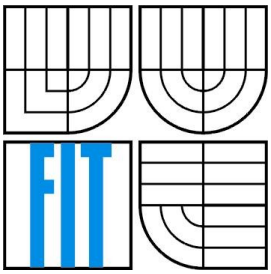
AUTOR PRÁCE
AUTHOR

Bc. Tomáš Varga

BRNO 2015



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 OBJEKTŮ NA DESCE PRACOVNÍHO STOLU

TABLETOP OBJECT DETECTION

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. Tomáš Varga

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. Michal Španěl, Ph.D.

BRNO 2015

Abstrakt

Tato práce se zabývá problematikou detekce objektů na pracovní desce v mračne bodů. Mračno bodů je zaznamenáno pomocí senzoru Kinect. Navržené řešení je postaveno na algoritmu RANSAC pro detekci plochy, dále na algoritmu Euklidovo zhlukování pro segmentaci a nakonec na algoritmu ICP pro detekci objektů. Algoritmus ICP je modifikovaný a dokáže detekovat hlavně rotačně symetrické objekty a objekty, které nejsou nijak transformovány vůči modelům. Řešení využíva platformu ROS, na kterém se výsledný balíček vyvíjí. Výsledky dosažené nad vlastní datovou sadou byly dobré i přes omezenou funkčnost detektoru.

Abstract

This work describes the issue of tabletop object detection in point cloud. Point cloud is recorded with Kinect sensor. Designed solution uses algorithm RANSAC for plane detection, algorithm Euclidean clustering for segmentation and ICP algorithm for object detection. Algorithm ICP is modified and mainly it can detect rotational symetric objects and objects without any transformation against it's models. The final package is build on platform ROS. The achieved results with own dataset are good despite of the limited functionality of the detector.

Klíčová slova

počítačové vidění, mračno bodů, Kinect, ROS, PCL, detekce roviny, detekce objektů, detekce pracovní desky

Keywords

computer vision, point cloud, Kinect, ROS, PCL, plane detection, object detection, tabletop detection

Citace

Tomáš Varga: Detekcia objektov na doske pracovného stola, diplomová práce, Brno, FIT VUT v Brně, rok 2015

Detekcia objektov na doske pracovného stola

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením Ing. Michala Španěla Ph.D.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Bc. Tomáš Varga
13.1.2015

Poděkování

Chci poděkovat svému vedoucímu Ing. Michalu Španělovi Ph.D. za pomoc a cenné rady při tvorbě této práce.

Obsah

1 Úvod

Počítačové videnie je veda, ktorá vznikla začiatkom šesťdesiatych rokov minulého storočia. V princípe je to veda, ktorá sa snaží, aby počítače pochopili obrázky a videá. Za pár desaťročí prešla obrovským vývojom, ale aj tak má ešte dlhú cestu pred sebou. Prečo je počítačové videnie dôležité? Predstavme si nejaké vozidlo, napríklad automobil, ktorý má v sebe nainštalovanú kameru. Touto kamerou dokáže vnímať svoje okolie, ako sú dopravné značky, ľudí prechádzajúcich cez cestu, ostatné automobily, línie vozovky. Znalosťou všetkých týchto informácií sa môže predísť rôznym haváriám a nešťastiam a zachrániť veľa ľudských životov. Tak isto veľa životov môžu zachrániť senzory v zdravotníctve, ktoré dokážu v skorom štádiu detekovať ťažké choroby. V niektorých prípadoch človek potrebuje pomoc, ako sú napríklad prírodné katastrofy, kedy môže byť uveznený pod troskami zrútenej stavby a pomôcť mu môže robot, ktorý ho dokáže detekovať a privolať ľudskú pomoc. Týmto sa dostávame k ďalšiemu uplatneniu počítačového videnia, čim je robotika. Roboti môžu byť nápomocný taktiež v priemysle alebo domácnosti, hlavne pre starších alebo postihnutých ľudí, ktorým môžu uľahčiť ich život, či už vykonávaním rôznych domácich prác alebo podávaním požadovaných objektov.

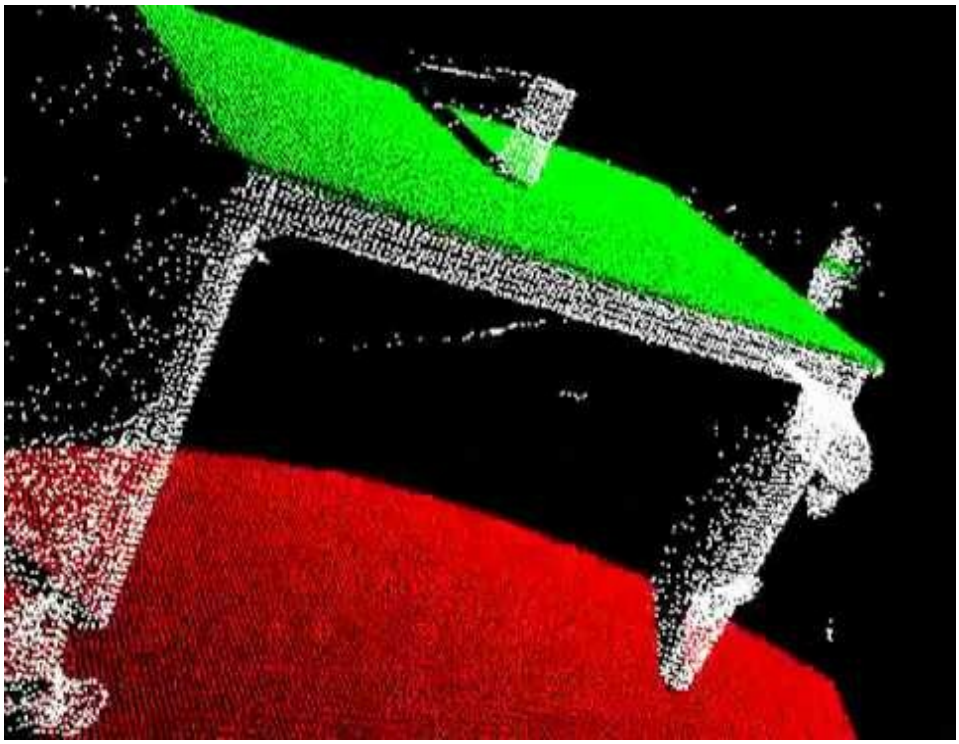
Cieľom tejto práce je navrhnúť modul v systéme ROS[1] pre detekciu čiastočne zaplnenej dosky pracovného stolu a jednotlivých objektov pre robotické aplikácie. Tento nástroj má za úlohu najprv spracovať mračno bodov nahrané senzorom Kinect. Následne z tohoto mračna bodov detekovať pracovnú dosku využitím algoritmu RANSAC. Ďalej nasleduje segmentácia, ktorá rozdelí toto mračno na segmenty, a to použitím metódy Euklidovho zhlukovania, pomocou ktorého získame súradnice bodov jednotlivých segmentov. Na tieto segmenty sa potom použije metóda ICP, ktorá slúži na detekciu objektov v mračne bodov, ktorá nám identifikuje dané objekty. Táto metóda bude využívať modely objektov uložených v databázi, voči ktorým bude detekciu prevádzať.

Práca je rozdelená do deviatich kapitol. V kapitole 2 bude rozobraný problém detekcie pracovnej dosky v mračne bodov, kde bude vysvetlený pojem mračno bodov a budú popísané algoritmy RANSAC a 3D Houghova transformácia, ktoré slúžia na detekciu roviny v mračne bodov. Kapitola 3 je zameraná na riešenie problému segmentácie a detekcie objektov v mračne bodov. Postupne bude popísaný segmentačný algoritmus Eulerovo zhlukovanie a algoritmus na detekciu objektov ICP. Kapitola 4 popisuje senzor Microsoft Kinect, kde bude vysvetlený princíp, akým vníma okolie a získava z neho mračno bodov. Kapitola 5 je zameraná na framework ROS, ktorý bude použitý na vývoj cieľového nástroja. Budú uvedené základné pojmy a princípy, ktorý tento systém používa a nakoniec bude popísaný robot PR2. V kapitole 6 bude bližšie popísaný návrh výsledneho nástroja. Kapitola 7 rieši implementáciu vyvíjaného balíčku. V kapitole 8 budú popísané experimenty a ich výsledky, ktoré boli vykonané s výsledným balíčkom. Nakoniec v kapitole 9 bude zhrnutý záver tejto práce.

1 http://www.ros.org/

2 Problém detekcie roviny

Na snímanie obrazu môže byť použito viacero techník. Jednou z nich je snímanie pomocou Kinectu, ktoré produkuje tzv. mračno bodov. Postupne bude popísané čo to vlastne mračno bodov je, spôsob akým je možné v ňom detekovať rovinu a nakoniec princíp získavania dát Kinectom a jeho matematický model. V podkapitole zameranej na detekciu roviny v mračne bodov budú popísané najznámejšie algoritmy slúžiace tomuto účelu, a to RANSAC a 3D Houghova transformácia. Na obrázku 2.1 je vidieť detekciu dvoch rovín, a to detekciu pracovnej dosky a podlahy.

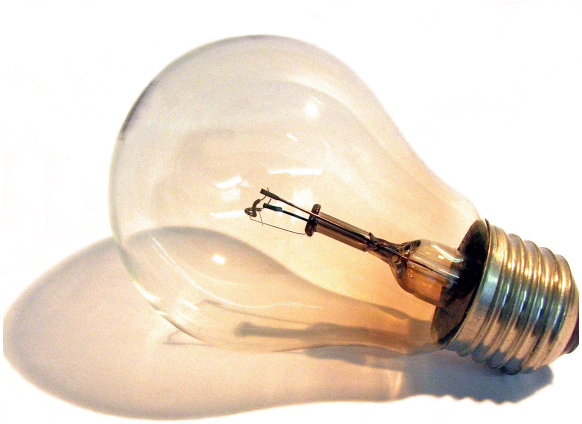


Obrázok 2.1: Detekcia roviny[2].

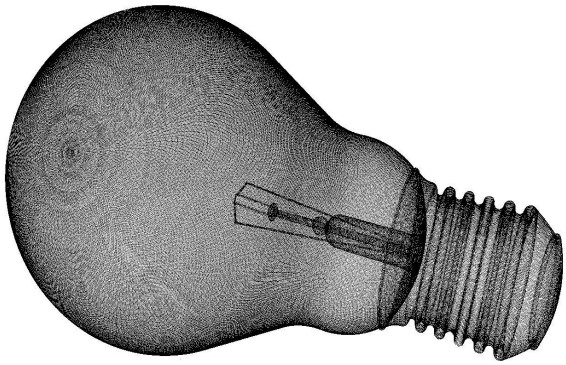
2.1 Mračno bodov

Mračno bodov, tzv. point cloud, je veľký súbor bodov, rádovo milióny bodov, pričom každý bod má zadefinovanú polohu v priestore pomocou troch súradníc *x*, *y* a *z*. Mračno bodov nám detailne a veľmi presne zachytáva a popisuje priestorové objekty a následne je využité na tvorbu priestorových modelov týchto objektov v počítačovom prostredí. Na obrázku 2.2 je možné vidieť obyčajnú žiarovku a na obrázku 2.3 jej reprezentáciu v point cloude.
Mračno bodov vieme získat rôznymi meracími technikami ako je napr. laserové skenovanie, optické skenovanie, fotogrametria a ďalšie. Medzi laserové skenovanie môžeme zaradiť prístroj Kinect, ktorý je dôležitým základom pre výslednú diplomovú prácu a bude bližšie popísaný v kapitole 4.

2 Prevzaté z: http://i.ytimg.com/vi/ZTK7NR1Xx4c/hqdefault.jpg



Obrázok 2.2: Žiarovka originál[3].



Obrázok 2.3: Žiarovka v point cloude[2].

2.2 Detekcia roviny v mračne bodov

Detekcia roviny patrí medzi nevyhnutné podmienky pri širokom spektre úloh zameraných na počítačové videnie. Detekcia roviny je použitá v aplikáciách ako je 3D rekonštrukica a analyzovanie scény, rozpoznávanie objektov, segmentácia a rozšírená realita. V tejto kapitole si uvedieme dve najviac využívané metódy na detekciu roviny, a to RANSAC a 3D Houghovu transformáciu.

RANSAC

Princíp algoritmu RANSAC (Random Sample Consensus) [1] spočíva v hľadaní najlepšej roviny v 3D mračne bodov. V rovnakom čase, zmenšuje počet iterácií, aj keď je počet bodov veľmi veľký. Pre tento účel vyberá náhodne tri body a vypočíta parametre príslušiacej roviny. Potom rozpozná všetky body z východzieho mračna bodov patriaceho do vypočítanej plochy, vzhľadom na nejaký prah. Následne to zopakuje n-krát. V každom opakovaní to porovná z predchádzajúcim výsledkom. Keď je nový výsledok lepší, tak nahradí posledný uložený. Vstupom tohto algoritmu sú štyri vstupné dáta:

- 3D mračno bodov, čo je matica troch súradníc x, y a z
- tolerancia prahu vzdialenosti t medzi vybranou plochou a ďalšími bodmi. Jeho hodnota je vo vzťahu s výškovou presnosťou mračna bodov
- *„predvídateľná podpora“* (*forsseable_support*) je maximálny pravdepodobný počet bodov patriacich do tej istej plochy. Je odvodené z hustoty bodov a maximálneho predvídateľného povrchu plochy
- Pravdepodobnosť α je minimálna pravdepodobnosť nájdenia aspoň jednej dobrej sady pozorovaní v N pokusoch. Zvyčajne leží medzi 0.90 a 0.99.

Algoritmus 1 popisuje pseudokód algoritmu RANSAC pre detekciu roviny. V tomto algoritme ε značí percento pozorovaní, ktoré môžu byť chybné, funkcia *pts2plane* vypočíta parametre roviny z troch vybraných bodov. Funkcia *dist2plane* vypočíta vzdialenosti medzi sadou bodov a danou rovinou.

3 Prevzaté z: https://www.sfdm.scad.edu/faculty/mkesson/vsfx319/wip/best_winter2011/ali_jafargholi/project_1.html

Algoritmus 1: RANSAC pre detekciu roviny

```
 1 bestSupport = 0
 2 bestPlane(3,1) = [0,0,0]
 3 bestStd = ∞
 4 i = 0
 5 ε = 1 – forseeable_support / length(point_list)
 6 N  = round(log(1-α) / log(1-(1-ε)^3))
 7 while (i <= N):
 8       pl = pts2plane(j)
 9       dis = dist2plan(pl, point_list)
10       s = find(abs(dis) <= t)
11       st = Standard_deviation(s)
12       if (length(s) > bestSupport or
         (length(s) = bestSupport and st < bestStd)):
13             bestSupport = lenght(s)
14             bestPlan = pl
15             bestStd = st
16             i = i+1
```

3D Houghova transformácia

Klasická Houghova transformácia (2D) [2] je metóda určená na detekovanie parametrizovaných objektov, typicky používaná pre čiary a kruhy. Aj keď veľa postupov využívajúcich túto transformáciu pracuje s klasickými obrázkami na svojom vstupe, tak pri tejto metóde to nie je nutnosť. Roviny sú zvyčajne reprezentované vzdialenosťou ρ od počiatku súradnicového systému a smernice m_x a m_y v smere súradnice x a y. Tento vzťah nám vyjadruje rovnica 2.1.

$$z = m_x x + m_y y + \rho \tag{2.1}$$

Rovina je daná bodom p na rovine, normálovým vektorom n, ktorý je kolmý na rovinu a vzdialenosťou ρ od počiatku súradnicového systému, ktorý sa dá vyrátať pomocou rovnice 2.2.

$$\rho = p.n + p_y n_y + p_z n_z = \rho \tag{2.2}$$

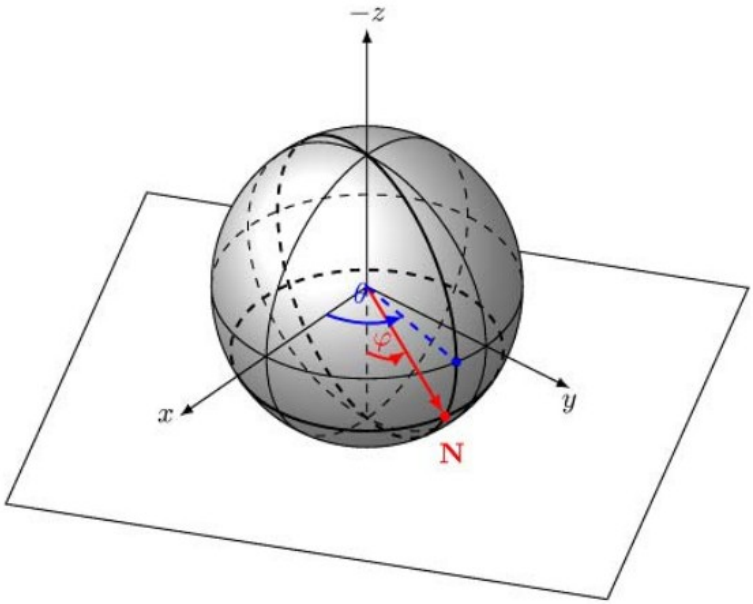
Uvažovaním uhlov medzi normálovými vektormi a súradnicovým systémom, sú súrádnice n faktorizované tak ako je vidieť v rovnici 2.3.

$$p_x.\cos\theta.\sin\varphi + p_y.\sin\varphi.\sin\theta + p_z.\cos\varphi = \rho \tag{2.3}$$

s uhlom θ normálového vektoru na rovine xy a uhlom φ medzi rovinou xy a normálovým vektorom v smere z ako je vidieť na obrázku 2.4. φ, θ a ρ definujú troj dimenzionálny Houghov priestor (φ, θ, ρ), tak že každý bod v Houghovom priestore patrí jednej rovine v $\mathbb{R}^3$.

Pre nájdenie roviny v sade bodov, je potrebné vypočítať Houghovu transformáciu pre každý bod. Pre daný bod p v karteziánskych súradniciach, je potrebné nájsť všetky roviny na ktorých tento bod leží, tj. nájsť všetky φ, θ a ρ, ktoré platia pre rovnicu 2.3. Označením týchto bodov v Houghovom

priestore vedie k 3D sínusovej krivke ako je vidieť na obrázku 2.5. Prienik dvoch kriviek v tomto priestore značí roviny, ktoré sú rotované okolo priamky tvorenej dvoma bodmi. A preto prienik troch kriviek v Houghovom priestore odpovedá polárnym súradniciam, ktoré definujú rovinu klenutú troma bodmi. Na obrázku 2.5 je prienik označený čierne. Keď máme množinu *P*, ktorá obsahuje body v karteziánskych súradniciach. Transformujú sa všetký body $p_i \in P$ do Houghovho priestoru. Čím

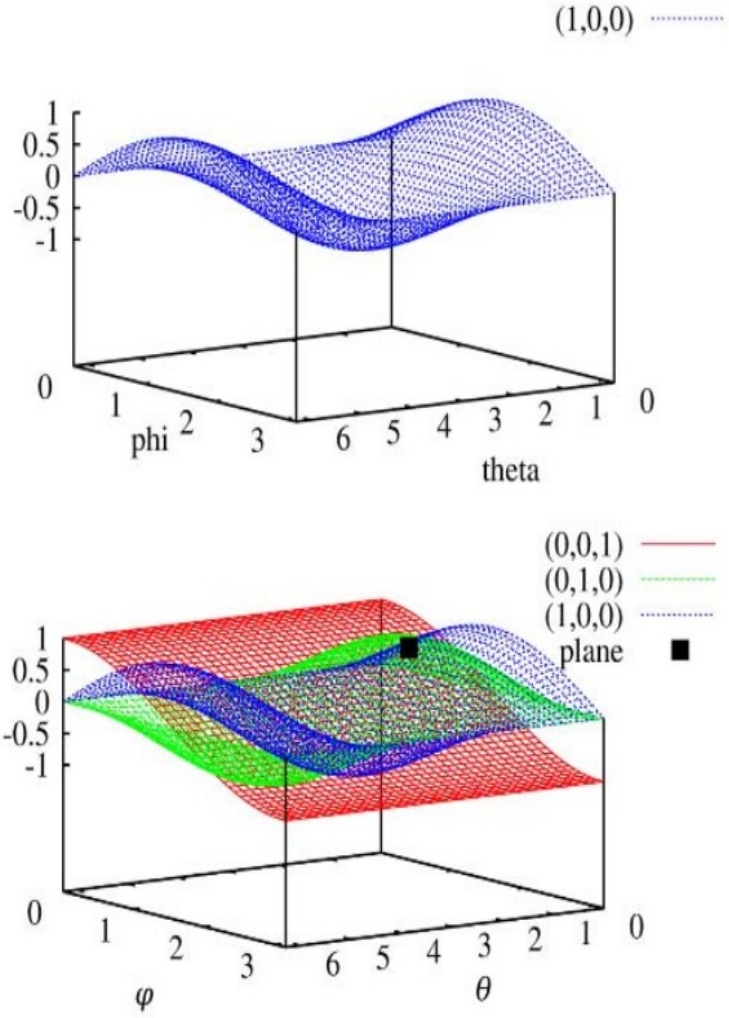


Obrázok 2.4: Normálový vektor v polárnom súradnom systéme [2].

viac kriviek sa pretína v $h_j \in (\theta, \varphi, \rho)$, tým viac bodov leží na rovine reprezentovanej h_j a o to je väčšia pravdepodobnosť, že h_j je získaná z *P*.

Pre praktické aplikácie sa používa diskretizácia Houghovho s *φ*, *θ*, *ρ* rozďelujúcich rozsah každej bunky v príslušiacom smere v Houghovom priestore. Na uloženie všetkých buniek so skóre je potrebná nejaká datová štruktúra. Táto datová štruktúra sa nazýva akumulátor. V nasledujúcom výklade bude inkrementácia bunky znamenať zvýšenie skóre o 1. Pre každý bod p_i sú inkrementované všetky bunky, ktoré sa náležia svojej Houghovej transformácie. Proces inkrementácie je často nazývaný hlasovanie, tj. každý bod hlasuje pre všetky množiny parametrov (*φ*, *θ*, *ρ*), ktoré definujú rovinu, na ktorej leží, tj. keď je eukleidovská vzdialenosť k rovine reprezentoaná stredom bunky menšia ako prah. Bunka s najvýššími hodnotami reprezentuje najviac význačné roviny, rovinu, ktorá pokrýva najviac bodov mračna bodov.

Potom ako všetky bunky hlasovali sú vybrané výherné roviny. Kvôli diskretizácii Houghovho priestoru a šumu vo vstupných datách sa odporúča vyhľadať nielen jednu bunku s maximálnym skóre, ale maximum súm v malom regióne akumulátoru. V procedúre posúvneho okna je definované malé 3D okno, ktoré je navrhnuté tak, aby pokrylo celé rozloženie maxima. Najvýznačnejšia rovina je zhodná so stredným bodom kocky v Houghovom priestore s maximálnou sumou akumulačných hodnôt. Kroky procedúry sú ukázene v algoritme 2.



Obrázok 2.5: Transformácia troch bodov do Houghovho priestoru (φ, θ, ρ) [*2*].

Algoritmus 2: Štandardná Houghova transformácia

```
1 pre každý bod p_i v sade P:
2       pre každú bunku (φ, θ, ρ) v akumulátore A:
3             ak bod p_i leží na rovine definovanou (φ, θ, ρ):
4                   Inkrementuj bunku A(φ, θ, ρ)
5 Vyhľadaj najvýznačnejšie bunky v akumulátore, ktoré definujú
  detekované roviny v P
```

Nevýhodou tejto metódy je vysoká časová výpočetná náročnosť, a preto je skôr nepraktická, a to hlavne pre real-time aplikácie. Preto bolo vymyslených niekoľko variant tejto metódy. K nim patria napríklad náhodná Houghova transformácia a pravdepodobnostná Houghova transformácia, ktorá sa delí na adaptívnu a progresívnu. Podrobnejší popis týchto variant možno nájsť v [2].

3 Problém segmentácie a detekcie objektov v mračne bodov

V tejto kapitole bude popísaný problém segmentácie objektov v mračne bodov, ktorá je dôležitá pri rozpoznávaní a detekcii objektov. Ďalej budú zhrnuté niektoré typy metód, ktoré sa používajú na segmentáciu a bude bližšie popísaný algoritmus Euklidovo zhlukovanie. Po získaní segmentov, či možných objektov je možné prejsť k detekcii týchto segmentov. V tejto kapitole budú tiež zhrnuté niektoré typy metód, ktoré sa používajú na detekciu objektov v mračne bodov a bližšie bude popísaná modifikovaná forma algoritmu ICP, a to Picky ICP.

3.1 Segmentácia objektov v mračne bodov

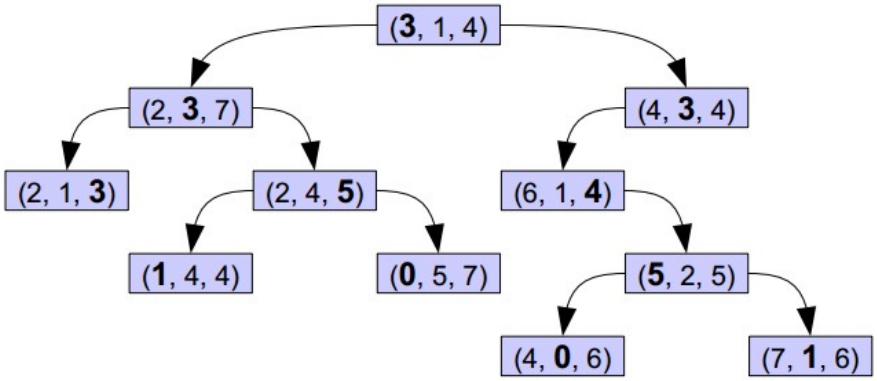
Na segmentáciu v mračne bodov existuje mnoho algoritmov, medzi ktoré patrí napríklad algoritmus založený na K-means [http://www.academia.edu/2349096/3D_Point_Cloud_Segmentation], ďalej algoritmus založený na rezoch v grafe [Y. Boykov and G. Funka-Lea. Graph cuts and efficient nd image segmentation. International Journal of Computer Vision, 70(2):109– 131, 2006.], potom algoritmus založený na hľadaní *k*-najbližších susedov, čím je Euklidovo zhlukovanie alebo metóda Mean Shift [http://www.cs.stolaf.edu/wiki/images/f/fb/080390_bg_paper.pdf]. V tejto kapitole si priblížíme metódu Euklidovo zhlukovanie a taktiež štruktúru KdTree, ktorú využíva. Táto štruktúra sa v segmentácii používa na vyhľadanie najbližších susedov.

KdTree

KdTree (k-dimensional tree) [5], taktiež *k*-dimenzionálny strom, je generalizáciou binárneho vyhľadávacieho stromu, ktorý ukladá body v *k*-dimenzionálnom priestore. Na obrázku 3.1 je znázornený *k*-dimenzionálny strom, ktorý ukladá body v troj-rozmernom priestore. Je možné si všimnúť, že v každej úrovni tohoto stromu je vyznačená komponenta každého uzlu tučným písmom. Keď sa budú indexovať komponenty každého uzlu od nuly, tj. prvá komponenta má index nula, druhá komponenta index jedna, atď., tak v úrovni *n* tohoto stromu je každá komponenta s indexom *n%3* vyznačená tučne. Dôvodom, že tieto hodnoty sú zvýraznené je, že každý uzol sa chová ako uzol v binárnom vyhľadávacom strome, ktorý rozlišuje na základe tejto zvýraznenej komponenty. Napríklad prvá komponenta každého uzlu v ľavom podstrome je menšia ako prvá komponenta v koreni stromu, kým prvá komponenta každého uzlu v pravom podstrome je väčšia alebo rovná ako prvá komponenta v koreni stromu. Podobne sa uvažuje aj v ľavom podstrome *k*-dimenzionálneho stromu. Koreň tohoto stromu má hodnotu (2, **3**, 7). Všetky komponenty na druhej pozícii v jeho ľavom podstrome sú menšie ako tri. Podobne v pravom podstrome je vidieť, že druhá komponenta každého uzlu je najmenej tri. Takto to pokračuje v celom strome.

Táto šturktúra sa dá využiť na ukladanie kolekcie bodov v Karteziánskej rovine v troj-dimenzionálnom priestore. Najzaujímavejšou operáciou využitia *k*-dimenzionálneho stromu je

hľadanie najbližšieho suseda, ktorá je bližšie popísana v [5]. Túto operáciu využíva segmentačný algoritmus Euklidovo zhlukovanie, ktorý je popísaný nižšie.



Obrázok 3.1: Príklad KdTree [5].

Euklidovo zhlukovanie

Euklidovo zhlukovanie [6] je segmentačná metóda, ktorá delí mračno bodov na niekoľko častí, z ktorej každá časť obsahuje nejaký izolovaný objekt. To je docielené porovnávaním Euklidovskej vzdialenosti medzi susediacimi bodmi. Zhlukovanie je vykonávané pomocou algoritmu *k*-najbližších susedov. Hľadanie najbližších susedov začína vybraním náhodného bodu z aktuálneho mračna bodov a vyrátaním Euklidovskej vzdialenosti medzi týmto bodom a jeho najbližšími susedmi. Body, ktoré sú vzdialené do istého prahu sú označené ako časť objektu. Prehľadávanie zastaví, keď sa nenájde žiadny sused, ktorého vzdialenosť by bola menšia ako daný prah. Zatiaľ nájdené body sú označené do jednej skupiny, sú súčasťou jedného objektu, a prehľadávanie začína znova odstránením už nájdenej skupiny bodov z mračna bodov a náhodným vybraním ďalšieho neoznačeného bodu. Zhlukovanie zastavuje, keď všetky body sú označené. Pri hľadaní najbližšieho suseda sa použije binárna KdTree implementácia. Tento prístup rozdelí mračno bodov do binárneho stromu a umožní ľahké a rýchle hľadanie najbližšieho suseda. Pseudokód tohoto algoritmu je popísaný algoritmom 2.

Algoritmus 2: Segmentácia mračna bodov pomocou Euklidovho zhlukovania[4].

```
1 vytvor KdTree reprezentáciu pre vstupné mračno bodov P
2 vytvor prázdny zoznam zhlukov C
3 vytvor frontu Q , ktorá obsahuje body, ktoré musia byť skontrolované
4 pre každý bod p_i∈P :
5       pridaj p_i do Q
6       pre každý bod p_i∈Q :
7             hľadaj množinu P_k^i susedných bodov bodu p_i v priestore s
              polomerom r<d_th
```

4 Prevzaté z: http://www.pointclouds.org/documentation/tutorials/cluster_extraction.php

```
8           pre každého suseda p_i^k ∈ P_i^k skontroluj či bol daný bod
            spracovaný a nebol pridaný do Q
9       keď všetky body v Q boli spracované pridaj Q do zoznamu
        zhlukov C a vyprázdni Q
10  algoritmus končí keď všetky body p_i ∈ P boli spracované a teraz sú
    súčasťou zoznamu zhlukov C
```

3.2 Detekcia objektov v mračne bodov

Na detekciu 3D objektov existuje tiež mnoho metód, medzi ktoré patria napríklad algoritmy založené na lokálnych [10, 11] alebo globálnych príznakoch [12, 13]. Medzi ďalšie metódy patria algoritmy založené na registrácii modelov. Tieto algoritmy sa snažia nájsť čo najväčšiu zhodu medzi známymi modelmi objektov rôznymi transformáciami k skúmaným objektom. K týmto metódam patrí napríklad algoritmus ICP, ktorý bude popísaný nižšie.

ICP

Algoritmus ICP (Iterative Closest Point) [7][8] slúži na registráciu dvoch sád bodov. V základe tento algoritmus iteratívne vykonáva dve operácie až kým nekonverguje. Prvá operácia pozostáva z nájdenia najbližšieho bodu z prvej sady bodov pre každý bod v druhej sade bodov. V druhej operácii je pohyb medzi bodmi odhadnutý pomocou zodpovedajúcich párov bodov. Ako väčšina algoritmov slúžiacich na minimalizáciu tak aj algoritmus ICP potrebuje dobrú inicializáciu. Inicializácia môže byť vykonaná použitím znalosti o pozícii 3D senzorov alebo užívateľom definovaným vstupom. Ďalej bude popísaná modifikácia tohoto algoritmu, tzv. Picky ICP algoritmus [9].

Riešenie registrácie pre množinu 3D datových bodov *A* a množiny modelových bodov *B* môže byť dané dvoma rovnocennými spôsobmi. Buď môže byť špecifikované množinou párov bodov *C*, kde:

$$C = \{(i, j) | a_i \in A \, a \, b_j \in B\} \quad , \tag{3.1}$$

kde *a* je bod z množiny datových bodov, *b* je bod z množiny modelových bodov a *i*,*j* sú indexy bodov v jednotlivých množinách *A* a *B*. Alebo môže byť použitý pohyb (*R, t*) zarovnávajúci tieto dve sady bodov. Pohybové parametre zahŕňajú rotačnú maticu $R \in \mathbb{R}^{3x3}$ a posuvný vektor $t \in \mathbb{R}^3$.

Pre získanie párov bodov musia byť vybrané kontrolné body zo sád bodov. Štandardný ICP algoritmus používa jednoduchú stratégiu pre výber kontrolných bodov. Všetky datové body sú použité ako kontrolné body. Tento algoritmus bol rozšírený pridaním hierarchického výberu bodov. Najprv každý 2^h-ty bod je vybraný ako kontrolný bod, kde $h+1$ je číslo udávajúce úroveň hierarchie. Po konvergencii registračného algoritmu pre jednu množinu kontrolných bodov výpočet pokračuje na ďalšej úrovni hierarchie. Predovšetkým pre veľké sady bodov, toto rozšírenie urýchli výpočet.

Prvá operácia v hlavnej smyčke ICP algoritmu je vypočítanie párov bodov. Pre každý kontrolný bod zo sady datových bodov je nájdený najbližší modelový bod pomocou algoritmu hľadania najbližšieho suseda. Tento krok registračného algoritmu je výpočetne najnáročnejší, a preto

predstavovaná metóda Picky ICP používa pre maximálny výkon vysoko optimalizovaný algoritmus hľadania najbližšieho suseda využívajúci štruktúru KdTree popísanú v kapitole 3.1.

Po nájdení zodpovedajúceho bodu pre každý kontrolný bod, môžu byť chybné páry bodov vypustené. Keď nie je dostupná žiadna doplňujúca informácia, len vzdialenosť dvoch bodov v páre, tak táto informácia môže byť použitá na rozlíšenie dobrých bodov od extrémov. Hlavnou myšlienkou je robustne odhadnúť smerodajnú odchýlku vzdialeností, a vypustiť páry bodov s vzdialenosťou väčšou ako vybraný násobok tejto štandardnej odchýlky. Ďalším rozšírením implementovaným v Picky ICP je možnosť zabrániť tomu, aby modelový bod bol prítomný vo viac ako jednom páre. Keď je takýto modelový bod nájdený, tak všetky páry okrem tých s najmenšou vzdialenosťou sú vypustené. Táto metóda je dôležitá najmä keď sady bodov *A* a *B* sú prekryté len čiastočne. Vypustenie párov bodov zvýši robustnosť zašumenia na súradniciach 3D bodov a extrémov, ale má to nevýhodu v spomalení konvergencie algoritmu.

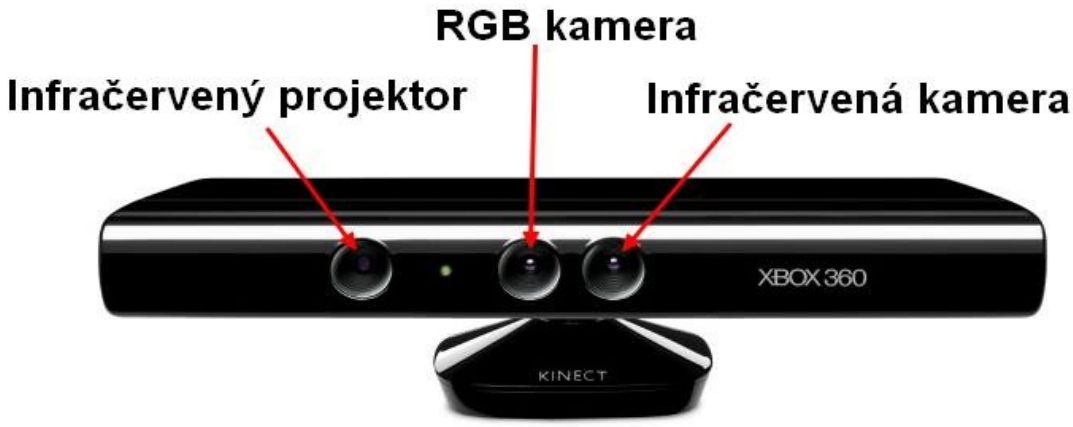
Výpočet pohybu je druhá operácia v hlavnej smyčke algoritmu ICP. Na zmeranie chybovosti sa použije suma štvorcových vzdialeností zodpovedajúcich párom bodov ako znázorňuje rovnica 3.2.

$$(R', t') = \underset{R,t}{argmin} \sum_{(i,j)\in C} \left\| b_j - Ra_i - t \right\|^2 \quad . \tag{3.2}$$

4 Kinect

Nízko nákladové 3D senzory sú atraktívnou alternatívou oproti drahým laserovým skenerom v aplikačných oblastiach ako napríklad vnútorné mapovanie, robotika a forenzné vedy. V spotrebiteľskej vrstve je známy senzor Microsoft Kinect (ďalej Kinect), ktorý bol vydaný v novembri v roku 2010. Kinect bol určený predovšetkým pre interakciu človeka a počítačovej hry, aj keď charakteristika dát zachytených Kinectom zaujala pozornosť hlavne výskumníkov z oboru 3D modelovania a mapovania interiéru. Oblasti využitia Kinectu v počítačovom videní možno rozdeliť na nasledovné [3]:

- Sledovanie a rozpoznávanie objektov
  - Detekcia a sledovanie objektov
  - Rozpoznanie objektov a scén
- Analyzovanie ľudskej aktivity
  - Odhadovanie pozície človeka
  - Rozpoznanie aktivity človeka
- Analyzovanie pohybu rúk
  - Detekcia ruky
  - Odhadovanie pozície ruky
  - Klasifikácia gestikulácie
- Interiérové 3D mapovanie



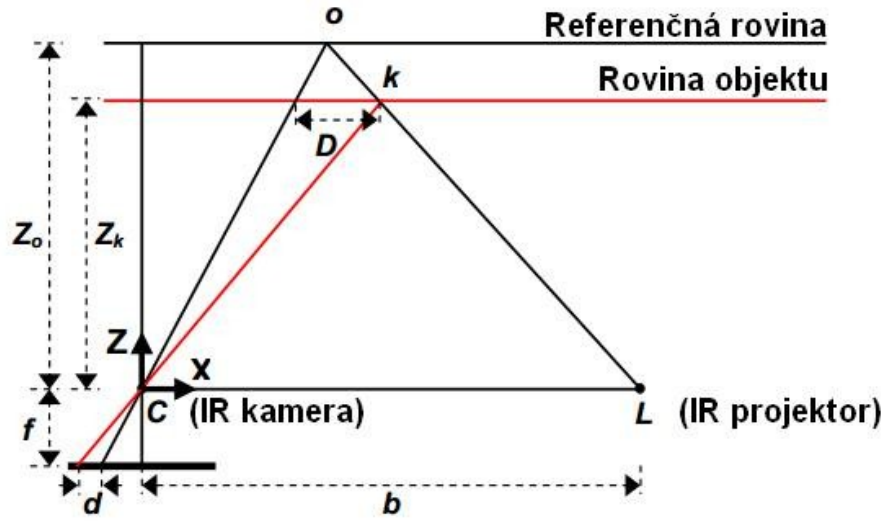
Obrázok 4.1: Popis častí Kinectu.[5]

Senzor kinectu zaznamenáva hĺbkové a farebné obrázky súčasne rýchlosťou 30 snimkov za sekundu. Integrácia hĺbkových a farebných dát vyúsťuje vo farebné mračno bodov, ktoré obsahuje okolo 300 000 bodov v každom snímku. Registráciou po sebe idúcich obrazov možno získať väčšiu hustotu bodov, ale taktiež vytvoriť kompletné mračno bodov prostredia v reálnom čase.

5 Prevzaté z: http://www.gameinformer.com/b/news/archive/2013/03/06/microsoft-kinect-demonstrates-new-hand-grip-technology.aspx

4.1 Princíp merania

Ako vidieť na obrázku 4.1 senzor Kinectu pozostáva z infračerveného laserového projektoru, infračervenej kamery a RGB kamery. Vynálezcovia popisujú meranie hĺbky ako triangulačný proces. Laser emituje lúč, ktorý je difrakčnou mriežkou rozdelený do viacerých lúčov, aby vytvoril konštantný vzor škvŕn premietaných na scénu. Tento vzor je zachytený infračervenou kamerou a je korelovaný vzhľadom na referenčný vzor. Referenčný vzor sa získa tým, že zachytí plochu v známej vzdialenosti od senzoru a je uložený v pamäti senzoru. Keď je škvrna premietaná na objekt, ktorého



Obrázok 4.2: Schématická reprezentácia relácie hĺbka rozdiel [*4*].

vzdialenosť od senzoru je menšia alebo väčšia ako vzdialenosť od referenčnej plochy, pozícia škvrny v infračervenom obraze bude posunutá v smere základni medzi projektorom laseru a centrom perspektívy infračervenej kamery. Tieto posuny su merané pre všetky škvrny pomocou jednoduchej korelačnej procedúry, ktorý vracia rozdielnosť obrazu. Pre každý pixel môže byť vzdialenosť od senzoru vyrátaná zo zodpovedajúceho rozdielu [4].

4.2 Matematický model

Na obrázku 4.2 vidieť reláciu medzi vzdialenosťou bodu objektu *k* od senzoru relatívnu k referenčnej ploche a nameraného rozdielu *d*. Na vyjadrenie 3D súradníc bodov objektu uvažujeme hĺbkový súradnicový systém s bázou v centre perspektívy infračervenej kamery. Osa *Z* je ortogonálna k ploche obrazu smerom k objektu, osa *X* je kolmá na osu *Z* v smere od základne *b* medzi stredom infračervenej kamery a laserového projektoru. Predpokladajme, že objekt je na referenčnej ploche vo vzdialenosti Z_0 k senzoru a škrvna na objekte je zachytená na obrazovom plátne infračervenej kamery. Keď je objekt posunutý bližšie k (alebo ďalej od) senzoru, pozícia škrvny na obrazovom plátne bude

nahradená v smere *X*. Toto je namerané v priestore obrazu ako rozdiel *d* náležiaci bodu *k* v priestore objektu. Z podobnosti trojuholníkov vyplýva vzťah znázornený rovnicami 4.1 a 4.2:

$$\frac{D}{b}=\frac{Z_0-Z_k}{Z_0} \tag{4.1}$$

$$\frac{d}{f}=\frac{D}{Z_k} \tag{4.2}$$

kde Z_k je vzdialenosť (hĺbka) bodu *k* v priestore objektu, *b* je dĺžka základne, *f* je ohnisková vzdialenosť infračervenej kamery, *D* je posunutie bodu *k* v priestore objektu a *d* je pozorovaný rozdiel v priestore objektu. Dosadením *D* z rovnice 4.2 do rovnice 4.1 a vyjadrením Z_k z hľadiska ostatných premenných dostávame vzťah:

$$Z_k=\frac{Z_0}{1+\frac{Z_0}{fb}d} \tag{4.3}$$

Rovnica 4.3 je základný matematický model pre deriváciu hĺbky z pozorovaného rozdielu za predpokladu, že konštantné parametre Z_0, *f* a *b* môžu byť určené kalibráciou. Súradnica *Z* daného bodu spolu s *f* definuje rozsah zobrazenia pre tento bod. Polohové súradnice každého bodu objektu môžu byť vypočítané z obrazových súradníc a rozsahu.

$$X_k=-\frac{Z_k}{f}(x_k-x_0+\delta x) \tag{4.4}$$

$$Y_k=-\frac{Z_k}{f}(y_k-y_0+\delta y) \tag{4.5}$$

V rovniciach 4.4 a 4.5 vyjadrujú x_k a y_k obrazové súradníce bodu, x_0 a y_0 sú súradníce hlavného bodu a δx a δy sú korekcie skreslenia objektívu, pre ktoré existujú rôzne modely s rôznymi koeficientmi. Poznamenajme, že súradnicový systém obrazu je paralelný s líniou základne a teda aj s hĺbkový súradnicovým systémom [4].

5 PR2 a Robot Operating System

V tejto kapitole budú bližšie popísané základné kamene systému, na ktorom sa výsledná práca vyvíja. Je to framework Robot Operating System[6] (ďalej ROS), prispôsobený na vývoj softwaru pre robotov. Obsahuje veľa nástrojov, knižníc a konvencií, ktoré majú za úlohu zjednodušiť vytváranie komplexných a robustných aplikácií v širokej škále robotických platforiem. ROS je vyvíjaný na platforme na bázi Unixu, a to Ubuntu.

5.1 Základné pojmy

Balíček

Aplikácie vytvárané v ROS sú organizované v balíčkoch. Balíček môže obsahovať uzly, nezávislé knižnice, datové sady, konfiguračné súbory, softvér tretích strán alebo niečo iné, čo logicky predstavuje užitočný modul. Cieľom balíčkov je poskytovať užitočnú funkcionalitu konceptu znovupoužiteľnosti a jednoduchosti v použití iným softvérom.

Balíčky sa vytvárajú jednoducho, či už ručne alebo pomocou nástrojov na to určených. Balíčky sú najmenšou jednotkou zostavenia a vydania projektu. ROS balíček je vlastne adresár, ktorý obsahuje viac adresárov a súborov. Medzi základné súbory patrí *package.xml*. Tento súbor definuje vlastnosti balíčku ako názov balíčku, číslo verzie, autorov, editorov a závislosti na ďalších balíčkoch. Ďalším dôležitým súborom je *CMakeLists.txt*, ktorý je vstupom pre zostavovací systém *CMake*, ktorý má za úlohu zostaviť balíček. Tento súbor obsahuje popis ako balíček zostaviť a kam ho nainštalovať. Balíček ďalej obsahuje nasledovné zložky:

- msg/
- src/nazov_balicku
- srv/
- scripts/
- include/nazov_balicku
- launch

Zložka *msg* obsahuje súbory typu správa (message), zložka *src* obsahuje zdrojové kódy balíčku (uzly), adresár *srv* obsahuje súbory typu služba (service), adresár *scripts* obsahuje spustiteľné skripty, zložka include obsahuje externé balíčky použité vo vytváranom balíčku a zložka launch obsahuje súbory typu launch. Použitie a obsah jednotlivých súborov bude popísaný nižšie.

Správa

Uzly komunikujú medzi sebou publikovaním správ do tém. Správa je jednoduchá datová štruktúra, ktorá môže obsahovať štandardné datové typy alebo zložitejšie vnorené šturktúry. ROS používa zjednodušenú formu na popis hodnôt týchto dát. Tento popis umožňuje ďalším nástrojom automatickú generáciu zdrojového kódu pre správy v rôznych jazykoch. Popis správ je uložený v súboroch s príponou *.msg* v podadresári *msg* daného balíčku. Týmto je daný typ správy v systéme. Skladá sa z

6 Čerpané z: http://www.ros.org/

názvu balíčku a názvu správy oddelený symbolom „/“, kde každý riadok tohoto súboru pozostáva z typu a názvu, ktoré sú oddelené medzerou.

Uzol

Uzol je proces, ktorý vykonáva potrebný výpočet. Uzly sú spojené do grafu a komunikujú medzi sebou pomocou tém, služieb alebo parametrového serveru. Jednotlivé aplikácie sú zložené z väčšieho počtu uzlov. Napríklad, jeden uzol ovláda plánovanie cesty, jeden uzol lokalizáciu a ďalší ovláda motory kolies robota. Použitie uzlov v ROS poskytuje niekoľko výhod vrámci celého systému. Je odolný proti chybám a pády sú izolované na jednotlivých uzloch. Zložitosť kódu je znížená, rozdelením výsledného produktu na viac častí. Z menších podsystémov sa postupne skladá komplexnejší systém. Všetky spustené uzly majú svoj názov v grafe prostriedkov, ktorý jednoznačne identifikuje daný uzol.

Téma

Téma, tzv. topic, je kanál cez ktorý si uzly vymieňajú správy. Topicy používajú sémantiku publikovania (publish) a odoberania (subscribe). Vo všeobecnosti, uzly nevedia s kým komunikujú len odoberajú dáta z kanálu alebo prispievajú dátami do tohoto kanálu. Môže existovať naraz viac uzlov, ktoré publikujú alebo odoberajú z jedného topicu. Každá téma je striktne typovaná, čiže cez ňu môžu prúdiť správy len jedného typu.

Služba

Služba je ďalší spôsob ako môžu spolu uzly komunikovať. Je to založené na princípe *RPC* (remote procedur call), vzdialeného volania procedúr – žiadosť a odpoveď. Sužba je definovaná párom správ, jedna pre žiadoť a druhá pre odpoveď. Uzol poskytuje službu pod nejakým názvom a nejaký ďalší uzol volá túto službu poslaním správy, ktorú tvorí žiadosť a čaká na odpoveď. Služby sú definované pomocou súborov s príponou *.srv* a sú uložené v balíčku v podaresári *srv*, čím definujú typ služby rovanko ako pri definícii typu pri správach.

Parametrový server

Parametrový server je zdieľaný, viac účelový slovník, ktorý je dostupný cez API. Uzly používajú tento server na uloženie a načítanie parametrov pri behu. Vzhľadom k tomu, že nie je určený pre vysoký výkon, je vhodné ho použiť pre statické a nebinárne data, ako sú konfiguračné parametre. Má byť globálne viditeľný, aby nástroje mohli ľahko kontrolovať stav konfigurácie systému a upraviť ho ak je to nutné.

Parametre sú pomenované podľa klasickej konvencie pomenovania v ROS. To znamená, že ROS parametre majú hierarchiu, ktorá sa zhoduje s menným priestorom pre témy a uzly. Táto hierarchia slúži na ochranu parametrových mien od kolízie. Hierarchická schéma taktiež umožňuje, aby parametre boli prístupné individuálne alebo ako strom (slovník, či asociatívne pole).

5.2 Roslaunch

Balíček roslaunch je nástroj pre jednoduché spustenie viacerých uzlov lokálne alebo vzdialene cez SSH, a taktiež nastavuje parametre parametrového serveru. Prijíma jeden alebo viac konfiguračných súborov vo formáte XML, tzv. launch súbor. V tomto súbore sú špecifikované parametre, ktoré budú nastavené a ktore uzly a s akými argumentmi budú spustené a taktiež stroje na ktorých dané uzly budú bežať.

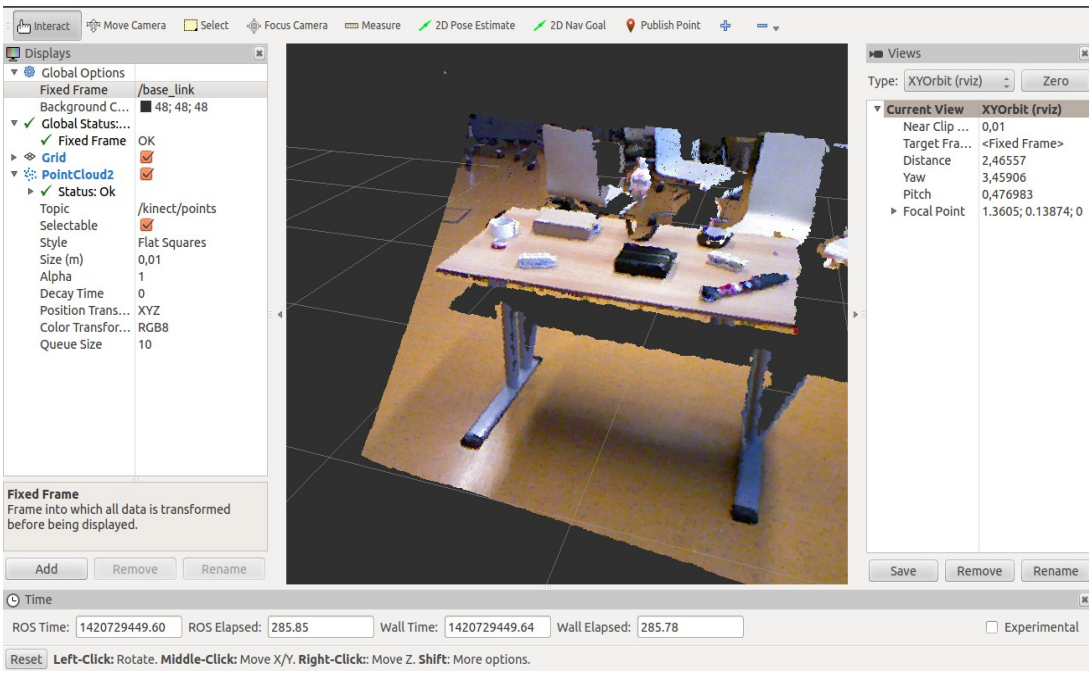
5.3 Rosbag

Balíček rosbag[7] poskytuje nástroj pre prácu s *bag* súbormi. Je to typ súborov, do ktorého sa ukladajú ROS správy z jedného alebo viacerých ROS topicov v serializovanej forme. Tieto súbory potom môžu byť znova prehrané v ROS do toho istého topicu z ktorého boli nahrané alebo dokonca sa môžu premapovať na iný topic.

Súbory *bag* sú primárnym prostriedkom v ROS pre logovanie dát, čo znamená, že majú rozličné offline využitie. Výskumníci využívajú tieto súbory na nahrávanie datových sád, ich vizualizáciu, popis a ukladanie pre budúce použitie.

5.4 Rviz

Vizualizácia senzorických informácií je dôležitou súčasťou vývoju a ladenia balíčkov vyvíjaných v ROS. Na tento účel v systéme ROS slúži nastroj rviz (ROS visualization), ktorý je 3D vizualizátorom



Obrázok 5.1: Prostredie nástroja rviz s vizualizáciou pracovnej dosky v mračne bodov.

7 Čerpané z: http://wiki.ros.org/Bags

pre zobrazenie senzorických dát a stavových informácií. Pomocou tohoto nástroja je možné zobraziť reprezentácie senzorických hodnôt prechádzajúcich ROS topicmi a to spolu s datami z rôznych kamier, ako napríklad pointcloud zo stereo kamery alebo Kinectu. Prostredie nástroja rviz spolu s vizualizáciou pracovnej dosky v mračne bodov získaného z Kinectu je znázornený na obrázku 5.1.

5.5 Nástroje ROSu pre prácu s mračnom bodov a detekciu objektov na pracovnej doske

PCL

PCL[8] (Point Cloud Library) je open source knižnica pre spracovanie 2D/3D obrazov a mračna bodov. Tento framework obsahuje mnoho algoritmov vrátane filtrovania, odhadu rysov, rekonštrukcie povrchu, registrácie a segmentácie. Tieto algoritmy môžu byť použité, napríklad na filtrovanie odľahlých hodnôt pri zašumených datách, zoskupenie 3D mračna bodov, segmentovanie relevantných častí scény, extrahovanie kľúčových bodov a vypočítanie deskriptorov na rozpoznanie objektov založených na ich geometrickom vzhľade a vytvorenie povrchov z mračna bodov a ich vizualizácia.

Je to multiplatformný framework, ktorý sa dá použiť na systémoch Linux, MacOS, Windows, Android a iOS. Pre zjednodušenie vývoja je PCL rozdelená do viacerých menších knižníc, ktoré sa dajú skompilovať samostatne. Táto modularita je dôležitá pri distribúcii PCL na platformách s menšými výpočetnými, či pamäťovými možnosťami .

ROS balíček tabletop_object_detector

Tento balíček[9], ktorý bol vydaný pre ROS distribúciu *Groovy*, slúži na segmentáciu a jednoduché rozpoznávanie objektov pre obmedzené scény. Používa dve hlavné komponenty pre vnímanie objektov, a to segmentáciu objektov a rozpoznanie objektov. Pre svoju funkcionalitu používa knižnicu PCL, ktorá je popísaná nižšie.

ROS balíček household_object_database

Tento balíček[10], slúži ako rozhranie pre databázu obsahujúcu 3D modely vo forme trojuholníkových meshov. Poskytuje operácie nad databázou vo forme služieb. Medzi tieto služby patrí napríklad vkladanie nového modelu do databáze, výber modelu z databáze, vrátenie identifikátorov modelov, ktoré databáza obsahuje a iné.

Trojuholníkový mesh je typ polygonového meshu, ktorý obsahuje množinu trojuholníkov, ktoré sú pospájané hranami alebo vrcholmi a spolu definujú tvar objektu. Balíček *household_object_database* pracuje s meshmi uloženými v súbore s príponou *.ply.*

8 Čerpané z: http://pointclouds.org/about/

9 Dostupné na: http://wiki.ros.org/tabletop_object_detector

10 Dostupné na: https://github.com/ros-interactive-manipulation/household_objects_database

5.6 Robot PR2

Medzi roboty ktoré používajú ROS, patrí aj robot PR2[11], ktorý je vyvinutý spoločnosťou Willow Garage. Tento robot sa nachádza aj na našej fakulte a experimenty a testovanie výsledného nástroja tejto práce budú prebiehať práve na ňom. Robot PR2 je znázornený na obrázku 5.2.

Obrázok 5.2: Robot PR2. [12]

Obrázok 5.3: Možnosti pohybu robota PR2. [13]

Skladá sa zo všesmerovej základne a z teleskopického trupu, čo mu umožňuje pohyb vo všetkých smeroch, ako je vidieť na obrázku 5.3. Ďalej sa skladá z dvoch robotických ramien s ôsmymi stupňami voľnosti a hlavy.

Záladňa robota je vybavená dvoma servermi s procesorom i7 Xeon, ktorý má osem jadier, ďalej pamäť RAM o veľkosti 24 GB, interný pevný disk o veľkosti 500 GB a externý pevný disk o veľkosti 1,5 TB. Základňa je ďalej vybavená senzorom lidar Hokuyo UTM-30LX. Trup robota je umiestnený na základni a jeho výška je nastaviteľná v rozmedzí 1330 - 1645mm (merané od podlahy až po hlavu). Vo vrchnej časti trupu sa nachádza lidar toho istého typu ako na základni. Obe robotické ramená sa skladajú z troch častí, a to hornej časti ramena, zápestia a chápadla. Nad zápästím má umiestnenú farebnú kameru, ktorá umožňuje pohľad na prácu chápadla. Chápadlo ďalej obsahuje troj-osý akcelerometer. Hlavou robota PR2 je možné otáčať v rozsahu 350° a nakláňať v rozsahu 115°. V hlave má zabudované dva páry stereokamier s úzkym a širokým záberom, ďalej päť megapixelovú farebnú kameru, LED projektor spúšťaný so stereokamerou s úzkym záberom a Kinect. Maximálna rýchlosť jeho pohybu je 1 m/s.

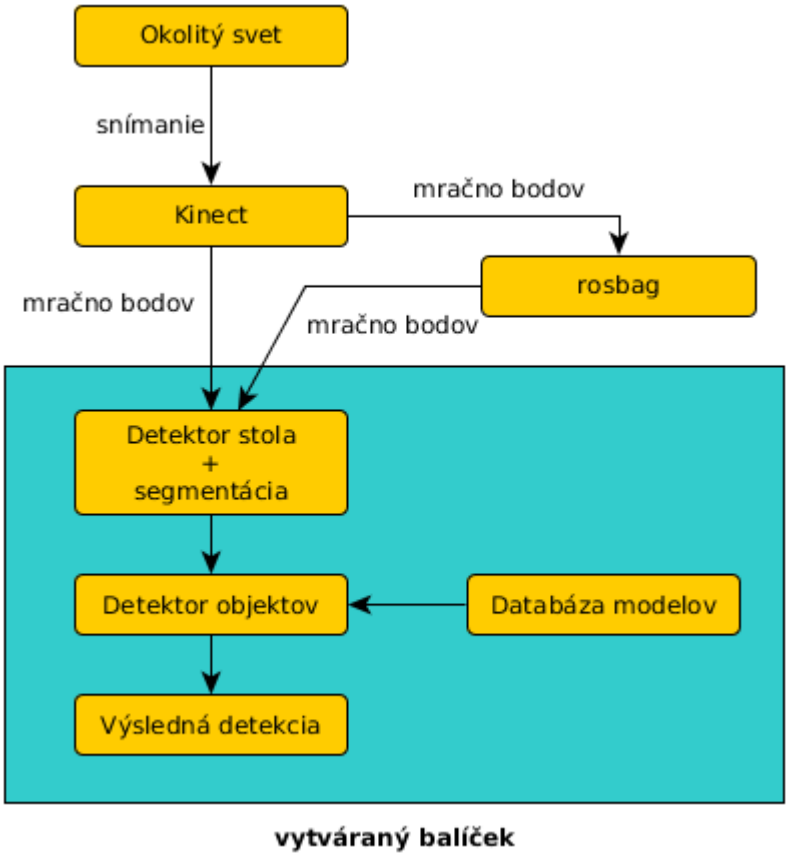
11 Čerpané z: https://www.willowgarage.com/pages/pr2/specs

12 Prevzaté z: https://www.willowgarage.com/blog/2010/01/15/pr2-beta-program-call-proposals-out

13 Prevzaté z: https://www.willowgarage.com/pages/pr2/design

6 Návrh ROS modulu pre detekciu objektov na pracovnej doske

Návrh riešenia tejto práce je založený na predchádzajúcej analýze problému detekcie objektov na pracovnej doske. Cieľom je vytvoriť balíček pre systém ROS a distribúciu *Hydro*, ktorý dokáže spracovať hĺbkové data z Kinectu, následne rozpoznať a detekovať v mračne bodov pracovnú dosku a rozpoznať objekty ležiace na nej. Na obrázku 6.1 je pomocou vývojového diagramu znázornený celý systém detekcie objektov na pracovnej doske a modrým rámčekom je označená časť, ktorá náleží vyvíjanej aplikácii. Z diagramu je vidieť, že výsledný balíček sa bude skladať z dvoch častí. Prvá časť bude plniť úlohu detektoru stolu spolu so segmentáciou a druhá časť zas detektoru objektov. V tejto kapitole si postupne popíšeme návrh na implementáciu daných detektorov a popíšeme knižnice a moduly, ktoré budú použité.



Obrázok 6.1: Vývojový diagram aplikácie

Základnou ideu je vytvoriť ROS balíček, ktorý bude obsahovať dva uzly. Prvý uzol bude detekovať pracovnú dosku a bude vykonávať segmentáciu a bude bližšie popísaný v kapitole 6.1. Ďalší uzol bude na základe výstupu z prvého uzlu vykonávať detekciu objektov a bude bližšie popísaný v kapitole 6.2.

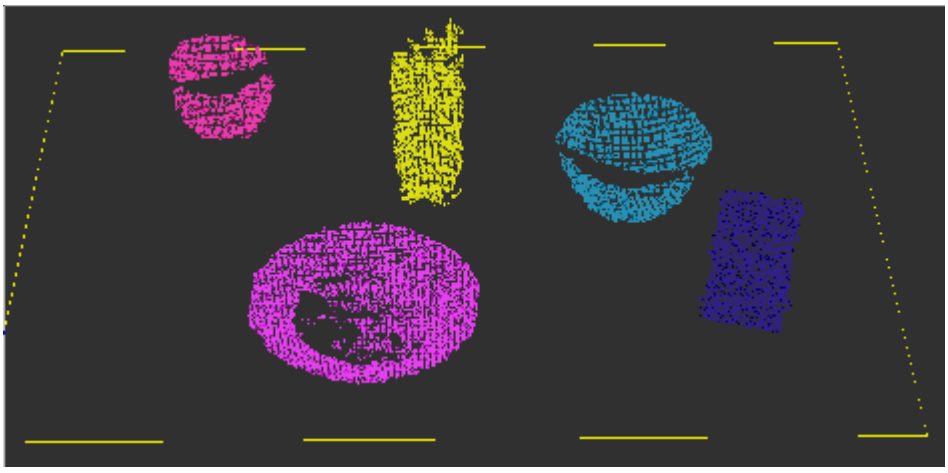
6.1 Detektor pracovnej dosky a segmentácia objektov

Pre detekciu pracovnej dosky a segmentáciu bude vytvorená služba, ktorá bude fungovať na základe služby *tabletop_segmentation* z balíčku *tabletop_object_detector*, ktorý bol spomínaný v kapitole 5.5 a to je nasledovne. Vstupným parameterom bude mračno bodov nasnímané pomocou Kinectu alebo z bagfile. Potom sa vykoná segmentácia, ktorá má nasledovné kroky:

- stôl je detekovaný nájdením dominantnej plochy v mračne bodov využitím metódy RANSAC popísanej v kapitole 2.2.
- body nad stolom sa považujú za body, ktoré patria uchopiteľným objektom. Na tieto body sa použije zhlukovacia metóda Eulerovo zhlukovanie popísaná v kapitole 3.1, ktorá rozdelí body na zhluky, kde každý zhluk predstavuje jeden objekt.

Výstupom tohoto uzlu budú body detekovaného stola a body jednotlivých zhlukov. Na obrázku 6.3 je vidieť výsledok segmentácie, kde žltá čiarkovaná čiara značí obrysy pracovnej dosky a ostatné farebné mračná bodov sú jednotlivé zhluky rozpoznané segmentátorom.

Obrázok 6.2: Scéna objektov

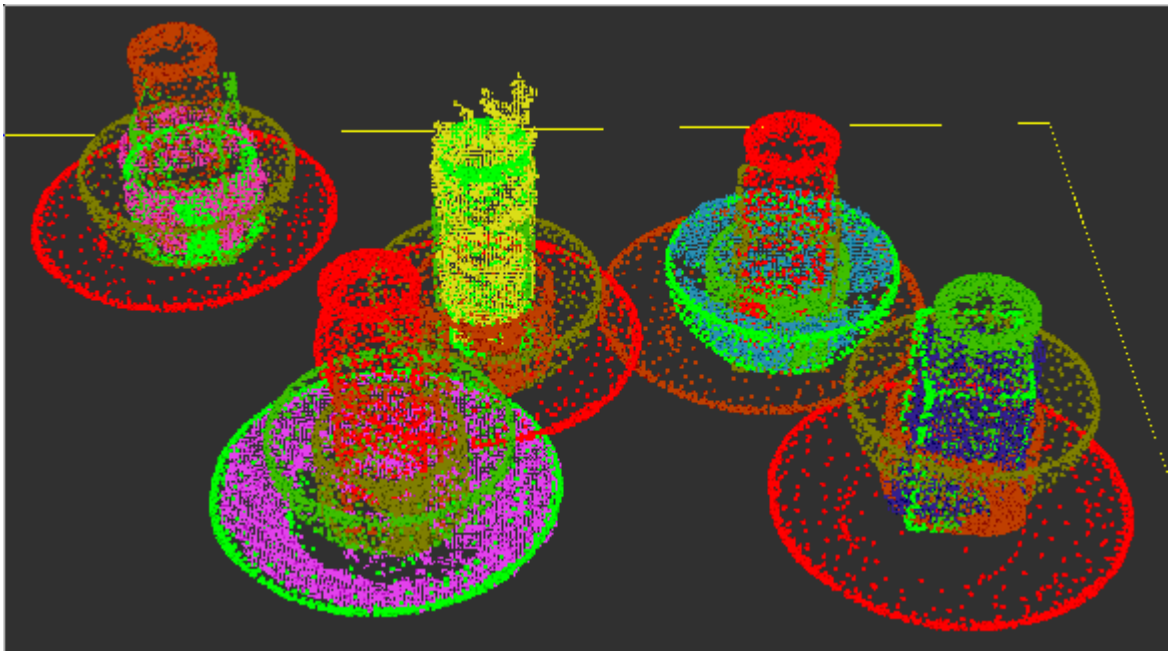


Obrázok 6.3: Výsledok detekcie stola a segmentácie nad scénou z obrázku 6.2

6.2 Detektor objektov

Tento uzol bude mať za úlohu detekovať objekty z mračna bodov. Vstupom bude databáza modelov, kde budú modely uložené vo forme meshov. Ďalším vstupom bude výstup vyššie zmieneného uzlu, tzn. body detekovaného stola a body jednotlivých zhlukov. Na tieto body zhlukov sa použije metóda ICP, ktorá bola spomínaná v kapitole 3.2, a bude sa snažiť nájsť zhodu s nejakým modelom objektu z databáze objektov. Výstupom tohoto detektoru budú identifikátory modelov objektov z databáze, ktoré sa najviac zhodujú s objektmi na snímanej pracovnej doske. Na obrázku 6.4 je vidieť výstup detektoru pre scénu z obrázku 6.2, ktorý je prekrytý s výstupom segmentátora za účelom porovnania detekcie voči segmentom. Farby na škále medzi zelenou a červenou značia modely v databázi, ktoré boli porovnávané voči farebným segmentom z obrázku 6.3. Najviac zelený súbor markerov značí najlepšie skóre pre daný model a najviac červený súbor markerov značí najhoršie skóre pre daný model. Ostatné súbory markerov modelov majú skóre medzi nimi. Význam ostatných farieb už bol spomínaný vyššie.

Z obrázku 6.4 je vidieť, že detekcia prebehla úspešne, pretože tie najviac zelené markery modelov sú zhodné s objektmi zo scény na obrázku 6.2.



Obrázok 6.4: Výsledok detekcie objektov nad scénou z obrázku 6.2

7 Implementácia

V tejto kapitole bude popísaná implementačná časť tejto práce. Postupne bude rozobraný prevod uzlov a služieb knižnice *tabletop_object_detector* z verzie ROS *Groovy* do verzie ROS *Hydro.* Potom budú bližšie popísané služby *tabletop_segmentation* a *tabletop_object_recognition*. Následne bude popísané prepojenie služby detekcie objektov s databázov objektov a priebeh vytvárania sady modelov. Nakoniec bude rozobraný výstup aplikácie a postup pre spustenie výsledného balíčka.

7.1 Prevod z Groovy do Hydra

Použitá knižnica *tabletop_object_detector* bola vydaná pre verziu ROS Groovy. Lenže robot PR2, ktorý sa nachádza v robolabe na Fakulte informačných technológií funguje na verzii Hydro. Preto bolo potrebné najprv previesť niektoré časti kódu tejto knižnice do verzie ROS Hydro, aby sa výsledná aplikácia mohla pripraviť na použitie na tomto robotovi.

Hlavné zmeny boli prevedené v súbore *CMakeLists.txt.* Tento súbor obsahoval parametre a príkazy, ktoré pracovali s kompilačným programom *rosbuild*, ktoré bolo potrebné prerobiť tak, aby sa dali spustiť pomocou programu *catkin_make.* Najprv bol pridaný príkaz *find_package(),* ktorý slúži na nájdenie všetkých požadovaných balíčkov a správ, ktoré budú potrebné pri kompilácii. Ku klasickým balíčkom, ktoré sú uvedené v tomto príkaze patria *roscpp, rospy* a *std_msgs.* Tieto základné balíčky boli rozšírené o správy *shape_msgs* a *household_objects_database_msgs.* Druhý zmienený typ správy slúži na prácu s databázou, ktorá bude bližšie popísaná v kapitole 7.3. Ďalej bolo potrebné pridať parametre *message_generation* a *genmsg*, ktoré slúžia na vygenerovanie správ. V poslednom rade boli pridané balíčky *moveit_core* a *pcl_ros*. *Moveit_core* obsahuje knižnice, ktoré boli súčasťou verzie ROS Groovy a bohužiaľ už nie sú súčasťou verzie ROS Hydro. *Pcl_ros* je balíček obsahujúci triedy, funkcie a typy na prácu s mračnom bodov. Ďalej boli zo súboru *CMakeList.txt* odstránené všetky príkazy s predponou rosbuild, a to *rosbuild_init()*, ktorý slúži na počiatočnú inicializáciu balíčku, ďalej *rosbuild_genmsg()* a *rosbuild_gensrv()* , ktoré spracúvajú špecifikácie správ resp. služieb, ktoré sú uložené v zložke */msg* resp. /*srv* a boli nahradené nasledovným kódom:

```
add_message_files(
      FILES
      Table.msg
      TabletopDetectionResult.msg
)
```

resp.:

```
add_service_files(
      FILES
      AddModelExclusion.srv
      ClearExclusionsList.srv
      NegateExclusions.srv
      SegmentObjectInHand.srv
      TabletopDetection.srv
```

```
        TabletopObjectRecognition.srv
        TabletopSegmentation.srv
    )
```

Následne bolo potrebné ešte pridať kód pre generovanie správ a služieb spolu s ich závislosťami, a to nasledovne:

```
    generate_messages(
        DEPENDENCIES
        std_msgs
        geometry_msgs
        shape_msgs
        sensor_msgs
        household_objects_database_msgs
    )
```

Medzi ďalšie rosbuild príkazy, ktoré bolo potrebné nahradiť patria príkazy *rosbuild_add_library*, *rosbuild_add_executable* a *target_link_libraries*. Táto trojica príkazov má tento tvar:

```
rosbuild_add_library(marker_generator src/marker_generator.cpp)
rosbuild_add_executable(tabletop_segmentation
src/tabletop_segmentation.cpp)
target_link_libraries(tabletop_segmentation marker_generator)
```

kde príkaz *rosbuild_add_executable* obsahuje názov výsledného binárneho súboru a cestu k zdrojáku, z ktorého sa má vytvoriť. Príkaz *rosbuild_add_library* pridá knižnicu, ktorú výsledný binárny súbor používa a nakoniec *target_link_libraries* nalinkuje tieto súbory dokopy. Zmienené príkazy boli nahradené podobnými príkazmi s menšími zmenami, a to:

```
add_executable(tabletop_segmentation
src/tabletop_segmentation.cpp src/marker_generator.cpp)
target_link_libraries(tabletop_segmentation ${catkin_LIBRARIES})
```

kde príkazy *rosbuild_add_executable* a *rosbuild_add_library* boli zlúčené do jedného príkazu *add_executable*. A príkazu *target_link_libraries* bol nahradený druhý parameter za parameter obsahujúci referenciu na všetky knižnice catkin.

Balíček *moveit_core*, pre nahradenie ďalej nepoužívaných súčastí ROS knižníc bol použitý v súboroch:

- *include/tabletop_object_detector/iterative_distance_fitter.h*
- *src/tabletop_object_detector/model_fitter.cpp*

kde bol nahradený import `<distance_field/propagation_distance_field.h>` importom `<moveit/distance_field/propagation_distance_field.h>`. V druhom zmienenom súbore *model_fitter.cpp* bolo potrebné spraviť ešte jednu zmenu. Funkcia *addPointsToField()* triedy *distance_field::PropagationDistanceField*, ktorá pôvodne prijímala jeden parameter typu *const std::vector<tf::Vector3> &points*, teraz v novej verzii prijíma parameter typu

const EigenSTL::vector_Vector3d &point. Keďže táto zmena nebola obsiahnutá v balíčku *core_moveit*, tak ju trebalo vykonať manuálne, a to tak, že riadok:

```
distance_voxel_grid_->addPointsToField(points);
```

bol nahradený riadkami:

```
EigenSTL::vector_Vector3d pts(points.size());
for(size_t i = 0; i < points.size(); ++i)
      pts[i] = Eigen::Vector3d(points[i].x(), points[i].y(),
               points[i].z());
distance_voxel_grid_->addPointsToField(pts);
```

kde vidieť, že bola vytvorená inštancia *EigenSTL::vector_Vector3d* a bola naplnená obsahom premennej *points*.

Tak ako niektoré knižnice prešli zmenami pri prechode na novšiu verziu distribúcie ROS, tak isto nastali zmeny aj v knižnici *pcl*, ktorá slúži na prácu s mračnom bodov. Medzi tieto zmeny patrí napríklad to, že pôvodne funkcie tejto knižnice prijímaly aj parametre typu *sensor_msgs::PointCloud2*, ale teraz už neprijímajú a je ich potrebné previesť na typ *pcl::PCLPointCloud2*. Kvôli týmto zmenám bola použitá knižnica *pcl_conversions*, ktorá obsahuje funkcie *fromPCL* a *toPCL* na prevod medzi zmienenými typmi. Príkladom je hlavička mračna bodov v súbore *src/tabletop_segmentation.cpp*, pre ktorú bolo potrebné vykonať opísanú zmenu z kódu:

```
table_points.header = table.header;
```

na kód:

```
table_points.header = pcl_conversions::fromPCL(table.header);
```

Ďalší problém nastal pri časovom razítku v hlavičke mračna bodov, ktoré pôvodne bolo typu *ros::Time*, ale teraz sa ako razítko ukladá číslo určujúce počet mikro sekúnd. Tento problém bol vyriešený tak, že sa na premennú typu *ros::Time* použila metóda *toNSec()*, ktorá previedla čas na nano sekundy a potom sa táto hodnota vydelila konštantou *1e3*, čím bol dosiahnutý požadovaný výsledok.

Medzi posledné zmeny patrí premenovanie súboru *manifest.xml* na *package.xml* a všetky tagy *depend* boli nahradené tagmi *build_depend* a *run_depend*. Medzi zmenené súbory patrí aj src/*ping_tabletop_node.cpp*, ktorý bol prerobený podľa vlastných potrieb a bude bližšie popísaný v kapitole 7.5. Nakoniec bol odstránený súbor *Makefile*, ktorý už nebol potrebný.

7.2 Segmentácia objektov

O segmentáciu objektov sa stará služba s názvom *tabletop_segmentation*, ktorá čaká na topicu *cloud_in* na prijatie mračna bodov. Po prijatí mračna sa spustí služba a najprv vyfiltruje len body, ktoré zodpovedajú zadaným kritériám. Tieto obmedzujúce kritériá sú zadané pomocou parametrov, ktoré budú bližšie popísané v kapitole 7.6. Na filtrovanie sa použije trieda *pcl::PassThrough.* Najprv sa pomocou metódy *setInputCloud* zadá mračno bodov, ktoré sa má filtrovať. Potom sa pomocou metódy *setFilterFieldName* nastaví osa podľa ktorej sa body budú filtrovať a následne sa nastaví rozsah bodov, ktoré majú byť vyfiltrované pomocou metódy *setFilterLimits* a jej dvoch parametrov, ktoré určujú minimálnu a maximálnu hodnotu bodu v danej ose. Nakoniec sa do ukazateľa na typ *pcl::PointCloud<Point>::Ptr* uloži vyfiltrované mračno bodov. Tento postup sa postupne zopakuje pre jednotlivé osy *x*, *y* a *z.*

Po vyfiltrovaní bodov vo všetkých osách sa ešte zmenší počet bodov použitím triedy *pcl*::*VoxelGrid* a jej metódy *filter,* ktorá slúži na prevzorkovanie a to tak, že zostaví 3D mriežku nad daným mračnom bodov a prevzorkuje každú časť tejto mriežky pomocou, ťažiska, stredového bodu danej časti mriežky. Následne sa toto prevzorkované mračno bodov použije na odhadnutie normálov, ktoré je vykonané pomocou triedy *pcl::NormalEstimation* a jej metódy *compute.* Potom sa vykoná planárna segmentácia, ktorá slúži na získanie bodov pracovnej dosky. Na to sa využije trieda *pcl::SACSegmentationFromNormals,* ktorej sa pomocou metód *setInputCloud* resp. s*etInputNormals* zadá prevzorkované mračno bodov resp. odhadnuté normály a následne pomocou metódy *segment* vráti body patriace pracovnej doske. Body nad pracovnou doskou sa potom rozdelia do Euklidovských zhlukov pomocou metódy *extract* z triedy pcl::EuclideanClusterExtraction, ktorá je založená na štruktúre KdTree a metódy Euklidovho zhlukovania, ktoré sú popísané v kapitole 3.1. Nakoniec sa tieto zhluky, mračná bodov, vrátia spolu so súradnicami pracovnej dosky ako odozva služby *tabletop_segmentation* a ešte sa aj publikujú markery do topicu *tabletop_segmentation_markers,* ktoré sa potom dajú zobraziť v nástroji *Rviz* ako je vidieť na obrázku 6.2.

7.3 Prepojenie s databázou

Aby detekcia mohla fungovať, bolo potrebné stiahnuť balíček *household_objects_database*, spomínaný v kapitole 5.5 a pripraviť databázu. Databáza bola vytvorená z najnovšieho záložného súboru *household_objects-0.7_groovy_prerelease_1.backup*, ktorý je dostupný z repozitára[14] záložných súborov obsahujúcich modely objektov z domácnosti.

Zmienená databáza obsahuje viaceré tabuľky, ale pre potreby tejto práce sú dôležité len tabuľky mesh, model_set, a original_model. Tabuľka mesh sa skladá zo stĺpcov original_model_id, mesh_vertex_list, mesh_triangle_list, ktorých obsah je zrejme jasný. Ďalšia tabuľka model_set obsahuje stĺpce model_set_name a original_model_id. Stĺpec model_set_name určuje názov sady modelov, ktorý je predávaný ako parameter pre službu *tabletop_object_recognition*. Z tejto tabuľky sa dá vyčítať, ktorej sade prislušia ktoré modely. Jeden model môže byť priradený súčasne viacerým sadám. Posledná tabuľka original_model obsahuje 13 stĺpcov. Ich názvy nie sú podstatné, preto len zhrnieme aké informácie obsahujú. Dá sa v nej nájsť bližší popis daného modelu objektu, a to akého

14 Dostupný na: https://github.com/ros-interactive-manipulation/household_objects_database_backups

je model typu, či už tanier, šálka, pohár a iné, ďalej výrobca objektu a vlastnosti daného modelu, či je model rotačne symetrický, konkávne vyplnený alebo vnútorne konkávny a hlavne identifikátor modelu, čím je original_model_id spomínaný vyššie.

Na pridanie nového modelu do databáze slúži CLI aplikácia *insert_model* nachádzajúca sa v balíčku *household_object_database*. V tomto prípade je to v pracovnom priestore *catkin* v zložke *devel*/*lib*/*household_object_database*. Táto aplikácia prijíma dva parametre. Prvým je súbor s príponou *.ply* obsahujúci mesh daného modelu objektu a druhým je obrázok tohoto modelu vo formáte *.png*. Táto aplikácia obsahuje sprievodcu, ktorý nás postupne prevedie vkladaním modelu objektu do databáze. Zadajú sa informácie ako výrobca, názov modelu, prípadný čiarový kód a popis daného objektu niekoľkými slovami. Následne sa už aplikácia postará o zvyšok a naplní tabuľky original_model a mesh spomínané vyššie, s tým že automaticky priradí original_model_id vkladanému modelu.

V poslednom rade je potrebné nastaviť konfiguračný súbor formátu *yaml*[15]. V tomto prípade je to súbor *localhost.yaml* nachádzajúci sa v pracovnom priestore a v adresári s relatívnou cestou *src/household_objects_database/config.* Tento súbor má nasledovný obsah:

```
household_objects_database:
  database_host: localhost
  database_port: 5432
  database_user: willow
  database_pass: willow
  database_name: household_objects-0.7_groovy_prerelease_1
```

kde prvé štyri parametre sú zrejmé a posledný parameter *database_name* je názov databáze, na ktorú sa chceme pripojiť. V tomto prípade je to databáza *household_objects-0.7_groovy_prerelease_1*, ktorá bola vytvorená podľa postupu spomenutého na začiatku tejto podkapitoly.

7.4 Vytvorenie sady modelov

Pre otestovanie funkčnosti detektora bolo potrebné mať k dispozícii čo najpodobnejšie reálne objekty k modelom v databázi. Po preskúmaní obsahu databáze *household_objects-0.7_groovy_prerelease_1* a prehľadaní bytu som našiel približnú zhodu len v pár objektoch. Týmto kritériám odpovedal jeden tanier a jedna miska, a preto som ešte hľadal na internete nejaké ďalšie modely objektov z domácnosti, ktorými by som rozšíril vytvorenú databázu.

Na serveri github som našiel repozitár *object_models*[16] užívateľa CURG, ktorý obsahoval modely vo formáte *ply* a každý model mal svoju reprezentáciu aj vo forme obrázku. Po preskúmaní týchto obrázkov som našiel ďalšie tri objekty, ktoré boli aspon približne zhodné s reálnymi objektmi, ktoré som mal k dispozícii. Boli to šálka, tuba na tenisové loptičky a krabička od čaju. Využitím nástroja *MeshLab*[17] som tieto modely prispôsobil čo najviac dostupným reálnym objektom a následne

15 http://www.yaml.org/start.html

16 https://github.com/CURG/object_models/tree/master/models/cgdb/model_database

17 http://meshlab.sourceforge.net/

som ich uložil do databáze pomocou programu *insert_model*, ktorý bol spomínaný v kapitole 7.3.

Týchto päť objektov tvorilo základ pre otestovanie funkčnosti detektora. V databázi v tabuľke *model_set* boli vytvorené záznamy, kde stĺpec *model_set_name* mal hodnotu *DIPLOMA_SET* a stĺpec *original_model_id* mal hodnotu identifikátora, ktorý bol priradený daným modelom pri vkladaní do databáze. Reálne objekty na ktorom bol detektor testovaný je vidieť na obrázku 7.1. a ich modely je vidieť na obrázku 7.2.

Obrázok 7.1: Objekty, ktorých modely sú uložené v databázi

Obrázok 7.2: Modely objektov v databázi

7.5 Detekcia objektov

Detekciu objektov má na starosti služba *tabletop_object_recognition.* Jej vstupom je výstup služby *tabletop_segmentation.* Prvým krokom tejto služby je načítanie meshov modelov objektov z databáze, ktoré sa vykoná pomocou metódy *loadDatabaseModels.* Tejto metóde sa ako parameter predá názov sady modelov, ktorú ma nahrať do pamäte. Potom sa pre každý zhluk použije metóda *fitBestModels* s dvoma parametrami. Prvým je zhluk, pre ktorý sa má vykonať detekcia a druhý parameter je *numModels*, ktorý určuje počet najlepších zhôd z porovnania voči všetkým načítaným modelom. Táto metóda funguje tak, že porovnáva zhluk so všetkými načítanými modelmi a pre každý model vráti skóre, ktoré určuje ako presne sa zhoduje model zo zadaným zhlukom. Nakoniec zoradí všetky skóre v porovnaní so všetkými modelmi a zoradí ich od najlepšieho k najhoršiemu. Z toho vyberie najlepšie zhody o počte *numResults.* Výstupom tejto služby je zoznam, v ktorom pre každý zhluk obsahuje zoznam modelov o veľkosti *numModels* spolu so skóre detekcie.

Nevýhodou tohoto detektoru je, že jeho funkcionalita je obmedzená. Detekciu vykonáva „posúvaním“ modelu po pracovnej doske a hľadá čo najväčšiu zhodu s objektmi na stole. A preto nedokáže detekovať objekty, ktoré sú inak pretočené ako daný model, tzn. že pre správnu detekciu musí byť objekt rotačne symetrický a musí mať známu orientáciu. K tomuto zisteniu došlo bohužiaľ neskoro a nebol k dispozícii dostatočný čas na prerobenie jeho funkcionality.

7.6 Použitie aplikácie

Nastavenie parametrov

Hlavným spúšťačom celého detektoru je launch súbor *tabletop_complete.launch*. Tento súbor najprv spustí uzol *objects_database_node* z balíčka *household_objects_database*, ktorý vytvorí spojenie s databázou.

Ďalším krokom tohoto launch súboru je spustenie služby *tabletop_segmentation*, ktorá je spustená pomocou vo vnorenom launch súbore, a to *tabletop_segmentation.launch*. Tento súbor obsahuje ďalšie nastavenia argumentov, ktoré sú už prednastavené použitím argumentu *default* a príslušnej hodnoty. Pravdaže tieto argumenty je možné nastaviť aj v hlavnom launch súbore, a potom nie je potrebné používať viac týchto súborov. Medzi nastaviteľné argumenty patrí napríklad *cloud_in*, ktorý určuje na akom topicu má služba *tabletop_segmentation* očakávať vstupné mračno bodov. Medzi ďalšie nastaviteľné argumenty patria argumenty, ktoré sa použijú na filtrovanie mračna bodov, ako bolo písané v kapitole 7.2. Kde v každej ose je možné určiť akú minimálnu a maximálnu hodnotu má mať bod v tomto mračne bodov. Sú to argumenty x*_filter_min, x_filter_max, y_filter_min, y_filter_max, z_filter_min, z_filter_max.* Ďalej je tu argument *cluster_distance*, ktorý určuje aká má byť minimálna vzdialenosť pre jednotlivé zhluky, aby ich segmentátor určil ako rôzne segmenty. V poslednom rade argument *min_cluster_size* určujúci počet bodov, ktorý musí zhluk obsahovať, aby ho segmentátor považoval za zhluk.

Posledným krokom launch súboru je spustenie služby *tabletop_object_recognition.* Táto služba má tiež vlastný launch súbor, a to *tabletop_object_recognition.launch*. V prvom rade sa táto služba stará o nahranie modelov z databáze do pamäte. Výber modelov, ktoré má nahrať je určený argumentom s názvom *model_set*, ktorý má hodnotu *DIPLOMA_SET*. Z databáze z tabuľky

model_set, ktorá bola spomínaná v kapitole 7.3, vyberie tie riadky kde hodnota stĺpca *model_set_name* je rovný *DIPLOMA_SET* a nahrá všetky modely určené identifikátorom v stĺpci *original_model_id*.

Uzol ping_tabletop_node

Tento uzol slúži na riadenie celej detekcie. Najprv čaká na spustenie služby *tabletop_segmentation*. Potom čaká na jej výstup, ktorý následne pošle na vstup služby *tabletop_object_recognition* a tiež na vstupe nastaví premennú *num_models*, ktorá určuje voči koľkým modelom sa má porovnávať jeden zhluk, tak ako bolo popísané v kapitole 7.4. Výstup tejto služby vypíše na štandardný výstup prerobený do formátu *json*[18]. Výstup obsahuje všetky segmenty nájdené službou *tabletop_segmentation* indexované od 0 a každý obsahuje informácie o stredovom bode tohoto segmentu a zoznam modelov o veľkosti *num_models*, ktorý je zoradený podľa skóre detekcie. Každá položka tohoto zoznamu obsahuje identifikátor modelu, jeho stredový bod a skóre vzhľadom k detekcii. Príklad výstupu tohoto uzlu, kde *numModels* sa rovná 2 a pri detekcii boli správne detekované dva objekty, ktoré sa nachádzajú na pracovnej doske je vidieť v prílohe A.

Spustenie

Návod na inštaláciu tejto knižnice a potrebné nastavenia systému je možno nájsť v súbore *README* na priloženom CD. Z tohoto dôvodu budú uvedené len základné príkazy na spustenie detektora. Najprv je potrebné spustiť všetky potrebné služby, čo sa učiní pomocou pripraveného launch súboru, a to nasledovne:

```
$ roslaunch tabletop_object_detector tabletop_complete.launch
```

V ďalšom okne terminálu spustíme balíček *openni_launch*, ktorým získame prístup k hĺbkovým dátam nahravaným senzorom Kinect, a to nasledovne:

```
$ roslaunch openni_launch openni.launch
```

Ak chceme vykonať detekciu na už nahraných bag súboroch, tak to môžeme previesť nasledovnými príkazmi:

```
$ rosparam use_sim_time true
$ rosbag play --loop cesta_k_bagfile.bag
```

Prvý príkaz slúži na nastavenie simulovaného času. Je to kvôli spusteným službám, aby si nemysleli, že prijímajú už staré a neplatné dáta, a teda nevracali chybu. Druhý príkaz prehrá potrebný bagfile. Parameter `--loop` slúži na prehrávanie súboru bag dookola, čo zaručí, že služba *tabletop_segmentation* príjme mračo bodov a spustí segmentáciu.

18 http://www.json.org/json-cz.html

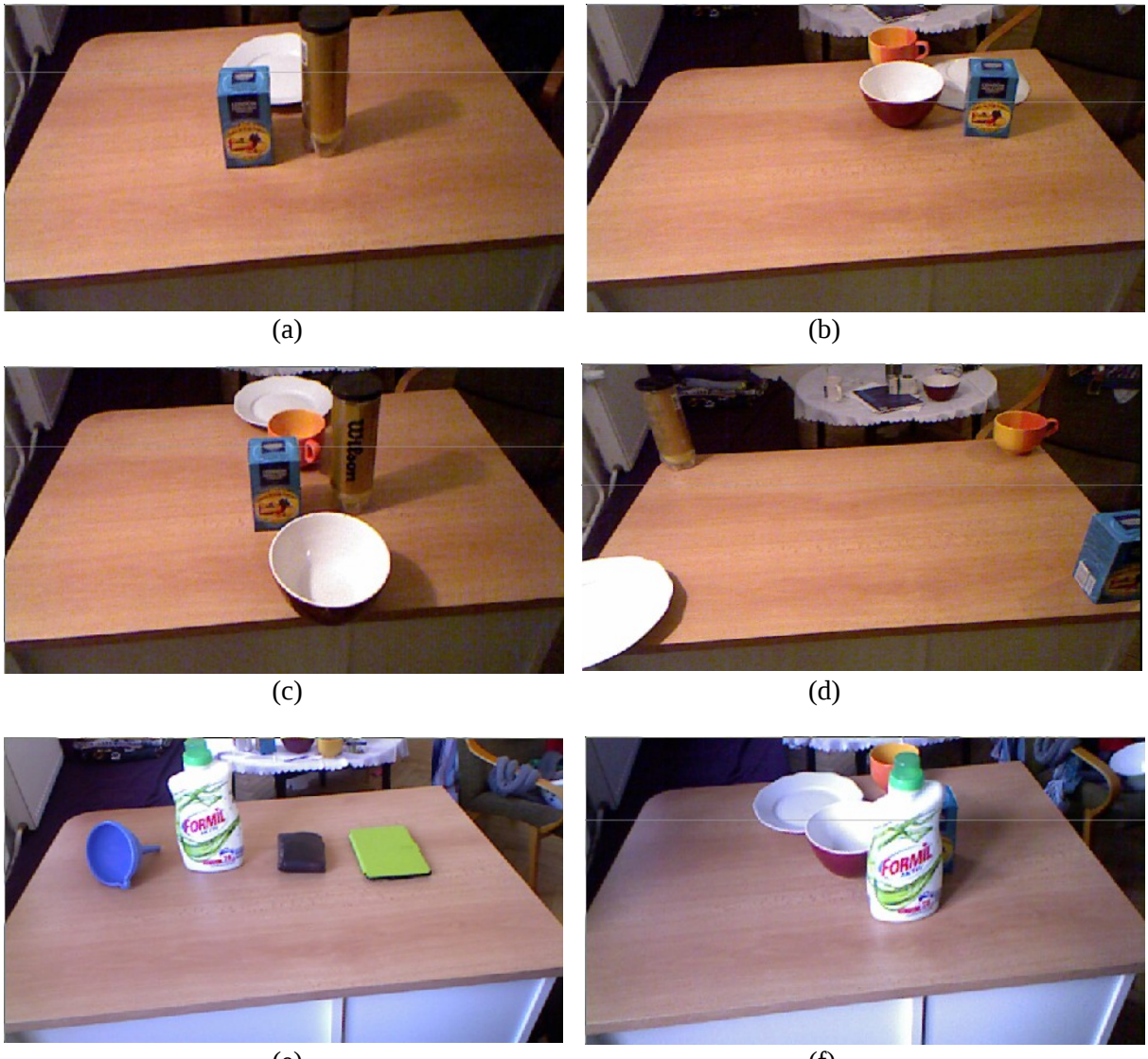
Nakoniec sa spustí uzol *ping_tabletop_node*, ktorý riadi celú detekciu, a to príkazom:

```
$ rosrun tabletop_object_detector ping_tabletop_node
```

Po vykonaní príkazu vráti *json* obsahujúci výslednú detekciu, vo formáte ako je vidieť v prílohe B.

8 Experimenty

Táto kapitola popisuje experimenty, ktoré boli vykonané s prerobenou knižnicou *tabletop_object_detector*. Pre tieto experimenty bola vytvorená datová sada, ktorej popis a obsah bude tiež popísaný nižšie. Ďalej budú uvedené parametre knižnice, ktoré boli pri experimentoch rôzne nastavované a budú uvedené ich hodnoty pri ktorých mala detekcia najlepšie výsledky. Na záver budú zhrnuté výsledky všetkých experimentov.



Obrázok 8.1: Príklady scén z vytvorenej datovej sady.

8.1 Datová sada

Pre overenie presnosťi detektoru bola vytvorená vlastná datová sada. Táto sada obsahuje sto rôznych scén pracovnej dosky s objektmi uloženými na nej a bola prispôsobená obmedzenej funkčnosti detektoru. Jednotlivé scény obsahujú jeden až osem objektov. Niektoré scény sú znázornené na obrázkoch 8.1(a-f). Táto sada pokrýva napríklad aj problém prekrytia objektov, ako vidieť na obrázkoch 8.1(a-c) alebo obsahuje aj scény, kde sú objekty položené na kraji pracovnej dosky, ako vidieť na obrázku 8.1(d). Do tejto datovej sady boli pridané aj scény s objektmi, ktoré nie sú uložené v databázi. Neznáme objekty sú štyri, a to: lievik, peňaženka, elektronická čítačka kníh a obal od tekutého pracieho prášku. Scény s týmito objektmi sú znázornené na obrázkoch 8.1(e-f).

Pre každú scénu bol vytvorený bagfile spolu s obrázkom scény a anotáciou, ktorá určuje pre každý objekt na stole číslo modelu z databáze objektov, ktorému daný objekt prísluší a ďalej popisuje reálnu pozíciu stredov jednotlivých objektov na pracovnej doske. Ak objekt v databázi uložený nie je, tak mu je v anotácii priradený model s číslom 0. Na tejto datovej sade boli následne prevedené experimenty, ktoré budú popísané nižšie spolu s ich výsledkami.

8.2 Nastavenie parametrov

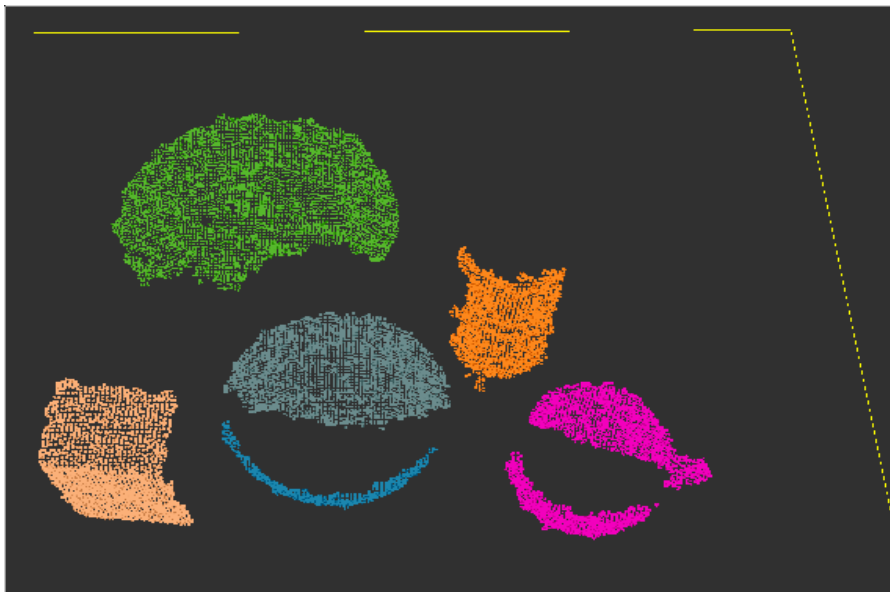
Pred samotnou kontrolou kvality detektora bolo potrebné nastaviť jeho vstupné parametre. Tieto parametre boli popísané v kapitole 7.6. Najväčší vplyv na výsledok detekcie mali parametre *cluster_distance* a *min_cluster_size.* Vplyv týchto nastavení bude rozobraný nižšie a budú uvedené ich optimálne nastavenia, ktoré boli dosiahnuté experimentovaním s rôznymi scénami.

Parameter cluster_distance

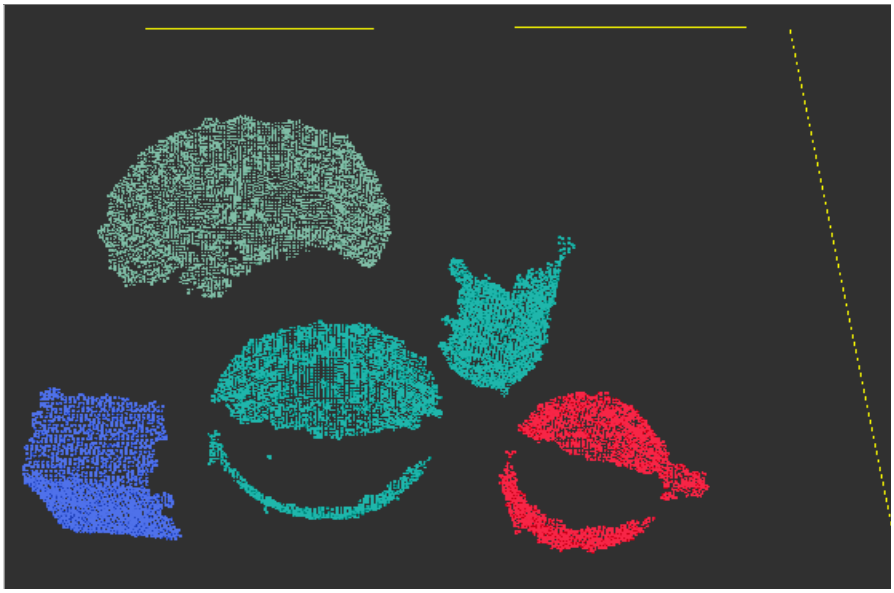
Tento parameter, ako už bolo písané vyššie, má za úlohu určiť minimálnu vzdialenosť, ktorá určí rozdelenie mračna bodov na jednotlivé segmenty. Dôležitosť tohoto nastavenia je značná. Ak minimálna vzdialenosť bude nastavená na malú hodnotu, tak objekty, ktoré sa nachádzajú na pracovnej doske blízko seba budú detekované ako jeden segment. Naopak ak bude vzdialenosť nastavená na väšiu hodnotu, tak môže reprezentáciu reálneho objektu rozdeliť na dva rôzne segmenty.

Obrázok 8.2: Scéna pracovnej dosky s objektmi.

Názorný príklad ukážeme na scéne, ktorá je znázornená na obrázku 8.2. Pri nastavení parametru *cluster_distance* na hodnotu *0.25* je vidieť na obrázku 8.3 (žltá čiarkovaná čiara znázorňuje obrysy pracovného stola), že rozdelí misku na dva rôzne segmenty. Z časti je to spôsobené aj uhlom snímania. Ako si môžeme všimnúť, tak mračno bodov misky nie je prepojené. Toto je spôsobené tým, že Kinect nasnímal len vonkajšie body misky na prednej časti a vnútorné body misky na zadnej časti. Naopak, ak sa parameter *cluster_distance* nastavil na hodnotu *0.3*, tak pri rovnakej scéne spojil tubu na loptičky a misku do jedného segmentu. Toto chovanie je vidieť na obrázku 8.4.



Obrázok 8.3: Výsledok segmentácie po nastavení parametru cluster_distance na hodnotu 0,25.

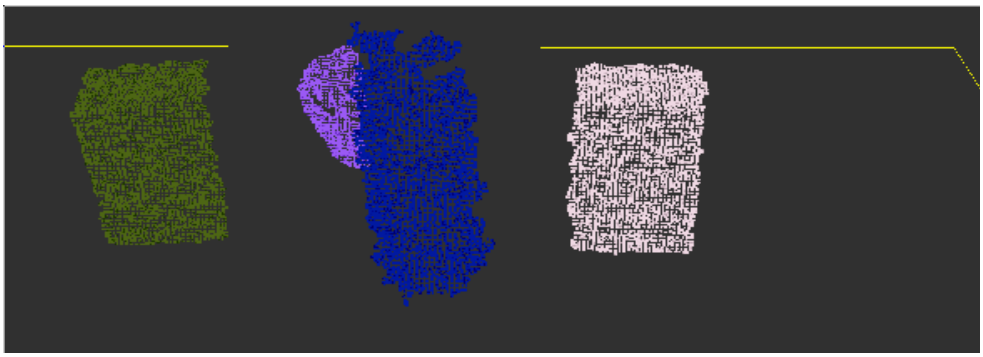


Obrázok 8.4: Výsledok segmentácie po nastavení parametru cluster_distance na hodnotu 0,3.

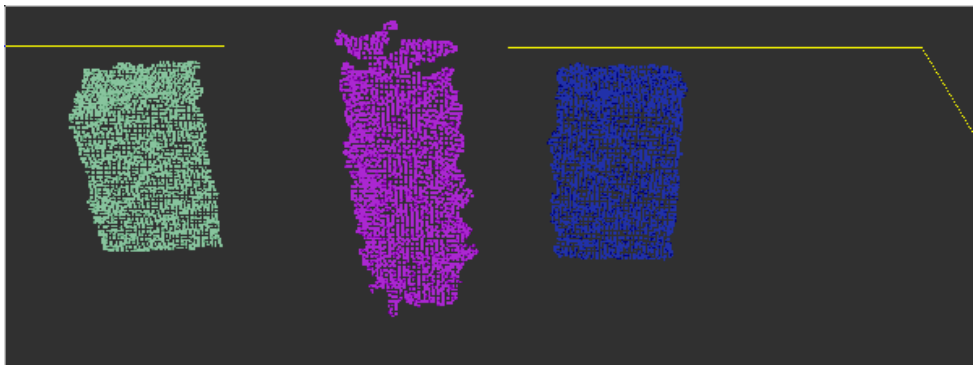
Parameter min_cluster_size

Ďalším dôležitým parametrom je *min_cluster_size*. Toto nastavenie dáva informáciu segmentátoru, že ak detekuje zhluk, ktorý neobsahuje aspon taký počet bodov, ktorý je určený hodnotou *min_cluster_size*, tak ho nemá ako zhluk prijať. V tomto prípade si to predvedieme na scéne z obrázku 8.5. Pri nastavení tohoto parametru na hodnotu *300*, bolo potrebné aby zhluk bol zložený aspoň z tristo bodov. Na obrázku 8.6 je vidieť, že to splňujú všetky objekty, aj miska, ktorá je schovaná za tubou od tenisových loptičiek. Pri nastavení na hodnotu *700* , to už ale z časti zakrytá miska nesplňuje, ako vidieť na výstupe segmentátora z obrázku 8.7.

Obrázok 8.5: Scéna pracovnej dosky s objektmi.



Obrázok 8.6: Výsledok segmentácie po nastavení parametru min_cluster_size na hodnotu 300.



Obrázok 8.7: Výsledok segmentácie po nastavení parametru cluster_distance na hodnotu 700.

Po experimentovaní s týmito parametrami na rôznych scénach, som dospel k záveru, že najlepšie funguje segmentácia, ak je parameter *min_cluster_size* nastavený na hodnotu *300* a parameter *cluster_distance* na hodnotu *0,27*. Tieto optimálne parametre boli nastavené aj pri experimentovaní a vyhodnocovaní detektoru na vytvorenej datovej sade, ktoré bude popísane nižšie.

8.3 Dosiahnuté výsledky

Experimentovanie s detektorom bolo prevádzané na vytvorenej datovej sade opísanej v kapitole 8.1. Parametre výsledného balíčku boli nastavené, tak ako sú popísané v kapitole 8.2. Pre každý bagfile bola spustená detekcia tak, ako je popísaná v kapitole 7.6. Výstup detekcie prevedenej nad jednou scénou bol uložený do samostatného súboru. Taktiež bol vytvorený skript v jazyku Python, ktorý spracoval tieto výsledky. Jazyk Python bol zvolený z dôvodu ľahkej práce so súbormi vo formáte *json.*

Dosiahnuté výsledky boli rozdelené na dve časti. V prvej si ukážeme kvalitu jednotlivých detektorov modelov. V druhej si ukážeme kvalitu detektoru ako celku. Kvalita jednotlivých detektorov modelov je znázornená ROC krivkami, ktorej body sú tvorené pomocou *recall* a *precision.* Výpočet *recall* a *precision* ukazujú vzorce 8.1 a 8.2.

$$recall = \frac{true\_positive}{true\_positive + false\_negative} \tag{8.1}$$

$$precision = \frac{true\_positive}{true\_positive + false\_positive} \tag{8.2}$$

True_positive znamená, že detektor detekoval objekt správne. *False_positive* znamená prijatie nesprávneho objektu detektorom. *False_negative* znamená, že detektor neprijal správny objekt a existuje ešte *true_negative*, ktoré znamená, že detektor odmietol nesprávny objekt, ten sa ale vo vzorci nepoužíva.

Vytvorený skript vytváral hodnotenie pre každý detektor modelu samostatne. Najprv zozbieral skóre zo všetkých detekcií všetkých scén pre daný model a nad každou scénou vytvoril párovanie detekovaného modelu s anotáciou k danej scéne, a to na základe pozície modelu detekovaného objektu a pozície modelu objektu v anotácii. Potom postupne pre každé skóre vykonával nasledujúce operácie. Ak sa v detekcii objavil model práve skúmaného detektoru, tak sa nad párovaním detekcie k anotácii testovalo, či sa rovnajú identifikátory modelov v detekcii a anotácii. Ak táto podmienka bola splnená a zároveň aktuálne skóre bolo lepšie alebo rovné ku skóre danej detekcie tak sa inkrementoval čítač *true_positive.* Ak skóre bolo horšie ako aktuálne skóre, tak sa inkrementoval čítač *false_negative.* Keď nad párovaním detekcie k anotácii vznikol pár, ktorý nemal rovnaký identifikátor modelu a skóre bolo lepšie ako aktuálne skóre, tak sa inkrementoval čítač *false_positive* a nakoniec ak skóre bolo horšie ako aktuálne skóre, tak sa inkrementoval čítač *true_negative*.

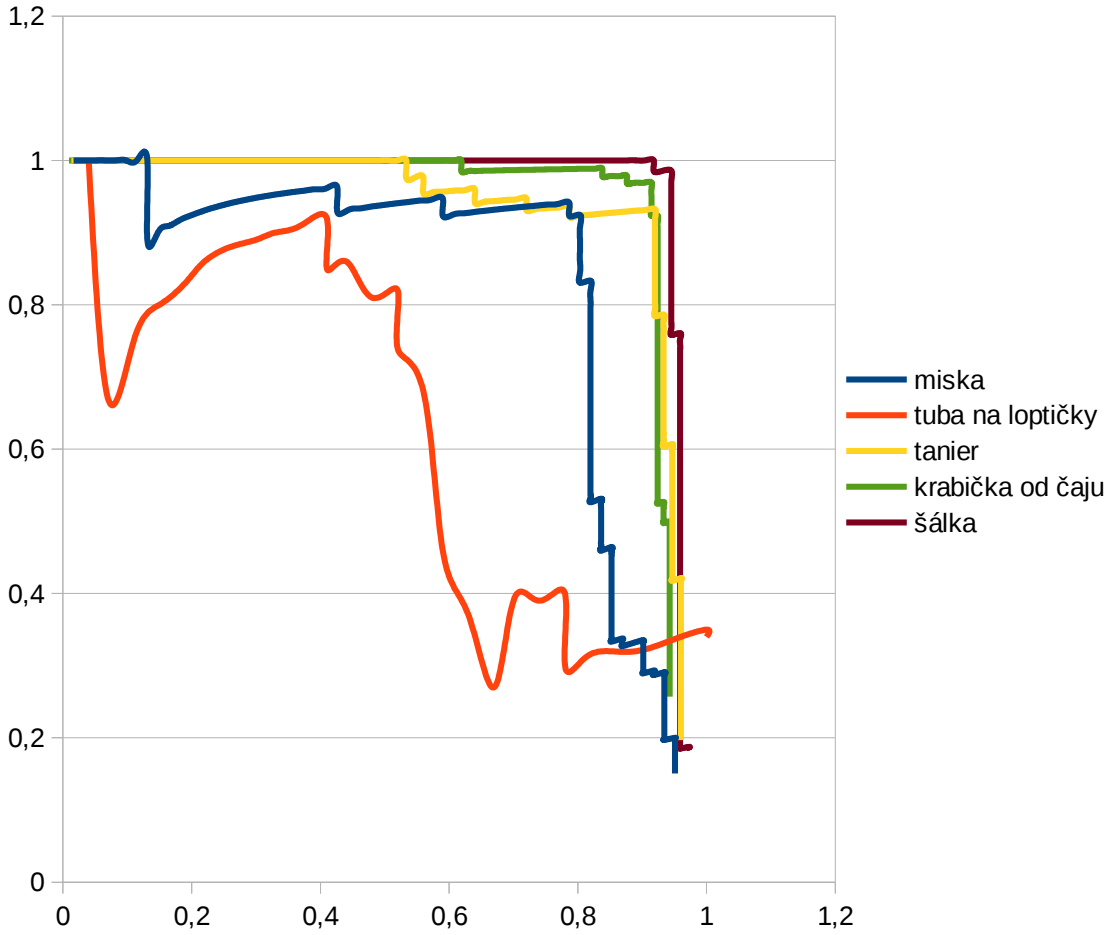
8.3.1 Výsledky jednotlivých detektorov

Detektor taniera

Tento detektor nemal väčšie problémy. Pri preskúmaní výsledkov bolo zistené, že chyby nastali len zriedkavo. Tieto chyby pramenili z toho, že detektor pri niektorých prípadoch nesprávne detekoval peňaženku a čítačku elektronických kníh ako tanier, inak všetko prijal a odmietol správne. Jeho priemerný *precision* je preto 0,91.

Detektor krabičky od čaju

Tento detektor dosiahol vynikajúce výsledky, keďže priemerná *precision* v tomto prípade je 0,927. Tento objekt ma svoj špecifický hranatý tvar a podobný mu nebol žiadny iný objekt z databáze, či z neznámych objektov, čo tiež prispelo k tomuto výsledku.



Obrázok 8.8: ROC krivky jednotlivých detektorov

Detektor misky

Výsledok detektoru misiek už nie je tak výborný ako výsledky predchádzajúcich detektorov. V tomto prípade je priemerný *precision* rovný 0,802. Mohlo to byť spôsobené tým, že parameter *cluster_distance* nebol pre segment misky dostatočný, a teda rozdelil misku na dva segmenty, ako bolo vidieť na obrázku 8.3, a tým pádom to detektor zle vyhodnotil. Tento problém by šlo riešiť prestavením parametru *cluster_distance* na vyššiu hodnotu alebo použitím inej metódy na segmentáciu, a to napríklad Mean Shift, ktorá by tento problém mala zvládnuť.

Detektor tuby na tenisové loptičky

Najhoršie výsledky zo všetkých detektorov mal detektor tuby na tenisové loptičky. Po preskúmaní výsledkov som prišiel k záveru, že problém bol v detektore, keď vo viacerých prípadoch prijal obal od tekutého pracieho prášku ako tubu na loptičky. A taktiež ak scéna obsahovala tubu na loptičky položenú v miske, tak to segmentátor vyhodnotil ako jeden segment a nie ako dva rôzne segmenty. Primerný *precision* má hodnotu 0,6.

Detektor šálky

Detektor šálok dosiahol vynikajúce výsledky. Ako vidieť aj na ROC krivke, tak tento detektor nemal skoro žiadne problémy a jeho priemerný *precision* má hodnotu 0,94.

ROC krivky pre jednotlivé detektory sú znázornené na obrázku 8.8. Pre porovnanie jednotlivých detektorov vidieť priemerné *precision* v nasledujúcej tabuľke:

Typ detektoru	**Priemerná *precision***
tanier	0,91
krabička od čaju	0,927
miska	0,802
tuba na tenisové loptičky	0,6
šálka	0,94

8.3.2 Výsledok celého detektoru

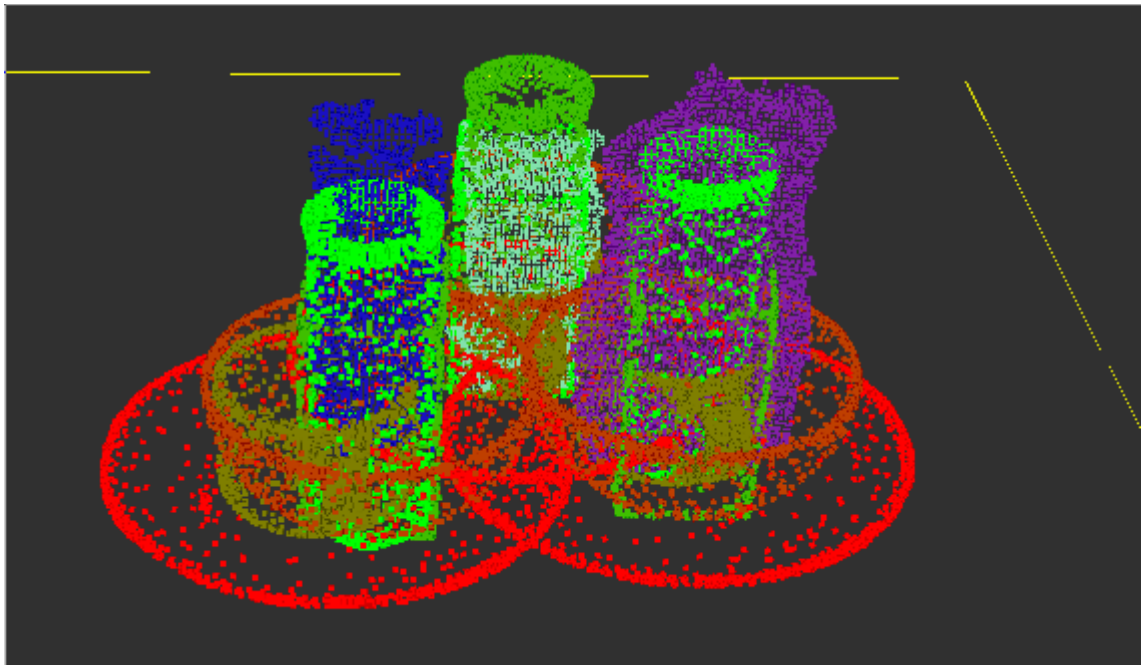
Pre ohodnotenie detektoru ako celku bola vykonaná detekcia nad celou datovou sadou, ktorá obsahuje spolu na všetkých scénach 399 objektov. Zisťoval sa počet segmentov, ktoré dokáže segmentátor rozoznať a z toho počet objektov, ktoré dokáže detektor správne rozpoznať. Pre porovnanie bola vykonaná detekcia tri krát a hodnota *cluster_distance* bola za každým rôzna, a to 0,27, 0,3 a 0,33.

Pre hodnotu *cluster_distance* 0,27 rozoznal segmentátor 395 segmentov, čo je výborná úspešnosť 98,9%. Z rozoznaných segmentov detektor detekoval 350 objektov správne, a teda presnosť detektoru pri danom nastavení je 88,6%.

Ak bol parameter *cluster_distance* nastavený na hodnotu 0,3, tak segmentátor dokázal rozpoznať 386 segmentov, čo činí 96,7% úspešnosť segmentátora. Z toho detektor detekoval 342 objektov správne. Celková presnosť detektoru pri tomto nastavení činí tiež 88,6%. Ako vidieť , tak presnosť detektoru je pre oba parametre zhodná, lenže segmentátor bol presnejší pri predchádzajúcom nastavení.

Pre hodnotu parametru *cluster_distance* 0,33 segmentátor rozpoznal 367 segmentov, čo je 92% z celkového počtu objektov. Z rozpoznaných segmentov toho správne detekovaných 330, tzn. že presnosť detektoru pri tomto nastavení je 89,9%. Presnosť detektoru pre toto nastavenie je najlepšie, ale presnosť segmentátoru najhoršia.

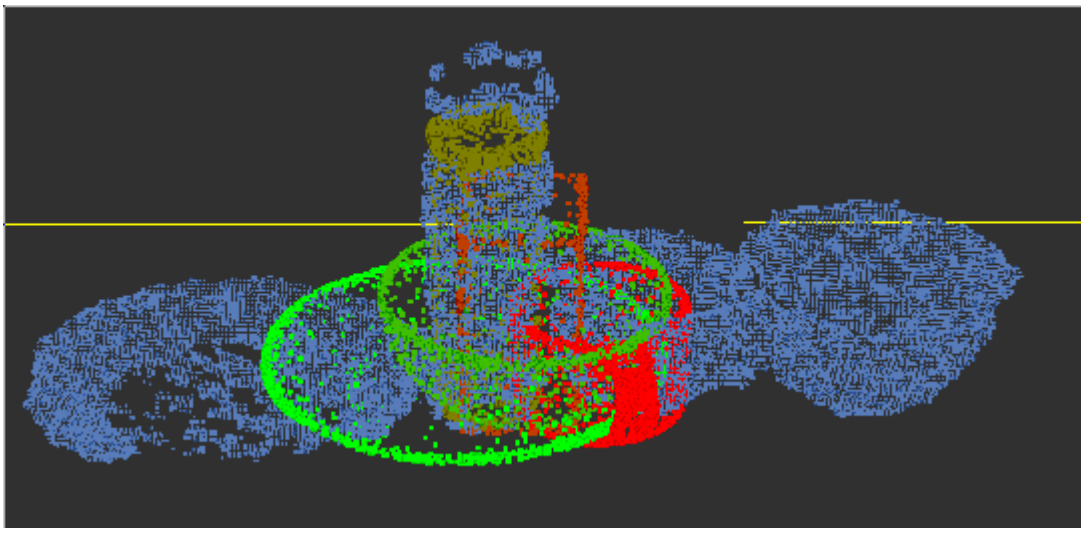
Obrázok 8.9: Scéna s neznámym objektom



Obrázok 8.10: Neúspešná detekcia jedného objektu zo scény na obrázku 8.9

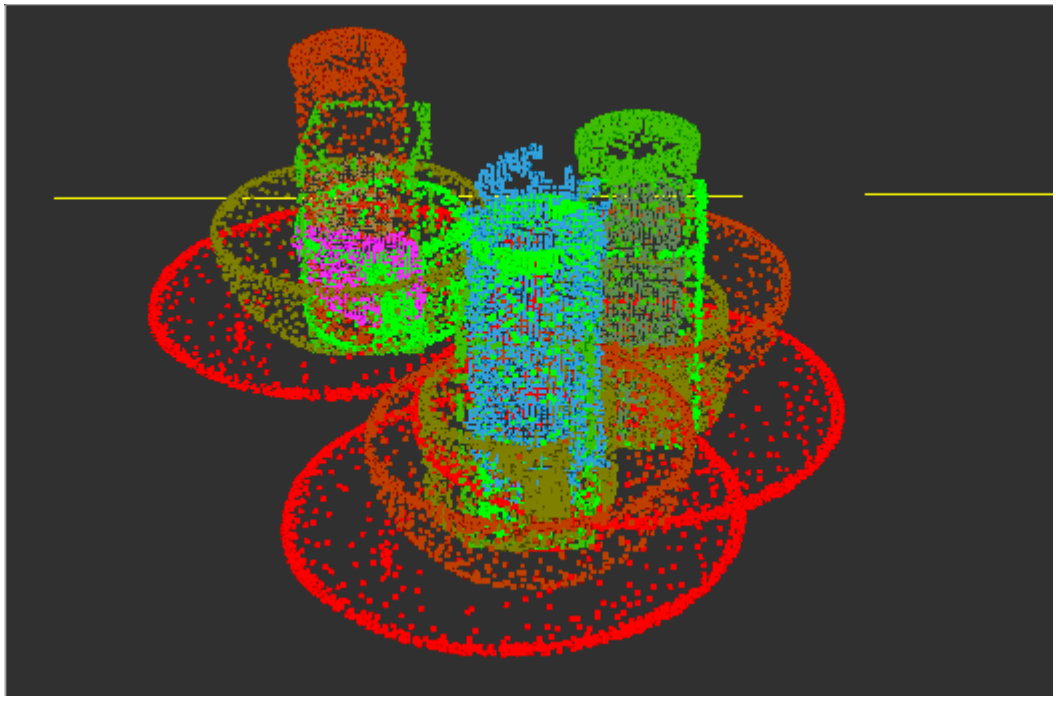
Detektor nie vždy zvládal rozoznávať objekty, ktorých modely nie sú uložené v databázi. Napríklad preto dosahoval zlé výsledky detektor tuby na loptičky, čo potvrdzuje aj obrázok 8.10, ktorý je výsledkom detekcie scény z obrázku 8.9. V tomto prípade nesprávne neodmietol obal od tekutého pracieho prášku a prijal ho ako tubu na loptičky. Ďalej detektor zvládal zle aj scény, kde boli objekty príliž blízko vedľa seba, ako je vidieť napríklad na obrázku 8.11. V tomto prípade dopadla detekcia zle, pretože segmentátor to vyhodnotil ako jeden segment, čo vidieť na obrázku 8.12 (modrá farba). Na druhú stranu, ak boli objekty čiastočne prekryté a boli známe ich modely, tj. boli uložené v databázi, tak detekcia prebehla v poriadku. Správnu detekciu prekrytých objektov zo scény na obrázku 8.13 je vidieť na obrázku 8.14.

Obrázok 8.11: Scéna s objektmi, ktoré sú uložené veľmi blízko seba



Obrázok 8.12: Príklad chyby segmentátora a zlej detekcie scény z obrázku 8.11

Obrázok 8.13: Scéna s prekryvom objektov



Obrázok 8.14: Úspešná detekcia prekrytých objektov zo scény na obrázku 8.13

Po predchádzajúcich výsledkoch segmentátora a samotného detektoru môžeme dospieť k záveru, že nastavenie parametru *cluster_distance* na hodnotu 0,27 je naozaj najoptimálnejším nastavením pre tento detektor, ako bolo predtým uvedené v kapitole 8.2. V prípadoch keď nie sú objekty položené úplne vedľa seba, alebo ak je dobre nastavený prah skóre na odmietnutie neznámeho objektu, tak detekcia funguje dobre, inak má menšie problémy.

Pre zaujímavosť sú uvedené časy prvotného spustenia balíčku. V tomto čase je zarátaná réžia okolo spustenia balíčku v ROS, spustenie služieb a aj načítanie modelov z databáze. Uvedený čas je priemerom časov troch spustení. Porovnanie pre rôzny počet modelov v databázi je vidieť v nasledujúcej tabuľke:

počet načítaných modelov z databáze	**čas (v sekundách)**
1	2,8
3	4,5
5	5,6

Čas vykonania samotnej detekcie sa líši. Záleží od počtu načítaných modelov z databáze a aj počtu objektov na pracovnej doske. Napriek tomu je výsledný čas detekcie relatívne nízky. Je to spôsobené obmedzenou funkčnosťou detektoru, ktorá bola popísana vyššie. V tomto prípade bol opäť čas spriemerovaný z troch nezávislých spustení. Výsledky časov spustení je vidieť v nasledujúcej tabuľke:

počet načítaných modelov z databáze	**počet objektov na stole**	**čas (v sekundách)**
1	1	1,02
	2	1,10
	5	1,51
	8	1,84
3	1	0,92
	2	1,14
	5	1,72
	8	2,10
5	1	1,15
	2	1,21
	5	1,25
	8	2,41

9 Záver

Cieľom tejto práce bolo navrhnúť nástroj, ktorý bude detekovať čiastočne zaplnenú dosku pracovného stolu a jednotlivých objektov pre robotické aplikácie. Prácu som započal štúdiom rôznych odborných prácí so zameraním na detekciu plochy a detekciu objektov v mračne bodov. Ďalej som sa zoznámil s frameworkom ROS a pochopil jeho fungovanie a prácu s ním.

Na základe získaných znalostí som vytvoril balíček, ktorý je založený na knižnici *tabletop_object_detector*. Táto knižnica bola vydaná pre verziu ROS Groovy, tak som ju prerobil do verzie ROS Hydro, aby ju bolo možné spustiť aj na robotovi PR2, ktorý sa nachádza na našej fakulte. Základ vytvoreného balíčku tvoria dve služby. Prvou je služba vykonávajúca detekciu pracovnej dosky v mračne bodov a následne vykonáva segmentáciu nad bodmi, ktoré sa nachádzajú nad pracovnou doskou, tzv. segmentátor. Druhá služba, detektor, slúži na detekciu objektov zo segmentov vrátených prvou službou. Na detekciu využíva databázu, ktorá je naplnená modelmi objektov. Základ databáze som vytvoril zo záložného súboru nájdenom na internete, ktorý obsahoval modely objektov z domácnosti. Následne som ju rozšíril o niekoľko objektov, na ktorých som potom testoval presnosť detektoru. Bohužiaľ som neskôr zistil, že funkčnosť detektoru tejto knižnice je obmedzená, a z časových dôvodov som už nemal čas na jeho prerobenie. Po implementácii som pomocou Kinectu vytvoril datovú sadu o veľkosti sto scén, kde každá scéna bola tvorená niekoľkými objektmi ležiacimi na pracovnej doske a bola prispôsobená k funkcionalite detektoru. Pre každú scénu bol nahraný bagfile, ktorý obsahoval mračno bodov tejto scény a bola uložená obrazová reprezentácia a anotácia tejto scény. Experimenty s detektorom ukázali, že pre vytvorenú datovú sadu funguje dobre a vracia dobré výsledky. Presnosť segmentátora je na úrovni 98,9% a presnosť detektora je na úrovni 88,6%. Na experimentoch som si overil aj správne nastavenie vstupných parametrov segmentátora s detektorom.

V budúcnosti by sa mohla rozšíriť metóda detektora, prípadne prerobiť na robustnejšiu metódu, ktorá dokáže zvládnuť rôzne preklopenia objektu a aj zložitejšie scény objektov. Následne by sa mohol celý balíček spojazdniť na robotovi PR2, ktorý by našiel využitie napríklad v domácnosti pre starších ľudí, kde by im mohol pomôcť s podávaním rôznych objektov.

Literatúra

[1] YANG, Michael Ying a FÖRSTNER. Plane Detection in Point Cloud Data. In: *Plane Detection in Point Cloud Data* [online]. Bonn, 2010 [cit. 2015-01-09]. Dostupné z: http://www.ipb.uni-bonn.de/fileadmin/publication/pdf/Yang2010Plane.pdf

[2] BORRMANN, Dorit, Jan ELSEBERG, Kai LINGEMANN a NÜCHTER. The 3D Hough Transform for Plane Detection in Point Clouds: A Review and a new Accumulator Design. In: *The 3D Hough Transform for Plane Detection in Point Clouds: A Review and a new Accumulator Design* [online]. 2011 [cit. 2015-01-09]. Dostupné z: http://plum.eecs.jacobs-university.de/download/3dresearch2011.pdf

[3] HAN, Jungong, Ling SHAO, Dong XU a Jamie SHOTTON. Enhanced Computer Vision with Microsoft Kinect Sensor: A Review. *Enhanced Computer Vision with Microsoft Kinect Sensor: A Review* [online]. 2013, vol. 43, issue. 5, 1318 - 1334 [cit. 2015-01-09]. Dostupné z: http://www3.ntu.edu.sg/home/DongXu/TC-Kinect.pdf

[4] KHOSHELHAM, K. ACCURACY ANALYSIS OF KINECT DEPTH DATA. *ACCURACY ANALYSIS OF KINECT DEPTH DATA* [online]. 2011, vol. 38, issue 5, s. 133-138 [cit. 2015-01-09]. Dostupné z: http://www.int-arch-photogramm-remote-sens-spatial-inf-sci.net/XXXVIII-5-W12/133/2011/isprsarchives-XXXVIII-5-W12-133-2011.pdf

[5] Assignment 3: KDTree. In: *Assignment 3: KDTree* [online]. 2014 [cit. 2015-01-09]. Dostupné z: http://web.stanford.edu/class/cs106l/handouts/assignment-3-kdtree.pdf

[6] BATIA, Shashank a Stephan K. CHALUP. Segmenting Salient Objects in 3D Point Clouds of Indoor Scenes Using Geodesic Distances. *Segmenting Salient Objects in 3D Point Clouds of Indoor Scenes Using Geodesic Distances* [online]. 2013, issue 4, s. 102-108 [cit. 2015-01-09]. Dostupné z: http://www.scirp.org/journal/PaperDownload.aspx?paperID=38079

[7] Y. Chen and G. Medioni, “Object Modelling by Registration of Multiple Range Images,” Image and Vision Computing, vol. 10, no. 3, pp. 145–155, 1992.

[8] Paul J. Besl and Neil D. McKay, “A Method for Registration of 3-D Shapes,” IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 14, no. 2, pp. 239–256, 1992.

[9] ZINSSER, Timo, Jochen SCHMIDT a Heinrich NIEMANN. A REFINED ICP ALGORITHM FOR ROBUST 3-D CORRESPONDENCE ESTIMATION. In: *A REFINED ICP ALGORITHM FOR ROBUST 3-D CORRESPONDENCE ESTIMATION* [online]. 2003 [cit. 2015-01-09]. Dostupné z: http://pdf.aminer.org/000/322/756/a_refined_icp_algorithm_for_robust_d_correspondence_estimation.pdf

[10] GUO, Yulan, Mohammed BENNAMOUN, Ferdous SOHEL, Min LU a Jianwei WAN. 3D Object Recognition in Cluttered Scenes with Local Surface Features: A Survey. IEEE Transactions on Pattern Analysis and Machine Intelligence [online]. 2014, vol. 36, issue 11, s. 2270-2287 [cit. 2015-05-25]. DOI: 10.1109/tpami.2014.2316828.

[11] MENG, Xianjing, Yilong YIN, Gongping YANG a Xiaoming XI. Retinal Identification Based on an Improved Circular Gabor Filter and Scale Invariant Feature Transform. Sensors [online]. 2013, vol. 13, issue 7, s. 9248-9266 [cit. 2015-05-25]. DOI: 10.3390/s130709248.

[12] ALDOMA, Aitor, Federico TOMBARI, Luigi DI STEFANO a Markus VINCZE. A global hypotheses verification method for 3d object recognition. *ECCV'12 Proceedings of the 12th European conference on Computer Vision* [online]. 2012, (vol. 3): 511-524 [cit. 2015-05-25]. Dostupné z: http://vision.deis.unibo.it/fede/papers/eccv12.pdf

[13] OSADA, Robert, Thomas FUNKHOUSER, Bernard CHAZELLE a David DOBKIN. Shape distributions. ACM Transactions on Graphics [online]. 2002, vol. 21, issue 4, s. 807-832 [cit. 2015-05-25]. DOI: 10.1145/571647.571648.

Príloha A

Obsah DVD

Priložené DVD obsahuje:

- vytvorený balíček
- túto technickú správu
- plagát
- datovú sadu s obrázkami, bagfilemi a anotáciami
- meshe modelov z databáze
- súbor README.txt, kde je popísaný návod na spustenie spolu s požiadavkami na spustenie

Príloha B

Výstup služby *tabletop_object_recognition*

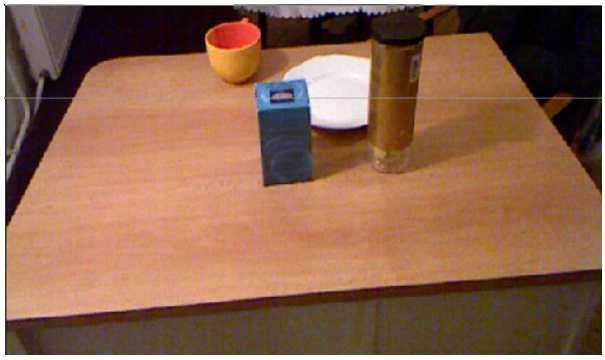
Príklad výstupu uzlu *ping_tabletop_node* pri správnej detekcii dvoch objektov na pracovnej doske a nastavení premennej *num_models* na hodnotu 2


```
{
  "clusters": {
    "0": {
      "centroid": {
        "x":0.0388941,
        "y":0.000713153,
        "z":0.756166
      },
      "models": [
        {
          "model_id": 19018,
          "pose": {
            "x": 0.0423939,
            "y": 0.0579826,
            "z": 0.832662
          },
         "score": 0.00328376
        },
        {
          "model_id": 19012,
          "pose": {
            "x": 0.0451293,
            "y": 0.0641074,
            "z": 0.82373
          },
         "score": 0.0072028
        },
      ]
    },
    "1": {
      "centroid": {
        "x":0.213734,
        "y":-0.0161419,
        "z":0.824928
      },
      "models": [
        {
          "model_id": 19007,
```

```
          "pose": {
            "x": 0.221845,
            "y": 0.0199796,
            "z": 0.874761
          },
         "score": 0.00596627
        },
        {
          "model_id": 19021,
          "pose": {
            "x": 0.218093,
            "y": 0.0275257,
            "z": 0.864239
          },
         "score": 0.0174492
        },
      ]
    }
  }
}
```

Príloha C

Ukážky scén z vytvorenej datovej sady








Príloha D

Plagát

Obrázok D.1: Plagát prezentujúci vytvorený balíček a príklady užitia