VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA STROJNÍHO INŽENÝRSTVÍ
FACULTY OF MECHANICAL ENGINEERING

ÚSTAV AUTOMATIZACE A INFORMATIKY
INSTITUTE OF AUTOMATION AND COMPUTER SCIENCE

BIN PICKING A ROBOTICKÉ VIDĚNÍ

BIN PICKING AND ROBOTIC VISION

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — Bc. Jan Múčka
AUTHOR

VEDOUCÍ PRÁCE — doc. Ing. Radomil Matoušek, Ph.D.
SUPERVISOR

BRNO 2019

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA STROJNÍHO INŽENÝRSTVÍ

Zadání diplomové práce

Ústav:	Ústav automatizace a informatiky
Student:	**Bc. Jan Múčka**
Studijní program:	Strojní inženýrství
Studijní obor:	Aplikovaná informatika a řízení
Vedoucí práce:	**doc. Ing. Radomil Matoušek, Ph.D.**
Akademický rok:	2018/19

Ředitel ústavu Vám v souladu se zákonem č.111/1998 o vysokých školách a se Studijním a zkušebním řádem VUT v Brně určuje následující téma diplomové práce:

Bin Picking a robotické vidění

Stručná charakteristika problematiky úkolu:

Jde o řešení velmi praktické úlohy výběru objektů, tj. součástek z kontejneru. Netriviálnost úlohy plyne ze složitosti dílů, i jejich předem nedefinované prostorové orientaci. Matematické postupy vyhledání shody modelu dílu s nasnímanými daty jsou stále předmětem výzkumu. Úloha bude řešena s reálným kooperativním robotem.

Cíle diplomové práce:

1. Rešerše přístupů robotického vidění pro aplikaci bin picking.
2. Popis technologií Intel RealSense a MS Kinect.
3. Tvorba aplikace pro robotické vidění s využitím ROS.
4. Aplikace pro segmentaci a geometrickou interpretaci několika zvolených objektů.

Seznam doporučené literatury:

PAJDLA, Tomáš. Global optimization in camera and robot calibration: Globální optimalizace v kalibraci kamer a robotů. V Praze: České vysoké učení technické, [2017]. ISBN 978-80-01-06114-5.

KOLÍBAL, Zdeněk. Roboty a robotizované výrobní technologie. Brno: Vysoké učení technické v Brně - nakladatelství VUTIUM, 2016. ISBN 978-80-214-4828-5.

Termín odevzdání diplomové práce je stanoven časovým plánem akademického roku 2018/19

V Brně, dne

L. S.

doc. Ing. Radomil Matoušek, Ph.D. ředitel ústavu	doc. Ing. Jaroslav Katolický, Ph.D. děkan fakulty

ABSTRAKT

Cílem této diplomové práce je popsat přístup robotického vidění pro aplikaci Bin Picking a vytvořit aplikaci pro realizaci této úlohy. Tato aplikace bude schopna rozlišit několik zvolených objektů na základě dat z kamery s hloubkovým vjemem a měla by umět hledaný objekt lokalizovat, rozeznat ho a v daném místě určit jeho polohu a orientaci. Řešení úlohy Bin Picking patří mezi největší výzvy v automatizaci a robotizaci dnešní doby.

ABSTRACT

The aim of this master's thesis is to describe the Robotic Vision for Bin Picking usage and creating an application for the realization of this task. This application will be able to distinguish several objects based on data from a camera with deep perception and should find the location of object, recognize it and determine its location and orientation. Bin Picking is one of the biggest challenges in today's automation.

KLÍČOVÁ SLOVA

Bin Picking, robotické vidění, PointCloud, 3D vize, Kinect, RealSense, podvzorkování, přiřazení šablony, povrchová normála, sousedství bodů, deskriptor, uzel

KEYWORDS

Bin Picking, Robot Vision, PointCloud, 3D Vision, Kinect, RealSense, Downsampling, Template Alignment, Surface Nomal, k-neighborhood, Descriptor, Node

BIBLIOGRAFICKÁ CITACE

MÚČKA, Jan. *Bin Picking a robotické vidění* [online]. Brno, 2019 [cit. 2019-05-22]. Dostupné z: https://www.vutbr.cz/studenti/zav-prace/detail/117275. Diplomová práce. Vysoké učení technické v Brně, Fakulta strojního inženýrství, Ústav automatizace a informatiky. Vedoucí práce Radomil Matoušek.

PODĚKOVÁNÍ

Chtěl bych poděkovat své rodině za podporu při psaní této diplomové práce, ochotným doktorandům našeho ústavu a vedoucímu doc. Ing. Radomilu Matouškovi, Ph.D.

ČESTNÉ PROHLÁŠENÍ

Prohlašuji, že tato práce je mým původním dílem. Zpracoval jsem ji samostatně pod vedením doc. Ing. Radomila Matouška, Ph.D. a s použitím literatury uvedené v seznamu literatury.

V Brně dne 24. 5. 2019 ……

Jan Múčka

OBSAH

1 ÚVOD

Svět se potýká s nejnižší nezaměstnaností za téměř 50 let a potřeba automatizace základních lidských činností výrazně narůstá. V USA, kde 38 % pracovních pozic vykonává přesun předmětu mezi zásobníkem a výrobním strojem [39], tedy provádí tzv. Bin Picking, zůstává 500 000 neobsazených pracovních míst. Automatizační průmysl se snaží tuto potřebu naplnit. Každoročně se na veletrzích automatizace představuje několik nových společností, které tvrdí, že konečně vyřešily tuto složitou úlohu. Bin Picking je anglický pojem, který se běžně nepřekládá a vyjadřuje techniku přesunutí neuspořádaných součástek ze zásobníku na pás či do výrobního stroje.

Pro člověka je Bin Picking každodenní praxí. Je schopen přesouvat objekty, třídit je a skládat, aniž by o tom přemýšlel. Například je schopen vybrat broskev z krabice, kde se nachází i jablka, housenky a jiné biologické objekty. Dalším takovým úkolem je například čištění či úklid stolu, což je pro člověka triviální úkol. Lidé automaticky dokážou aplikovat správnou sílu díky zkušenostem a rozhodovacím schopnostem.

Všechny výše uvedené situace jsou však pro robota velmi obtížné pro vyhodnocení a zpracování. Standardně by řešení tohoto úkolu vyžadovalo extrémní množství času pro programování a nastavení robota, aby byl schopen takové úkony dělat. Avšak ve spolupráci s chytrými senzorickými systémy je robot schopen skenovat povrch výrobků tak, aby je mohl vybrat a přesunout na jiné místo. Tyto procesy předcházející samotnému pohybu robota jsou úlohou robotického vidění. [38]

Cílem této diplomové práce je popsat přístup robotického vidění pro aplikaci Bin Picking a vytvořit aplikaci pro realizaci této úlohy. Tato aplikace bude schopna rozlišit několik zvolených objektů na základě dat z kamery s hloubkovým vjemem. Aplikace by měla umět hledaný objekt lokalizovat, rozeznat ho a v daném místě určit jeho polohu a orientaci.

Pro softwarové řešení této úlohy je využit systém ROS a knihovna Point Cloud Library obsahující mnoho implementovaných metod a algoritmů užitečných v problematice uvedeného typu. Z hlediska hardwaru jsou v této práci využity kamery s hloubkovým vjemem Intel RealSense R200 a především Microsoft Kinect Xbox 360. K realizaci samotné techniky Bin Picking je využit kooperativní robot UR3.

Stručně popsaný proces, který tato aplikace vykonává, se skládá ze zpracování snímku kamery ve formátu velké množiny bodů snímaného povrchu (tzv. mrak bodů, point cloud), výpočtu deskriptorů popisujících geometrii menších oblastí bodů a jejich přiřazení k několika vytvořeným šablonám hledaných objektů. Předměty jsou dopředu známy, tudíž pro ně není problém tyto šablony přichystat. Výstupem aplikace je výsledná translace a rotace šablony, tedy objektu v prostoru. Zbývá už jen navigovat robota na dané místo a tento předmět uchopit a přemístit.

V teoretické části práce jsou podrobně popsány prostředky využité k její realizaci a metody a algoritmy pracující s daty trojrozměrných bodových množin. Dále se pokračuje praktickou částí, kde je již popsána kompletní a nezbytná konfigurace

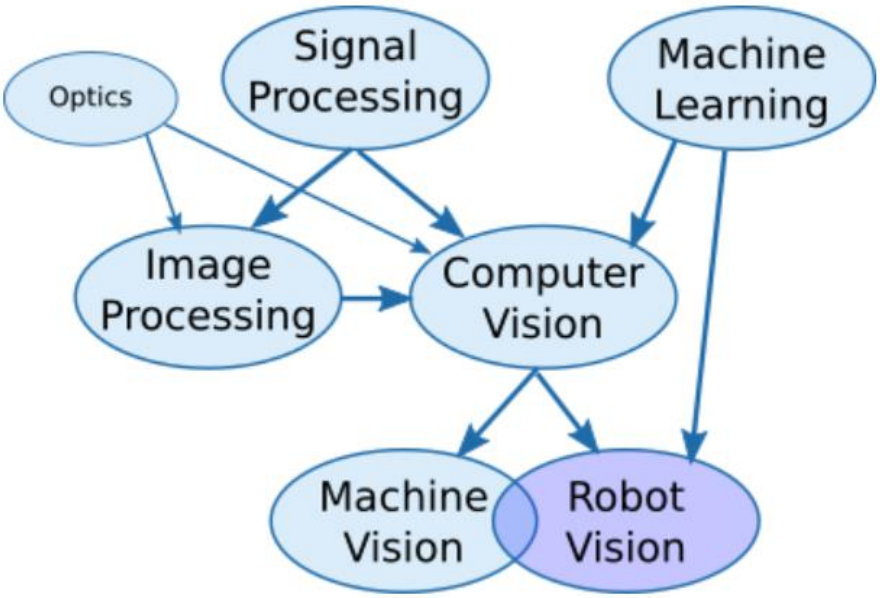
prostředků k tomu, aby tato aplikace mohla fungovat. V následující kapitole je podrobně popsaný zdrojový kód obsahující metody zpracovávání 3D obrazu, které jsou převzaté z open-source knihovny Point Cloud Library. V závěrečné kapitole jsou uvedeny výsledky realizace a provedeného testování.

2 ROBOTICKÉ VIDĚNÍ A BIN PICKING

2.1 Robotické vidění

V základním pojetí zahrnuje robotické vidění použití kombinace hardwaru kamery a počítačových algoritmů, které dovolí robotům zpracovávat vizuální data okolního prostředí. Systém může mít například 2D kameru detekující objekt, který má robot zvednout. Komplexnějším příkladem by mohlo být použití 3D stereo kamery pro navigaci robota k montáži kol na pohybující se vozidlo. [37]

Bez robotického vidění je daný robot v podstatě slepý. To sice není problém pro mnoho robotických úloh, ale pro některé aplikace je robotické vidění užitečné, nebo dokonce nezbytné. Robotické vidění úzce souvisí se strojovým viděním. A obě jsou úzce spjaty s počítačovým viděním. Kdyby se vytvořil rodokmen těchto pojmů, počítačové vidění by mohlo být považováno za jejich „rodiče“. Aby se dalo v těchto významově podobných pojmech vyznat, je zde uveden rodokmen souvisejících pojmů obsahující: optiku (Optics), zpracování signálu (Signal Processing), zpracování obrazu (Image Processing), počítačové vidění (Computer Vision), strojové učení (Machine Learning), strojové vidění (Machine Vision), robotické vidění (Robot Vision). [37]

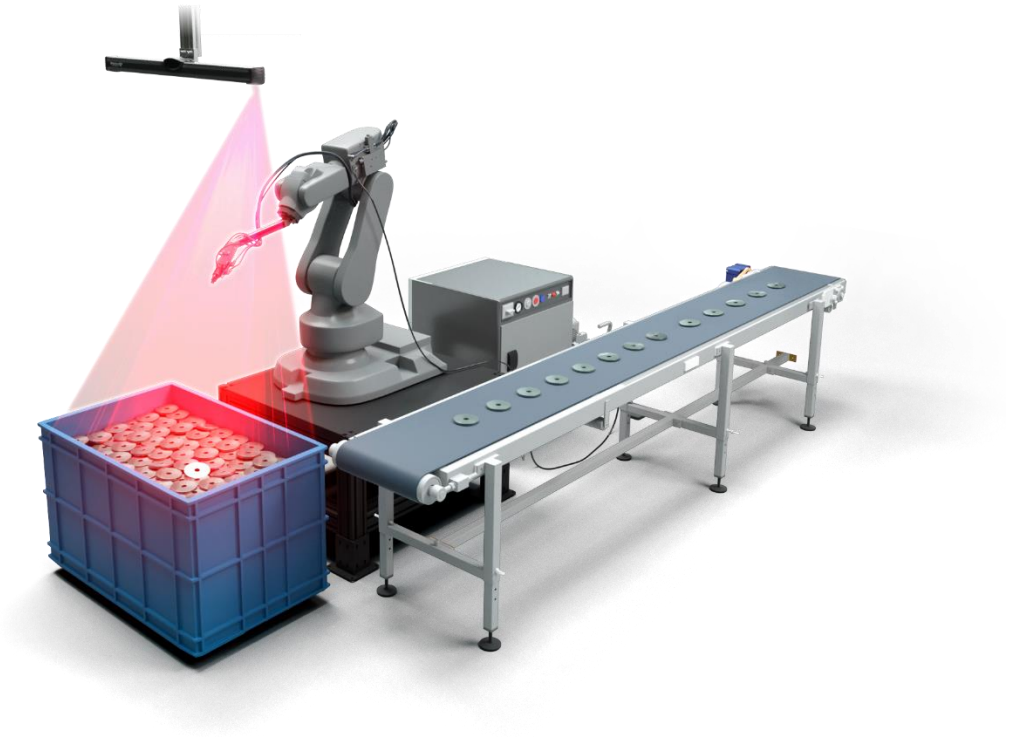


Obr. 1: Rodokmen technologických pojmů [37]

Velmi konkrétní aplikací robotického vidění je řešení úlohy Bin Picking. Tento problém je součást náplně této diplomové práce, a proto je mu věnována následující podkapitola.

2.2 Bin Picking

Bin Picking je název techniky, kterou robot používá k uchopení předmětů náhodně umístěných uvnitř krabice nebo na paletě. Tato technika umožňuje robotovi rychle a samostatně uchopit objekty bez nutnosti jeho zdlouhavého programování. Jedná se o velké zvýšení efektivity a pravděpodobně o jednu z největších výzev, se kterou se v současné době v oblasti robotických technologií lze setkat. [38]



Obr. 2: Příklad úlohy Bin Picking [40]

Manipulátory ve výrobních podnicích jsou velmi efektivní, protože jsou naprogramovány pro neustálé opakování stejných úkolů. Čas přípravného programování robotů je tak rozdělen mezi mnoho produktů. Tyto roboty jsou sice velmi rychlé, ale svoje vlastní rozhodovací schopnosti nemají. [38]

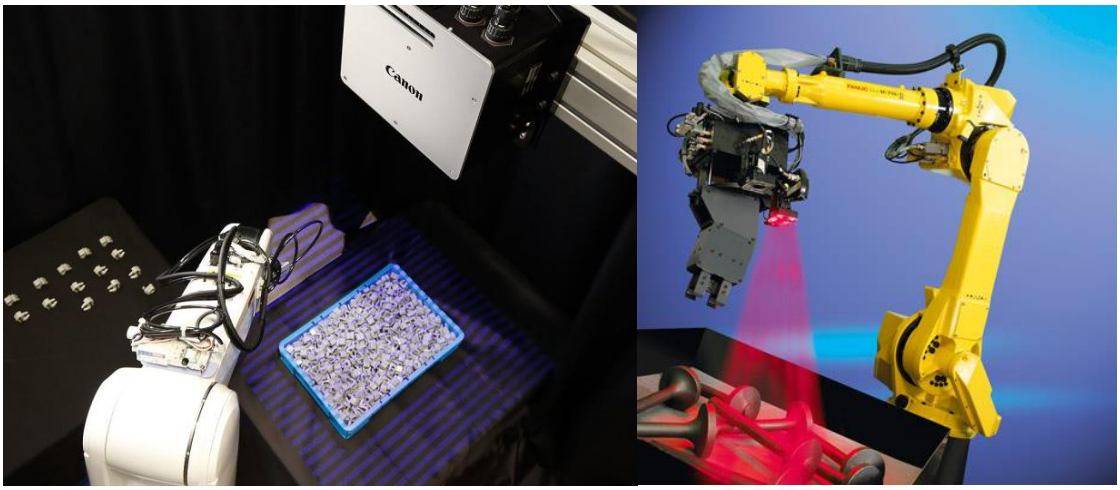
Technika Bin Picking je pro roboty obtížným úkolem, protože jednotlivé cykly robota jsou stěží stejné. Výrobky jsou roztroušeny náhodně uvnitř zásobníku a jedná se tak o nedeterministický úkol. Robot musí být vybaven vhodnými senzory pro rozlišení produktů a jejich přesnou lokalizaci. Navádění a plánování cesty robotického ramena je další z netriviálních úloh, robot se musí umět přesunout k předmětu bez jakýchkoliv kolizí s okolními objekty. Tento proces volá po opravdové revoluci v řízení robotů a v následujících podkapitolách jsou uvedeny některé velké firmy, které se problémem robotického vidění či celkové aplikaci Bin Picking zabývají. [38]

2.2.1 Fanuc

Firma Fanuc založená v roce 1956 se věnuje automatizaci v mnoha oblastech strojního inženýrství. Vyrábí průmyslové roboty a realizuje jejich aplikaci třeba i pro úlohy Bin Picking. Patří mezi největší špičku mezi firmami, které tento složitý problém umí řešit. Senzory využité k 3D vizi fungují na principech stereofonního vidění a infračerveného vidění. Firma vyvíjí širokou škálu senzorů a v aplikaci robotického vidění se jejich rozhodovací čas pohybuje kolem 100 až 300 milisekund. [41]

2.2.2 Cannon

Společnost Cannon se také zapojila do vývoje 3D vidění a vyvinula systém rozpoznávání objektů RV300 a RV500. Tento systém vyžaduje CAD data snímaných dílů a na jejich základě určí pozici a prostorovou orientaci daných objektů. Produkty, které firma Cannon představila, jsou ideální pro aplikaci robotického vidění úlohy Bin Picking. Přibližná doba trvání měření a rozpoznání objektů je 1,8 sekundy s přesností 0,1 až 0,15 mm. Počet dílu, které lze registrovat v jedné aplikaci, je až 200 variant modelů objektů. [43]



Obr. 3: Aplikace Bin Picking produktem Canon RV500 (vlevo) a firmou Fanuc (vpravo) [43][42]

2.2.3 Další

Většina velkých automatizačních firem se nějakým způsobem účastní vývoje v této problematice ať už výrobou 3D senzorů, (např. Kuka, Cognex, Fanuc), aplikací robotického vidění (např. Pickit, Cannon, Fanuc, Kinali soft), nebo samotnou realizací úlohy Bin Picking (Fanuc a jiné menší firmy).

3 POUŽITÉ TECHNOLOGIE

V této kapitole jsou popsány systémy a přístroje použité v této diplomové práci, které disponují technologií využitelnou pro robotické vidění. Produkty nabízející velké firmy uvedené v předchozí kapitole jsou velmi vyspělé a určené pro konkrétní využití automatizace v komerčním průmyslu. V této práci jsou však pro akademické účely využity univerzální a mnohem dostupnější technologie.

3.1 Kamery

3.1.1 Intel RealSense R200

Kamera R200 je produktem firmy Intel, která vyvíjí technologie hloubkového vidění s názvem RealSense. Toto zařízení funguje na USB 3.0 a může poskytovat barevné, hloubkové a infračervené videostreamy. Hloubkové videostreamy jsou podobné klasickému barevnému obrazu například z kamery telefonu, avšak s výjimkou toho, že každý pixel nese namísto informace o barvě hodnotu reprezentující vzdálenost od kamery. [28]

Obr. 4: Kamera Intel RealSense R200 [27]

Skládá se z infračerveného laserového projekčního systému, dvou infračervených senzorů a barevné full HD kamery. Hloubkový videostream je generován pomocí technologie stereofonního vidění s využitím infračerveného laserového projektoru a dvou infračervených senzorů. Modul R200 sice není primárně určen k pořizování fotografií, lze ho ale synchronizovat s fotoaparátem s vysokým rozlišením pro fotografování a aplikace hloubkového vidění. [28]

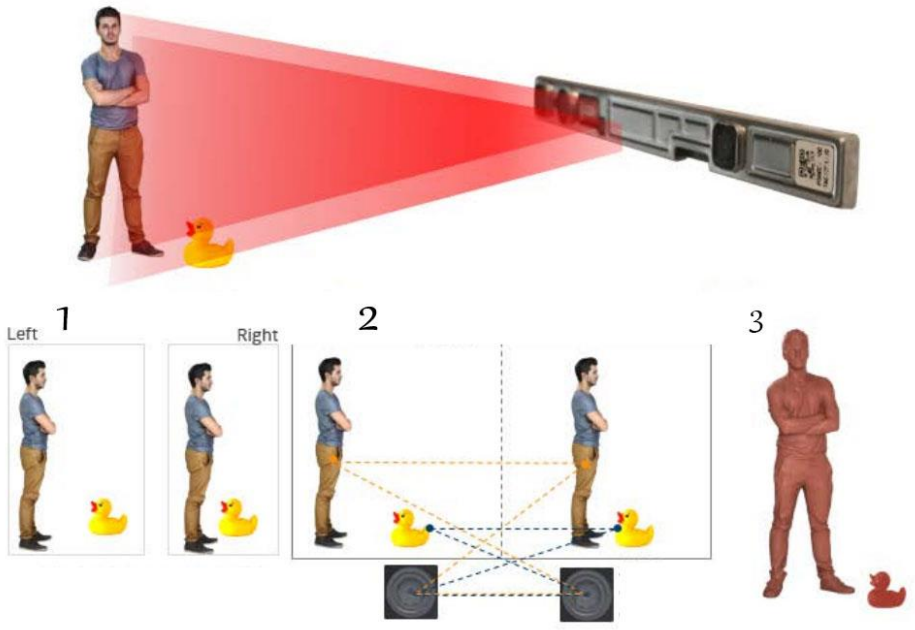
Obr. 5: Umístění komponent v kameře R200 [27]

Obr. 6: Příklad možných typů obrazů kamery R200 [27]

Modul R200 používá pro výpočet hloubky stereofonní vidění. Implementace tohoto vidění se skládá z levé infračervené kamery, pravé infračervené kamery a infračerveného laserového projektoru. Levá a pravá data kamery jsou odesílána do zařízení R200 ASIC. ASIC vypočítá hodnoty hloubky pro každý pixel v obraze. Infračervený projektor slouží ke zvýšení schopnosti systému vypočítat hloubku ve scénách s nevýraznou texturou. Scény, jako jsou rovné stěny, představují výzvu pro stereofonní systémy a pro jejich výpočet hloubky. [28]

Na následujícím obrázku jsou znázorněny kroky získání hloubkového obrazu:

1) Projekce infračerveného vzoru na prosté povrchy bez textur, např. stěny. Každá kamera vidí poněkud odlišný pohled.
2) Posunutí bližších objektů je větší než u těch vzdálenějších. ASIC vypočítá posunutí každého pixelu mezi levým a pravým snímaným obrazem a na základě triangulace dopočítá jejich hloubku.
3) Výsledná hloubková mapa bodů. [28]



Obr. 7: Přehled funkce stereofonního vidění [28]

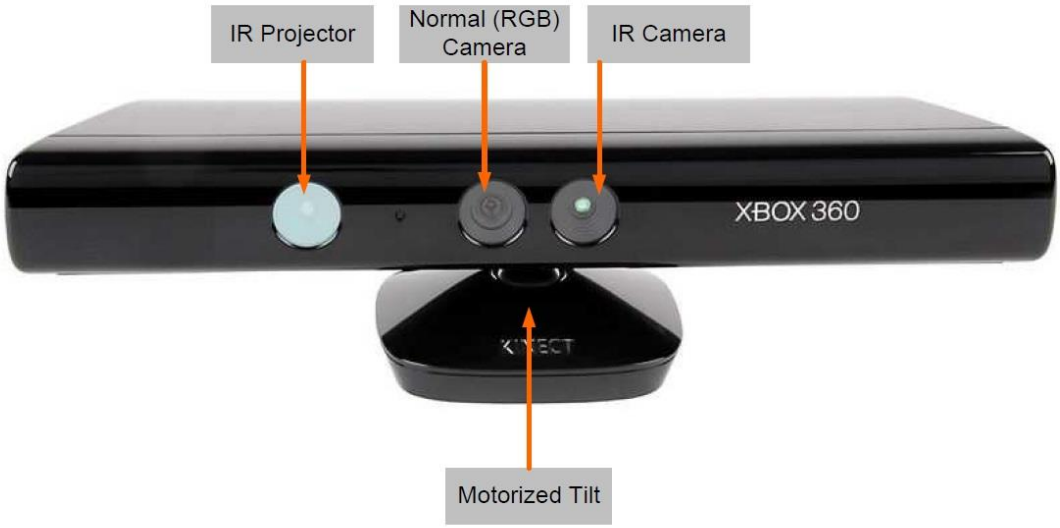
Kamera je velmi kompaktní a vzhledem k její nízké váze a malým rozměrům nemá na trhu konkurenci. Na zadní straně její konstrukce má malý magnet, který ji dělá velmi snadno připevnitelnou na jakýkoli kovový materiál. Její technické parametry jsou uvedeny v následující tabulce:

	Barva	Hloubka / IR
Aktivní pixely	1920x1080	640x480
Poměr stran	16:9	4:3
Snímková frekvence	30 fps	30/60 fps
Zorné pole (D x V x H)	77° x 43° x 70° (kužel)	70° x 46° x 59° (kužel)
Operační hloubka		0,4 – 2,8 m

Tab. 1: Popis parametrů zobrazovacích komponent kamery R200 [27]

3.1.2 Microsoft Kinect Xbox 360

Kinect pro Xbox 360 byl unikátní kamerou postavenou pro zábavový průmysl společností Microsoft. Hardware zahrnoval technologii čipové sady od izraelského vývojáře PrimeSense, který vyvinul systém sestávající z infračerveného projektoru a kamery se speciálním mikročipem, který generuje mřížku, ze které lze zjistit umístění snímaného objektu ve 3 rozměrech. Tento systém 3D skeneru s názvem Light Coding využívá rekonstrukci 3D obrazu. [29]



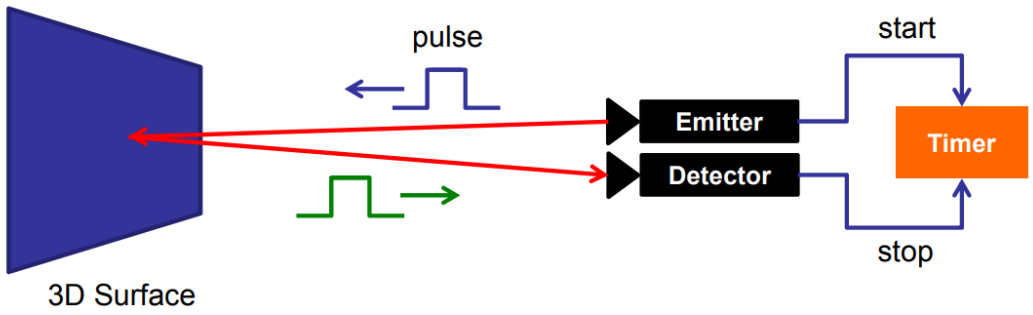
Obr. 8: Popis komponent kamery Kinect Xbox 360 [31]

Kamera Kinect s hloubkovým vjemem je snímací zařízení připevněné na malém stojanu s motorizovaným otočným čepem, který umožňuje vertikální natáčení kamery až o 27°. Přístroj je vybaven RGB kamerou, hloubkovým senzorem a mikrofonním snímačem pole s proprietárním softwarem. Tyto komponenty zajišťují spolehlivé zachycení 3D pohybu, rozpoznávání obličeje a rozpoznávání hlasu. [29]

Mikrofonní pole senzoru Kinect umožňuje aplikaci Xbox 360 provádět lokalizaci akustického zdroje spolu s potlačením okolního hluku. To umožňuje použití i v rušných prostředích. [29]

Snímač hloubky se skládá z infračerveného laserového projektoru kombinovaného s černobílým snímačem CMOS, který zachycuje video data ve 3D za jakýchkoliv podmínek okolního osvětlení. Snímací rozsah hloubkového snímače je nastavitelný a software Kinect je schopen automaticky kalibrovat senzor dle fyzického prostředí v okolí kamery a přizpůsobit ho například přítomnosti nábytku nebo jiných překážek. [29]

Technologie hloubkového vjemu kamery Kinect využívá podobné komponenty jako u kamery RealSense R200. Zásadní rozdíl je, že kamera Kinect má jen jednu infračervenou kameru, což ve výsledku ničemu nevadí, protože všechny její komponenty jsou výkonově na mnohem vyšší úrovni. Kombinace IR projektoru a snímače jednoduše počítají vzdálenost všech pixelů scény metodou ToF (Time-of-Flight), jak je znázorněno na obrázku níže. [31]



Obr. 9: Princip měření vzdálenosti ToF [31]

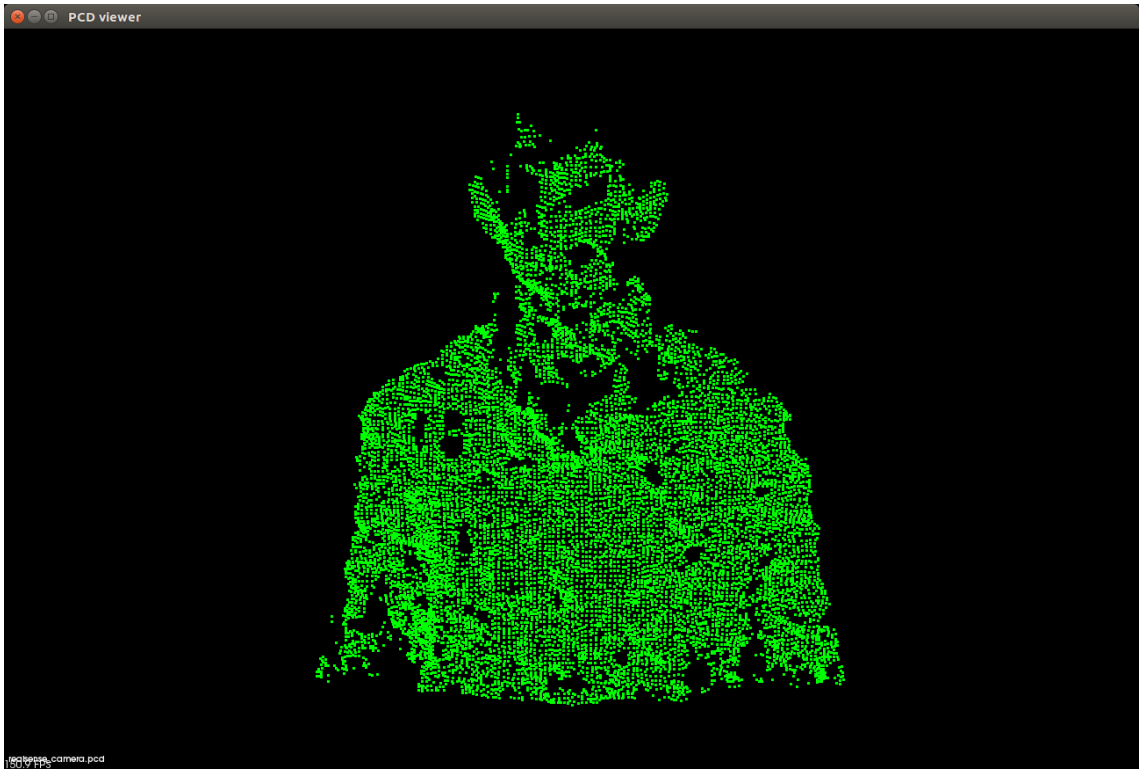
V následující tabulce jsou uvedeny technické parametry:

	Barva	Hloubka / IR
Aktivní pixely	640x480	640x480
Poměr stran	4:3	4:3
Snímková frekvence	30 fps	30 fps
Zorné pole (V x H)	43° x 70°	43° x 57°
Operační hloubka		0,7 – 6 m
Možný vertikální náklon	27°	

Tab. 2: Popis parametrů zobrazovacích komponent kamery Xbox 360 Kinect [30]

3.2 Srovnání kamer

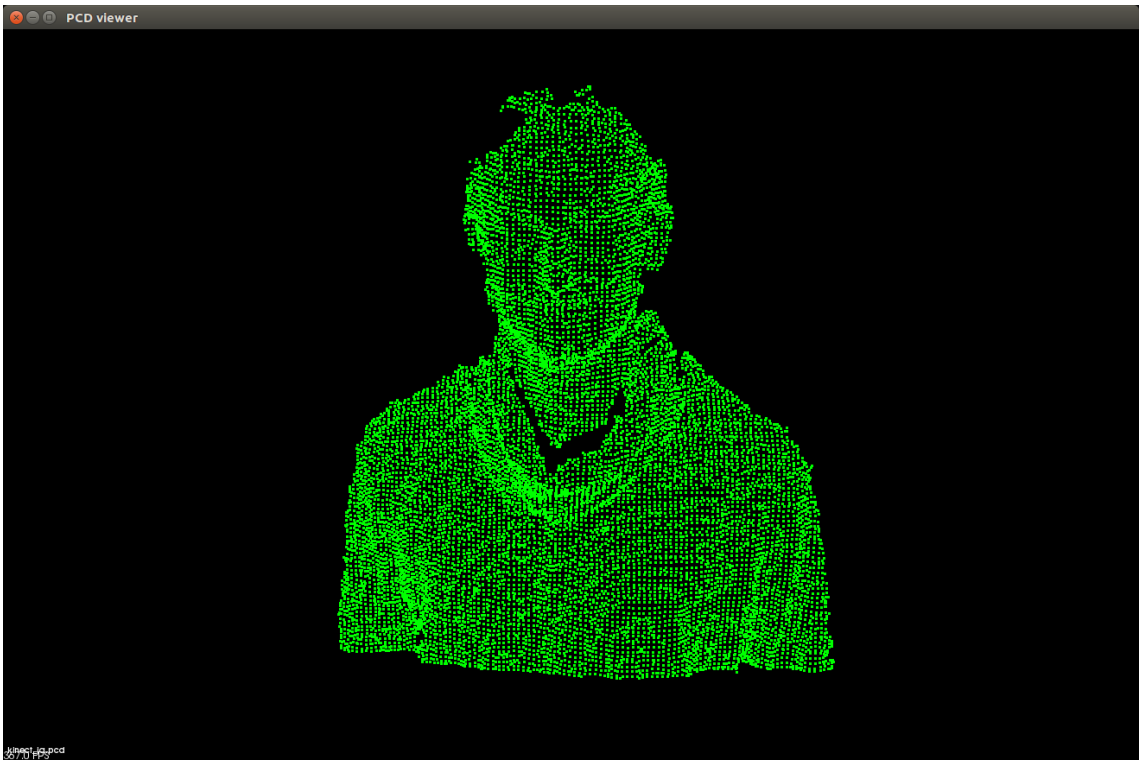
Kamera Intel RealSense R200 je velice kompaktní malé zařízení, které lze zakoupit za velmi nízkou cenu a cílí tak na poměrně jednoduché aplikace ideální například pro lokalizaci překážek, vytváření 3D map pro prostorovou orientaci nějakého vozítka či k hrubým rekonstrukcím 3D objektů, ke kterým je však nutno využít mnoho rekonstruktivních operací. Pro příklad je níže uveden snímek bodové mapy lidské postavy.



Obr. 10: Snímek bodové mapy lidské postavy kamerou R200

Kamera Microsoft Kinect Xbox 360 je poměrně větší zařízení, které není přímo určené k výzkumným účelům. Avšak díky své vyšší úrovni senzorů a rozsáhlejšímu softwaru se dá použít jak na většinu aplikací podobných pro kameru R200, tak na aplikace mnohem složitější. Díky tomu, že byla tato kamera koncipována pro hraní videoher, obsahuje užitečné implementace, jako jsou rozpoznávání obličeje, pohybů člověka a hlasu. Nejdůležitějšími vlastnostmi tohoto zařízení (v oblasti hloubkového vjemu) jsou v porovnání s kamerou R200 vyšší přesnost, konzistentnost, menší náchylnost vůči netexturovaným či lesklým povrchům a větší snímaná vzdálenost. Pro výzkumné účely této diplomové práce byla v závěrečném testování použita právě kamera Kinect.

Na následujícím obrázku je pro srovnání s kamerou R200 ukázka snímku pořízeného kamerou Kinect na stejné lidské postavě.



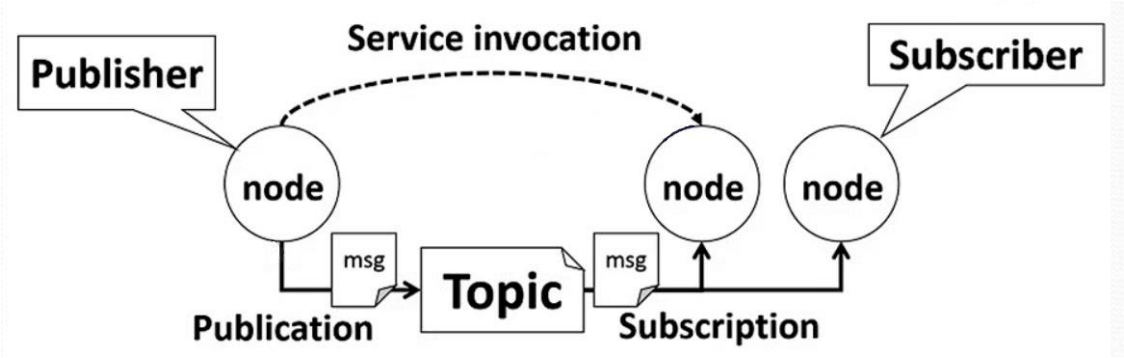
Obr. 11: Snímek bodové mapy lidské postavy kamerou Kinect

3.3 ROS

The Robot Operating System (ROS) je flexibilní framework k vývoji softwaru pro roboty. Jedná se o soubor nástrojů, knihoven a konvencí, jejichž cílem je zjednodušit úpravu již existujících řešení nebo úlohu vytváření komplexních a robustních systémů chování robotů v celé řadě robotických platforem. Hlavní výhodou tohoto systému je velká sada ovladačů a implementovaných algoritmů široce používaných v robotice. [34]

Obr. 12: Logo systému ROS [36]

ROS je navržen jako volně spojený systém, kde se proces nazývá uzel (node) a každý uzel by měl být zodpovědný za jeden úkol. Uzly mezi sebou komunikují pomocí zpráv (messages), které procházejí logickými kanály nazývanými témata (topics). Každý uzel může odesílat nebo získávat data z jiného uzlu modelem publish/subscribe. [35]



Obr. 13: Schéma komunikační struktury ROS [33]

3.3.1 Uzly (nodes)

Základní jednotka ROSu se nazývá uzel. Uzly mají na starosti manipulační zařízení nebo výpočetní algoritmy, každý uzel pro samostatný úkol. Uzly mohou mezi sebou komunikovat pomocí témat nebo služeb. Software ROS je distribuován v balíčcích. Jeden balíček je obvykle vyvinut pro provádění jednoho typu úlohy a může obsahovat jeden nebo více uzlů. [34]

3.3.2 Témata (topics)

Téma je datový kanál používaný pro výměnu informací mezi uzly. Používají se k odesílání častých zpráv jednoho typu, např. data senzoru nebo rychlost motoru. Každé téma je registrováno pod jedinečným názvem a definovaným typem zpráv. Uzly se s tématy mohou spojit pomocí metody publisher nebo subscriber a posílané zprávy tak vysílat nebo přijímat. Pro dané téma na něm nemůže jeden uzel současně vysílat a přijímat zprávy, ovšem není zde žádné omezení v počtu publikování nebo odběru různých uzlů. [34]

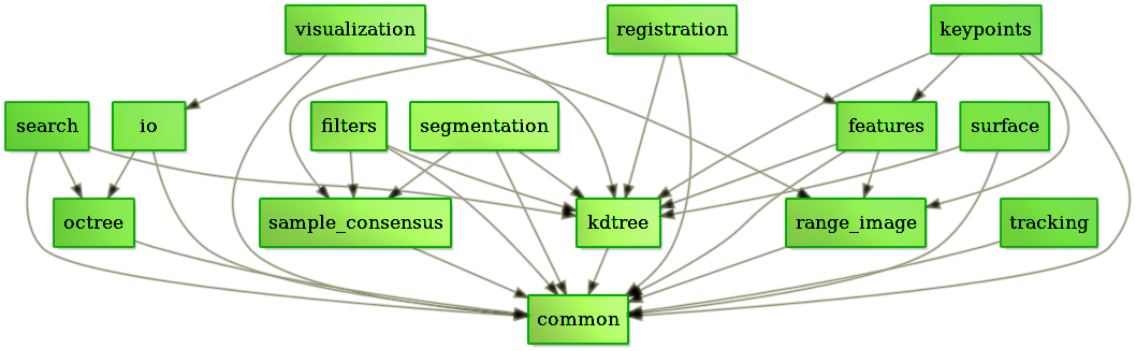
3.3.3 Služby (services)

Komunikace službami se podobá modelu klient-server. V tomto režimu je v systému registrován jeden uzel (server) a později může o tuto službu požádat jakýkoli jiný uzel a získat odpověď. Na rozdíl od témat umožňují služby obousměrnou komunikaci, protože požadavek může obsahovat i nějaká data. [34]

3.4 Point Cloud Library (PCL)

Knihovna Point Cloud Library (PCL) je rozsáhlý open-source projekt pro zpracování 2D/3D obrazu a PointCloud. Framework PCL obsahuje mnoho nejmodernějších algoritmů včetně filtrace, odhadu funkcí, rekonstrukce povrchu, registrace, přiřazení modelu a segmentace. Tyto algoritmy mohou být použity například pro filtraci hodnot z dat obsahujících mnoho šumu, spojování 3D PointCloud dohromady, segmentaci relevantních částí snímku, extrakce klíčových bodů, výpočet deskriptorů k rozpoznávání objektu na základě jeho geometrického tvaru, vytváření PointCloud a jejich vizualizaci atd. [11]

PCL je vydávána v souladu s třetí klauzulí BSD licence a jedná se tak o open-source software. Je zdarma pro komerční i výzkumné účely. Je také multiplatformní a lze tak využít v systémech Linux, MacOS, Windows a Android/iOS. Knihovna je implementovaná v C++ a pro její zjednodušení jsou kódy PCL rozděleny do řady menších knihoven, lze je tak kompilovat samostatně. Využívá-li se PCL na platformách s nižším výpočetním výkonem nebo velikostí operační paměti a úložného místa, je tato modularita poměrně důležitá. Pro představu je zde uveden graf C++ knihoven: [11]



Obr. 14: Grafické znázornění C++ knihoven [11]

4 TEORETICKÝ ZÁKLAD

Lokalizace 3D objektů v prostoru je velmi náročný a komplikovaný problém. Pro aplikaci Bin Picking je však nezbytné, aby byl manipulátor při odebírání neuspořádaných objektů neustále navigován a řízen, protože se jeho trajektorie cyklu pokaždé liší.

Osamocený objekt má v třídimenzionálním prostoru šest stupňů volnosti, tedy tři posuvné na osách *x*, *y*, *z* a tři rotační kolem nich, které jsou dané buď Eulerovými úhly, nebo rotační maticí. Úlohou robotického vidění je tedy správně odhadnout translaci a rotaci daného tělesa vůči robotu, který má být k tomuto tělesu naveden.

Využívá-li se kamera s hloubkovým vjemem, měl by celý systém fungovat tak, že se nejprve získá snímek bodů tvořících povrch scény a vypočítají se k nim deskriptory popisující geometrii daných okolí bodů, to se porovná se známými šablonami objektů, které má systém rozpoznat. Následně se pak program v reálném čase snaží ve snímaném povrchu nalézt s těmito šablonami shodnou orientaci a translaci. [10]

V případě této práce je k tomuto účelu využit suboperační systém ROS [20], ve kterém se spouští kamera Microsoft Xbox 360 Kinect a řídicí aplikace, jejíž algoritmy a metody pocházejí z open-source knihovny Point Cloud Library (PCL) [19]. Tato knihovna se zabývá podobnými typy problémů a má pro ně implementované četné funkce a zpracovanou rozsáhlou dokumentaci. Metody využité v diplomové práci jsou v této kapitole podrobně technicky popsány.

Nejprve je popsaný datový formát bodů v prostoru PointCloud, dále pak význam klíčových bodů a podvzorkování, datové struktury K-d tree sloužící k hledání okolí bodů, výpočet povrchových normál, které se využívají k jednoduché identifikaci geometrie povrchu, a jsou zde vysvětlena různá úskalí tohoto odhadu. Následuje popis metody výpočtu deskriptorů, které již obsahují více informací o okolí bodů a nesou tak lokální data připravená pro srovnávání. Nakonec jsou vysvětleny procesy párování deskriptorů a shlukování korespondencí, které vykonává algoritmus Sample Consensus Initial Alignment (SAC-IA).

4.1 PointCloud

PointCloud (volně přeloženo – Mračno bodů) je zaužívaný termín představující množinu bodů uspořádaných v třídimenzionálním souřadnicovém systému. V PCL knihovně se jedná o datovou strukturu, která krom souřadnic jednotlivých bodů může obsahovat i informaci o barvách bodů daných modelem RGB aj. PointCloud bývá typickým výstupem z kamer s hloubkovým vjemem nebo 3D skenerů. [10]

4.2 Klíčové body

Kamera Kinect snímá PointCloud s 307 200 body, tj. rozlišení 640x480. V případě této práce se sice zpracovává pouze jeden snímek, ale i tak toto číslo bodů představuje vysoký

nárok na výpočetní výkon. Provedení jednoduché operace pro každý bod v PointCloud by odpovídalo časové složitosti $O(n)$, kde n je počet bodů, např. 307 200. Porovnává-li se každý bod s k sousedními body v jeho oblasti sousedství, časová složitost narůstá na $O(nk)$. [10] Informace získané z výpočtů na datové množině bodů v této aplikaci by byly v případě plného rozlišení zbytečně redundantní.

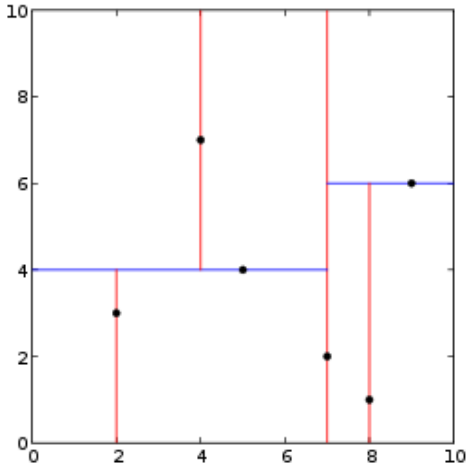
Proto se využívají metody, které odstraní nepotřebné body z PointCloud a sníží tak jeho výpočetní složitost. Těmito procesy se získají tzv. klíčové body, což jsou body dostatečně charakteristické, aby reprezentovaly celé oblasti původních bodů. Jejich identifikace probíhá na základě detekčního kritéria. Způsob využitý v této práci, kterým získáváme klíčové body, se nazývá podvzorkování. [10]

4.2.1 Podvzorkování

Kvůli úspoře výpočetního výkonu se redukuje množství bodů v PointCloud, proto se původní body překryjí mřížkou nových bodů s pravidelným vzorkováním, které by měly zachovat původní tvar i přesnost. Tato metoda se nazývá voxelizace (voxel – 3D pixel). Nejprve se PointCloud rozdělí do mnoha oblastí tvaru krychle (voxelů) s požadovaným měřítkem vzorkování. V těchto oblastech se nacházejí skupiny bodů, a jelikož cílem je, aby uvnitř krychle zůstal jenom jeden bod, vypočítá se centroid oblasti. Tento proces by se dal přirovnat ke snížení rozlišení obrázku, kde se sníží počet pixelů ovšem ve 2D. V tomto případě se tedy jedná o snížení počtu voxelů v třídimezionálním prostoru. [10]

4.3 K-d tree

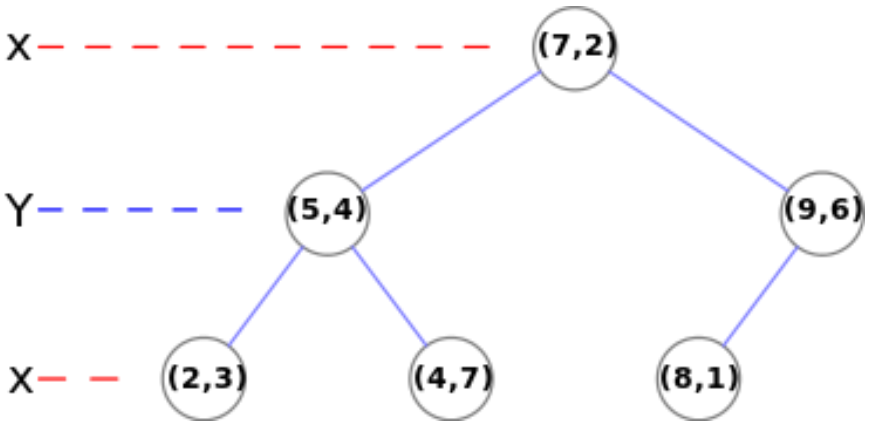
K-d tree nebo také K-dimensional tree je datová struktura používaná v informatice pro uspořádání určitého počtu bodů v k-dimenzionálním prostoru. Jedná se o binární vyhledávací strom splňující vhodné podmínky pro účely tohoto příkladu. K-d stromy jsou velmi užitečné pro hledání vzdáleností a vyhledávání nejbližších sousedních bodů. [7]



Obr. 15: Příklad 2D K-d stromu s daty: (2,3), (5,4), (9,6), (4,7), (8,1), (7,2) [7]

V úloze této metody se řeší případ, kdy je struktura PointCloud tvořena třemi dimenzionálními rozměry, K-d stromy jsou tak také třídimenzionální. Každá úroveň K-d stromu rozděluje všechny potomky po dané dimenzi využitím hyperplochy, která je kolmá na odpovídající osu. V kořenu stromové struktury jsou potomci rozděleni dle první dimenze následovně: Je-li první souřadnice menší než kořen stromu, umístí se do levého substromu, a pokud je větší, tak do pravého. Každá nižší úroveň ve stromu se dělí na další dimenzi a k první dimenzi se vrací, jakmile jsou všechny ostatní vyčerpány. [6]

Nejúčinnějším způsobem, jak vytvořit K-d strom, je použít dělící metodu a stejně jako algoritmus Quick Sort určit mediánový bod jako kořen a všechny souřadnice s menší hodnotou umístit doleva a ty s větší doprava. Tato procedura se opakuje jak na levém, tak na pravém substromu, dokud se poslední stromy určené k rozdělení neskládají pouze z jednoho prvku. [6]



Obr. 16: Výsledný K-d strom z příkladové datové množiny [7]

4.4 Odhad povrchových normál v PointCloud

Povrchové normály jsou důležitými charakteristickými prvky geometrického povrchu a jsou často využívány v různých aplikacích počítačové grafiky týkajících se přesných světelných zdrojů, které generují stínování a další vizuální efekty. [8]

Vzhledem ke geometrickému povrchu je obvykle triviální odvodit směr normály v určitém bodě povrchu, a to jako vektor kolmý k tomuto povrchu v daném bodě. Jelikož však datové množiny bodů získané z kamery představují komplikované vzorky z reálného povrchu, existují dvě možnosti, jak vypočítat normály:

- získat plochu prolínající se s příslušnými daty PointCloud za použití technik surface meshing a dopočítat tak povrchové normály z vytvořené sítě;
- využití aproximační metody k odvození povrchových normál přímo z datové množiny PointCloud. [8]

V aplikaci Template alignment je dle možností knihovny PCL k výpočtu povrchových normál využita právě druhá možnost, a to aproximace přímo z PointCloud. Ačkoliv existuje mnoho různých metod pro odhad normál, využívá se v tomto případě ta

nejjednodušší. Problém určování normály k povrchu v daném bodě je zjednodušen na problém určení normály tečné plochy k povrchu. Tato plocha je aproximována za použití metody nejmenších čtverců. [8]

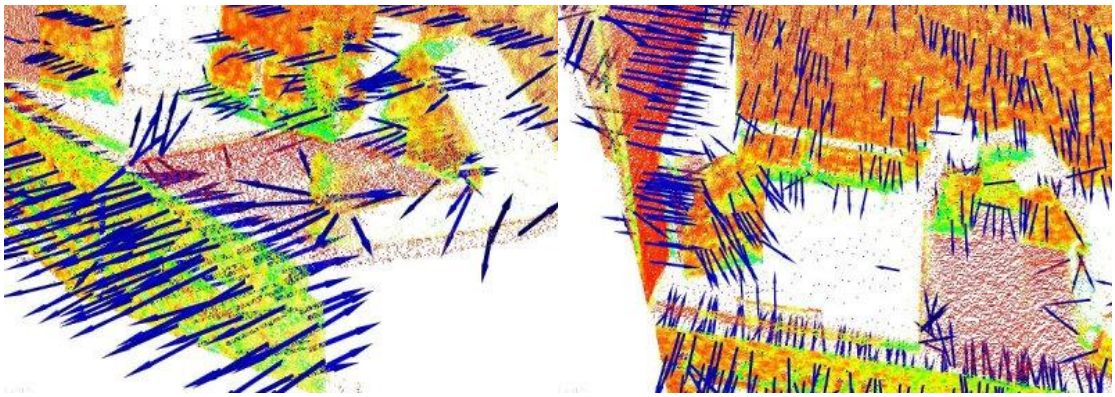
Řešení pro odhadování normál povrchu je proto redukováno na analýzu vlastních vektorů a vlastních čísel (nebo PCA – Analýza hlavních komponent) kovarianční matice, vytvořené z nejbližších sousedů dotazovaného bodu. Konkrétně pro každý bod p_i se sestaví kovarianční matice C: [8]

$$C = \frac{1}{k}\sum_{i=1}^{k} \cdot (p_i - \bar{p}) \cdot (p_i - \bar{p})^T , \quad C \cdot \vec{v_j} = \lambda_j \cdot \vec{v_j} , \quad j \in \{0, 1, 2\} \tag{4.1}$$

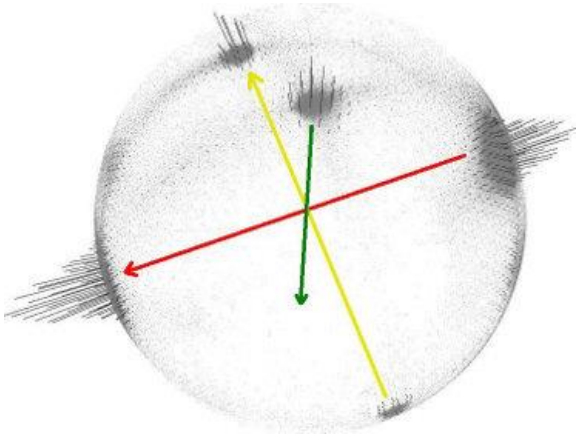
Kde k je počet sousedících bodů uvažovaných v sousedství p_i, $\bar{p}$ reprezentuje třídimenzionální těžiště nejbližších sousedů, λ_j je j-té vlastní číslo kovarianční matice a $\vec{v_j}$ je j-tý vlastní vektor. [8]

Protože obecně neexistuje žádný matematický způsob, jak řešit znaménko normály, je orientace vypočtená pomocí Analýzy hlavních komponent (PCA) nejednoznačná a není tak konzistentní v celé datové množině bodů PointCloud. Pro příklad níže uvedené dva obrázky ukazují důsledek tohoto efektu na dvou částech větší datové množiny bodů, zachycené z kuchyňského prostředí. [8]

Níže na obrázku číslo 5 je zobrazen rozšířený Gaussův obraz (EGI), známý také jako normálová koule, který popisuje orientaci všech normál z PointCloud. Vzhledem k tomu, že datové množiny jsou v tzv. 2,5D, byly tak získány z jediného zorného úhlu, a proto by měly být normály přítomny pouze na jedné polovině koule v obraze EGI. Nicméně vzhledem k nekonzistentnosti orientace normál jsou rozptýleny po celé sféře. [8]



Obr. 17: Příklad nekonzistentního určení orientace povrchových normál [8]

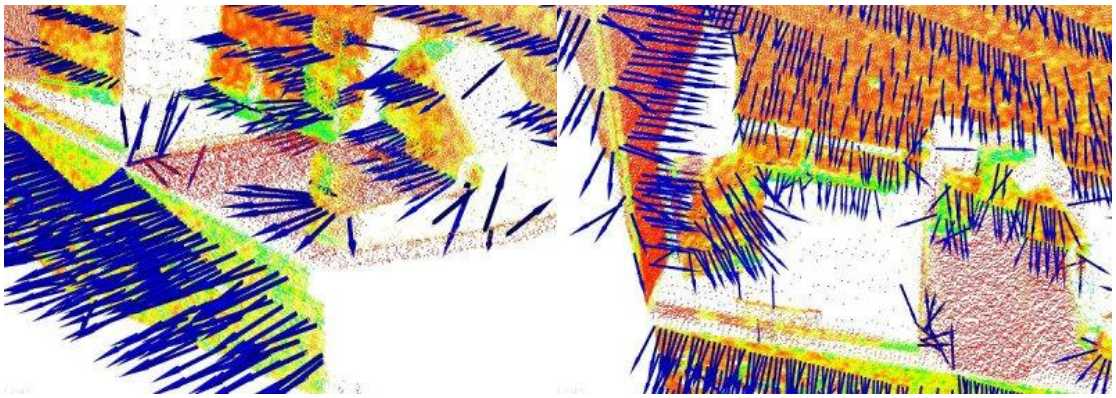


Obr. 18: Gaussův obraz orientace povrchových normál, po celé sféře [8]

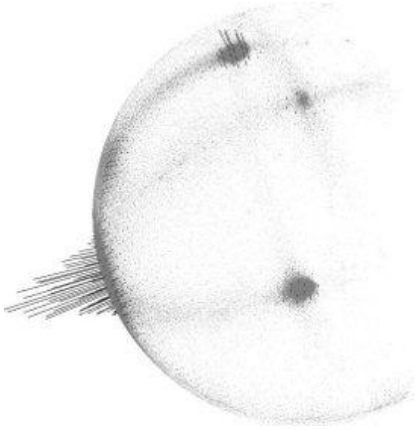
Řešení tohoto problému je triviální, pokud je pozice hlediska v_p známá. K tomu, aby byly všechny normály $\vec{n}_i$ orientovány souhlasně vůči pozorovacímu bodu, je zapotřebí splnit následující podmínku: [8]

$$\vec{n}_i \cdot (v_p - p_i) > 0 \tag{4.2}$$

Níže umístěné obrázky ukazují žádaný výsledek, kdy jsou všechny normály ze stejné datové množiny orientovány souhlasně vůči hledisku. [8]



Obr. 19: Příklad konzistentního určení orientace povrchových normál [8]



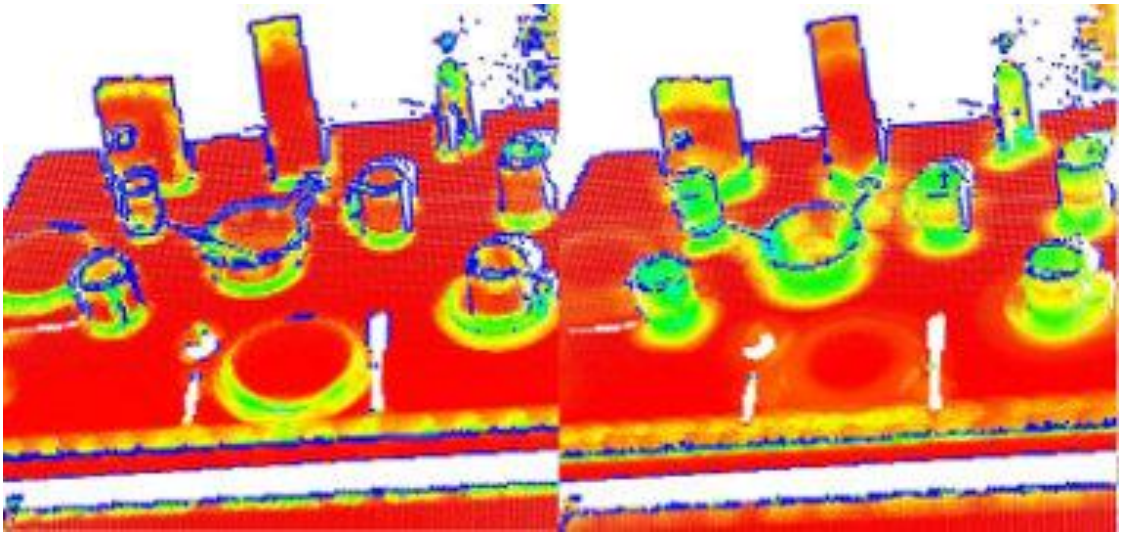
Obr. 20: Gaussův obraz orientace povrchových normál, na půlce sféry [8]

4.5 Výběr vhodného měřítka vzorkování

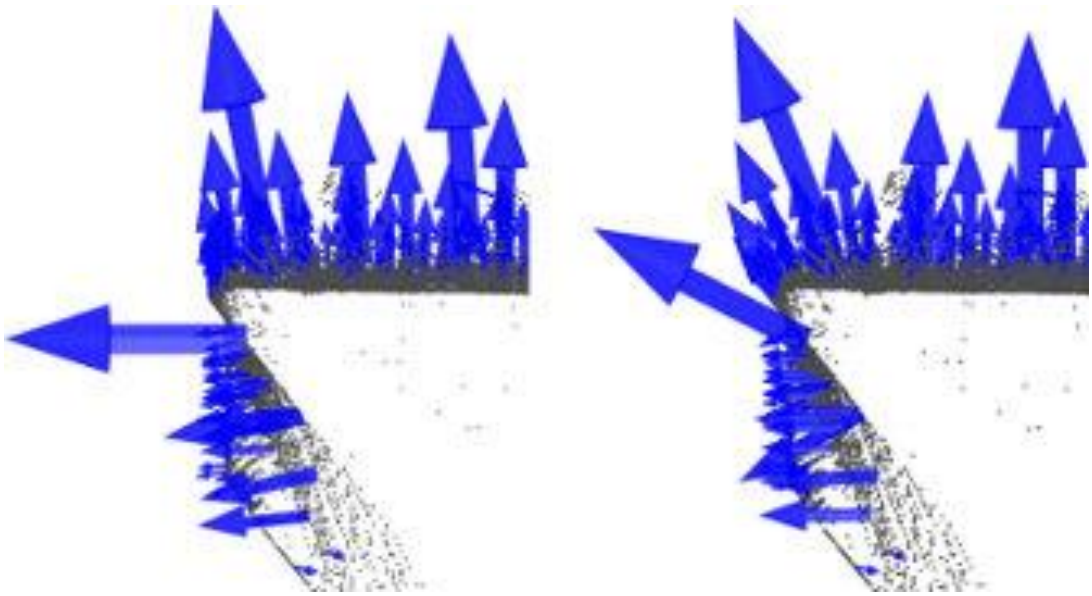
Jak již bylo vysvětleno, povrchová normála v bodě potřebuje ke svému určení sousedící body z jeho okolí (též se nazývá k-neighborhood). Specifika problému odhadu nejbližších sousedících bodů vedou k otázce určení správného měřítka vzorkování datové množiny PointCloud. Tedy, jaké jsou správné hodnoty proměnných **k** (pro metodu `pcl::Feature::setKSearch`) nebo **r** (pro `pcl::Feature::setRadiusSearch`), které jsou použity při určování skupiny nejbližších bodů. [8]

Tento problém má mimořádný význam a představuje omezující faktor v problematice automatického odhadu (tj. bez toho, aniž by uživatel zadal okrajové podmínky) bodově interpretovaných prvků. [8]

Pro lepší ilustraci tohoto problému je níže uvedeno srovnání dvou obrázků s různým nastavením (obr. č. 9). Jeden ukazuje účinky při výběru menšího měřítka (tj. malého **r** nebo **k**) proti většímu měřítku (tj. velkému **r** nebo **k**). Levá strana obrázků ukazuje poměrně rozumně zvolené měřítko vzorkování s povrchovými normálami určenými přibližně kolmě pro dva ploché povrchy a několik malých hran viditelných po celém stole. Pokud je však měřítko vzorkování příliš velké (pravá část), je tak oblast sousedství širší a je tvořena větším krycím povrchem okolo sousedících bodů. Důsledkem toho jsou odhadované bodové oblasti reprezentující povrch zkresleny. Příznakem jsou pak pootočené povrchové normály na hranách a okrajích ploch, jež mají za následek rozmazané hrany objektů a potlačení jemných detailů, jak je vidět na obrázku č. 8. [8]



Obr. 21: Porovnání kvality vytvořeného povrchu s různým měřítkem vzorkování [8]



Obr. 22: Porovnání kvality odhadu povrchových normál kolem rohu stolu [8]

K řešení této problematiky je nutné si uvědomit, že měřítko vzorkování pro výpočet sousedících bodů musí být vybráno na základě úrovně detailů, které tato aplikace vyžaduje. Jinými slovy, pokud je nějaké důležité zakřivení nebo hrana, která by měla být na snímaném objektu viditelná, mělo by být měřítko vzorkování menší a naopak. [8]

4.6 Deskriptory

Deskriptor je datový soubor nesoucí informace o geometrii okolí dotazovaného bodu. V kapitole 4.4 již byly představeny povrchové normály, které jsou v podstatě základními deskriptory popisujícími povrchové geometrie okolí bodů. Jejich parametry pro výpočet jsou **k** počet zvolených bodů ze sousedství dotazovaného bodu a poloměr **r** určující oblast sousedství. [10]

Deskriptory a jejich výpočetní techniky jsou však obecně mnohem složitější, přesnější a obsahují podstatně víc informací o geometrii povrchu v okolí daného bodu. K dosažení optimálních deskriptorů je potřeba znát následující kritéria, která by měla splňovat:

- Odolnost vůči transformacím – konzistentnost vzdálenosti mezi body při translačních nebo rotačních transformacích.
- Odolnost vůči šumu – nízký vliv šumu vzniklého nedokonalým měřením na odhad deskriptorů.
- Konzistentnost při změně měřítka vzorkování – minimální odlišnost odhadu deskriptorů při změně vzorkování PointCloud. [10]

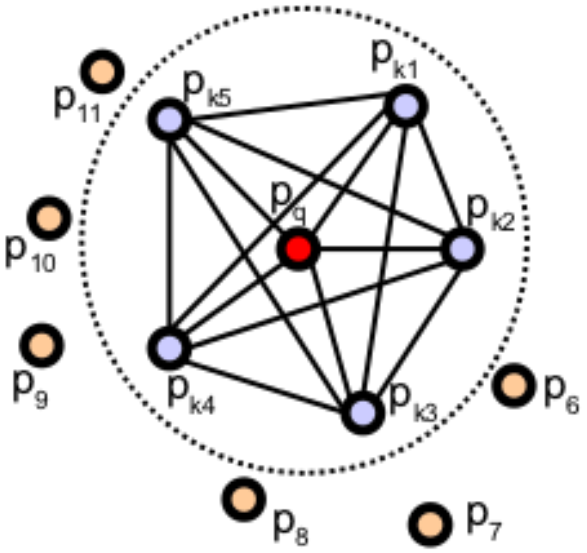
4.6.1 Deskriptory Point Feature Histograms (PFH)

Povrchy reprezentující bodové mapy vytvořené pomocí metod pro odhad normál a zakřivení povrchu jsou svou geometrií kolem určitého bodu poněkud jednoduché. Ačkoli jsou tyto metody extrémně rychlé a jejich výpočty snadné, nemohou zachytit příliš mnoho detailů, protože geometrii sousedících bodů aproximují s několika málo hodnotami. Důsledkem toho vzniká efekt, že většina scén obsahuje velké množiny bodů se stejnými nebo velmi podobnými hodnotami, čímž se snižuje jejich informativní význam. V této podkapitole je popsána metoda z rodiny 3D deskriptorů PFH. [9]

Cílem této metody je co nejpřesněji popsat geometrické vlastnosti sousedících bodů. Pomocí mnohodimenzionálního histogramu hodnot se snaží zobecnit střední zakřivení kolem bodu. Tento vysoce dimenzionální hyperprostor poskytuje deskriptorům informativní popis daných prvků a velmi dobře zvládá různé vzorkovací hustoty bodů nebo různé hladiny šumu v okolí bodu. [9]

Reprezentace deskriptorů Point Feature Histograms je založena na vzájemných vztazích dvojic bodů v sousedství a na jejich odhadnutých povrchových normálách. Všechny body jsou na sobě vzájemně závislé, protože dvojice bodů v oblasti sousedství tvoří každý bod s každým. PFH se pokouší nalézt co nejlepší variantu vzorkovaného povrchu s ohledem na všechny interakce mezi různě směrujícími odhadovanými normálami v jednotlivých bodech. Výsledný hyperprostor je tak závislý na kvalitě určených normál v každém bodě. [9]

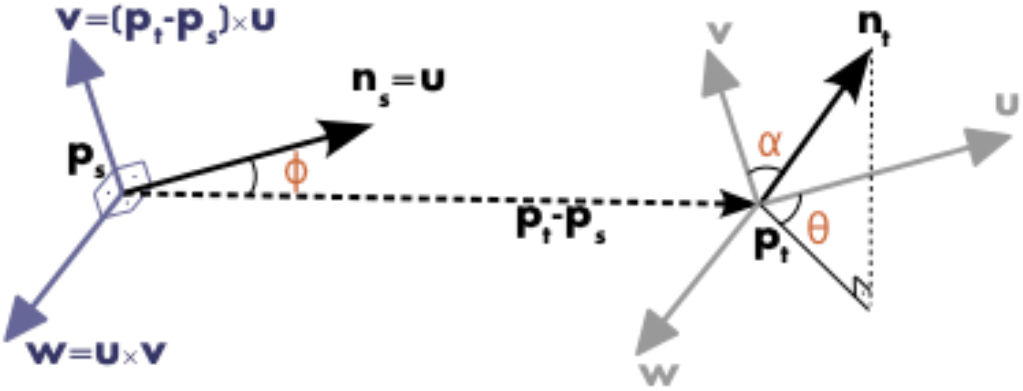
Na následujícím obrázku je znázorněn diagram ovlivněné oblasti výpočtu PFH pro počáteční bod p_q (umístěný uprostřed a označený červenou barvou). Dále kruh představující 3D sféru s poloměrem **r** a se všemi **k** sousedy (body s menší vzájemnou vzdáleností než poloměr **r**) propojenými ve společnou sít. Finální PFH deskriptor je vypočten jako histogram vztahů mezi všemi páry bodů v sousedství. Jeho výpočetní složitost tak odpovídá $O(k^2)$, kde k je počet sousedů ve výpočtu. [9]



Obr. 23: Diagram ovlivněné oblasti výpočtu PFH deskriptoru [9]

Pro výpočet relativního rozdílu mezi dvěma body p_s a p_t a jejich přiřazenými normálami $\vec{n}_s$ a $\vec{n}_t$ definujeme fixní souřadnicový systém (frame) (vektory $\vec{u}$, $\vec{v}$ a $\vec{w}$) pro jeden ze dvou bodů. Označíme-li počáteční bod p_s a cílový bod p_t, tak za p_s dosadíme bod svírající nejmenší úhel mezi spojnicí těchto dvou bodů p_s a p_t a svojí normálou $\vec{n}_s$. Souřadnicový systém v bodě p_s definujeme: [9]

$$\vec{u} = \vec{n}_s$$
$$\vec{v} = \vec{u} \times \frac{(p_t - p_s)}{\|p_t - p_s\|_2}$$
$$\vec{w} = \vec{u} \times \vec{v}$$



Obr. 24: Graficky znázorněný souřadnicový systém pro PFH úhlové deskriptory [9]

Pomocí tohoto souřadnicového systému $\vec{u}$, $\vec{v}$ a $\vec{w}$ lze rozdíl mezi dvěma normálami $\vec{n}_s$ a $\vec{n}_t$ definovat jako množinu úhlových deskriptorů následovně: [9]

$$\alpha = \vec{v} \cdot \vec{n}_t$$
$$\phi = \vec{u} \cdot \frac{(p_t - p_s)}{d}$$
$$\theta = \arctan(\vec{w} \cdot \vec{n}_t, \vec{u} \cdot \vec{n}_t)$$

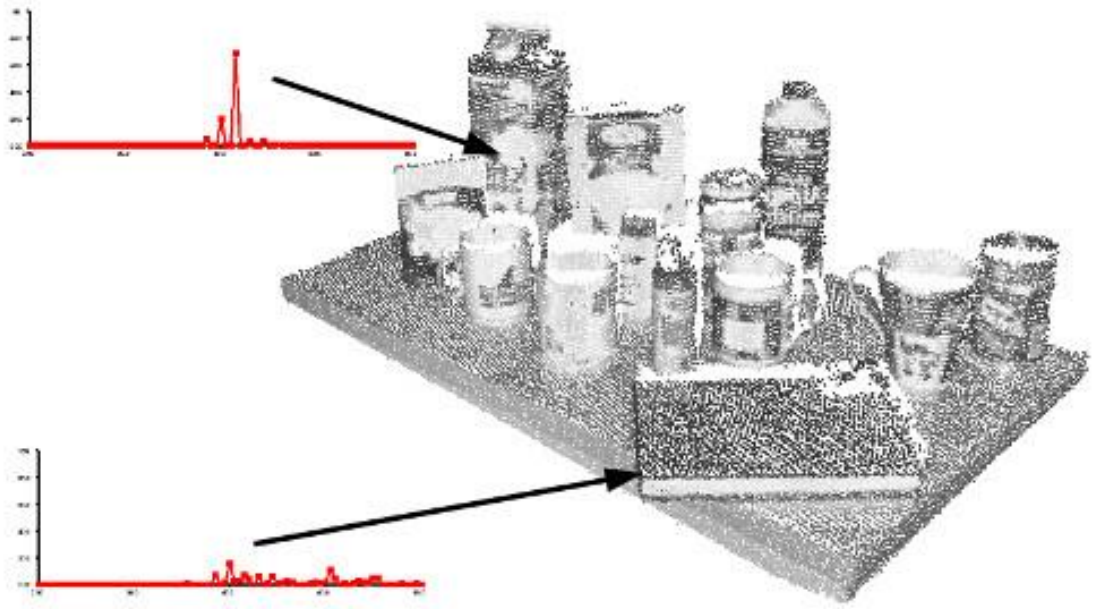
Kde d je Eukleidovská vzdálenost mezi dvěma body p_s a p_t, $d = \|p_t - p_s\|_2$. Tento čtyřkomplet $\langle \alpha, \phi, \theta, d \rangle$ je spočítán pro každý pár bodů v oblasti sousedících bodů, a tím se redukuje až 12 hodnot (souřadnice x, y, z a popis normál) pro dvojice bodů a jejich normály na 4 hodnoty. [9]

Pro vytvoření výsledné reprezentace PFH pro daný bod je množina všech odpovídajících čtveřic hodnot bodů rozdělena do histogramu. Proces pokračuje rozdělením rozsahu hodnot každého deskriptoru do b subintervalů a počítá se počet výskytu hodnot daných bodů v těchto intervalech. Protože 3 hodnoty z uvedeného čtyřkompletu jsou hodnoty úhlů mezi normálami, lze jejich hodnoty snadno normalizovat do stejného intervalu na jednotkové kružnici. Pro představu, rozdělením každého intervalu deskriptorů na stejný počet shodných částí se vytvoří histogram s b^4 sloupci

(nebo zásobníky) v souvislém prostoru. Přírůstek v určitém zásobníku histogramu odpovídá danému bodu, kterému odpovídají jeho všechny 4 hodnoty deskriptoru. [9]

Čtvrtá hodnota **d** pro datové množiny 2.5D nepředstavuje v obvyklých případech robotického vidění příliš velký význam. Děje se tak kvůli zvětšující se vzdálenosti sousedících bodů od pozorovacího bodu. Je vyzkoumáno, že vynechání **d** pro snímání míst, kde lokální hustota bodů ovlivňuje tuto hodnotu, je pro výpočet prospěšné. [9]

Na následujícím obrázku jsou uvedeny reprezentace Point Feature Histograms pro různé body v PointCloud.



Obr. 25: Reprezentace Point Feature Histograms pro různé body v PointCloud [9]

4.6.2 Deskriptory Fast Point Feature Histograms (FPFH)

Teoretická výpočetní složitost metody Point Feature Histogram pro danou PointCloud P s n body odpovídá $O(nk^2)$, kde k je počet sousedů pro každý bod p v P. Výpočet Point Feature Histograms v hustých oblastech sousedících bodů může být pro aplikace real-time nebo aplikace blížící se real-time největší překážkou. [21]

V této podkapitole se popisuje zjednodušení metody PFH nazvané Fast Point Feature Histograms, které snižuje výpočetní složitost algoritmu na $O(nk)$, přičemž si stále zachovává většinu diskriminačních schopností PFH. [21]

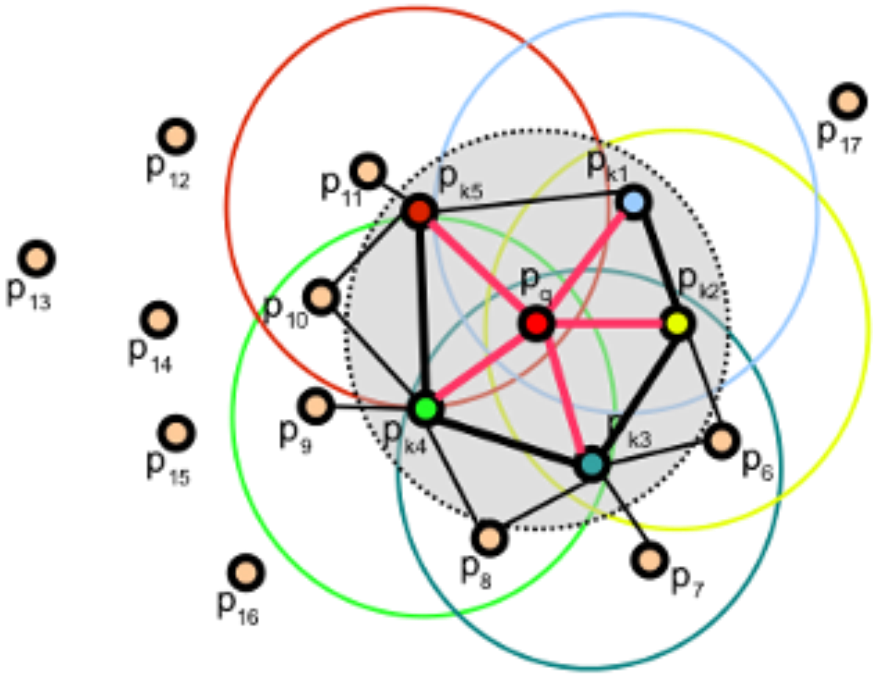
Pro zjednodušení výpočtu deskriptorů histogramu se postupuje následovně:

- V prvním kroku se pro každý dotazovaný bod p_q vypočítá sada hodnot α, ϕ, θ mezi ním a sousedícími body stejným způsobem, jak bylo popsáno v kapitole č. 4.6.1 – Deskriptory Point Feature Histograms (PFH). Dále bude tato metoda nazývána Simplified Point Feature Histogram (SPFH).
- V druhém kroku je pro každý bod znova určena oblast sousedství a sousední hodnoty SPFH jsou použity k vážení finálního histogramu daných bodů p_q, který se nazývá Fast Point Feature Histogram. [21]

Zapsáno rovnicí: [21]

$$FPFH(p_q) = SPFH(p_q) + \frac{1}{k}\sum_{i=1}^{k}\frac{1}{\omega_k}\cdot SPFH(p_k) \tag{4.3}$$

Kde váha ω_k představuje vzdálenost mezi tázaným bodem p_q a sousedním bodem p_k ve zvolené metrice, ohodnocuje tak pár (p_q, p_k). Pro správné pochopení, jak důležitý je tento váhový systém, je na následujícím obrázku znázorněn diagram vlivové oblasti pro sousedství bodů se středem p_q. [21]



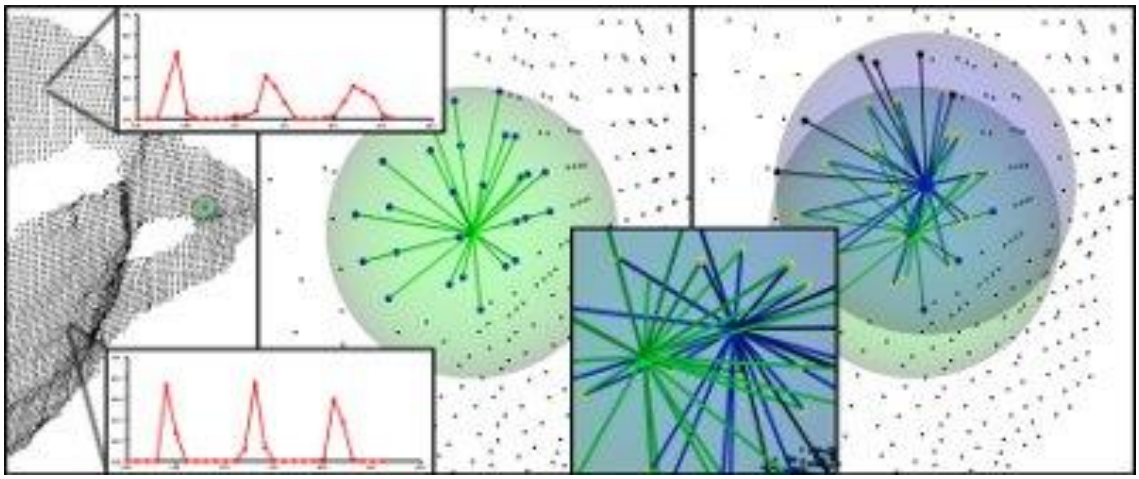
Obr. 26: Diagram vlivové oblasti pro sousedství bodů se středem p_q a jejich vztahů [21]

Algoritmus pro dotazovaný bod p_q nejprve vytvoří páry mezi daným bodem a jeho sousedy (zvýrazněné pomocí červených čar) a odhadne hodnoty SPFH. Tento proces se opakuje pro všechny body v datové množině. Následně se opět váží hodnoty SPFH pro bod p_q, avšak s využitím SPFH hodnot jeho sousedů, tímto je vytvořen Fast Point Feature Histogram pro bod p_q. Spojnice FPFH odpovídající dodatečnému váhovému schématu jsou na obrázku vyznačeny černými čarami. Tlusté čáry označují páry bodů, které byly počítány dvakrát. [21]

Zde jsou uvedeny hlavní rozdíly mezi formulací metod PFH a FPFH:

- Jak je vidět na obrázku č. 13, FPFH v oblasti sousedících bodů dotazovaného bodu p_q nepropojuje každý bod s každým. Díky tomu chybí některé hodnoty párových bodů, které by mohly přispět k zachycení přesnější geometrie povrchu kolem bodu p_q.
- PFH modeluje přesně stanovený povrch kolem dotazovaného bodu, zatímco FPFH do svého výpočtu zahrnuje i páry tvořené s body vně oblasti sousedství bodu p_q, tedy ve vzdálenosti větší než **r**, avšak maximálně do **2r**.
- Díky opakovanému přepočítávání vah hodnot diagramu FPFH kombinuje hodnoty SPFH a může tak znova získat některé hodnoty soudních párů. [21]

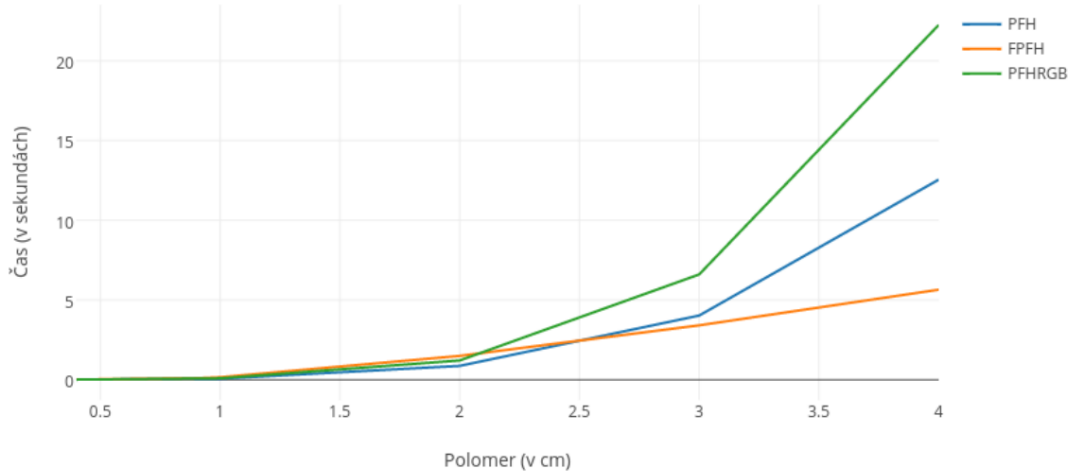
Výsledný histogram je zjednodušen separací hodnot, to znamená vytvoření **d** různých histogramů, každý pro jednu dimenzi daného deskriptoru; poté následuje spojení dohromady. [21]



Obr. 27: Znázornění separace dimenzí deskriptorů [21]

4.6.3 Srovnání metod

Na níže uvedeném grafu je zobrazeno srovnání časové náročnosti výpočtu deskriptorů několika metodami v okolí dotazovaného bodu při zvyšujícím se poloměru oblasti sousedství. Srovnávané algoritmy jsou PFH, FPFH a PFHRGB, což je rozšíření metody PFH o informaci nesoucí barvu daných bodů. Z obrázku lze vidět, že výsledný průběh funkcí algoritmů PFH a PFHRGB rostou kvadraticky, zatímco u FPFH roste lineárně. [10]



Obr. 28: Srovnání časové náročnosti metod PFH, PFHRGB a FPFH [10]

4.7 Párování deskriptorů a shlukování korespondencí

Poté, co jsou vypočítány lokální deskriptory jak na šablonách objektů, tak na cílovém PointCloud, následuje hledání možných dvojic deskriptorů s co nejvíce možnou shodou hodnot jimi popisované geometrie. Kritérium klasifikace párové korespondence je maximální korespondenční vzdálenost, která je volena uživatelem. V tomto případě se korespondenční vzdálenost mezi danými body vypočítává pomocí Eukleidovské čtvercové metriky. Na rozdíl od klasické Eukleidovské vzdálenosti není v jejím předpisu obsažena odmocnina, ale jen druhá mocnina. Počítání velmi velkého počtu odmocnin by z hlediska jejich vyšší náročnosti na výpočet mohlo zpomalovat celkový proces.

$$d(x,y)^2 = \sum_{i=1}^{n} (x_i - y_i)^2 \tag{4.4}$$

Kde d je Eukleidovská čtvercová vzdálenost, kde $x, y \in \mathbb{R}^n$. [22]

Výsledkem párování je seznam korespondencí mezi klíčovými body v PointCloud šablonách objektů a v PointCloud cílových snímcích scény. Stále však není celková shoda zaručena. Proto následuje metoda shlukování korespondencí, která má za úkol seskupit geometricky stabilní korespondence s tím, že už zavádí možnou změnu orientace

a pozice. K nalezení nejlepšího přiřazení v třídimenzionálním prostoru se pokouší transformovat dané korespondence rotací a posuvy, a proto musí uvažovat vždy shluky s minimálně třemi korespondencemi. [10]

4.8 Algoritmus Sample Consensus Initial Alignment (SAC-IA)

Algoritmus Sample Consensus Initial Alignment vychází z metody Greedy Initial Alignment, která je předchůdcem tohoto algoritmu. Tato metoda je velmi robustní, protože využívá vnitřní rotační invariantní funkce PointCloud. Výpočetní složitost je však poměrně vysoká, protože při kroku shlukování se prochází všechny možné páry korespondencí. Jak může z názvu vyplývat, jedná se o „nenasytnou“ metodu (dle slova greedy), existují totiž situace, kdy by se mohla dostat do lokálního minima. [15]

Proto byla vědci z Technické univerzity Mnichov vyvinuta metoda Sample Consensus, která se snaží zachovat stejné geometrické vztahy korespondencí, aniž by musela vyzkoušet všechny kombinace omezeného souboru korespondencí. Místo toho se odebere velké množství kandidátů na korespondenci a velmi rychle se ohodnotí dle následujícího schématu:

4) Vyber s vzorkových bodů z P a ujisti se, že jejich párové vzdálenosti jsou větší než uživatelem definovaná minimální vzdálenost d_{min}.
5) Pro každé vzorkové body najdi seznam bodů v Q, jejichž histogramy jsou podobné histogramům vzorkových bodů. Z nich jeden náhodně vyber a ten bude považován za společnou korespondenci vzorkových bodů.
6) Vypočítej rigidní transformaci definovanou vzorkovými body a jejich korespondencemi a spočítej chybovou metriku pro PointCloud, ze které se spočítá kvalita transformace. [15]

Toto schéma najde rychle dobrou transformaci tím, že sleduje velmi velké množství různých trojic korespondencí. Chybová metrika pro třetí krok je určena měřením Huberovy pokuty L_h:

$$L_h(e_i) = \begin{cases} \frac{1}{2} e_i^2 & \|e_i\| \leq t_e \\ \frac{1}{2} t_e(2\|e_i\| - t_e) & \|e_i\| > t_e \end{cases} \tag{4.5}$$

Tyto tři kroky se opakují, až se uloží transformace přinášející nejlepší chybovou metriku a provede se zarovnání. [15]

Následující tabulka ukazuje srovnání mezi metodou Greedy Initial Alignment (GIA) a Sampe Consensus Initial Alignment (SAC-IA) při odhadování nejlepší registrace na stejném datovém souboru s tím, že výsledné přiřazení je téměř identické. [15]

	GIA – 1. průběh	GIA – 2. průběh	SAC-IA
Čas běhu	> 17 min	> 43 min	34 sec
Počet uvažovaných bodů	200	250	10462
Počet možných kombinací	37186	58300	1000

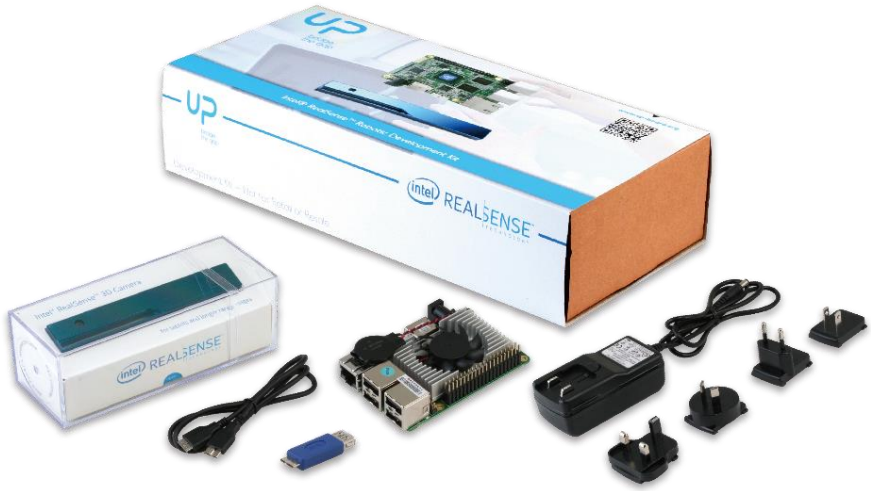
Tab. 3: Srovnání GIA a SAC-IA na shodném datovém souboru [15]

Kombinatorický charakter metody GIA ji dělá pro velké datové soubory extrémně pomalou, proto se pro toto řešení používá podvzorkovaná verze dat. Výsledným efektem je redukce FPFH deskriptorů jejich zprůměrováním, tudíž může být ztracena většina jejich detailů. Metoda SAC-IA založená na vzorkových shodách těmito nedostatky netrpí. [15]

5 KONFIGURACE

Konfigurací se rozumí, co je potřeba nachystat a nastavit před tím, než se přikročí k samotné implementaci zdrojového kódu aplikace. Neodmyslitelně k této činnosti patří proces učení, je totiž zapotřebí nakonfigurovat několik technologií, aby fungovaly společně, a proto je nutné každé z nich dostatečně porozumět.

Pro realizaci praktické části této diplomové práce je použit balíček zařízení Intel RealSense Robotic Development Kit. Ten obsahuje mikropočítač UP-board, kameru Intel RealSense R200, potřebnou kabeláž a napájecí zdroj 5V/4A. Dále se však využívá kamera Microsoft Xbox 360 Kinect kvůli vyšší přesnosti.



Obr. 29: Intel RealSense Robotic Development Kit [1]

5.1 UP-board

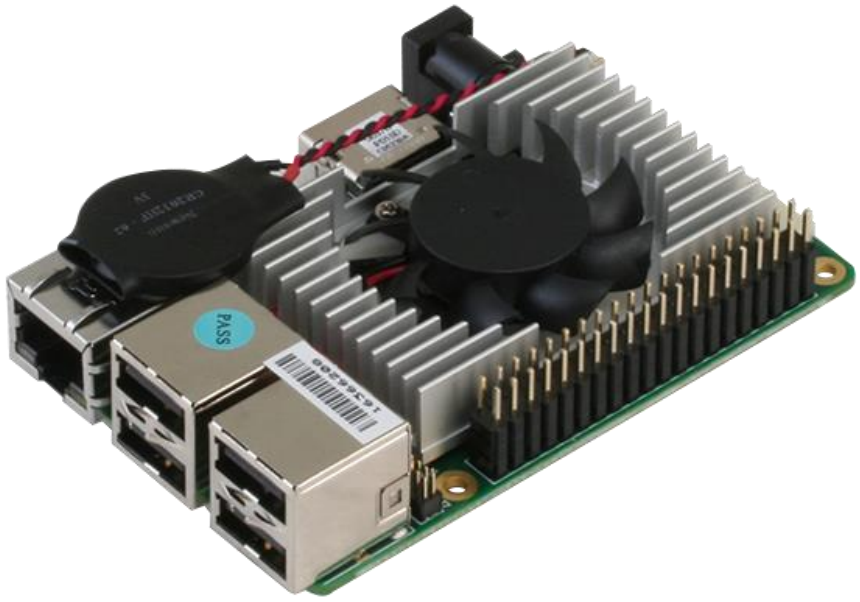
UP-board na první pohled připomíná Raspberry Pi, skládá se ale z odlišných komponentů. Mikropočítač vyvíjí firma Intel a disponuje poměrně vyšším výkonem také díky malému chladiči na přední straně. V následující tabulce jsou uvedeny některé jeho základní parametry:

Komponenta	Popis
Procesor	Intel Atom® x5-Z8350 Processor (2M Cache, 1.44 GHz až do 1.92 GHz) CPU architektura 64-bit Quad Core
Grafická karta	Intel® HD 400 Graphics
Operační paměť	4GB DDR3L-1600
Interní paměť	32GB eMMC

Tab. 4: Základní parametry související s výkonem UP-boardu [3]

Na UP-boardu je nainstalované Ubuntu 16.04 LTS Xenial Xerus desktop. Jelikož jde o open-source operační systém, je tato verze stažená z oficiálních stránek Ubuntu a nainstalována pomocí bootovacího flash disku. Dle instalačního návodu jsou také provedeny některé nezbytné úpravy. Nejprve instalace jádra linux-upboard-kernel zajišťující optimální hardwarové funkčnosti a následně balíčku linux-upboard, který také umožňuje funkce jako jsou piny GPIO a nativní formát natáčeného videa nezbytný pro kameru Intel RealSense. Tyto úpravy byly provedeny v příkazovém řádku terminálu operačního systému následovně: [1]

```
$   sudo apt update
$   sudo apt -y dist-upgrade
$   sudo add-apt-repository ppa:ubilinux/up
$   sudo apt update
$   sudo apt -y install linux-upboard
$   sudo apt -y autoremove --purge 'linux-.*generic'
$   sudo reboot
```



Obr. 30: Intel UP-board [1]

5.2 Konfigurace ROSu

Robotickému frameworku ROS se více věnuje kapitola č. 3.3. V této části jsou však popsány konfigurační úkony, které jsou potřebné pro zajištění funkčnosti tohoto projektu.

ROS je metaoperační systém, což znamená, že je privilegován na nejvyšším místě v operačním systému a díky tomu může umožňovat běh a komunikaci více procesů naráz. Tato vlastnost je velmi výhodná, chceme-li pracovat s daty z hloubkové kamery RealSense R200 nebo Microsoft Xbox 360 Kinect. Dále lze v ROSu využít mnoha

užitečných již implementovaných knihoven výrazně usnadňujících práci. Pro realizaci programu využívajícího robotické vidění je tento systém ideální.

5.2.1 Instalace ROSu

Je zde nainstalován ROS distribuce Kinetic Kame, který má prodlouženou podporu do roku 2021 a je kompatibilní s verzí Ubuntu 16.04 LTS Xenial Xerus. Jelikož pro tuto práci byla na začátku výzkumu využita kamera Intel RealSense R200, je zde nainstalována potřebná knihovna s ovladači pro toto zařízení.

Od doby vydání linux-upboard-kernel byly přidány různé další úpravy, například možnost zapojení více kamer RealSense naráz. Proto před instalací ROS s podporou RealSense musí mít systém přístup ke zdrojovým balíčkům jádra. [1]

```
$   wget -q -O - https://bit.ly/en_krnl_src | sudo /bin/bash
```

Přidání ROS repozitářů do Ubuntu package listu.

```
$   sudo add-apt-repository http://packages.ros.org/ros/ubuntu
$   sudo apt-key adv --keyserver hkp://ha.pool.sks-keyservers.net \
     --recv-key 0xB01FA116
$   sudo apt update
```

Instalace balíčků ROS.

```
$   sudo apt -y install ros-kinetic-desktop-full python-rosinstall
     ros-kinetic-realsense-camera
$   sudo rosdep init
$   rosdep update
$   echo "source /opt/ros/kinetic/setup.bash" >> ~/.bashrc
$   source ~/.bashrc
```

Po restartování systému by měla být instalace kompletní. [1]

5.2.2 Catkin

Při programování v ROSu lze využít několik IDE rozhraní jako např. Qt Creator, KDevelop nebo Eclipse především pro jejich snadnější kompilaci zdrojových kódů, protože ruční komplice uzlů není příliš jednoduchý a přehledný proces. Nicméně k samotné organizaci a kompilaci uzlů se nejčastěji používá Catkin v kombinaci s CMake, na kterém jsou založeny některé tyto IDE. [23]

V této práci byl využit právě čistě Catkin bez jakéhokoliv IDE. Bylo tedy nutné řešit linkování knihoven a další náležitosti při kompilaci ručně. Instalace Catkinu je provedena následovně:

```
$   sudo apt-get install ros-kinetic-catkin
```

Pro organizaci balíčku a jejich kompilaci je zapotřebí nejprve vytvořit catkin workspace. Inicializace Catkin workspace: [24]

```
$   mkdir -p ~/catkin_ws/src
$   cd ~/catkin_ws/
$   catkin_make
$   source devel/setup.bash
```

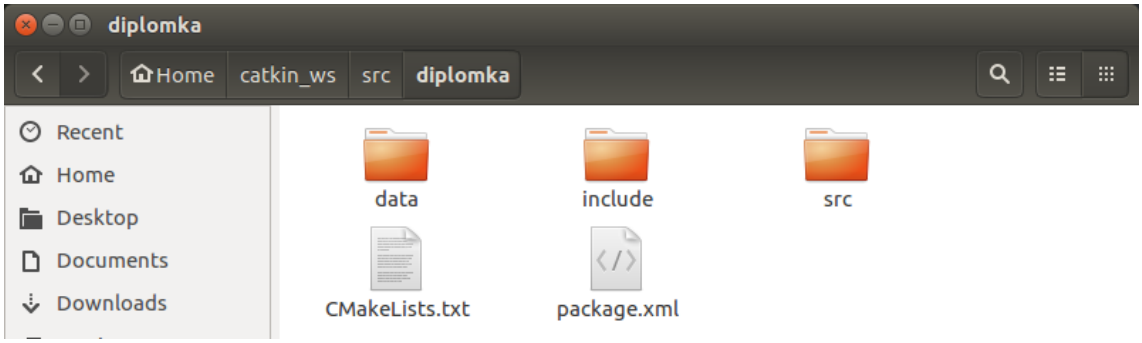
Při vytváření balíčku je ideální znát a rovnou zahrnout všechny závislosti (*dependencies*) na adresářích obsahujících hlavičkové soubory (*header files*), protože se při vytváření balíčku taky zároveň vytvoří soubor `package.xml` obsahující seznam závislostí. Nicméně při implementaci této aplikace byly některé závislosti přidány přesto ručně. Dále se pak automaticky vytvoří textový soubor `CMakeLists.txt`. Soubor CMakeLists funguje jako vstup kompilačního systému CMake, který využívá Catkin. CMakeLists popisuje, jak daný kód sestavit a kde ho nainstalovat. [25]

```
cmake_minimum_required(VERSION 2.8.3)
project(diplomka)
## Compile as C++11, supported in ROS Kinetic and newer
# add_compile_options(-std=c++11)
## Find catkin macros and libraries
find_package(catkin REQUIRED COMPONENTS
  pcl_ros
  roscpp
  rospy
  sensor_msgs
  std_msgs
)

find_package(Boost REQUIRED COMPONENTS system)
find_package(PCL REQUIRED)

include_directories(${PCL_INCLUDE_DIRS}) #include directories for PCL
link_directories(${PCL_LIBRARY_DIRS})

catkin_package(
  INCLUDE_DIRS include
  LIBRARIES data
  CATKIN_DEPENDS pcl_ros roscpp rospy sensor_msgs std_msgs
  DEPENDS system_lib
)

include_directories(include ${catkin_INCLUDE_DIRS})
add_definitions(${PCL_DEFINITIONS})

add_executable (dipl src/template_alignment.cpp) #creating .exe for node
target_link_libraries (dipl ${PCL_LIBRARIES})
target_link_libraries(dipl ${catkin_LIBRARIES})
add_executable (convertor src/convertor.cpp) #creating .exe for node
target_link_libraries (convertor ${PCL_LIBRARIES})
target_link_libraries(convertor ${catkin_LIBRARIES})
```

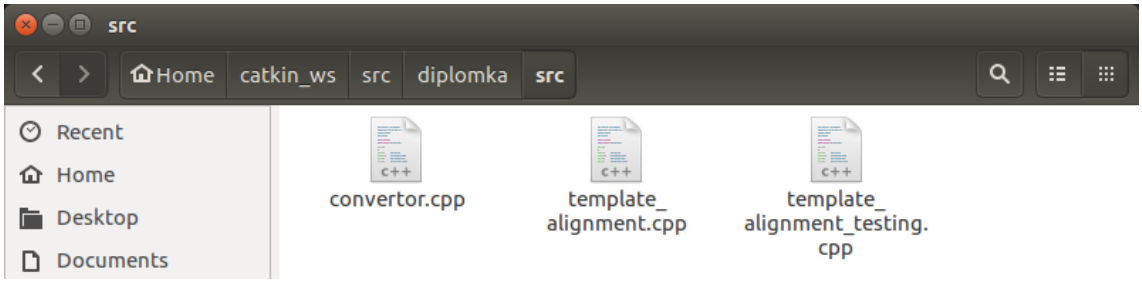
Příkaz pro vytvoření balíčku:

```
$ catkin_create_pkg diplomka pcl_ros roscpp rospy sensor_msgs std_msgs
```

Všechny zdrojové kódy v této práci představují uzly a jsou umístěny v balíčku s názvem *diplomka* ve složce *src*.



Obr. 31: Ukázka souborů obsažených v balíčku *diplomka*



Obr. 32: Ukázka souborů obsažených ve složce *src*

5.3 Příprava kamery

Při instalaci ROSu se zároveň instaluje balíček ros-kinetic-realsense-camera, který obsahuje mnoho souborů pro práci s kamerou realsense R200 včetně launchfiles, jako například `r200_nodelet_default.launch`, potřebných pro spuštění kamery.

Pro práci s kamerou Microsoft Xbox 360 Kinect je instalována knihovna libfreenect: [12]

```
$   sudo apt-get install freenect
```

A pro spuštění kamery lze využít příkaz:

```
$   roslaunch freenect_launch freenect.launch
```

5.4 Instalace PCL

V této práci pro implementaci programu Template Alignment bylo dle návodu PCL využito téměř všech subknihoven. Učinilo se tak rozhodnutí stáhnout a zkompilovat celou PCL knihovnu.

Nejprve je nutné stáhnout příslušnou verzi knihovny například z webu Github. V tomto projektu byla použita verze PCL 1.9.0. Dále se pak dekompresuje tar-bzip archiv a pomocí kompilátoru CMake se provede samotná kompilace: [13]

```
$   tar xvfj pcl-pcl-1.7.2.tar.gz
$   cd pcl-pcl-1.7.2 && mkdir build && cd build
$   cmake -DCMAKE_BUILD_TYPE=Release ..
$   make -j2
$   sudo make -j2 install
```

S kompilací knihovny na mikropočítači UP-board byly nejprve značné potíže kvůli nedostatečné operační paměti, která je v tomto případě 4GB, ale problém byl brzo vyřešen dočasným rozšířením záložního oddílu virtuální paměti někdy označovaného jako „swap“ na 10GB.

Také byly nainstalovány knihovny Boost, Eigen, FLANN, VTK.

6 IMPLEMENTACE

Jeden z prvních nápadů, jak lokalizovat objekt v prostoru pomocí kamery s hloubkovým vjemem, bylo využít segmentační postupy knihovny PCL a následnou rekonstrukci 3D modelu tělesa do formátu .STL. Jeho identifikaci, zjištění pozice a orientace by provedl vhodný algoritmus. Bylo učiněno několik pokusů implementace tohoto přístupu např. soubor *greedy_projection*, ale nakonec se od této metody upustilo.

Po neúspěchu s algoritmem pro vytvoření .STL formátu z PointCloud se začal zkoumat nový přístup řešení problému. Myšlenkou bylo zkusit zpracovávat data už na úrovni PointCloud, vyhnout se tak výpočetně náročné rekonstrukci 3D modelu tělesa a pracovat tak pouze s částmi povrchů objektů. Hlavním důvodem k tomuto úsudku bylo, že kamera snímající daný prostor s umístěnými tělesy je statická, tudíž data použitelná z hloubkové kamery budou vždy obsahovat pouze viditelnou část objektů z jednoho úhlu. Bylo by tedy velmi obtížné rekonstruovat celý 3D model snímaného tělesa.

V dokumentaci PCL knihovny, která je základním stavebním prvkem tohoto projektu, je k dispozici mnoho implementovaných funkcí a algoritmů pracujících s PointCloud, a proto jsou v této práci hojně využívány. Pro tyto účely byla využita aplikace Template Alignment, volně přeloženo: *přiřazení šablon*. Šablonou se myslí množina bodů PointCloud, která reprezentuje hledaný objekt, zpravidla představuje jen část jeho povrchu.

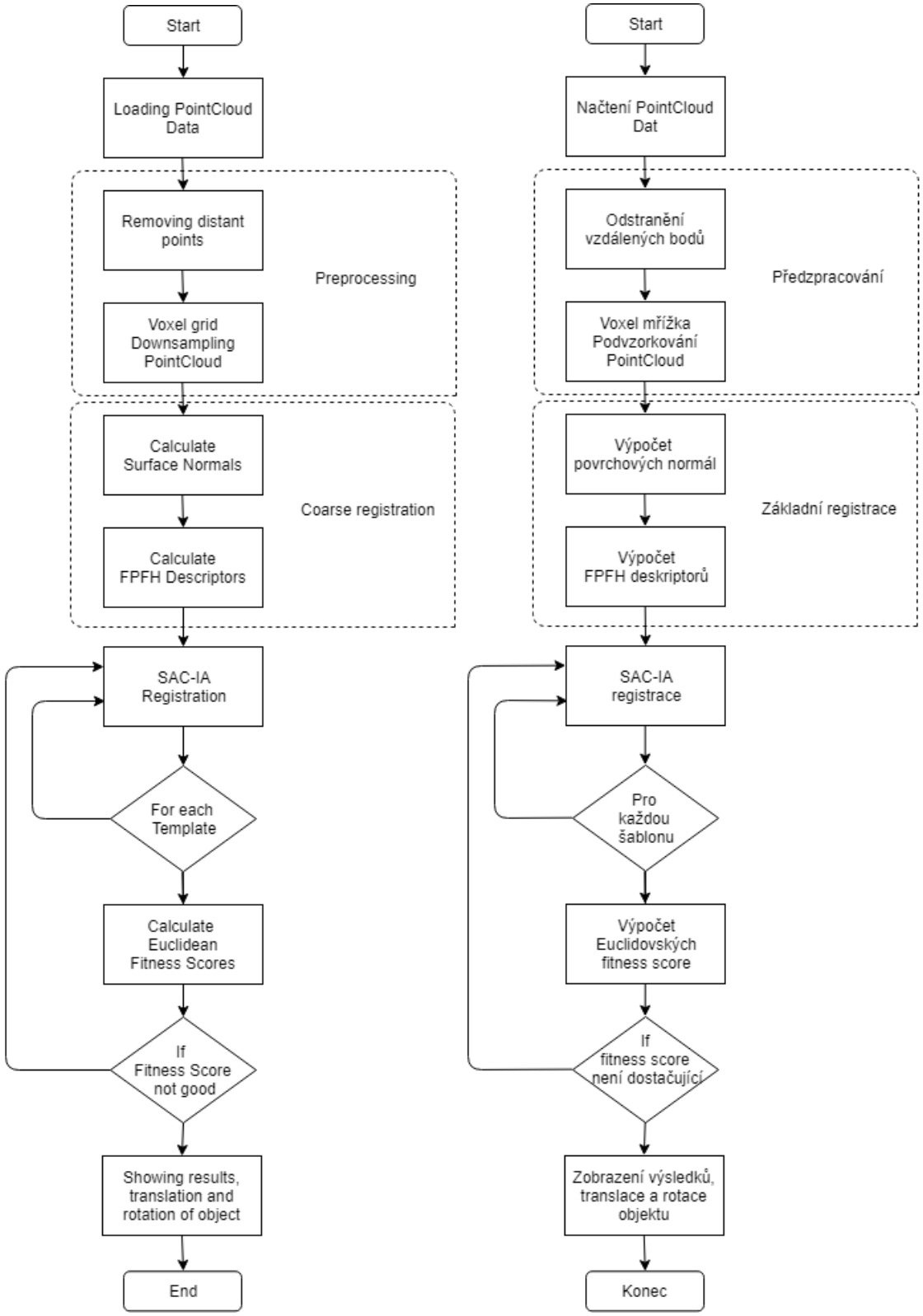
6.1 Aplikace Template Alignment

Co tato aplikace obsahuje za metody, jaký proces vykonává a jaký má účel?

Nejprve vysvětlení účelu aplikace: získá se snímek z kamery s hloubkovým vjemem a do něj se algoritmus snaží zasadit předem známou šablonu povrchu tělesa a dopočítat tak správnou polohu a orientaci hledaného objektu. Díky získaným souřadnicím lze navigovat robotický manipulátor na konkrétní místo, kde se těleso nachází, aby ho mohl uchopit a přemístit. Tím by se realizovala úloha Bin Picking.

Proces, který musí výpočetní program provést, je velmi komplikovaný a jeho výsledky jsou silně závislé na přesnosti kamery, která dodává vstup do tohoto procesu. Snímky z takové kamery jsou technicky jen velké množiny bodů umístěných v trojrozměrném prostoru, takový formát se nazývá PointCloud. Reprezentují tak čistě jen pohledovou část obrazu, kterou vidí kamera, a vytváří tak něco podobného povrchu těles.

Metody využité v celé aplikaci jsou podrobně popsány v kapitole č. 4 a je na ně patřičně odkazováno. Většinou se jedná o výpočetní algoritmy, které pracují s body v PointCloud. Jelikož by v reálné aplikaci hrál velkou roli i čas celého procesu, je mnohdy uplatněn kompromis mezi dokonalými výsledky a nejkratším možným výpočetním časem. Na následujícím obrázku lze vizualizovat schematický průběh celého procesu:



Obr. 33: Schematický diagram aplikace Template Alignment

6.2 Vytvoření šablon (object_template)

Aby aplikace fungovala správně, musí být vhodně vytvořeny šablony objektů určených k lokalizaci, a to ve stejném formátu, jako je snímek kamery, tedy formát PointCloud.

Šablona známého tělesa by měla představovat reprezentativní část hledaného objektu a měla by splňovat tyto předpoklady:

- Unikátnost, aby nedocházelo k více možnostem přichycení.
- Dobrá pozorovatelnost pro snímání z mnoha úhlů natočení.
- Vhodný tvar – tak, aby na základě tvaru šel identifikovat daný posuv a rotace.
- Věrohodnost, měla by přesně rozměrově odpovídat skutečnému tělesu.

Jelikož jsou hledaná tělesa známa, lze šablony pro tuto aplikaci předem nachystat. Vychází se z toho, že by pro aplikaci Bin piking byly dodány 3D modely objektů ve formátu .STL a ty se následně upravily a převedly na formát PointCloud, který se ukládá v souboru typu .PCD. Tyto soubory by si program Template alignment nahrál vždy na začátku procesu.

Postup v této práci je následovný: Nejprve se .STL model nahraje do programu MeshLab, kde se upraví a pokryje vrstvou bodů; následně se uloží jako formát .OBJ a skrze několik funkcí knihovny PCL se převede na PointCloud, podvzorkuje se a uloží do souboru .PCD. V této podkapitole jsou popsány jednotlivé kroky této konverze.

6.2.1 Format PCD

Formát .PCD (Point Cloud Data) pro ukládání PointCloud do souboru byl vyvinut knihovnou PCL. Dokáže totiž podporovat různá rozšíření pro PointCloud zpracovávaný v n-dimenzionálním prostoru, které jiné formáty nepodporují. [14]

PCD není prvním typem souboru, který dokáže ukládat data 3D PointCloud. Komunity počítačové grafiky a výpočtu geometrií časem nezávisle vytvořily mnoho formátů, kterými lze popsat libovolné polygony a PointCloud získané pomocí laserových skenerů. Některé z těchto formátů:

- PLY – polygonový formát vyvinutý na Stanfordské univerzitě Gregem Turkem a spol. [16]
- STL – formát souboru, který je nativní pro stereolitografii CAD vytvářený 3D systémy. Popisuje nestrukturovaný triangulovaný povrch. [17]
- OBJ – formát pro definici geometrií, který vyvinul Wavefront Technologies.
- X3D – formát ISO založený na standardu XML pro reprezentaci 3D dat počítačové grafiky
- A mnoho dalších. [14]

Všechny výše uvedené formáty souborů trpí několika nedostatky, což je naprosto přirozené, byly totiž vytvořeny pro jiné účely a využití, a to v různých časech – dříve, než byly vynalezeny dnešní technologické postupy a algoritmy počítačového vidění. [14]

Formát .PCD může být uložen buď jako *binary* (kompletní datová kopie paměti), nebo jako *ASCII*, což ukládá body přehledně každý zapsaný (souřadnice) na nový řádek souboru. Dále verze *ASCII* obsahuje hlavičku obsahující vstupní hodnoty, které musí být seřazeny vždy v daném pořadí:

- VERSION – udává verzi .PCD souboru. `VERSION .7`
- FIELDS – specifikuje názvy dimenzí a pole vyhrazené pro definici bodu.

```
FIELDS x y z                                # XYZ data
FIELDS x y z rgb                            # XYZ + colors
FIELDS x y z normal_x normal_y normal_z     # XYZ + surface normals
FIELDS j1 j2 j3                             # moment invariants
```

- SIZE – udává velikost každé souřadnice v bytech. `SIZE 4 4 4 4`
- TYPE – udává typování každé souřadnice. **I** – int8 (char), int16 (short) a int32 (int), **U** – unit8 (unsigned char), unit16 (unsigned short), unit32 (unsigned int), **F** – float. `TYPE F F F F`
- COUNT – určuje, kolik prvků má každá souřadnice. `COUNT 1 1 1 1`
- WIDTH – počet bodů v PointCloud na šířku. `WIDTH 213`
- HEIGHT – počet bodů v PointCloud na výšku. Je-li *1* jedná se o neorganizovanou PointCloud. `HEIGHT 1`
- VIEWPOINT – specifikuje polohu a orientaci hlediska vůči bodům v PointCloud. (translaci tx ty tz + quaternion qw qx qy qz) `VIEWPOINT 0 0 0 1 0 0 0`
- POINTS – celkový počet bodů. `POINTS 213`
- DATA – typ formátu. `DATA ascii` [14]

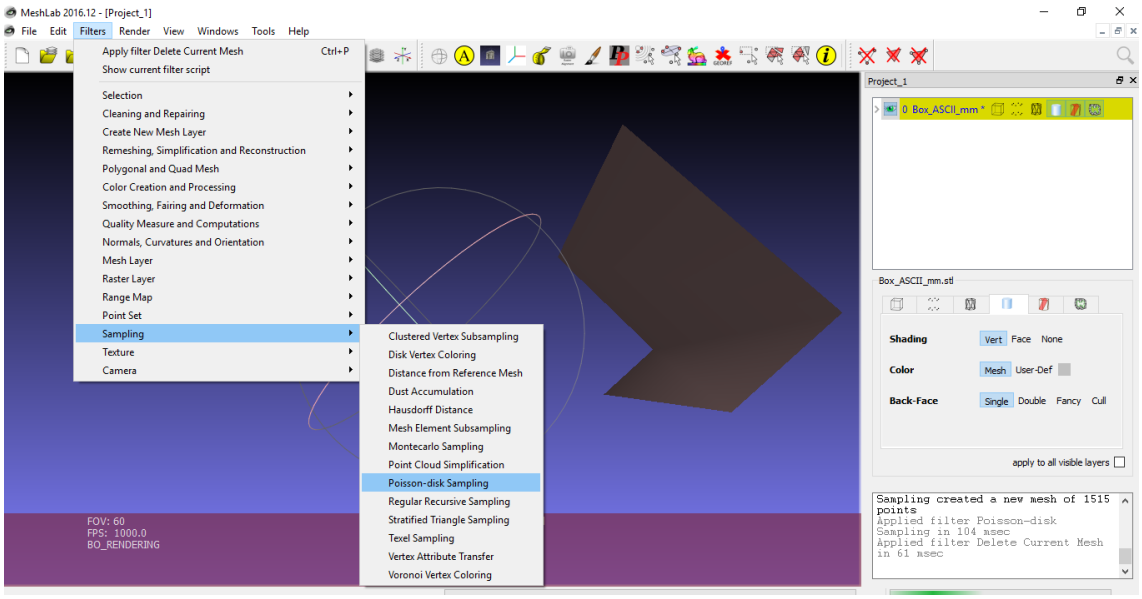
Největší výhodou tohoto formátu je schopnost ukládat a pracovat s organizovanými daty PointCloud. Tato přednost se stává extrémně významnou pro real-time aplikace a výzkumné oblasti, jako jsou virtuální realita, robotika aj. [14]

6.2.2 MechLab

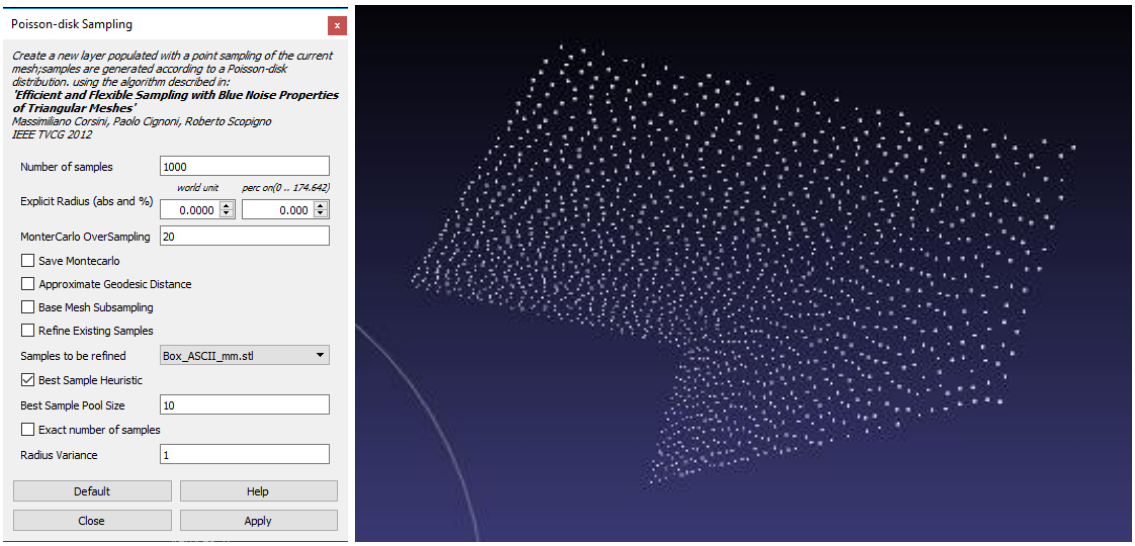
MechLab je open-source systém pro zpracovávání a editaci 3D trojúhelníkových sítí. Poskytuje širokou řadu nástrojů pro editaci, čištění, rekonstrukci, kontrolu, renderování, texturování a konverzi sítí. Nabízí funkce pro zpracování hrubých dat vytvořených pomocí 3D digitalizačních nástrojů a zařízení a pro přípravu modelů k 3D tisku. [18]

6.2.3 Příprava šablony v MechLabu

Po nahrání .STL souboru se model nejprve upraví tak, aby představoval jen reprezentativní část povrchu daného objektu a s využitím funkce filtru v MechLabu: *Poisson-disk samples points* se pokryje vrstvou bodů. Tato síť bodů se exportuje do souboru ve formátu .OBJ.



Obr. 34: Ukázka použití filtru v programu MechLab



Obr. 35: Vytvoření sítě bodů na povrchu 3D modelu .STL

Při zpracovávání 3D modelu objektu je ideální, aby byl již při tvorbě modelu .STL střed souřadného systému umístěn ve středu tělesa a tuto polohu si zachoval. Tímto se docílí výsledného souboru .PCD, který sice může obsahovat třeba jen zlomek povrchu daného objektu, ale pozice středu skutečného tělesa se díky konstrukci formátu .PCD zachovává. V konečných transformacích souřadnic při aplikaci Bin Picking hraje tato informace důležitou roli.

6.2.4 Konvertace z .obj na .pcd

Pro poslední krok vytvoření souboru .PCD byla implementována krátká aplikace sloužící ke konverzi šablony z formátu .OBJ na .PCD. Při tomto procesu se provede podvzorkování PointCloud šablony se stejným vzorkováním 5 mm jako u snímaného PointCloud.

```
// Declaration of PointCloud
pcl::PointCloud<pcl::PointXYZ>::Ptr cloud (new pcl::PointCloud<pcl::PointXYZ>);

int main(int argc, char **argv)
{
  // Loading .OBJ file
  pcl::io::loadOBJFile (argv[1], *cloud);
  // Downsampling the point cloud
  const float voxel_grid_size = 0.005f;
  pcl::VoxelGrid<pcl::PointXYZ> vox_grid;
  vox_grid.setInputCloud (cloud);
  vox_grid.setLeafSize (voxel_grid_size, voxel_grid_size, voxel_grid_size);
  pcl::PointCloud<pcl::PointXYZ>::Ptr tempCloud (new pcl::PointCloud<pcl::PointXYZ>);
  vox_grid.filter (*tempCloud);
  cloud = tempCloud;
  // Saving needed .PCD file
  pcl::io::savePCDFile ("object_template.pcd", *cloud);
  return (0);
}
```

Ke spuštění tohoto uzlu je samozřejmě nutné mít aktivní ROScore. Do terminálu se zadá příkaz obsahující název souboru `example.obj`, který se musí nacházet v aktuální složce, ve které byl příkaz proveden. Do této složky se uloží výsledný zpracovaný soubor `object_template.pcd`. Příkaz:

```
$   rosrun diplomka convertor example.obj
```

6.3 Implementace aplikace Template Alignment

V této kapitole je popsaný zdrojový kód uzlu `template_alignment.cpp`, který vykonává veškeré výpočetní úkony pro přiřazení šablony k snímané scéně. Je vytvořen na základě dokumentací knihovny PCL a spolu s potřebnými konfiguračními úpravami tvoří aplikaci robotického vidění pro snímání objektů a jejich lokalizaci. Výstupem z tohoto programu jsou vektor translace a rotační matice známého středu hledaného objektu. Na základě těchto informací lze souřadnice vhodně transformovat pro navigování manipulátoru, který by objekt přemístil a realizovat tak příklad Bin Picking.

6.3.1 ROS subscriber

Nejprve je nutné získat vstupní data z kamery s hloubkovým vjemem ve správném formátu. K tomuto účelu je využitý standardní ROS subscriber, který umožní odebírat data v PCL formátu PointCloud.

V následujícím kódu je definována proměnná *new_cloud_available_flag*, která informuje, zda je snímek kamery odebrán. Dále pak funkce *callback*, která je volána ROS subscriberem. Tato funkce uloží přijatou zprávu z konkrétního ROS topicu a pomocí funkce *fromROSMsg* ji převede do PCL formátu PointCloud. K informování uživatele

o průběhu výpočetního programu jsou v reálném čase vypisovány zprávy do konzole. [26]

```
bool new_cloud_available_flag; // Flag for availability of camera frame
// Create some point cloud object by PCL templates to hold data
pcl::PointCloud<pcl::PointXYZ>::Ptr cloud (new pcl::PointCloud<pcl::PointXYZ>);
    // ROS subscriber callback function
    void
    callback (const sensor_msgs::PointCloud2ConstPtr& pCloud)
    {
      // new cloud formation
      printf("Received a cloud message...\n");
      pcl::fromROSMsg (*pCloud, *cloud); // Conversion to PointCloud format
      new_cloud_available_flag = true;
      printf ("converted\n");
    }
```

Tato část kódu je umístěna přímo v hlavní funkci *main*. Jedná se o standardní využití ROS subscriberu. Nejprve se provede jeho inicializace a pak se volá daný topic, na kterém proudí stream zpráv (messages) neboli dat z kamery.

```
    // Load the target cloud PCD file
    ros::init(argc, argv, "subscriber_node");
    ros::NodeHandle nh;
    ros::Rate rate(30); // frequency of calling operation
    new_cloud_available_flag = false;
    // ROS subscriber calling callback function to read msg from this topic
    ros::Subscriber sub = nh.subscribe<sensor_msgs::PointCloud2>
("/camera/depth/points", 1, callback);
```

Těsně před voláním subscriberu se ještě nastaví proměnná *new_cloud_available_flag* na výchozí hodnotu *false*. Ta se změní na *true* ve funkci *callback* v případě, že se tato funkce úspěšně zavolá. To se stane, pokud topic `/camera/depth/points` disponuje daty z kamery. V případě, že tato proměnná nabývá hodnoty false, dostane se program do cyklu, kde čeká na první příchozí data daného topicu. Funkce `ros::spinOnce()` znova volá funkci *callback* a pokud data stále nejsou k dispozici program zůstává v tomto cyklu. [4]

```
    // Wait for the first frame:
    while(!new_cloud_available_flag)
    {
      printf ("waiting for first frame...\n");
      ros::Duration( 1 ).sleep(); // pause
      ros::spinOnce(); // function to call callback function again
    }
    printf ("processing...\n");
```

6.3.2 Třída FeatureCloud

Tato i následující kapitola popisuje implementaci tříd a jejich metod převzatých z open-source knihovny PCL. Třída *FeatureCloud* je definována tak, aby poskytovala pohodlné metody pro výpočty a ukládání PointCloud s lokálními deskriptory jednotlivých bodů.

Konstruktor vytvoří nový objekt: `pcl:`KdTreeFLANN<pcl::KdTreeFLANN>`` a inicializuje parametry poloměru, které budou použity při výpočtu povrchových normál a lokálních deskriptorů (podrobněji v kapitolách č. 4.4, 4.6). [5]

```
FeatureCloud () :
      search_method_xyz_ (new SearchMethod),
      normal_radius_ (0.0205f),
      feature_radius_ (0.0205f)
    {}
```

Dále jsou definovány metody pro nastavení vstupního PointCloud, a to buď pomocí předání sdíleného ukazatele na PointCloud, nebo poskytnutím názvu souboru PCD, který se má následně nahrát. V obou případech se tak stane po nastavení vstupu pomocí funkce *processInput*. Tato funkce vypočítává lokální deskriptory, jak je popsáno dále. [5]

```
    // Process the given cloud
    void
    setInputCloud (PointCloud::Ptr xyz)
    {
      xyz_ = xyz;
      processInput ();
    }

    // Load and process the cloud in the given PCD file
    void
    loadInputCloud (const std::string &pcd_file)
    {
      xyz_ = PointCloud::Ptr (new PointCloud);
      pcl::io::loadPCDFile (pcd_file, *xyz_);
      processInput ();
    }
```

Také jsou zde definovány některé metody pro přístup *public*, které lze použít k získání sdílených ukazatelů k bodům, povrchovým normálám a k lokálním deskriptorům. [5]

```
  // Get a pointer to the cloud 3D points
    PointCloud::Ptr
    getPointCloud () const
    {
      return (xyz_);
    }
```

```
    // Get a pointer to the cloud of 3D surface normals
    SurfaceNormals::Ptr
    getSurfaceNormals () const
    {
      return (normals_);
    }

    // Get a pointer to the cloud of feature descriptors
    LocalFeatures::Ptr
    getLocalFeatures () const
    {
      return (features_);
    }
```

Dále je definována metoda ke zpracování vstupní PointCloud, která nejprve vypočítá povrchové normály PointCloud a pak vypočítá jeho lokální deskriptory. [5]

```
// Compute the surface normals and local features
    void
    processInput ()
    {
      computeSurfaceNormals ();
      computeLocalFeatures ();
    }
```

Pro výpočet povrchových normál je využita třída z knihovny PCL: `pcl:`NormalEstimation <pcl::NormalEstimation>``. Aby to bylo možné učinit, je nutné specifikovat vstupní PointCloud. Nejdříve pomocí KdTree (podrobněji v kapitole č. 4.3) vyhledat sousedící body a dále určit poloměr, který definuje, co se počítá jako sousedství, a to u každého bodu. Pak se vypočítají povrchové normály a uloží se do členské proměnné pro pozdější použití. [5]

```
// Compute the surface normals
    void
    computeSurfaceNormals ()
    {
      normals_ = SurfaceNormals::Ptr (new SurfaceNormals);
      pcl::NormalEstimation<pcl::PointXYZ, pcl::Normal> norm_est;
      norm_est.setInputCloud (xyz_);
      norm_est.setSearchMethod (search_method_xyz_);
      norm_est.setRadiusSearch (normal_radius_);
      norm_est.compute (*normals_);
    }
```

Podobně z PCL používáme třídu: `pcl:`FPFHEstimation<pcl::FPFHEstimation>`` k výpočtu deskriptorů metodou Fast Point Feature Histograms. Výpočet se provádí z povrchových normál vstupního PointCloud. (Podrobněji v kapitole č. 4.6) [5]

```
// Compute the local feature descriptors
    void
    computeLocalFeatures ()
    {
      features_ = LocalFeatures::Ptr (new LocalFeatures);

      pcl::FPFHEstimation<pcl::PointXYZ, pcl::Normal, pcl::FPFHSignature33>
fpfh_est;
      fpfh_est.setInputCloud (xyz_);
      fpfh_est.setInputNormals (normals_);
      fpfh_est.setSearchMethod (search_method_xyz_);
      fpfh_est.setRadiusSearch (feature_radius_);
      fpfh_est.compute (*features_);
    }
```

6.3.3 Třída TemplateAlignment

Metody popsané výše slouží k sestavení úkonů pro výpočet deskriptorů a pro jejich uložení do paměti s odpovídajícím PointCloud. [5]

Následující část zkoumá třídu *TemplateAlignment*, která, jak vyplývá z názvu, slouží k realizaci přiřazení šablony (používají se také výrazy jako zarovnání, přizpůsobení, hledání shody, registrace). Šablona je tvořená typicky menší skupinou bodů ve formátu PointCloud, která má za úkol reprezentovat známou vystihující část většího objektu. Přiřazením šablony do snímaného obrazu ve formátu PointCloud můžeme určit polohu a orientaci tělesa, které šablona reprezentuje. [5]

Nejprve se definuje struktura pro ukládání výsledků přiřazení. Ta obsahuje hodnotu *fintness_score* s plovoucí desetinou čárkou, která představuje výsledek hodnotící funkce, tedy „vhodnost“ hledané shody. Cílem je tuto hodnotu minimalizovat, čím menší bude, tím lepší je výsledné přiřazení šablony. Tato struktura dále obsahuje transformační matici, která nese informaci o odhadované translaci a rotaci, jak by měl být PointCloud šablony zasazen v prostoru, aby co nejlépe odpovídal bodům ve snímaném PointCloud. [5]

```
// A struct for storing alignment results
    struct Result
    {
      float fitness_score;
      Eigen::Matrix4f final_transformation;
      EIGEN_MAKE_ALIGNED_OPERATOR_NEW
    };
```

V konstruktoru se inicializuje objekt z PCL: `pcl::`SampleConsensusInitialAlignment <pcl::SampleConsensusInitialAlignment>`` (SAC-IA), tedy algoritmus, který provádí samotné přiřazování (podrobněji v kapitole č. 4.8). Následně mu jsou poskytnuty hodnoty odpovídajících vstupních parametrů. [5]

Parametr *max_correspondence_distance* (maximální korespondenční vzdálenost) je specifikován jako Eukleidovská čtvercová vzdálenost. (Podrobněji v kapitole č. 4.7) Pro příklad, chceme-li ohraničit chybu horní hranicí 1 cm, dosadíme 0,01 čtverečního. [5]

```
    TemplateAlignment () :
      min_sample_distance_ (0.052f),
      max_correspondence_distance_ (0.015f*0.015f),
      nr_iterations_ (1250)
    {

      // Initialize the parameters in the Sample Consensus Initial Alignment
(SAC-IA) algorithm
      sac_ia_.setMinSampleDistance (min_sample_distance_);
      sac_ia_.setMaxCorrespondenceDistance (max_correspondence_distance_);
      sac_ia_.setMaximumIterations (nr_iterations_);
    }
```

Dále je definována metoda pro nastavení vstupního cílového PointCloud pro algoritmus SAC-IA, tj. snímaný PointCloud, ke kterému budou šablony přiřazeny. [5]

```
// Set the given cloud as the target to which the templates will be aligned
    void
    setTargetCloud (FeatureCloud &target_cloud)
    {
      target_ = target_cloud;
      sac_ia_.setInputTarget (target_cloud.getPointCloud ());
      sac_ia_.setTargetFeatures (target_cloud.getLocalFeatures ());
    }
```

Další definovaná metoda slouží k určení šablony nebo šablon, které se algoritmus bude pokoušet přiřadit k cílové PointCloud. Každé zavolání této metody přidá danou šablonu do interního vektoru *FeatureCloud* a připraví ji tak k budoucímu použití. [5]

```
 // Add the given cloud to the list of template clouds
    void
    addTemplateCloud (FeatureCloud &template_cloud)
    {
      templates_.push_back (template_cloud);
    }
```

Následuje definice metody přiřazovacího algoritmu. Tato metoda vezme jako vstup šablonu tělesa a přiřadí ji v odhadované pozici k cílovému PointCloud specifikovanému voláním metody: `pcl:`setInputTarget <pcl::Registration::setInputTarget>`. Daná šablona se nastaví jako zdrojový PointCloud pro algoritmus SAC-IA a pak se zavolá metoda: `pcl:`align <pcl::Registration::align>` k přiřazení zdroje k cílovému PointCloud. Metoda `pcl:`align <pcl::Registration::align>` chce z hlediska své definice přidat PointCloud, do kterého by byl uložen nově přiřazený zdrojový PointCloud, ale tento výstup se pro tuto aplikaci ignoruje. Na místo toho se zavolají

přístupové metody algoritmu SAC-IA, které předají hodnotu hodnotící funkce (fitness score) a transformační matici (rigidní transformace zdrojového PointCloud k cílovému) a uloží je do výsledné struktury. [5]

```
// Align the given template cloud to the target specified by setTargetCloud
()
    void
    align (FeatureCloud &template_cloud, TemplateAlignment::Result &result)
    {
      sac_ia_.setInputCloud (template_cloud.getPointCloud ());
      sac_ia_.setSourceFeatures (template_cloud.getLocalFeatures ());

      pcl::PointCloud<pcl::PointXYZ> registration_output;
      sac_ia_.align (registration_output);

      result.fitness_score = (float) sac_ia_.getFitnessScore
(max_correspondence_distance_);
      result.final_transformation = sac_ia_.getFinalTransformation ();
    }
```

Protože je tato třída navržena k práci s více šablonami, je nutné definovat metodu pro postupné přiřazování všech šablon k cílovému PointCloud a uložení výsledků ve vektoru výstupní struktury. [5]

```
// Align all of template clouds set by addTemplateCloud to the target
specified by setTargetCloud ()
    void
    alignAll (std::vector<TemplateAlignment::Result,
Eigen::aligned_allocator<Result> > &results)
    {
      results.resize (templates_.size ());
      for (size_t i = 0; i < templates_.size (); ++i)
      {
        align (templates_[i], results[i]);
      }
    }
```

Nakonec se definuje samotná metoda *findBestAlignment*, která přiřadí všechny šablony k cílovému PointCloud a vrátí index nejlepší shody a odpovídající výstupní strukturu. [5]

```
// Align all of template clouds to the target cloud to find the one with best
alignment score
    int
    findBestAlignment (TemplateAlignment::Result &result)
    {
      // Align all of the templates to the target cloud
      std::vector<Result, Eigen::aligned_allocator<Result> > results;
      alignAll (results);
```

```
    // Find the template with the best (lowest) fitness score
    float lowest_score = std::numeric_limits<float>::infinity ();
    int best_template = 0;
    for (size_t i = 0; i < results.size (); ++i)
    {
      const Result &r = results[i];
      if (r.fitness_score < lowest_score)
      {
        lowest_score = r.fitness_score;
        best_template = (int) i;
      }
    }

    // Output the best alignment
    result = results[best_template];
    return (best_template);
  }
```

6.3.4 Hlavní program – funkce main

Nyní, když je třída, která zpracovává přiřazování šablon objektů, definována, aplikuje se na řešený problém lokalizace těles pro úlohu Bin Picking. Ve složce *data* se nachází několik šablon PointCloud, které reprezentují hledané objekty. U každé z nich byl PointCloud převzorkován na měřítko 5 mm a byly manuálně ořízнuty a upraveny tak, aby obsahovaly pouze specifickou reprezentativní část povrchu těles. V následujícím kódu je popsáno, jak se tato třída *TemplateAlignment* využije k nalezení pozice a orientace daných objektů ve snímaném PointCloud. [5]

Nejprve se načte PointCloud šablon objektů. Tyto šablony jsou uloženy ve formátu .PCD ve složce *data*, kde se také nachází textový soubor `object_templates.txt` obsahující jednotlivé cesty k daným souborům šablon. Funkce *main* vyžaduje zadaný argument *argv[1]*, který představuje string zadaný do konzole při spouštění tohoto uzlu a měl by obsahovat cestu k souboru `object_templates.txt`. Text souboru se čte řádek po řádku a název souboru se pokaždé nahraje do *FeatureCloud* a uloží se do vektoru pro pozdější použití. Dále funkce *main* vyžaduje další dva argumenty specifikované na konci této kapitoly. [5]

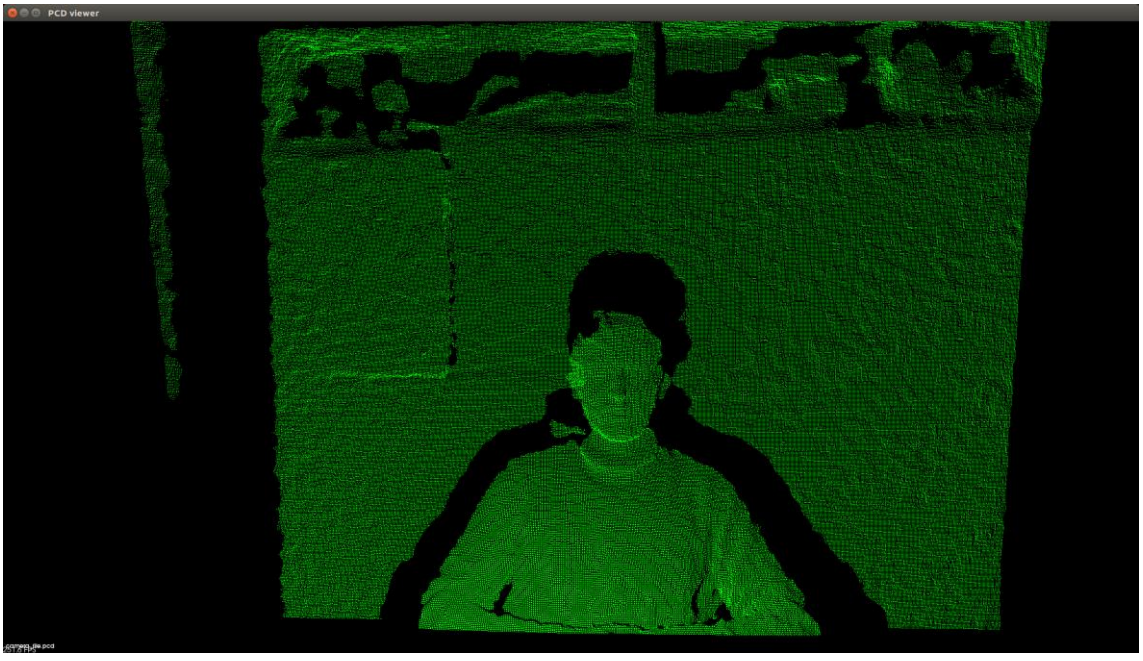
```
// Load the object templates specified in the object_templates.txt file
std::vector<FeatureCloud> object_templates;
// argv[1] contains the path to the object_templates.txt
std::ifstream input_stream (argv[1]);
object_templates.resize (0);
std::string pcd_filename;

while (input_stream.good ())
{
  std::getline (input_stream, pcd_filename);
  // Skip blank lines or comments
  if (pcd_filename.empty () || pcd_filename.at (0) == '#')
```

```
      continue;

    FeatureCloud template_cloud;
    template_cloud.loadInputCloud (pcd_filename);
    object_templates.push_back (template_cloud);
  }
  input_stream.close ();
```

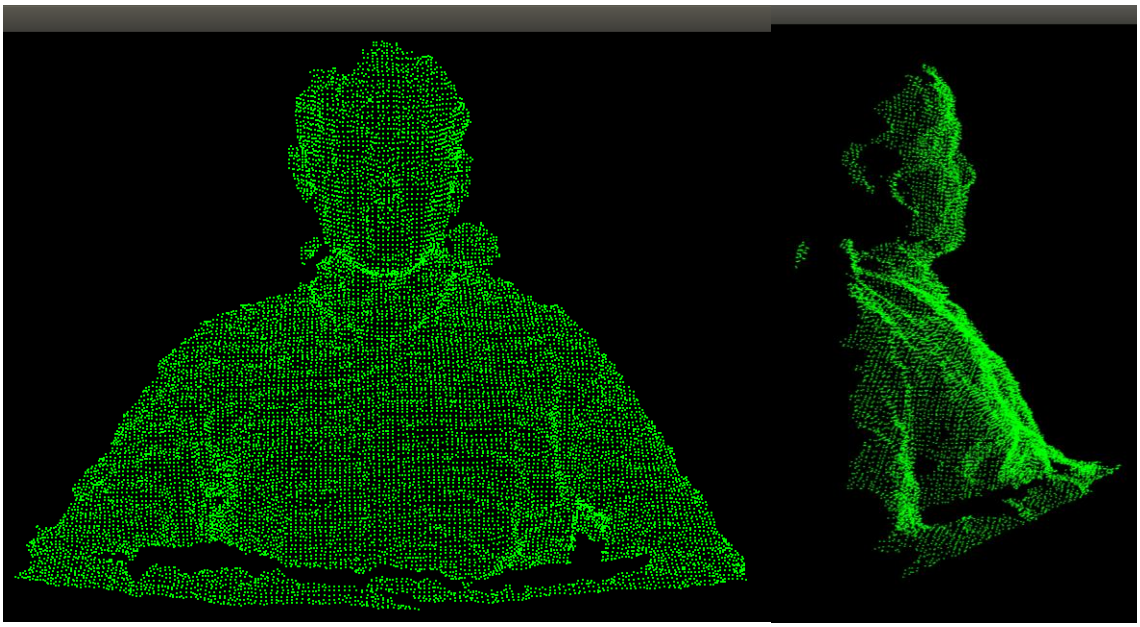
Následuje načtení cílového PointCloud pomocí ROS subscriber, které již bylo detailněji popsáno v kapitole 6.3.1.

Prostý PointCloud přijatý z kamery je nejdříve potřeba předběžně zpracovat, aby byl připravený k přiřazovacímu algoritmu. Prvním krokem je filtrace vzdálených bodů a bodů na pozadí. [5]

Při procesu snímání kamery jsou často některé body chybně zpracovány a můžou se pak nacházet na nesmyslných pozicích ve výsledných PointCloud. Příčinou těchto chyb často bývají infračervené paprsky projektoru kamery odražené od reflexních a lesklých materiálů. Dále, jelikož je v této práci cílem sledovat objekty v poměrně blízké vzdálenosti od kamery a není tak zapotřebí snímat vzdálené prostředí, které nic nevypovídá o hledaných objektech, pouze zvyšuje počet zpracovávaných bodů a zpomaluje tak pozdější výpočetní procesy, aplikuje se tzv. pass-through filtr, který filtruje snímané pole v z-tové dimenzi s limitem od 0 do 0,83 m. Správným nastavením pass-through filtru lze ve snímaném PointCloud zcela osamostatnit hledané objekty. Příklad srovnání snímků s nevhodně a vhodně nastaveným pass-through filtrem.



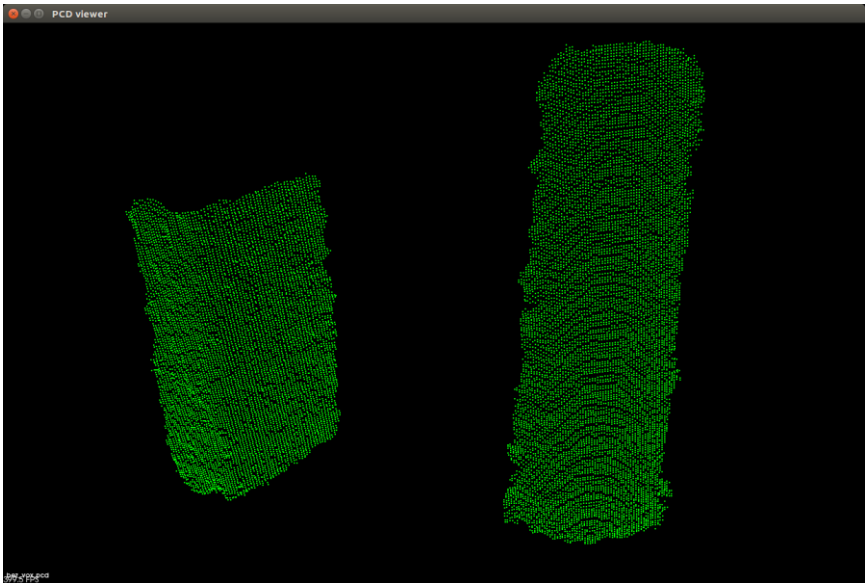
Obr. 36: Snímek s nevhodně vysoko nastavenou hodnotou hloubkového limitu



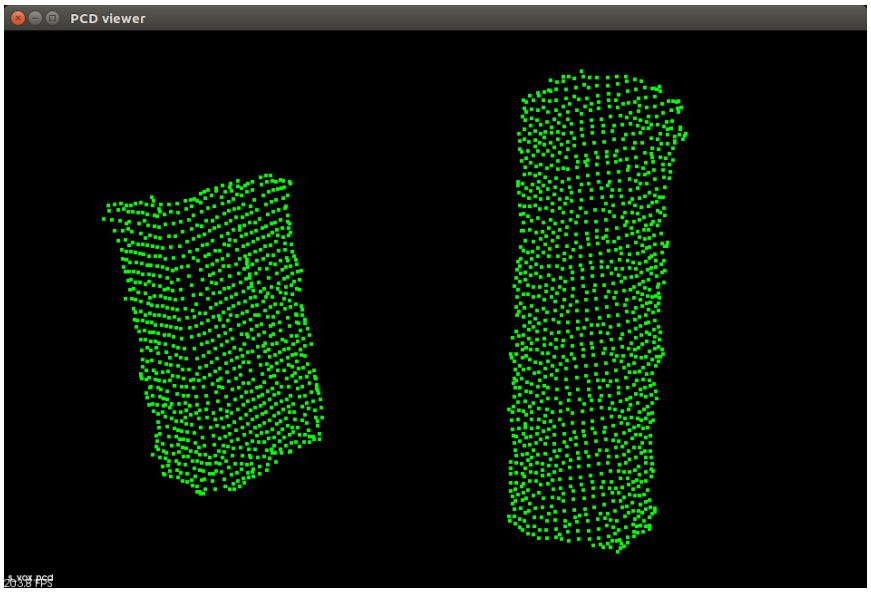
Obr. 37: Snímek s vhodně nastavenou hodnotou hloubkového limitu

```
// Preprocess the cloud by...
    // ...removing distant points
    const float depth_limit = 0.83;
    pcl::PassThrough<pcl::PointXYZ> pass;
    pass.setInputCloud (cloud);
    pass.setFilterFieldName ("z");
    pass.setFilterLimits (0, depth_limit);
    pass.filter (*cloud);
```

Kvůli úspoře výpočetního výkonu je nutné zredukovat množství bodů v PointCloud podvzorkováním, proto se původní body překryjí mřížkou nových bodů s pravidelným vzorkováním 5 mm, jak je vidět na následujícím srovnání obrázků. Více v kap. č. 4.2.1.



Obr. 38: Příklad snímaného PointCloud tvořeného hustou sítí bodů



Obr. 39: Příklad snímaného PointCloud po úpravě podvzorkováním

```
    // ... and downsampling the point cloud
    const float voxel_grid_size = 0.005f;
    pcl::VoxelGrid<pcl::PointXYZ> vox_grid;
    vox_grid.setInputCloud (cloud);
    vox_grid.setLeafSize (voxel_grid_size, voxel_grid_size, voxel_grid_size);
```

Potom, co je PointCloud předběžně zpracován, se vytvoří cílová *FeatureCloud* a data se uloží do souboru ve formátu PCD pro pozdější vizualizaci. [5]

```
    pcl::PointCloud<pcl::PointXYZ>::Ptr tempCloud
(new pcl::PointCloud<pcl::PointXYZ>);
    vox_grid.filter (*tempCloud);
    cloud = tempCloud;
    pcl::io::savePCDFileASCII ("camera_file.pcd", *cloud);
```

Dále se inicializuje objekt *TemplateAlignment*, přidají se všechny šablony a nastaví se cílový PointCloud. [5]

```
    // Assign to the target FeatureCloud
    FeatureCloud target_cloud;
    target_cloud.setInputCloud (cloud);

    // Set the TemplateAlignment inputs
    TemplateAlignment template_align;
    for (size_t i = 0; i < object_templates.size (); ++i)
    {
      template_align.addTemplateCloud (object_templates[i]);
    }
    template_align.setTargetCloud (target_cloud);
```

Nyní, když je objekt *TemplateAlignment* inicializován, je vše připraveno zavolat metodu *findBestAlignment* určující, která šablona nejlépe vyhovuje danému cílovému PointCloud. Výsledky přiřazení se následně uloží do proměnné *best_alignment*. [5]

```
    // Find the best template alignment
    while(!fitness_good)
      {
        TemplateAlignment::Result best_alignment;
        int best_index = template_align.findBestAlignment (best_alignment);
        const FeatureCloud &best_template = object_templates[best_index];
      …
```

Proces *findBestAlignment* je uzavřen v cyklu, který se bude opakovat, dokud se nesplní tato podmínka:

```
    if(best_alignment.fitness_score < atof(argv[2]) or iter == atoi(argv[3]))
    return fitness_good = true;
```

Podmínka umožňuje uživateli zadat maximální povolené *fitness_score* a maximální počet průběhů algoritmu pro přiřazování šablony.

Následuje předvedení výsledků vypsáním výstupních proměnných do konzole. První je hodnota fitness score *best_alignment.fitness_score*, která představuje ohodnocení, jak úspěšně bylo přiřazení provedeno. Dále je vypsána transformační matice *best_alignment.final_transformation* udávající polohu a orientaci objektu, který byl přiřazen do cílového PointCloud. Jedná se o rigidní transformaci, může být tedy rozložena do třídimenzionálního vektoru (t_x, t_y, t_z) a 3×3 rotační matice R, jak následuje: [5]

$$T = \begin{bmatrix} & & & t_x \\ & R & & t_y \\ & & & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

```
    // Print the alignment fitness score (values less than 0.00002 are good)
    printf ("Best fitness score: %f\n", best_alignment.fitness_score);

    // Print the rotation matrix and translation vector
    Eigen::Matrix3f rotation =
best_alignment.final_transformation.block<3,3>(0, 0);
    Eigen::Vector3f translation =
best_alignment.final_transformation.block<3,1>(0, 3);

    printf ("\n");
    printf ("    | %6.3f %6.3f %6.3f | \n", rotation (0,0), rotation (0,1),
rotation (0,2));
    printf ("R = | %6.3f %6.3f %6.3f | \n", rotation (1,0), rotation (1,1),
rotation (1,2));
    printf ("    | %6.3f %6.3f %6.3f | \n", rotation (2,0), rotation (2,1),
rotation (2,2));
```

```
    printf ("\n");
    printf ("t = < %0.3f, %0.3f, %0.3f >\n", translation (0), translation
(1), translation (2));
```

Na závěr se vezme nejlépe sedící šablona, tedy s nejnižší odpovídající fitness score a použije se na ni daná transformace, která ji zarovná s cílovým PointCloud. Tato šablona se uloží do .PCD souboru pro pozdější vizualizaci. [5]

```
    // Save the aligned template for visualization
    pcl::PointCloud<pcl::PointXYZ> transformed_cloud;
    pcl::transformPointCloud (*best_template.getPointCloud (),
transformed_cloud, best_alignment.final_transformation);
    pcl::io::savePCDFileASCII ("output.pcd", transformed_cloud);
```

Následuje příkaz sloužící k vizualizaci, která ukazuje cílový PointCloud a v něm zasazenou šablonu orientovanou dle výsledného přiřazení.

```
    system("pcl_viewer -fc 0,255,0 camera_file.pcd -fc 255,0,0 output.pcd
\"reload\"");
```

Spouštění této aplikace se provádí příkazem v terminálu. Program je umístěn v balíčku *diplomka* v *Catkin workspace* a funguje jako uzel se spouštěcí zkratkou *dipl*. Nejprve je však nutné spustit kameru pomocí jejího launchfile (podrobněji v kapitole č. 5.3), jednak právě proto, že skrze ni se získá potřebný snímek prostředí PointCloud, ale taky kvůli spuštění ROScore, který je nezbytný k tomu, aby bylo možné aplikaci v ROSu vůbec zapnout.

Dále se při spouštění zadávají tři argumenty:

7) *argv[1]* – cesta k souboru `object_templates.txt`
8) *argv[2]* – maximální dovolená hodnota *fitness_score*
9) *argv[3]* – maximální dovolený počet průběhů přiřazovacího algoritmu

7 REALIZACE A TESTOVÁNÍ

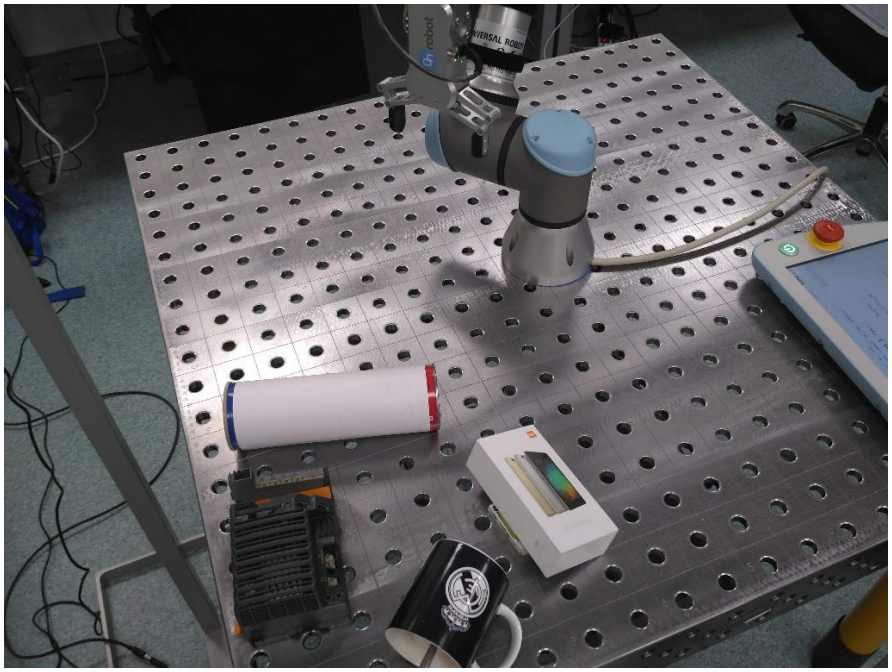
Praktické záležitosti robotického vidění byly dokončeny a následovala tak realizace s kooperativním robotem, společně s kterým by šla provést úloha Bin Picking. Realizace a testování tohoto projektu bylo provedeno ve výzkumném pracovišti Ústavu automatizace a informatiky. Pro realizaci robotického vidění s využitím uvedené aplikace byla použita kamera Microsoft Xbox 360 Kinect, která je výrazně přesnější než kamera Intel RealSense R200. Kamera Kinect byla připevněna nad pracovní stůl, který představoval výchozí scénu pro aplikaci Bin Picking. Na tomto stole je postaven robot UR3 s manipulační nadstavbou: gripper RG2.

7.1 Testování aplikace Template Alignment

7.1.1 Šablony

Byly určeny dva předměty, které slouží jako testovací objekty pro aplikaci Bin Picking. Krabička tvaru kvádru s rozměry 80 x 150 x 40 mm a tubus jako válec o průměru 75 mm a výšce 230 mm.

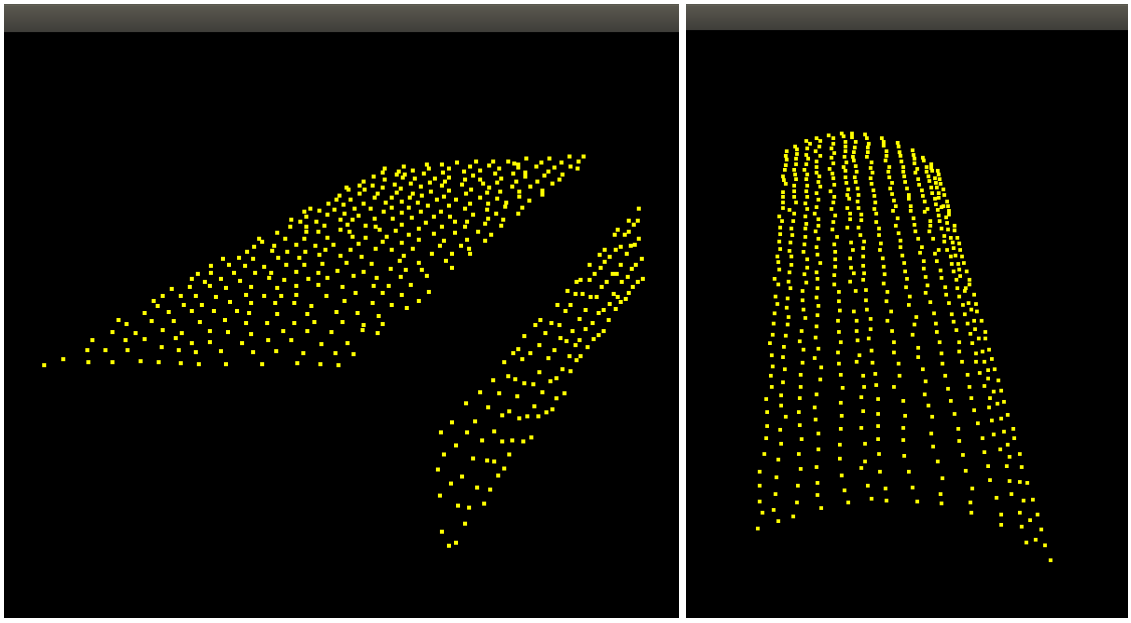
Byla testována následující scéna uspořádaných objektů obsahujících krabičku i váleček:



Obr. 40: Scéna uspořádaných objektů obsahujících krabičku i váleček

Podle těchto předmětů byly vytvořeny jejich odpovídající 3D modely a z nich metodou popsanou v kapitole č. 6.2.3 připraveny šablony ve formátu PointCloud pro zpracování aplikací Template Alignment. Volba reprezentativní části plochy šablony a její přesný tvar byla úloha vyžadující postupné vylepšování dle nově získávaných zkušeností.

Konečná verze šablony pro krabičku je vyobrazena na obrázku číslo 26. Je vidět, že se skládá ze dvou na sebe kolmých plošek s tím, že nárožní oblast je prázdná. Je to kvůli eliminaci nepřesností souvisejících s chybně vypočítanými povrchovými normálami v oblastech právě kolem hran krabičky (více v kapitole č. 4.5, obr. č. 9). Dále byla vytvořena šablona pro váleček, která byla poměrně méně komplikovaná a je vyobrazena na níže uvedeném obrázku:



Obr. 41: Nalevo šablona krabičky, napravo šablona válečku

7.1.2 Ladění programu a jeho test

Další fází je ladění programu, aby bylo přiřazení šablon do cílového PointCloud co nejpřesnější. Kamera Microsoft Xbox 360 Kinect není koncipovaná pro účely tohoto typu, a proto bylo tento optimalizační problém poměrně obtížné správně vyřešit. Nakonec se však pro tyto akademické účely nalezlo nastavení, které bylo i přes své nedokonalosti dostatečně uspokojující a mělo poměrně vysokou úspěšnost. Hodnotu procenta úspěšnosti činnosti této aplikace je obtížné získat v závislosti na několika silně nedeterministických algoritmech využitých v tomto procesu.

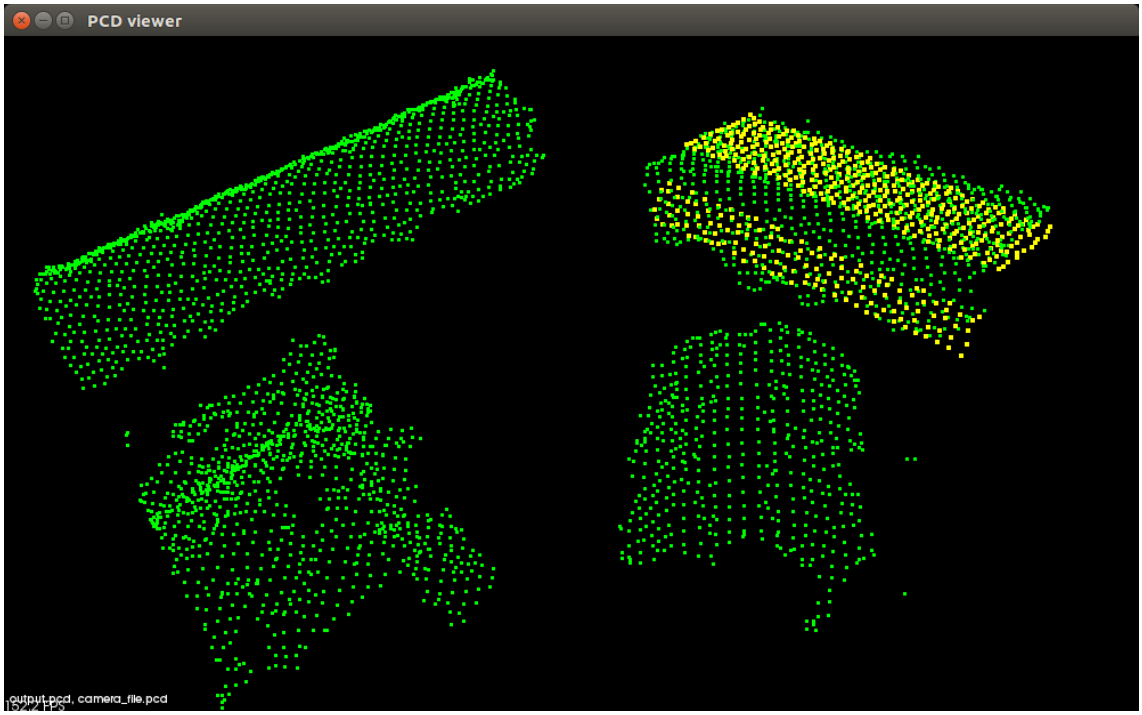
Ovlivňující parametry byly nastaveny následovně:

```
normal_radius_ = 0.0205f
feature_radius_ = 0.0205f
min_sample_distance_ = 0.052f
max_correspondence_distance_ = 0.015f*0.015f
nr_iterations_ = 1250
voxel_grid_size = 0.005f
depth_limit = 0.83
```

Vzorkování bylo nastaveno u šablon i cílového PointCloud stejně – 5 mm. Hloubkový limit byl určen tak, aby na snímaném PointCloud nebyl zachycen stůl, na kterém

předměty leží, a bylo tak ušetřeno mnoho výpočetního času. Dosahuje tedy zhruba 2 cm nad jeho povrch a díky tomu jsou ve výsledném záběru objekty od povrchu odděleny (viz obrázek níže).

Pro tento první příklad uspořádaných objektů byl spuštěn algoritmus Teamplate Alignment a výsledné přiřazení šablony k cílovému PointCloud dopadlo následovně:



Obr. 42: Výsledné přiřazení šablony krabičky (žlutě) k cílovému PointCloud (zeleně)

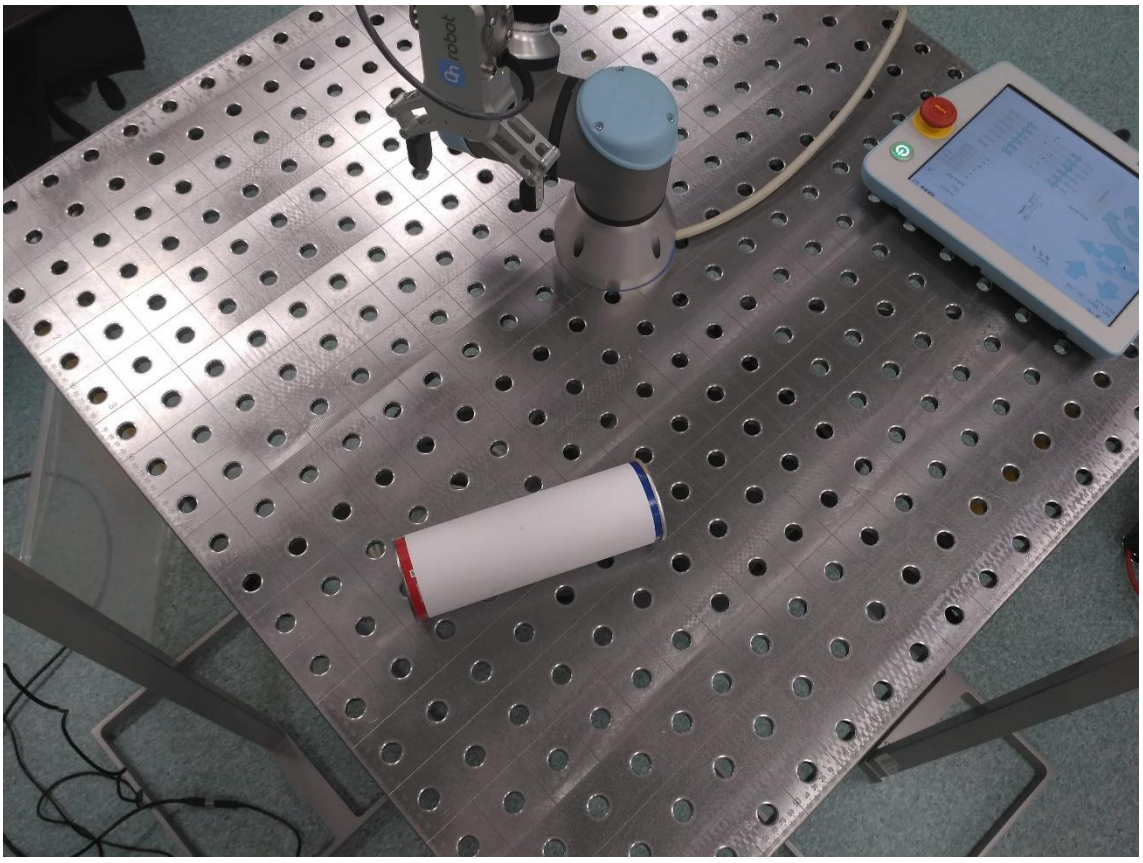
Při dokončení procesu se do terminálu vypíšou výsledné hodnoty, a to nejlepší nalezené *fitness_score*, počet iterací, rotační matice a translační vektor šablony vůči kameře. Dále pak pro představu Eulerovy úhly rotace. Jsou zde uvedeny také vektory translace a rotace se započítanými korekcemi os mezi robotem a kamerou. Realizaci samotného Bin Pickingu se však více věnuje v podkapitole č. 7.2. Výsledky tohoto testu jsou vypsány do terminálu následovně:

```
robot@robot-UP:~/catkin_ws
Best fitness score: 0.0000099689
number of iteration: 5

    |  0.507 -0.269  0.819 |
R = | -0.769  0.288  0.571 |
    | -0.389 -0.919 -0.060 |

t = < -0.107, 0.078, 0.811 >

t for robot = < 0.406, 0.127, 0.071 >

Euler angles:
R = < 84.0, -152.1, 125.0 >

Euler angles for robot
R_t = < 84.0, -152.1, -35.0 >
```

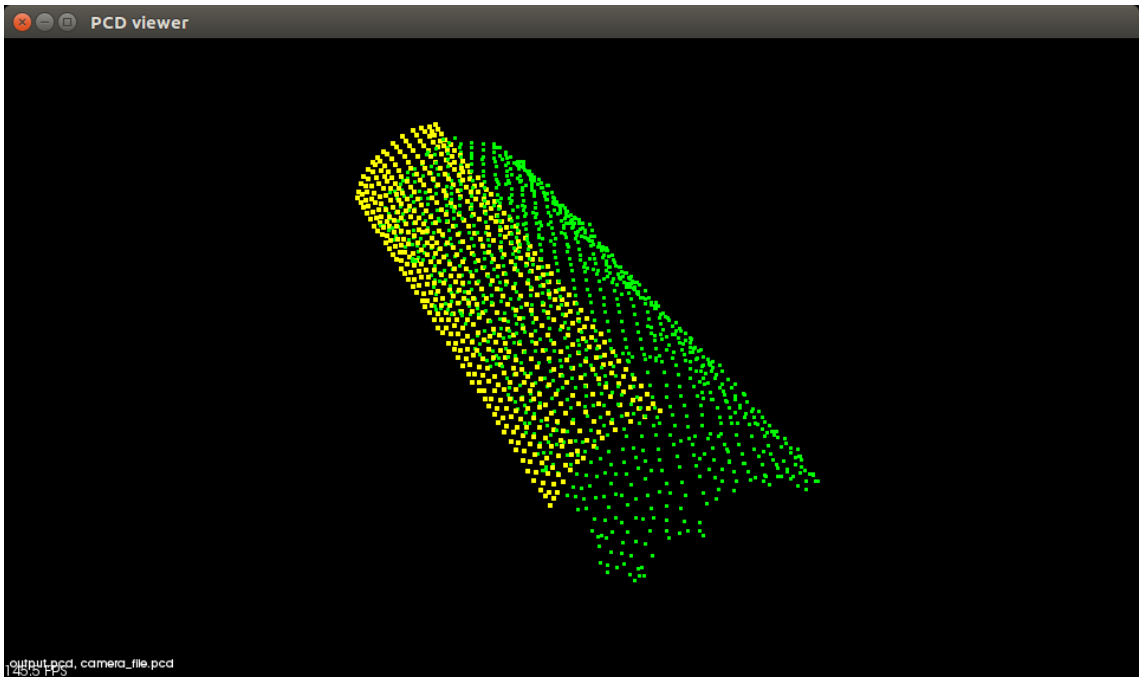
Obr. 43: Výpis výsledků přiřazovacího procesu do terminálu

Algoritmus Template Alignment zkouší přiřadit vždy všechny šablony, jejichž adresa je uvedená v textovém souboru `object_templates.txt` a vybere tu nejlepší shodnou korespondenční množinu, která představuje jeden z objektů. To znamená, že pokud se vyloženě neupraví textový soubor, algoritmus sám určuje, který objekt přiřadí první. Tato vlastnost může být žádoucí.

V tomto případě bylo nalezení krabičky snadné, což bylo pravděpodobně způsobeno tím, že je krabička tvořena rovnými plochami a povrch snímané části krabičky je tak relativně konzistentní. Daný objekt se tedy odebral a následoval druhý cyklus aplikace Template Alignment, kdy snímaná scéna obsahovala už jen jeden známý předmět. Nalezení tohoto válečku nebylo vůbec snadné. Problémem celého chybného procesu byla nedostačující přesnost použité kamery, která v případě zaobleného povrchu byla více nekonzistentní než u rovných ploch. Nakonec se muselo hledané těleso úplně osamostatnit, aby se docílilo přibližného výsledku, jak ukazují následující obrázky:



Obr. 44: Scéna osamostatněného objektu válečku



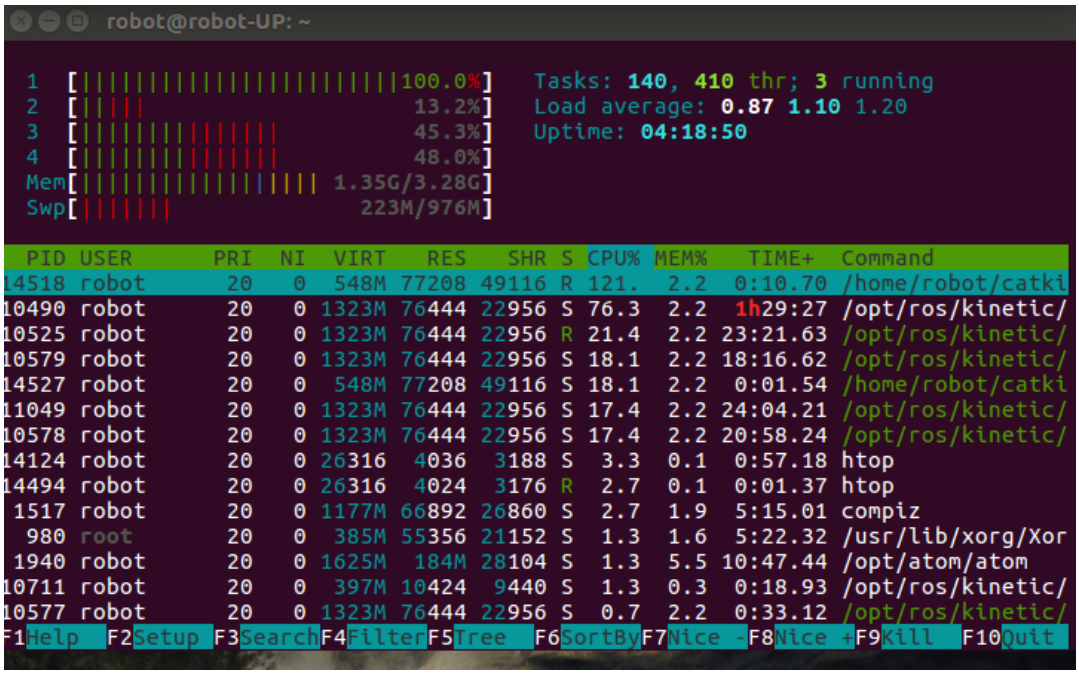
Obr. 45: Výsledné přiřazení šablony válečku (žlutě) k cílovému PointCloud (zeleně)

```
robot@robot-UP:~/catkin_ws
robot@robot-UP:~$ cd catkin_ws
robot@robot-UP:~/catkin_ws$ rosrun diplomka dipl /home/robot/catkin_ws/src/diplomka/data/object_
templates.txt 0.000005 20
waiting for first frame...
Received a cloud message...
converted
processing...
Best fitness score: 0.0000050749
number of iteration: 1
need better fitness...
Best fitness score: 0.0000047256
number of iteration: 2

    | -0.465  0.010 -0.885 |
R = |  0.885  0.025 -0.465 |
    |  0.018 -1.000 -0.021 |

t = < 0.002, 0.137, 0.818 >

t for robot = < 0.347, 0.018, 0.064 >

Euler angles:
R = < 92.5, -178.8, -62.3 >

Euler angles for robot
R_t = < 92.5, -178.8, 152.3 >

The viewer window provides interactive commands; for help, press 'h' or 'H' from within the wind
ow.
> Loading camera_file.pcd [done, 349 ms : 1840 points]
Available dimensions: x y z
> Loading output.pcd [done, 14 ms : 511 points]
Available dimensions: x y z
```

Obr. 46: Výpis výsledků druhého průběhu programu

7.1.3 Sledování výkonu Procesoru

Pro sledování výpočetního zatížení UP-boardu byl použit program Htop, který je implementován přímo do terminálu. Jak je vidět na níže uvedeném obrázku, operační paměti je dostatek, ovšem zatížení jednoho jádra procesoru je 100 %. Aplikace je spouštěna jako jeden uzel v ROSu a jelikož není implementována pro multithreading, musí celý proces probíhat jen na jednom jádře procesoru. Kvůli obraně proti přehřívání se však ve výpočtu všechna čtyři jádra postupně střídají.



Obr. 47: Program Htop při průběhu algoritmu

7.2 Realizace Bin Picking s využitím robota UR3

K realizaci úlohy Bin Picking již stačilo vykonat pár potřebných úkonů, aby se celý projekt dočkal reálného testu. Programování navigace manipulátoru bylo vykonáváno za spolupráce s výzkumným pracovníkem ústavu. Z časových důvodů nebyla řešena komunikace aplikace Template Alignment a řídicího programu robota. Výstupy z přiřazovacího algoritmu byly předávány ručně.

Rozvržení souřadnicových systémů kamery a robota bylo nutné sjednotit. Vybrala se výchozí pozice kamery vůči robotu tak, aby byly osy souřadnicových systémů rovnoběžné. Celková kalibrace kamery byla provedena poměrně zjednodušeně, protože na výsledek přiřazení šablony to nemělo vliv a pro navigaci robota stačilo přidat několik offsetů. Zbývající rotační transformace byly implementovány v řídicím programu robota. Výsledné záběry přemístění rozpoznaných objektů pomocí robota UR3 na základě získaných translačních a rotačních hodnot z aplikace Template Alignment jsou zachyceny na dvou videích nahraných na přiloženém CD.

8 ZÁVĚR

Cílem této diplomové práce bylo popsat přístup robotického vidění pro aplikaci Bin Picking a vytvořit aplikaci pro realizaci této úlohy. Tato aplikace je schopna rozlišit několik zvolených objektů na základě dat z kamery s hloubkovým vjemem a umí hledaný objekt lokalizovat, rozeznat a v daném místě určit jeho polohu a orientaci.

V této práci jsou vysvětleny pojmy robotické vidění a Bin Picking spolu s uvedením několika firem, které tuto úlohu řeší. Dále jsou popsány prostředky využité při realizaci – jak softwarové, tak hardwarové. Následující rozsáhlou kapitolu tvoří podrobný popis metod zpracovávání 3D obrazu a vysvětlení mnoha důležitých pojmů v této problematice. Kapitola 6 popisuje přípravu šablon objektů pro přiřazovací algoritmus a celou implementaci aplikace robotického vidění. V závěrečné kapitole jsou popsány výsledky realizace a testování.

Při některých konkrétních tvarech objektů hrály velkou roli značné nepřesnosti hloubkových snímků kamery, která nebyla koncipována pro účely této aplikace, a měly tak za následek neúspěšné přiřazení šablony předmětu k cílovému PointCloud. Nicméně při aplikaci jednodušších běžných předmětů byla funkce přiřazovacího procesu správná a objekty snadno přesně lokalizovala.

Lze konstatovat, že aplikace vytvořená pro tuto diplomovou práci vykonává proces naplňující cíle uvedené v zadání. Při testování na reálných objektech se podařilo nalézt a lokalizovat hledané předměty a díky využití výzkumného pracoviště Ústavu automatizace a informatiky bylo možné zrealizovat i samotnou techniku Bin Picking s využitím robota UR3, který dle aplikací získaných souřadnic uchopil těleso a přesunul jej na stanovenou pozici.

SEZNAM POUŽITÉ LITERATURY

[1] Georgina, Lilia. *01.org* [online] 2018 [cit. 2019-04-24]. Dostupné z: https://01.org/developerjourney/recipe/intel-realsense-robotic-development-kit#step2

[2] U*p-board.org* [online] 2018 [cit. 2019-04-24]. Dostupné z: https://up-board.org/up/specifications/

[3] *Intel.com* [online] 2017 [cit. 2019-04-24]. Dostupné z: https://www.intel.com/content/www/us/en/support/articles/000022699/emerging-technologies/intel-realsense-technology.html

[4] *Wiki.ros.org* [online] 2017 [cit. 2019-04-25]. Dostupné z: http://wiki.ros.org/roscpp/Overview/Callbacks%20and%20Spinning

[5] Dixon, Michael. *pointclouds.org* [online] 2011 [cit. 2019-05-2]. Dostupné z: http://pointclouds.org/documentation/tutorials/template_alignment.php

[6] *pointclouds.org* [online] 2011 [cit. 2019-05-8]. Dostupné z: http://www.pointclouds.org/documentation/tutorials/kdtree_search.php

[7] *wikipedia.org* [online] 2019 [cit. 2019-05-8]. Dostupné z: https://en.wikipedia.org/wiki/K-d_tree

[8] *pointclouds.org* [online] 2011 [cit. 2019-05-9]. Dostupné z: http://pointclouds.org/documentation/tutorials/normal_estimation.php#normal-estimation

[9] *pointclouds.org* [online] 2011 [cit. 2019-05-11]. Dostupné z: http://pointclouds.org/documentation/tutorials/pfh_estimation.php#pfh-estimation

[10] Herák, Patrik. *Rozpoznanie objektov a ich polohy s využitím 3D modelov objektov* Ostrava: Technická univerzita Ostrava, Fakulta elektrotechniky a informatiky, 2017. 16 – 29 s. Vedoucí diplomové práce doc. Dr. Ing. Eduard Sojka.

[11] *pointclouds.org* [online] 2011 [cit. 2019-05-13]. Dostupné z: http://pointclouds.org/about/

[12] *openkinect.org* [online] 2016 [cit. 2019-05-13]. Dostupné z: https://openkinect.org/wiki/Getting_Started

[13] *pointclouds.org* [online] 2011 [cit. 2019-05-13]. Dostupné z: http://pointclouds.org/documentation/tutorials/compiling_pcl_posix.php

[14] *pointclouds.org* [online] 2011 [cit. 2019-05-14]. Dostupné z: http://pointclouds.org/documentation/tutorials/pcd_file_format.php

[15] Rusu, Radu Bogdan, Blodow, Nico, Beetz, Michael. *cvl.iss* [online] 2017 [cit. 2019-05-14]. Dostupné z: http://www.cvl.iis.u-tokyo.ac.jp/class2017/2017w/papers/5.3DdataProcessing/Rusu_FPFH_ICRA2009.pdf

[16] *wikipedia.org* [online] 2018 [cit. 2019-05-15]. Dostupné z: https://en.wikipedia.org/wiki/PLY_(file_format)

[17] *wikipedia.org* [online] 2018 [cit. 2019-05-15]. Dostupné z: https://en.wikipedia.org/wiki/STL_(file_format)

[18] *meshlab.net* [online] 2016 [cit. 2019-05-15]. Dostupné z: http://www.meshlab.net

[19] *pointclouds.org* [online] 2011 [cit. 2019-05-17]. Dostupné z: http://pointclouds.org

[20] *ros.org* [online] 2013 [cit. 2019-05-18]. Dostupné z: http://www.ros.org

[21] *pointclouds.org* [online] 2011 [cit. 2019-05-10]. Dostupné z: http://pointclouds.org/documentation/tutorials/fpfh_estimation.php

[22] *datascience.stackexchange.com* [online] 2018 [cit. 2019-05-10]. Dostupné z: https://datascience.stackexchange.com/questions/28296/how-the-squared-euclidean-distance-is-an-example-of-non-metric-function

[23] *wiki.ros.org* [online] 2019 [cit. 2019-05-19]. Dostupné z: http://wiki.ros.org/IDEs

[24] *wiki.ros.org* [online] 2017 [cit. 2019-05-19]. Dostupné z: http://wiki.ros.org/catkin/Tutorials/create_a_workspace

[25] *wiki.ros.org* [online] 2017 [cit. 2019-05-19]. Dostupné z: http://wiki.ros.org/catkin/Tutorials/CreatingPackage

[26] Newman, Wyatt. *A Systematic Approach to Learning Robot Programming with ROS*. CRC Press, 2017. 270 s. ISBN 1498777848, 9781498777841

[27] *reconstructme.net* [online] 2018 [cit. 2019-05-20]. Dostupné z: http://reconstructme.net/qa_faqs/intel-realsense-r200-review/

[28] *software.intel.com* [online] 2016 [cit. 2019-05-20]. Dostupné z: https://software.intel.com/sites/default/files/managed/d7/a9/realsense-camera-r200-product-datasheet.pdf

[29] *wikipedia.org* [online] 2019 [cit. 2019-05-20]. Dostupné z: https://en.wikipedia.org/wiki/Kinect#Kinect_for_Xbox_360_(2010)

[30] *forum.ni-mate.com* [online] 2016 [cit. 2019-05-20]. Dostupné z: https://forum.ni-mate.com/t/documentation-kinect-for-windows-1-kinect-for-xbox-360-models-1414-1473/288

[31] *allaboutcircuits.com* [online] 2017 [cit. 2019-05-20]. Dostupné z: https://www.allaboutcircuits.com/news/teardown-tuesday-microsofts-xbox-360-kinect/

[32] *campar.in.tum.de* [online prezentace] 2011 [cit. 2019-05-20]. Dostupné z: http://campar.in.tum.de/twiki/pub/Chair/TeachingSs11Kinect/2011-DSensors_LabCourse_Kinect.pdf

[33] *i.ytimg.com* [online] 2010 [cit. 2019-05-21]. Dostupné z: https://i.ytimg.com/vi/crTa_fcixTg/maxresdefault.jpg

[34] *husarion.com* [online] 2019 [cit. 2019-05-21]. Dostupné z: https://husarion.com/tutorials/ros-tutorials/1-ros-introduction/

[35] *allaboutcircuits.com* [online] 2017 [cit. 2019-05-21]. Dostupné z: https://www.allaboutcircuits.com/technical-articles/an-introduction-to-robot-operating-system-ros/

[36] *moveit.ros.org* [online] 2019 [cit. 2019-05-21]. Dostupné z: https://moveit.ros.org

[37] *blog.robotiq.com* [online] 2016 [cit. 2019-05-21]. Dostupné z: https://blog.robotiq.com/robot-vision-vs-computer-vision-whats-the-difference

[38] *teqram.com* [online] 2018 [cit. 2019-05-21]. Dostupné z: https://teqram.com/robotic-systems/bin-picking/

[39] *therobotreport.com* [online] 2019 [cit. 2019-05-22]. Dostupné z: https://www.therobotreport.com/fully-automated-bin-picking-finally-here/

[40] *photoneo.com* [online] 2019 [cit. 2019-05-22]. Dostupné z: https://www.photoneo.com/bin-picking/

[41] *fanuc.eu* [online] 2019 [cit. 2019-05-22]. Dostupné z: https://www.fanuc.eu/cz/cs/roboty/příslušenství/vidění

[42] *fanuc.eu* [online] 2019 [cit. 2019-05-22]. Dostupné z: https://www.fanuc.eu/dk/en/industrial-applications/machine-tending

[43] *ictrevue.ihned.cz* [online] 2015 [cit. 2019-05-22]. Dostupné z: https://ictrevue.ihned.cz/c3-64418960-0ICT00_d-64418960-canon-vyvinul-3d-videni-pro-prumyslove-roboty

SEZNAM ZKRATEK, OBRÁZKŮ A TABULEK

Seznam zkratek

ROS	Robotic Operation System
UR3	Universal Robot 3
PCL	Point Cloud Library
3D	trojdimenzionální
2D	dvojdimenzionální
CAD	Computer aided design
USB	Universal Serial Bus
ASIC	Application Specific Integrated Circuit
RGB	model červená-zelená-modrá
CMOS	Complementary Metal–Oxide–Semiconductor
ToF	Time-of-Flight
SAC-IA	Sampe Consensus Initial Alignment
PCA	Analýzy hlavních komponent
PFH	Point Feature Histograms
FPFH	Fast Point Feature Histograms
SPFH	Simplified Point Feature Histogram
GIA	Greedy Initial Alignment
CPU	Centrální procesorová jednotka
IDE	Integrated Development Environment
FLANN	Fast Library for Approximate Nearest Neighbors
VTK	Visualization Toolkit
PCD	Poin Cloud Data
PLY	Polygon File Format
STL	Standard Triangle Language
OBJ	object file format
X3D	3D file format
XML	Extensible Markup Language

Seznam obrázků

Seznam tabulek