Skoltech

Skolkovo Institute of Science and Technology

Skolkovo Institute of Science and Technology

Master's Educational Program: Space Systems

Title: 3D pose estimation algorithm for intelligent box picking of warehouse automation robot

Author: / ________________________________

Certified by: / ________________________________

____

[Research (Thesis) Advisor's Signature, Name and Title]

Accepted by: / ________________________________

____________, Dean of Education, Skoltech

Moscow

____________________ 2017

ACKNOWLEDGEMENTS

I would like to express my gratitude to the head of Skoltech Intelligent Space Robotics Laboratory professor Dzmitry Tsetserukou for his guidance as the supervisor and provided opportunity to work at the laboratory and use its equipment for engineering and research. Also, I am very grateful to my colleges - master students Alexandr Petrovskii, Alexey Postnikov, and Evgeniya Yureva for their help and support in performing my Thesis research.

ABSTRACT

Order picking is one of the most costly and labor-extensive operations in a typical warehouse. It entails 50% of operational expenses, thus being a high priority area for automation and cost reduction. Several attempts have been made towards an automation of order picking by companies and research groups around the Globe. However, there is still no solution suitable for order picking in a warehouse not specifically equipped for this purpose.

The main bottleneck of applications where robots have to work in environment built for humans is a perception ability. An appearance of low-cost depth sensors started with Microsoft Kinect, has opened a new research field in robotics that tries to provide better perception abilities to robots using a combination of depth and RGB data. Many methods have been proposed over last 5 years. These algorithms mostly use RANSAC-based techniques or one of the 3D point descriptors in order to fit a model into image. Many methods for deriving primitive shapes and mapping surrounding environments using RGB-D images originate in Photogrammetry and require a human input on processing stage for choosing potential candidates for detection. There are also methods built upon collected databases of CAD-models of objects to be detected.

The goal of this work is to develop a computer vision algorithm for reliable registration and pose estimation of boxes on a pallet. We propose a novel approach to object detection and pose estimation of cuboid-like objects. It is based on a combination of several approaches (edge-based, descriptors-based and region growing clustering-based). Such combination allows for more robust object detection in a highly cluttered environment where objects are occluded by each other. Proposed approach requires neither manual input nor a preloaded database of models to be detected.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

LIST OF TABLES

LISTINGS

NOMENCLATURE

$2D$**.** Two-dimensional space is a geometric model of the planar projection of the physical world.

$3D$**.** In physics and mathematics, a sequence of n numbers can be understood as a position in n-dimensional space. When $n = 3$, the set of all possible such locations is called three-dimensional Euclidean space.

API**.** In computer programming, an application programming interface (API) is a set of definitions, protocols, and tools for simplifying the process of building application software.

CAD**.** Computer-aided design is the use of computer systems to aid in the creation, modification, analysis, or optimization of a design.

IR**.** Infrared radiation, or simply infrared is electromagnetic radiation with longer wavelengths than those of visible light, and is therefore invisible.

NI**.** In computing, a natural interface is a user interface that is effectively invisible, and remains invisible as the user continuously learns complex interactions.

PCL**.** The Point Cloud Library (PCL) is a standalone, large scale, open project for 2D/3D image and point cloud processing.

$RANSAC$**.** Random sample consensus. An iterative method to estimate parameters of a mathematical model from a set of observed data that contains outliers.

RGB**.** Additive color model in which red, green and blue light are added together in various ways to reproduce a broad variety of colors.

$RGB - D$**.** RGB-Depth. Image containing depth information about each pictured point as well as associated RGB values.

ROS**.** Robot Operating System provides libraries and tools for robot applications development.

TOF**.** Time-Of-Flight denotes a variety of methods that measure the time that it takes for an object, particle or wave to travel a distance through space.

$URDF$**.** Unified Robot Description Format. An specific version of XML format for representing a robot model.

WMS**.** A warehouse management system (WMS) is a software application that supports the day-to-day operations in a warehouse. WMS programs enable centralized management of tasks typical for a warehouse.

Chapter 1

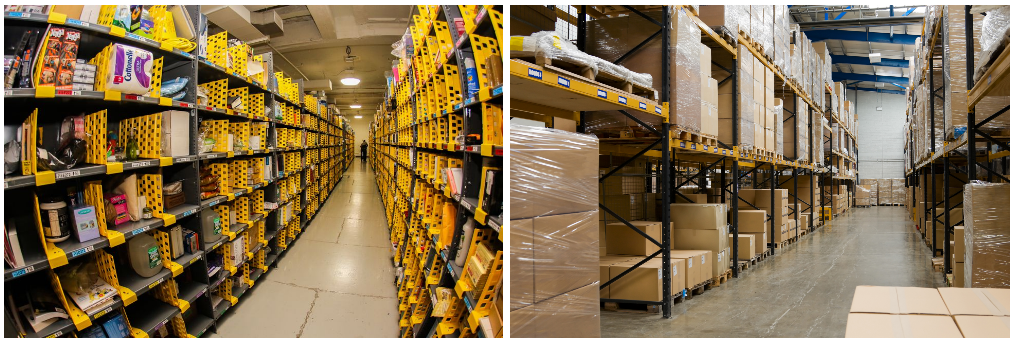
INTRODUCTION

1.1 Warehouse automation

According to the “Warehouses review in Moscow and Moscow Region” [1], warehouses occupy an area of more than 13 000 000 square meters only in Moscow region and this number grows rapidly. The Zebra Technologies Warehouse Vision Study [2] finds that 7 out of 10 key decision makers in logistics plan to accelerate their use of technology to create a smart warehouse system by 2020. Nearly half of responders indicated a concern about human labor performance in the order, pick, and fulfillment processes. This opens up an opportunity for new companies in these areas. No surprises, the worldwide industrial logistics robots market grows rapidly: Consideration of Robot Market Forecasts indicates that market at $16 billion will reach $31.3 billion by 2020 [3].

Order picking is the most labor-intensive and costly activity for typical warehouse. Different researches show that the cost of this activity is estimated to be as much as 50% of total warehouse operating expenses [4]. Hence, a lack of performance or failures in order picking results in an increase of operational cost not only for a warehouse but also for the whole supply chain. Based on this evidence, order picking is being considered as a highest-priority area for productivity improvements and automation. Currently, there is a big emphasis on warehouse automation in research papers. Moreover, there are several successful robotics solutions on the market aimed at solving this issue. Amazon robotics [5], Locus robotics [6], Fetch Robotics [7], just to name few, are currently developing their solutions. These companies focus on automation of different warehouse processes, in particular, Kiva Systems robots (the company was acquired by Amazon for $775 million [8]) are capable of moving the big shelves of goods, while picking process is performed by a human. Meanwhile, robots from Fetch Robotics, (the company received $20 million investment from SoftBank in 2015 [9]) are able to pick small separate goods from the shelves, therefore being useful for small warehouses storing separate goods (Figure 1.1a), but not for relatively big warehouses with goods packed into boxes (Figure 1.1b).

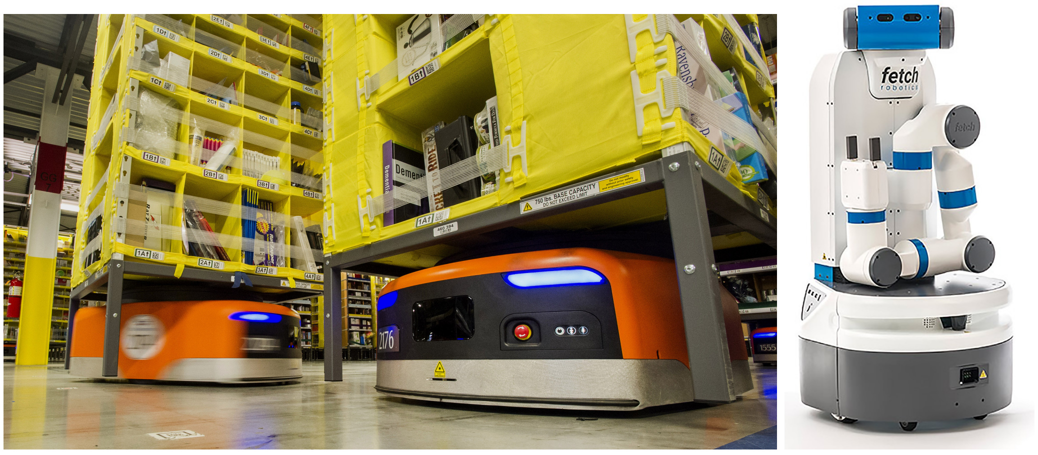
Despite the recent success of several research groups working on robotics intelligence [10, 11, 12], robotic systems are still not able to operate successfully in environments not built specifically for them. Due to this reason, the main drawback of warehouse solutions mentioned above is a requirement of serious changes, if not to say complete reorganization of a warehouse infrastructure. This brings additional costs which are not always comparable with profits obtained by introduction



(a) Warehouse storing separate goods. (b) Warehouse with goods packed into cardboard boxes.

Figure 1.1: Different types of warehouses.

of these systems. There are millions of square meters of warehouses already exist and need to be automatized as well.



(a) Kiva robots. (b) Fetch robots.

Figure 1.2: Existing warehouse automation robots.

1.2 Proposed solution for warehouse automation

We propose «Plug&Play» solution for automation of order picking process. It consist of several core components:

- Mobile platform able to autonomously navigate through warehouse.
- Industrial manipulator with adaptive gripper able to pick and manipulate objects (*i.e.* cardboard boxes) of different sizes.
- Computer vision system able to identify boxes lying on pallets and estimate there position in 3D space.

More detailed system decomposition is presented on Figure 1.3. Some parts are omitted for the simplicity.

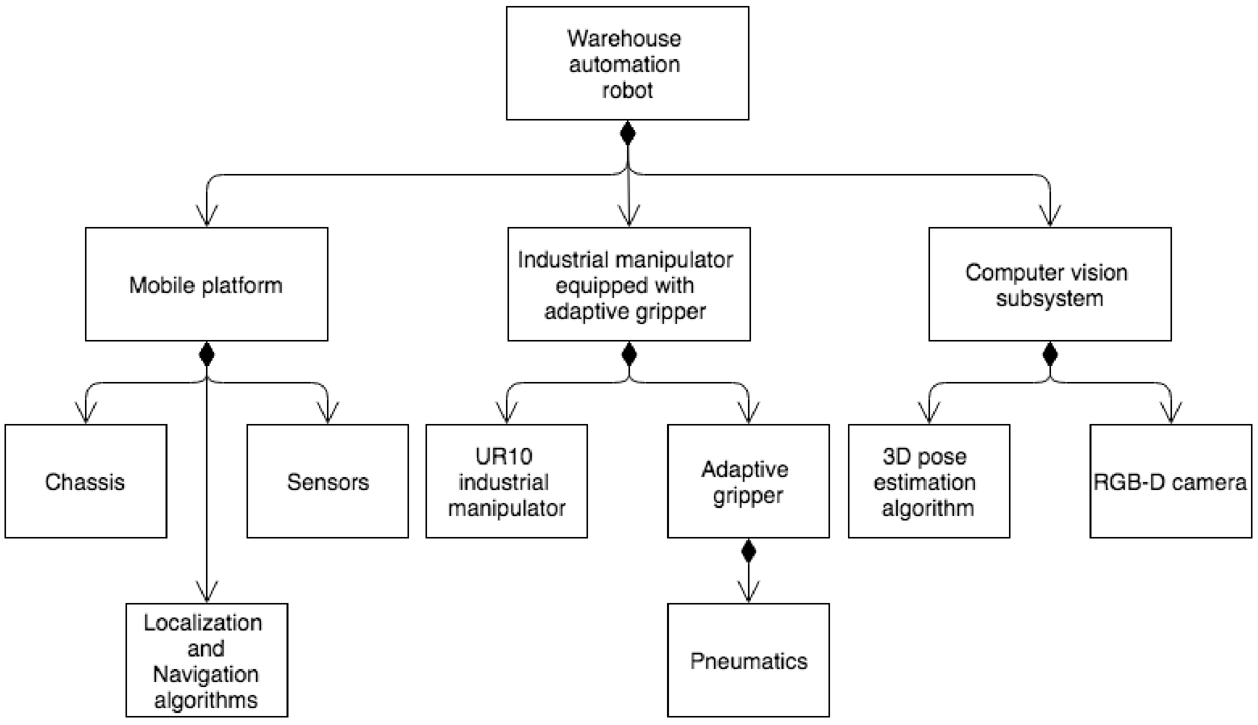
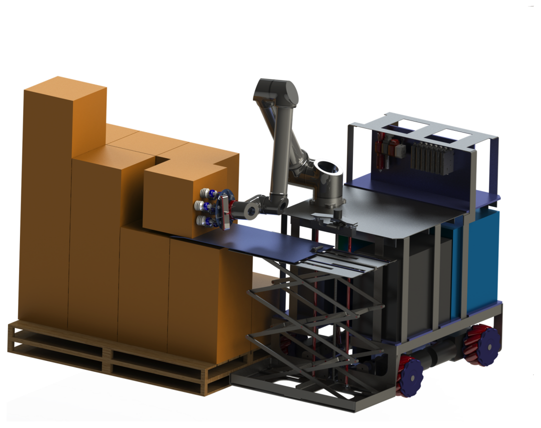

Warehouse automation robot
Mobile platform
Industrial manipulator equipped with adaptive gripper
Computer vision subsystem
Chassis
Sensors
UR10 industrial manipulator
Adaptive gripper
3D pose estimation algorithm
RGB-D camera
Localization and Navigation algorithms
Pneumatics

Figure 1.3: Order picking automation robot. System decomposition.

The 3D model and hardware prototype of the robot are presented on Figure 1.4a and Figure 1.4b respectively.


(a) SolidWorks 3D model.

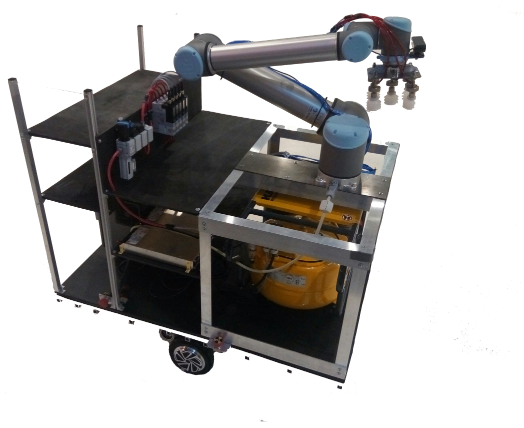
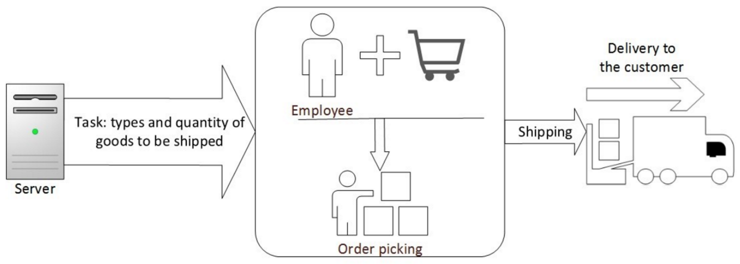

(b) Hardware prototype.

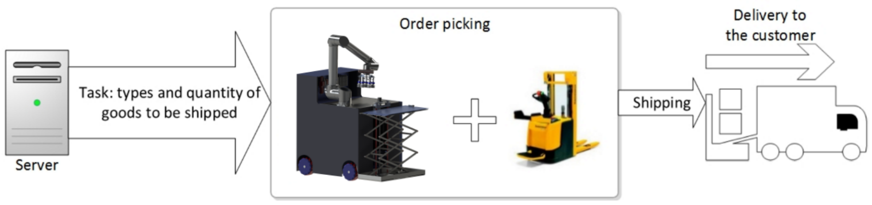
Figure 1.4: Warehouse automation robot.

Our solution focuses on order picking in warehouses where goods are represented by typical cardboard boxes or boxes with bottles. Figure 1.5a describes the typical order-picking process. First, the information on new order arrives from the customer to the server. An employee takes an empty pallet and walks through a warehouse collecting requested goods to form the pallet to be shipped. A reconsidered with an introduction of warehouse automation robot work-flow sequence is presented on Figure 1.5b. It consists of the following steps:

1. Information about new order arrives to the server;
2. Task is scheduled in order to be performed by vacant picking robot;
3. The optimal route is planned by the robot. In case of obstacles appearance, the robot will recalculate its path dynamically;
4. Robot followed by human with an empty pallet (or another robot, carrying a pallet) performs picking of required goods and fills the pallet;
5. The full pallet is sent to a customer.



(a) Typical work-flow.



(b) Reconsidered work-flow.

Figure 1.5: Order picking process.

1.3 Computer vision algorithm

The core component of the system is a computer vision algorithm performing recognition and 3D pose estimation of boxes containing goods. In order to be used in a real-world application such algorithm has to be reliable enough, because, as it was mentioned above, any failure in the order picking process would have a significant impact on the whole supply chain. Another important requirement for the pose estimation system is that it needs to be fast enough in order to outperform or at least be comparable with human performance.

There are many existing algorithms for object recognition and pose estimation. However, none of them satisfy previously mentioned requirements, thus can not be used directly. One possible way of overcoming this hurdle is to combine existing approaches in order to improve recognition and pose estimation quality. In this work we propose a novel approach to object recognition and 3D pose estimation of rectangular objects. The method is a combination of edge-based, descriptors-based and region growing clustering-based approaches. Such combination allows for more robust object detection in a highly cluttered environment where objects are occluded by each other. Importantly, proposed approach doesn't require manual input or a preloaded database of models to be detected. In following sections we discuss in details each individual algorithm and evaluate it on a dataset of RGB-D images, collected on a real warehouse. The combination of algorithms then described and evaluated against each individual algorithm. We also discuss an overall architecture of warehouse automation robot and show how developed algorithm is integrated with the rest of the system (in particular with UR-10 industrial manipulator) using Robot Operating System (ROS).

The rest of the work is structured as follows: (1) chapter two discusses related work. (2) chapter three elaborates on variety of depth sensors and discusses their principles of work. (3) chapter four contains a detailed description of all individual algorithms discussed in a scope of this work and shows an approach to their combination. (4) chapter five reveals implementation details. (5) chapter six evaluates obtained result and draws a conclusion about efficiency of a proposed approach. (6) chapter seven contains conclusion and brief overview of the work done.

Chapter 2

RELATED WORK

2.1 Object recognition and 3d pose estimation

Object recognition and pose estimation is a well-studied problem in a computer vision due to its wide applications in robotics, virtual reality, mapping, *etc.* Many methods have been proposed over the last years. These algorithms mostly one of the point descriptors in order to find correspondences between model and scene. There are two main approaches to the object recognition based on descriptors: local (*e.g.* [13, 14, 15]) and global ones (*e.g.* [16, 17]). Global feature-based methods are based on the whole visual appearance of the object. In other words, they are able to describe the object with the single vector. The drawback of this group of methods is that they can not capture shape details and require prior segmentation of the potential candidate from the scene. Therefore, global-based methods can not be used for recognition of partially occluded objects in cluttered scenes. In contrast, local descriptors-based methods are based on interpretation of local geometric features extracted from the points. They are more suitable for cluttered scenes and occlusions. In [18] authors performed a deep analysis of all descriptors currently implemented in PCL library [19]. PCL library on its own is a powerful tool, containing various algorithms for point cloud processing. The framework contains all individual algorithm required for designing an efficient 3D recognition system.

Many methods for deriving primitive shapes and mapping surrounding environments using RGB-D images, such as one presented in [20], originate in Photogrammetry. The majority of these methods can be classified as semi-automatic methods since they require a human input on processing stage for choosing potential candidates for detection. Moreover, these methods are usually not fast enough to be used in robotics, where near real time is required in most applications

There are also methods built upon collected databases of CAD-models of objects to be detected [21]. They show good performance, however, require preliminarily collected database of objects, thus being not flexible if a new object is introduced to the system and needs to be recognized as well.

2.2 Depth sensors

The appearance of cheap depth-cameras on the market has boosted the development of 3D computer vision area and its applications in real-life problems. There are several ways to get the depth information and consequently types of depth-sensors:

- Time-Of-Flight cameras [22].
- Stereo-pair-based cameras [23].
- IR based cameras [24].

The Microsoft Kinect sensor which became the first low-cast depth-sensor and accelerated tremendously 3D vision systems development in robotics is studied in depth in [25], while comparison of market-available sensors is presented in [26, 27].

Chapter 3

RGB-D SENSORS

Available on the market RGB-D cameras, their principals of work and accuracy are discussed in this chapter. Calibration process of used for this project camera is described in details. Calibration is done via ROS, which significantly simplifies the process.

3.1 Market available RGB-D cameras

In November 2010, Kinect RGB-D sensor (Figure 3.1) was released by Microsoft as a new User Interface for XBOX 360 gaming platform. The Kinect provides color information as well as the estimated distance for each pixel. The unprecedented low cost of the sensor has dramatically raised interest to RGB-D sensors and their applications in such areas as Robotics, Natural User Interfaces, 3D mapping and others.

The Kinect sensor is based on a technology developed by PrimeSense company [28]. The core principle of depth estimation is emitting of known infrared (IR) speckle pattern (Figure 3.2). The IR sensor then captures light reflected from a scene and estimates per-pixel depth of the surface based on the disparity between the known IR pattern and received image. In other words, so called pseudo-stereo pair is formed. The depth data provided by the infrared sensor is then correlated to a RGB camera. This produces an RGB image with a depth associated with each pixel.

Figure 3.1: Microsoft Kinect sensor [29].

Since the appearance of Kinect, many more depth-cameras were introduced. One of them is Asus Xtion PRO LIVE [31] which was chosen for the purpose of this work. This sensor exploits the same principle of depth estimation. Comparison of technical parameters of Microsoft Kinect, Asus Xtion PRO LIVE, and several more market available cameras are presented in Table 3.1.

Although the resolution of Asus Xtion PRO LIVE IR sensor is 1280x1024 pixels at 30 Hz, the image is being downsampled to 640x480 in order to fit in the USB throughput. The nominal operation range of the camera lies between 0.8 m and 3.5 m which suit the purpose of our application.

The are other types of depth-cameras such as stereo cameras and Time-Of-Flight (TOF) sensors.

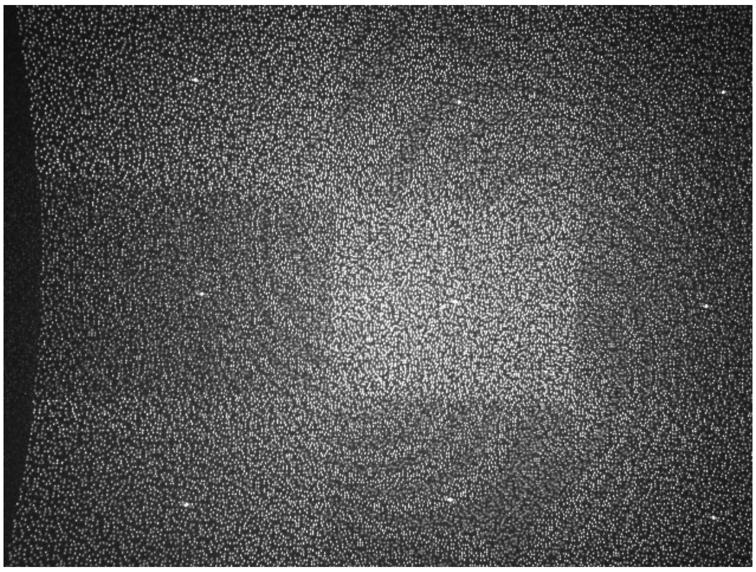


Figure 3.2: IR pattern emitted by Microsoft Kinect [30].

Table 3.1: Comparison of popular RGB-D cameras

	Asus Xtion PRO LIVE	**Microsoft Kinect**	**Orbbec Persee** [32]	**RealSense SR300** [33]
Released	July 2011	June 2011	December 2016	March 2016
Price	$300	$100	$240	$150
Tracking method	IR	IR	IR	IR
Range	0.8m – 3.5m	0.5m – 4.5m	0.4m – 8m	0.2m – 1.2m
RGB image	1280x1024	640×480, 30 FPS	1280×720, 30 FPS	1920×1080, 30 FPS
Depth image	640×480, 30 FPS	320×240, 30 FPS	640×480, 30 FPS	640×480, 60 FPS
Interface	USB 2.0	USB 2.0	Ethernet	USB 3.0

We will not discuss them here since the principal of their work is out of the scope of this research. For more information about these cameras, a reader can refer to [22].

3.2 Camera calibration

The calibration of RGB-D camera consists of two stages. Firstly, RGB camera is calibrated. Then depth camera calibration can be performed.

RGB camera calibration

Camera parameters include extrinsic, intrinsic and distortion coefficients. Extrinsic camera parameters map a point from 3D world coordinates to camera's 3D coordinate system while Intrinsic parameters map 3D coordinates in camera frame to the 2D image plane via projection. To estimate the camera parameters, we will need to have 3D world points and their corresponding 2D image points. We can obtain these correspondences through the use of multiple images of a calibration pattern. Usually, a checkerboard pattern is used. Using the correspondences, we will solve for the camera parameters. Luckily, as a part of *image_pipeline* ROS has a package called *camera_calibration* which uses the camera model proposed by Jean-Yves Bouguet [34] and allows for easy camera calibration. The model proposed by Jean-Yves Bouguet consist of:

– The pinhole camera model.

– Lens distortion.

Pinhole camera model is a model without any lens and with tiny aperture. Light rays pass through the aperture and form an inverted image on the opposite side of the camera Figure 3.3.

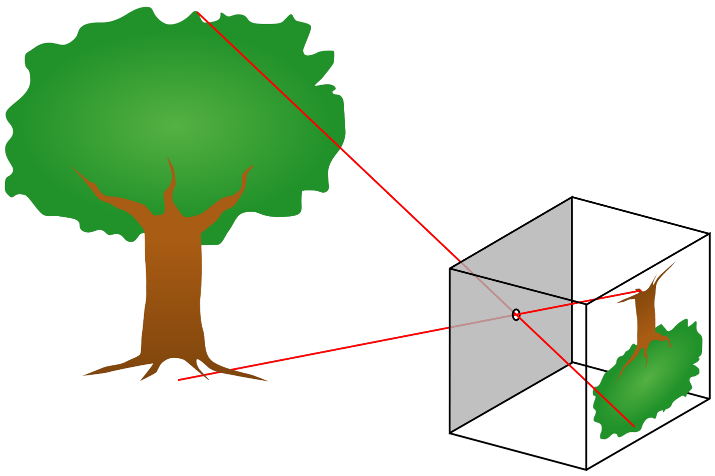


Figure 3.3: Pinhole camera model [35].

The pinhole model can be represented by 4-by-3 matrix called camera matrix, P, which maps the 3D world scene to the image plane:

$$w\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = P\begin{bmatrix} x_w \\ y_w \\ z_w \end{bmatrix}, \tag{3.1}$$

where w is a scale factor, $\begin{bmatrix} x & y & 1 \end{bmatrix}^T$ - coordinates of a point on the image plane, $\begin{bmatrix} x_w & y_w & z_w \end{bmatrix}^T$ - coordinates of a point in the world frame. As it was already mentioned camera parameters include intrinsic and extrinsic ones:

$$P = \begin{bmatrix} R & T \end{bmatrix} K, \tag{3.2}$$

where R is rotation, T is translation, their combination $\begin{bmatrix} R & T \end{bmatrix}$ is extrinsic and K is extrinsic camera parameters.

$$K = \begin{bmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}, \tag{3.3}$$

where f_x and f_y represent focal length in terms of pixels, $f_x = \frac{F}{p_x}$. F is focal length is world units (typically mm) and p_x is the size of a pixel in world units. c_x and c_y represent the principal point, which in ideal case would be at the center of the image. s is a skew coefficient, which is non-zero if the image axes are not perpendicular.

The pinhole camera model does not account for lens distortion due to the absence of a lens in an ideal pinhole camera. To precisely represent a real camera, the full camera model used by the algorithm includes two types of distortion: the radial and tangential lens distortion.

Radial distortion occurs when light rays bend more close to the edges of a lens than they do near its optical center. The smaller the lens, the greater the distortion. The radial distortion takes place when light rays bends more on the edges than at the center of the lens. Several types of radial distortion are distinguished (Figure 3.4).

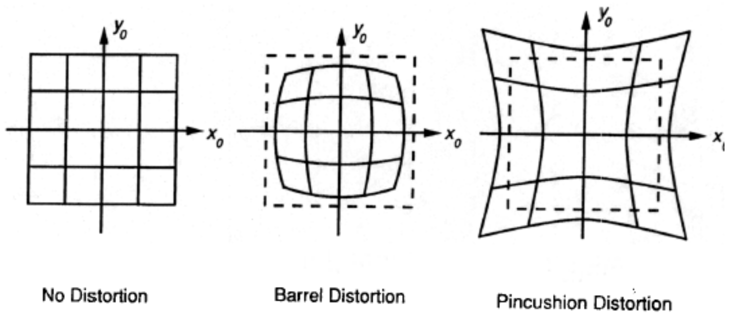


Figure 3.4: Radial distortion.

Radial distortion can be modeled with several coefficients:

$$\begin{aligned} x_{dist} &= x(1 + k_1 r^2 + k_2 r^4 + k_3 r^6), \\ y_{dist} &= y(1 + k_1 r^2 + k_2 r^4 + k_3 r^6), \end{aligned} \tag{3.4}$$

where (x_{dist}, y_{dist}) - distorted coordinates, (x, y) - undistorted pixel location, k_1, k_2, k_3 - radial distortion coefficients of the lens and $r^2 = x^2 + y^2$. Usually, two coefficients are sufficient for calibration. Tangential distortion occurs when the lens and the image plane are not parallel. Tangential distortion can be modeled with several coefficients:

$$\begin{aligned} x_{dist} &= x + [2p_1 xy + p_2(r^2 + 2x^2)], \\ y_{dist} &= y + [p_1(r^2 + 2y^2) + 2p_2 xy], \end{aligned} \tag{3.5}$$

where p_1 and p_2 — tangential distortion coefficients of the lens.

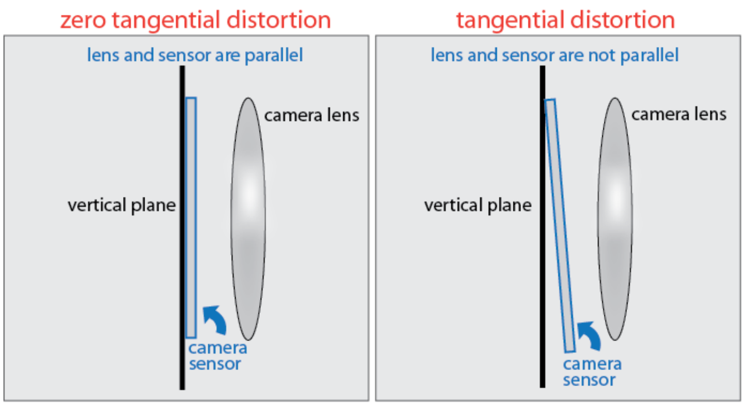


Figure 3.5: Tangential distortion [35].

Examples of camera calibration process using ROS *camera_calibration* package are presented on Figure 3.6.

As an output package provides derived values of all parameters discussed above (see Listing 3.1).

```
('D = ', [0.0754133, -0.174507, -0.004616, 0.007702, 0.0])
('K = ', [552.02, 0.0, 318.753939, 0.0, 554.605594, 228.936335, 0.0, 0.0, 1.0])
('R = ', [1.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 1.0])
('P = ', [554.479492, 0.0, 322.607826, 0.0, 0.0, 559.896606, 226.712239, 0.0, 0.0, 0.0, 1.0,
    0.0])
Width
640
Height
480
Camera matrix
552.020080 0.000000 318.753940
0.000000 554.605595 228.936336
0.000000 0.000000 1.000000
Distortion
0.075413 -0.174508 -0.004617 0.007703 0.000000
Rectification
1.000000 0.000000 0.000000
0.000000 1.000000 0.000000
0.000000 0.000000 1.000000
Projection
554.479492 0.000000 322.607826 0.000000
0.000000 559.896606 226.712239 0.000000
0.000000 0.000000 1.000000 0.000000
```

Listing 3.1: Camera calibration coefficients derived by *camera_calibration* package.

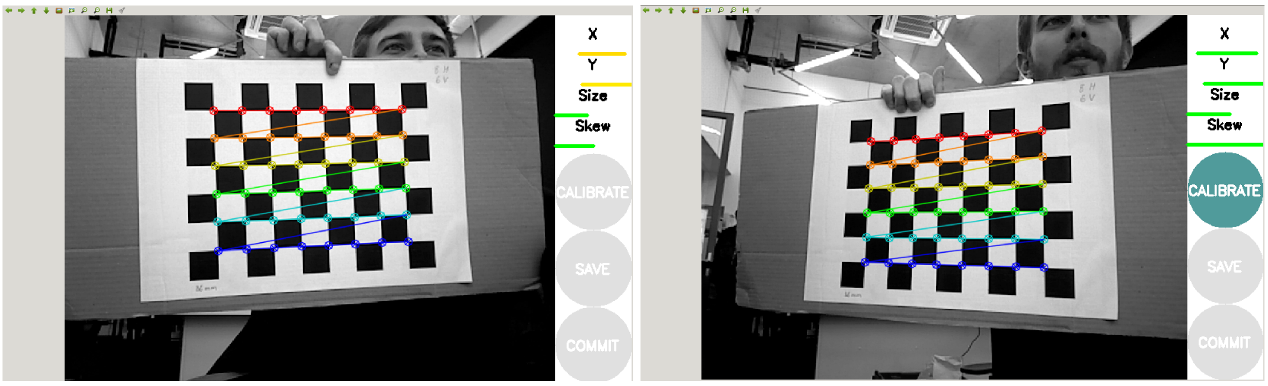


Figure 3.6: Camera calibration examples with ROS *camera_calibration* package.

Depth camera calibration

Once Intrinsic calibration has been performed we can start calibrating depth sensor. Depth camera process is based on the comparison of Pose (estimated by RGB camera while looking at checkerboard pattern) and uncalibrated Point Cloud. Then logistic regression is used to estimate an error. Calibration process is presented on Figure 3.7.

Figure 3.7: The process of depth camera calibration [36].

Chapter 4

PROPOSED APPROACH

4.1 Problem formulation

Let us first define the problem in a more formal way and introduce terms which will be frequently exploited throughout the further chapters.

In this work we will refer to a set of 3D points as a *point cloud* P. Point cloud represents a core data structure in 3D perception systems. It consist of individual points $p_i \in P$. Each individual point p_i contains coordinates $\{x_i, y_i, z_i\}$ measured from the fixed coordinate system, having its origin at the sensing device, used to acquire data. An example of point cloud dataset acquired with Asus Xtion PRO LIVE is presented on Figure 4.1. An important feature of a point cloud data representation is that each point p_i can contain additional information about point such as normal vector $\{n\_x_i, n\_y_i, n\_z_i\}$, principal curvatures $\{pc\_x_i, pc\_y_i, pc\_z_i\}$, RGB values $\{r_i, g_i, b_i\}$ and so on.

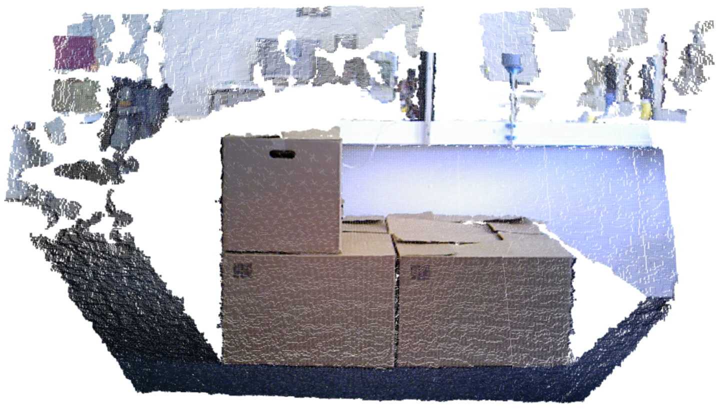


Figure 4.1: An example of point cloud acquired with Asus Xtion PRO LIVE.

The problem of identification of a specific object on an input image and finding its position and orientation relative to some coordinate system is known as *pose estimation*. The combination of position and orientation of the object on the image is refereed as *pose*.

In described above context, the aim of this work is to develop computer vision algorithm, receiving as an input a point cloud, containing a pallet with boxes in it, from measuring device and producing as an output estimated poses of all boxes lying on the pallet. We also assume that pose of the depth camera is known. In the real system sensor is mounted on an industrial manipulator. Using *tf* ROS package we can easily calculate transformations between all local coordinate systems of the robot and therefore derive camera pose [37]. Another input of the system, box dimensions, is coming

Table 4.1: Euro-pallet dimensions

Length	1200 mm
Width	800 mm
Hight	144 mm

from warehouse management system (WMS). The warehouse management system - software application that supports operations in a warehouse. WMS performs centralized management of tasks typical for a warehouse and contains different sorts of information including position of goods in a warehouse, their dimensions and so on.

As well as input data we use several assumptions in order to simplify the algorithm. First of all, we assume that the pallets are standard European pallets as specified by the European Pallet Association (EPAL). They have dimensions presented in the Table 4.1.

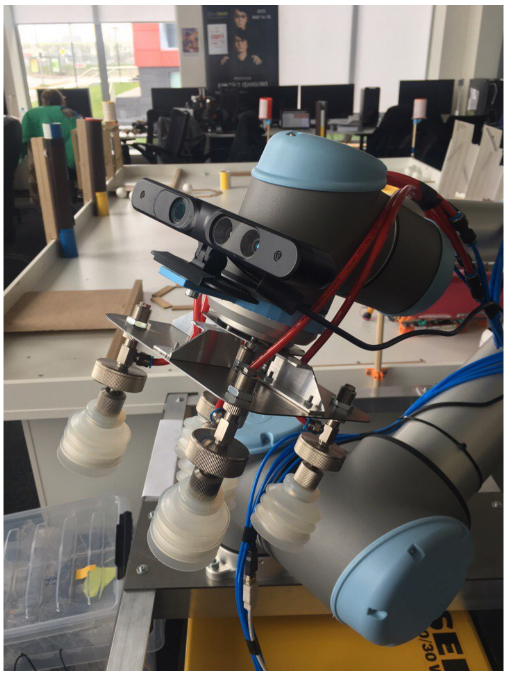


Figure 4.2: Depth camera mounted on industrial manipulator.

Secondly, we assume that thanks to a localization and navigation subsystems of the robot the preliminary layout of the scene is known. By saying that we mean that we expect the robot to arrive to the required pallet and maneuver in a way that the front side of the robot is parallel to the front side of boxes staying on the pallet (see Figure 4.3).

The additional constrain here is a camera field of view. In order to be able to capture the whole pallet full with boxes in one shot, the camera has to be not closer than a particular distance from a pallet. According to the specification (see Table 3.1) Asus Xtion field of view in the horizontal plane α is equal 58° while in the vertical plane β it is 45°. According to a standard (EUR-pallet)

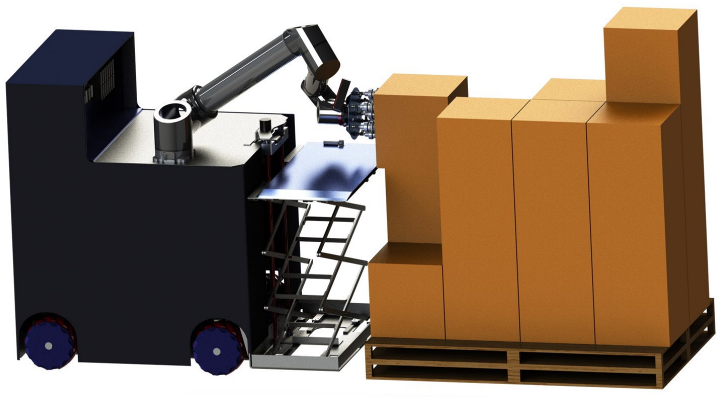


Figure 4.3: .

the maximum height of the pallet filled with boxes, h, is 1800 mm. Then:

$$\begin{aligned} l &= a\cos 61, \\ a &= \frac{l}{\cos 61} = 1277.6mm, \\ d_{horizontal} &= \sqrt[2]{a^2 - l^2} = 1128mm, \end{aligned} \tag{4.1}$$

where l is a length of the pallet, $d_{horizontal}$ is a minimum distance from a camera origin to the pallet in order to capture the whole pallet on the single image, a is a hypotenuse of a triangle formed with l and $d_{horizontal}$ (see Figure 4.4a).

$$\begin{aligned} h &= a\cos 67.6, \\ a &= \frac{h}{\cos 67.6} = 2351.8mm, \\ d_{vertical} &= \sqrt[2]{a^2 - h^2} = 2172.8mm, \end{aligned} \tag{4.2}$$

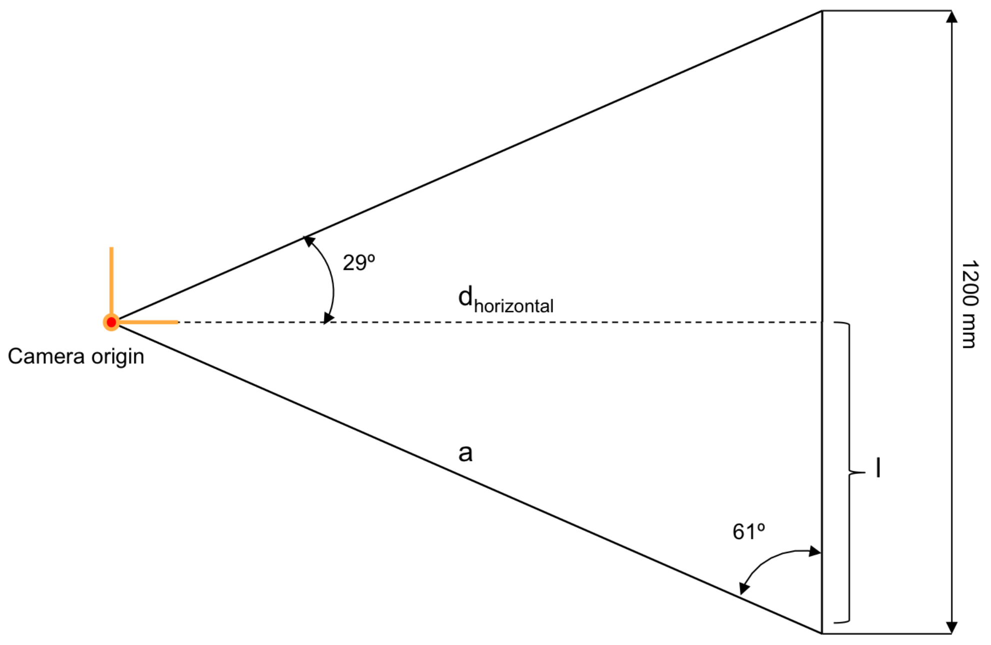
where $d_{vertical}$ is a minimum distance from a camera origin to the pallet in order to capture the whole pallet on the single image, a is a hypotenuse of a triangle formed with h and $d_{vertical}$ (see Figure 4.4b). Thereby, the minimum distance can be found as $max\{d_{horizontal}, d_{vertical}\}$ and equals to 2172.8 mm.

4.2 Pose estimation pipeline

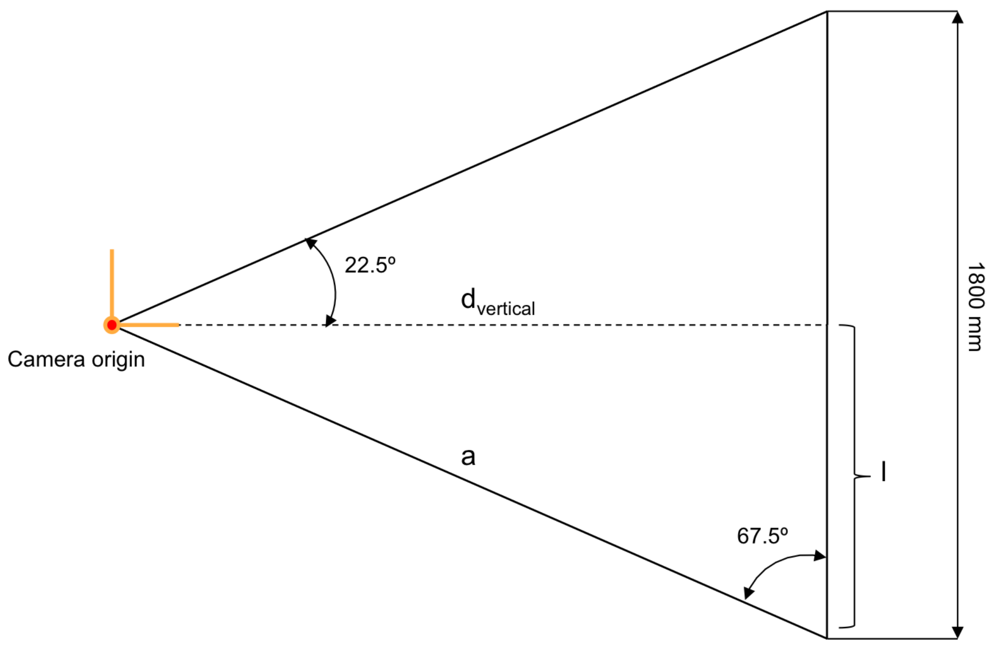
The pose estimation pipeline used in this work is presented on a Figure 4.5. Following sections describe all individual steps in details.

4.3 Affine transformation

Many of described further algorithms and processing steps exploit the geometric properties of the scene. Using information about camera position in 3D space, which can be derived from the manipulator position, we can align camera in a way, that $(X \| F, Y \| F, Z \perp F)$, where F denote a plane,



(a) Horizontal plane.



(b) Vertical plane.

Figure 4.4: Camera field of view and the size of the pallet.

aligned with a front side of boxes on a pallet. Such an alignment simplifies further steps and makes an analysis of the scene more intuitive. It is performed through the use of affine transformation (the

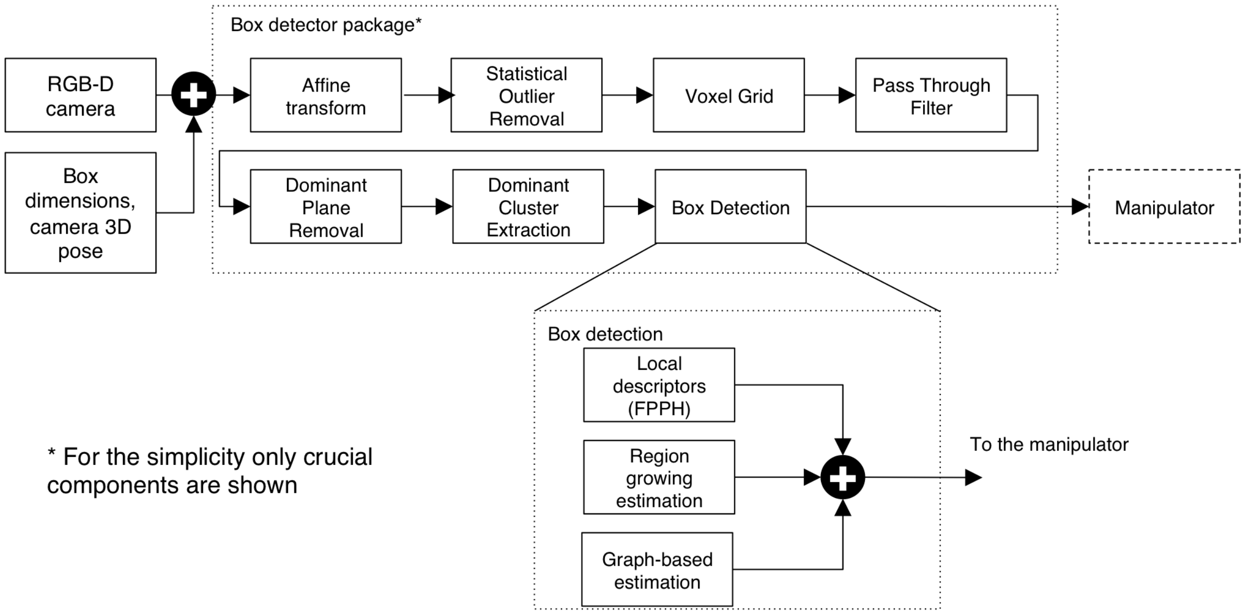


Figure 4.5: Pose estimation pipeline proposed in this work.

matrix shown bellow corresponds to the rotation around X axis):

$$T = \begin{bmatrix} 1 & 0 & 0 & x \\ 0 & \cos\theta & -\sin\theta & y \\ 0 & \sin\theta & \cos\theta & z \\ 0 & 0 & 0 & 1 \end{bmatrix}, \tag{4.3}$$

where θ is the angle of rotation, x, y, z - translation along the basis axes of the coordinate system. Point cloud Library uses *Eigen* C++ library for the linear algebra operations including Affine transformation.

4.4 Point cloud filtration

One of the most important parameters of any sensor is the level and type of noise it introduces. There are several widely used filtration techniques developed for point clouds. Some of them are aimed at reducing the amount of points in order to speed up computations. Others are used to discard outliers and obtain geometric layout better describing the real world. In this work, we exploit a simple and commonly used filtration pipeline (see Figure 4.6) which has been proven to be an effective combination of methods in several works [38]. It consists of three parts: Pass-through filtering, Statistical Outliers Removal, and Voxel Grid downsampling. Let us describe these three techniques in more details.

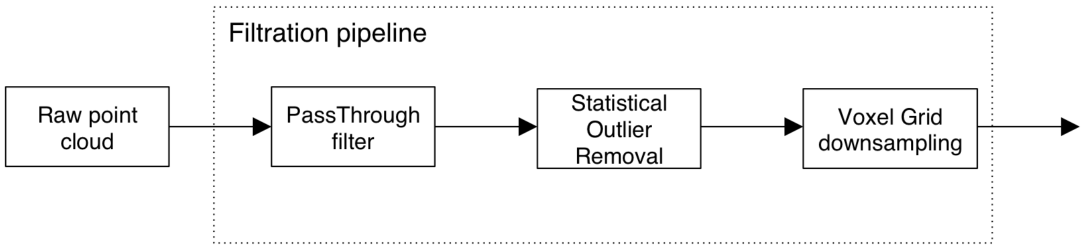


Figure 4.6: Filtration pipeline used in this work to filter out the noise and downsample an input point cloud.

PassThrough filter

PassThrough PCL class allows filtering along a specified dimension – that is, cuts off values that are either inside or outside of a given user range. The example of usage is presented in Listing 4.1 [19].

```
pcl::PassThrough<PointType> ptfilter (true); // Initializing with true will allow us to extract the removed indices
ptfilter.setInputCloud (cloud_in);
ptfilter.setFilterFieldName ("x");
ptfilter.setFilterLimits (0.0, 1000.0);
ptfilter.filter (*indices_x);
// The indices_x array indexes all points of cloud_in that have x between 0.0 and 1000.0
indices_rem = ptfilter.getRemovedIndices ();
// The indices_rem array indexes all points of cloud_in that have x smaller than 0.0 or larger than 1000.0
// and also indexes all non-finite points of cloud_in
```

Listing 4.1: Example of PCL *PassThrough* class usage.

Due to our assumption about the position of the pallet on the image, we can cut-off the big part of the image (keeping some margin) and thereby drastically speed up further computations. The example of application of *PassThrough* filter is shown on Figure 4.7. Left image presents an initial image, while the right one shows the point cloud after filtration.

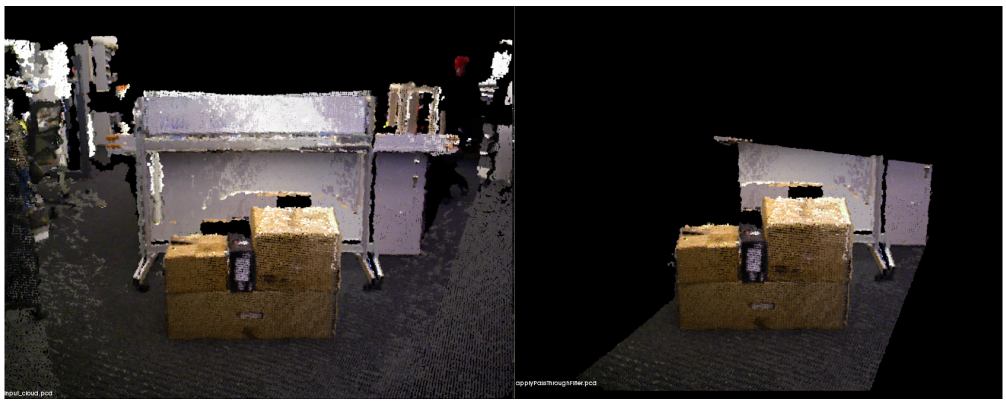


Figure 4.7: Example of *PassThrough* filter application.

Statistical Outlier Removal

The majority of the depth sensors generate point cloud datasets of with variations in density of points. Additionally, they introduce measurement errors that lead to the appearance of outliers. This complicates the estimation of local point cloud characteristics such as surface normals or curvature. This type of noise can be filtered by performing a statistical analysis on each point's neighborhood and discarding those points which do not meet certain criteria. *StatisticalOutlierRemoval* PCL class is based on the computation of the distribution of point to neighbors distances in the input dataset. For each point, it computes the mean distance from it to all its neighbors. By assuming that the resulted distribution is Gaussian with a mean and a standard deviation, all points whose mean distances are outside an interval defined by the global distances mean and standard deviation can be considered as outliers and filtered-out from the dataset [19].

The listing demonstrating the usage of *StatisticalOutlierRemoval* PCL class is presented in Listing 4.2 [19].

```
pcl::StatisticalOutlierRemoval<PointType> sorfilter (true); // Initializing with true will allow
    us to extract the removed indices
sorfilter.setInputCloud (cloud_in);
sorfilter.setMeanK (8);
sorfilter.setStddevMulThresh (1.0);
sorfilter.filter (*cloud_out);
// The resulting cloud_out contains all points of cloud_in that have an average distance to their
    8 nearest neighbors that is below the computed threshold
// Using a standard deviation multiplier of 1.0 and assuming the average distances are normally
    distributed there is a 84.1% chance that a point will be an inlier
indices_rem = sorfilter.getRemovedIndices ();
// The indices_rem array indexes all points of cloud_in that are outliers
```

Listing 4.2: Example of PCL *StatisticalOutlierRemoval* class usage.

Voxel grid downsampling

The *VoxelGrid* PCL class creates a 3D voxel grid (voxel grid can be imagined as a set of small 3D boxes in space) over the input point cloud data. After the grid is created each point of the initial point cloud is approximated (*i.e.* downsampled) with the centroid, calculated for the voxel it falls in. The example of *VoxelGrid* class usage is presented in Listing 4.3 [19]. Figure 4.8 shows the result of applying *Voxel Grid* filtering to the point cloud.

```
  // Create the filtering object
  pcl::VoxelGrid<PointType> sor;
  sor.setInputCloud (cloud_in);
  sor.setLeafSize (0.01f, 0.01f, 0.01f);
  sor.filter (*cloud_filtered);
```

Listing 4.3: Example of PCL *VoxelGrid* class usage.

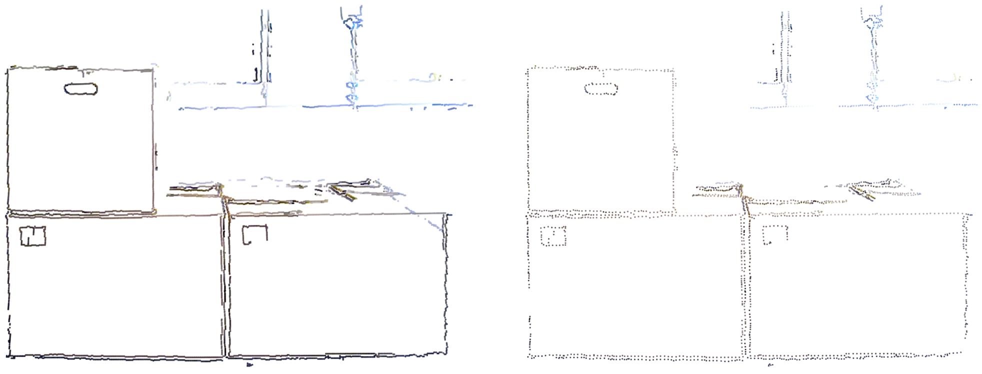


Figure 4.8: The result of point cloud *voxel grid downsampling*.

4.5 Floor plane removal and dominant cluster extraction

After performing all the filtering steps described above we are left with the point cloud containing mostly a region of our interest - pallet standing on the floor. As a next processing step, we must identify a floor plane and remove it from the image. The procedure is based on the assumption that the floor plane is parallel (with some angle tolerance) to the plane formed with camera local coordinate frame X and Z axes. It is guaranteed by Affine transformation performed previously. In order to identify all the points within a point cloud that support a plane model the plane *SACSegmentation* class can be used. In this work, we used the RANSAC method ($pcl :: SAC\_RANSAC$) as the robust estimator of choice. The decision was motivated by RANSAC's simplicity (other robust estimators use it as a base and add additional, more complicated concepts) [39]. The usage example of *SACSegmetation* is presented in Listing 4.4 [19]. In order to segment the floor, we identify all the vertical planes, compute the average Y coordinates and choose the one whose points have the biggest one (Y axis of the camera local frame is directed towards the floor). The identification of all the vertical planes requires some additional time but guarantees the reliable floor detection.

```
pcl::ModelCoefficients::Ptr coefficients (new pcl::ModelCoefficients);
pcl::PointIndices::Ptr inliers (new pcl::PointIndices);
// Create the segmentation object
pcl::SACSegmentation<PointType> seg;
seg.setOptimizeCoefficients (true); // Enables model coefficient refinement
seg.setModelType (pcl::SACMODEL_PLANE) ; // Model type
seg.setMethodType (pcl::SAC_RANSAC); // Method type
seg.setDistanceThreshold (0.01); // Sets the threshold of the distance from query point to the
  estimated model in order to be considered as an inlier

seg.setInputCloud (cloud);
seg.segment (*inliers, *coefficients);
```

Listing 4.4: Example of PCL *SACSegmetation* class usage.

Even after all described preprocessing steps there is a possibility, that some unnecessary objects (*e.g.* part of the shelves or other pallets) might be still present in a point cloud. Since they can prevent correct recognition of boxes some additional step needs to be applied in order to remove them. Assuming that the pallet is a dominant cluster on the image we apply PCL *EuclideanClusterExtraction* class in order to extract it. *Euclidean cluster extraction* is implemented by making use of a 3D grid subdivision of the space using fixed width boxes. This representation is very fast to build. The algorithm of Euclidean cluster extraction used in PCL library is presented in Algorithm 1.

Algorithm 1 Euclidean clustering

```
Require: $P, p_i \in P, i \in (0, .., n)$
 1: C (list of clusters) $\leftarrow 0$
 2: Kd-tree $\leftarrow 0$
 3: Q (list of points to be checked) $\leftarrow 0$
 4: for $i \leftarrow 0$ to $n$ do
 5:     Q $\leftarrow p_i$
 6:     for every $p$ in $Q$ do
 7:         search for the set $P^i_k$ of point neighbors of $p_i$ in a sphere with radius $r < d_{th}$ // $d_{th}$ is a user-defined threshold
 8:         for every neighbor $p^i_k \in P^i_k$, check if the point has already been processed, and if not add it to $Q$
 9:     end for
10:     when the list of all points in $Q$ has been processed, add $Q$ to the list of clusters $C$, and reset $Q$ to an empty list
11: end for
12: the algorithm terminates when all points $p_i \in P$ have been processed and are now part of the list of point clusters $C$
13: return C
```

The usage example of *EuclideanClusterExtraction* is presented in Listing 4.5 [19]. The result of the floor removal and dominant cluster extraction is shown on the Figure 4.9.

```
pcl::EuclideanClusterExtraction<PointType> ec;
ec.setClusterTolerance (0.02); // 2cm
ec.setMinClusterSize (100);
ec.setMaxClusterSize (25000);
ec.setSearchMethod (tree);
ec.setInputCloud (cloud);
ec.extract (cluster_indices);
```

Listing 4.5: Example of PCL *EuclideanClusterExtraction* class usage.

4.6 3D pose estimation

The following sections contain a detailed description of all approaches to 3D pose estimation which were studied in the scope of this work:

- Region growing based.
- Graph-based.

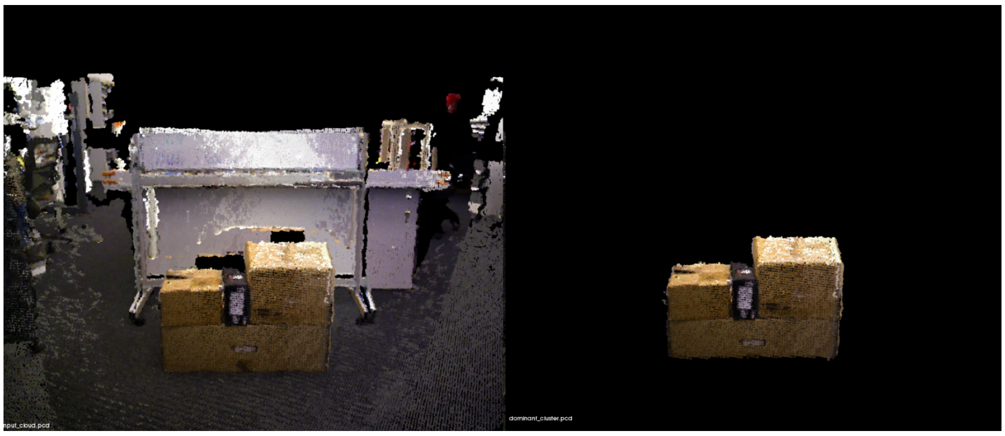


Figure 4.9: The result of floor plane removal and dominant cluster extraction.

– Local point descriptors-based.

Region growing-based approach

The high level of occlusion is typical for the images used as a data set for this work since boxes usually stand side by side and hardly can be distinguished from each other. Due to this reason, there are images where we partially or fully miss any geometric information about layout and basically have only a front plane of aligned boxes (see Figure 4.10) available for analysis.

Taken described above problem into consideration we have developed an approach which can work with such input data. The pseudocode algorithm is described in Algorithm 2.

Algorithm 2 Region growing-based pose estimation algorithm

```
Require: $P, p_i \in P, i \in (0, .., n)$
 1: R (list of estimated poses) $\leftarrow 0$
 2: C (list of clusters) $\leftarrow 0$
 3: V (list of vertical planes) $\leftarrow 0$
 4: Extract all vertical planes, $V \leftarrow SACSegmentation$
 5: for each vertical plane in $V$ do
 6:     Extract clusters, $C \leftarrow RegionGrowingRGB$
 7:     for each cluster in $C$ do
 8:         Apply StatisticalOutlierRemoval
 9:         Compute ConcaveHull
10:         Fit bounding rectangle
11:         Estimate pose, R
12:     end for
13: end for
14: return R
```

As a first step of the described above algorithm, we find all vertical planes in the scene parallel with some tolerance to the plane formed by X and Y axis of the camera frame, assuming that this gives us front planes of the visible boxes on the pallet. The need to detect all vertical planes arises from

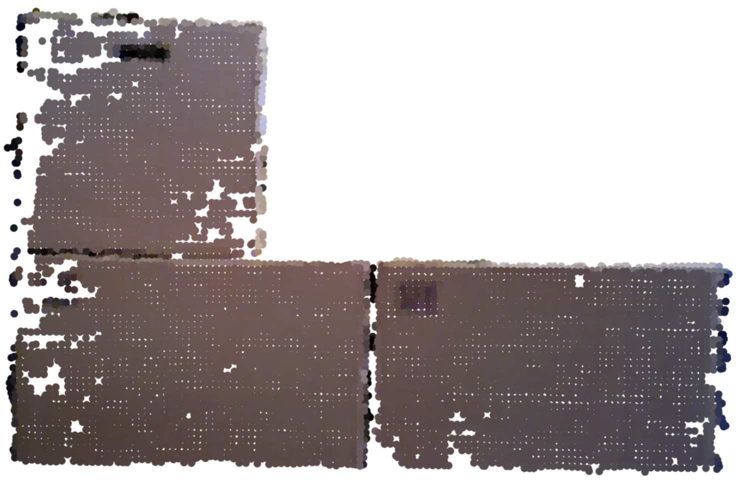


Figure 4.10: The result of floor plane removal and dominant cluster extraction.

the fact, that boxes might be not aligned to the side we are imaging from and therefore there might be multiple planes with front sides of the boxes.

Then, *RegionGrowingRGB* PCL class is used in order to extract separate clusters. *RegionGrowingRGB* class implements the Region Growing algorithm used for segmentation based on color of points [40]. The RGB data allows us to reliably distinguish clusters in the absence of the geometric information. The usage of the *RegionGrowingRGB* with the exact parameters used in this work is shown in Listing 4.6 [19].

```
pcl::RegionGrowingRGB<pcl::PointXYZRGB> reg;
reg.setInputCloud (cloud_plane);
reg.setSearchMethod (tree);
reg.setDistanceThreshold (0.1); // Distance between two points to consider them to be neighbours
reg.setPointColorThreshold (3); // Threshold to decide that two points belongs to one cluster
reg.setRegionColorThreshold (5); // Threshold to decide that two clusters are different
reg.setMinClusterSize (600);
```

Listing 4.6: Example of PCL *RegionGrowingRGB* class usage.

As a next step *Statistical Outlier removal* is applied again in order to filter errors related to cluster segmentation. After that, a *2D Concave Hull* is calculated for each cluster. It is important to note here, that the *Concave Hull* has shown better results in approximation boxes in comparison with *Convex Hull* and therefore was chosen for the purpose of this work.

Knowing the *Concave Hull* for each cluster we fit a bounding rectangle into it and compare with the known size of the box to be recognized (see Figure 4.11). It allows us to discard clusters which do not correspond to the known dimensions and relaunch Region Growing algorithms with less strict *RegionColorThreshold* which specifies the threshold value for the color test between the regions. If the difference between segments color is less than the threshold value, then they are merged together.

Some results of box detection using the described approach are shown on Figure 4.12. Green dots show the found center of the front plane of each box.

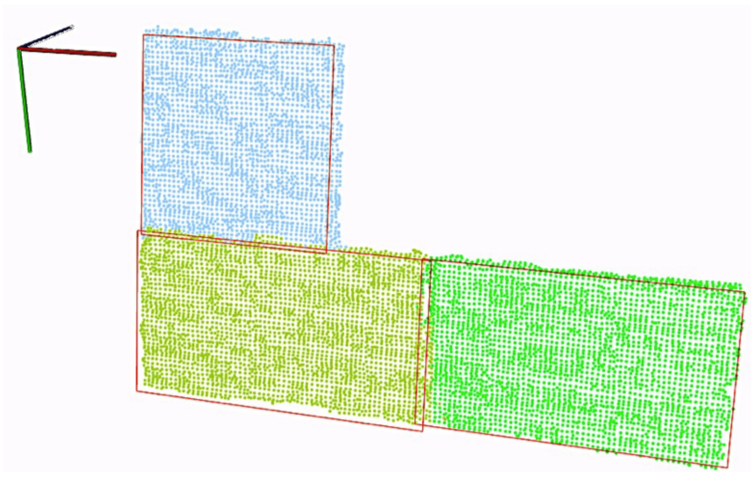


Figure 4.11: Results of *RegionGrowingRGB* clustering and bounding box estimation. Clusters assigned to recognized boxes are shown in different colors. Bounding boxes are shown in red.

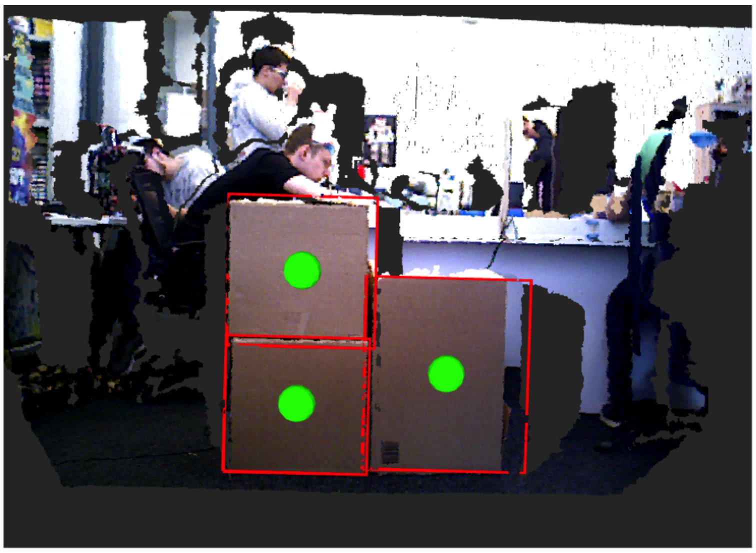


Figure 4.12: Region growing-based pose estimation result. Red lines show the estimated edges of boxes' front surfaces. Green dots denote the estimated centers of the front surfaces.

Although described in this section approach is very simple and intuitive it is very fast, provides good recognition quality and relatively easy to implement. But obviously, it has serious pitfalls. Thus, since it is based on RGB data, lighting conditions and all other problems common for 2D images have a big impact on recognition quality.

Graph-based approach

Another approach is based on the widely used in computer vision operation - edge detection. The general scheme of the *Graph-based approach* is presented on Figure 4.13.

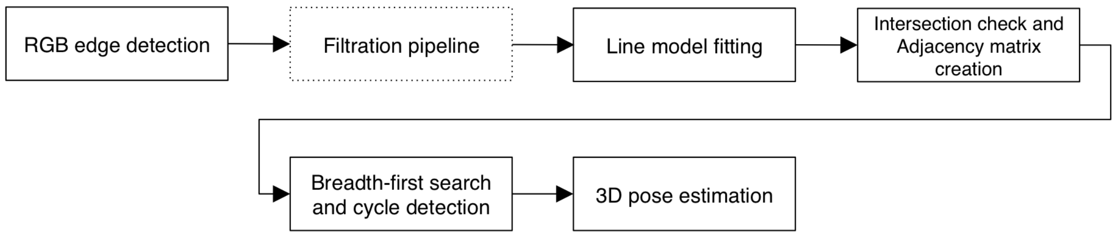

RGB edge detection
Filtration pipeline
Line model fitting
Intersection check and Adjacency matrix creation
Breadth-first search and cycle detection
3D pose estimation

Figure 4.13: Graph-based algorithm scheme.

PCL library provides *OrganizedEdgeFromRGBNormals* class to extract edges from *organized* point cloud. This class combines several approaches and allows a user to choose which edges he/she wants to detect. Currently, there are following options implemented:

- Occluding boundary edges (edge of occluding object).
- Occluded boundary edges (edge on surface being occluded).
- High curvature edge.
- 2D edge (image only edge extracted from RGB).
- Combinations of the above mentioned options.

The results of extractions of all types of edges can be seen on Figure 4.14.

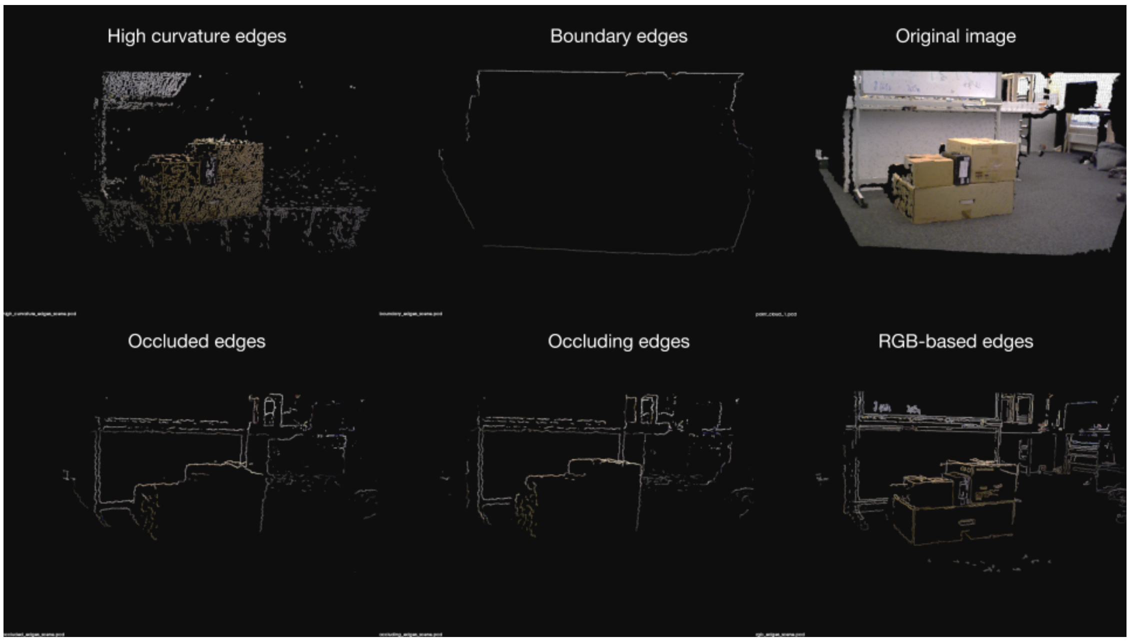


Figure 4.14: Detection of different edges types performed with *OrganizedEdgeFromRGBNormals* PCL class.

The big drawback of *OrganizedEdgeFromRGBNormals* class is that it requires *organized point cloud* as an input (at the time of writing this thesis there is no other option for edge detection implemented in PCL). A point cloud is called *organized* if it resembles an organized image (or matrix) like structure, where the data is split into rows and columns (as on typical 2D image). The advantages of an organized dataset is that by knowing the relationship between adjacent points (*e.g.* pixels), nearest neighbor operations are much more efficient, thus speeding up the computation and lowering the costs of certain algorithms in PCL. Application of any operation (*e.g.* filtering or geometric transformation) to the raw point cloud, coming from the sensor, erode its structure and the point cloud loses its organized quality. That is why edge detection should be applied prior any other operations as a first processing step. The described previously filtration pipeline is then applied to the extracted edges. The floor detection and dominant cluster extraction are not necessary for this algorithm.

As a next step we perform line model fitting into extracted edges with the constrain that found models are required to be parallel to the plane formed by X and Y axes of the camera frame. This allows us to partially discard the unfiltered edges which are parts of the object which are not in our interest (*e.g.* shelves, pallet). This is done with *SACSegmentation* class which was discussed in the **Region growing-based** approach section. To extract the lines parallel to the specific plane *pcl::SampleConsensusModelParallelLine<PointType>* model is used.

Having all line models we can find intersections between each pair of lines and create a graph in which vertices will represent lines and edge between any of two vertices will mean intersection of this lines in the 3D space. Intersection is checked with *pcl::lineWithLineIntersection()* function implemented in PCL. In Figure 4.15 found lines are shown. Intersections are shown as red spheres.

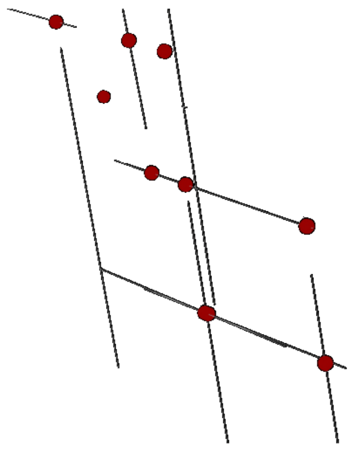


Figure 4.15: Fitted into edges line models.

Using an *adjacency matrix* representation we then search for cycles of four in the created graph using *Breadth-first search* algorithm. The general algorithm is presented in 3.

Algorithm 3 Breadth-first search

```
Require: $A$, adjacency matrix, root
 1: $Q$, empty queue $\leftarrow 0$
 2: $S$, empty set $\leftarrow 0$
 3: $C$, (current node) $\leftarrow 0$
 4: Add root to$S$, $S \leftarrow root$
 5: while $Q$ is not empty do
 6:     $C \leftarrow Q.top()$
 7:     if  then$C$ is a goal
 8:         return $C$
 9:     end if
10:     for each node $n$ that is adjacent to current do
11:         if  then$n \notin S$
12:             $S \leftarrow n$
13:             $n.parent \leftarrow currentnode$
14:             $Q \leftarrow n$
15:         end if
16:     end for
17: end while
```

As the last step of graph-based pose estimation approach, the check of all found cycles is performed against a known area of the box surface and expected orientation in space. Pose is computed using an information about the front surface pose and predefined box dimensions.

Although this method showed a good precision in identification poses of boxes, it gives a moderate precision and recall in terms of recognition of boxes in the scene. This is due to the problem that even small errors in line model fitting lead to the missing or false intersections between edges.

Local point descriptors-based approach.

The recognition approach based on the calculation of the point descriptors is the most commonly used technique to find a known object in the scene. There are two types of descriptors:

- **Local descriptors** are computed for each individual points in a point cloud. They have no notion of what an object is. All they do is just describe how the local geometry looks like around that point.

- **Global descriptors** encode the geometry of the whole object. They are not computed for individual points, but for a whole cluster that represents an object.

Many descriptors have been proposed for 3D data processing. An overview of the most widely used ones is presented in Table 4.2.

Table 4.2: 3D point descriptors comparison

Name	**Type**	**Size**
PFH (Point Feature Histogram)	Local	125
FPFH (Fast Point Feature Histogram)	Local	33
RSD (Radius-Based Surface Descriptor)	Local	289
3DSC (3D Shape Context)	Local	1980
USC (Unique Shape Context)	Local	1960
SHOT (Signatures of Histograms of Orientations)	Local	352
Spin image	Local	153*
RIFT (Rotation-Invariant Feature Transform)	Local	32*
NARF (Normal Aligned Radial Feature)	Local	36
RoPS (Rotational Projection Statistics)	Local	135*
VFH (Viewpoint Feature Histogram)	Global	308
CVFH (Clustered Viewpoint Feature Histogram)	Global	308
OUR-CVFH (Oriented, Unique and Repeatable Clustered Viewpoint Feature Histogram)	Global	308
ESF (Ensemble of Shape Functions)	Global	640
GFPFH (Global Fast Point Feature Histogram)	Global	16
GRSD (Global Radius-Based Surface Descriptor)	Global	21

Values marked with an asterisk (*) indicate that the descriptor's size depends on some parameters, and the one given is for the default values.

The local descriptors were used in this work since they describe the surface and suit better for our application. Let us take a look at the typical local recognition pipeline (see Figure 4.16) [41].

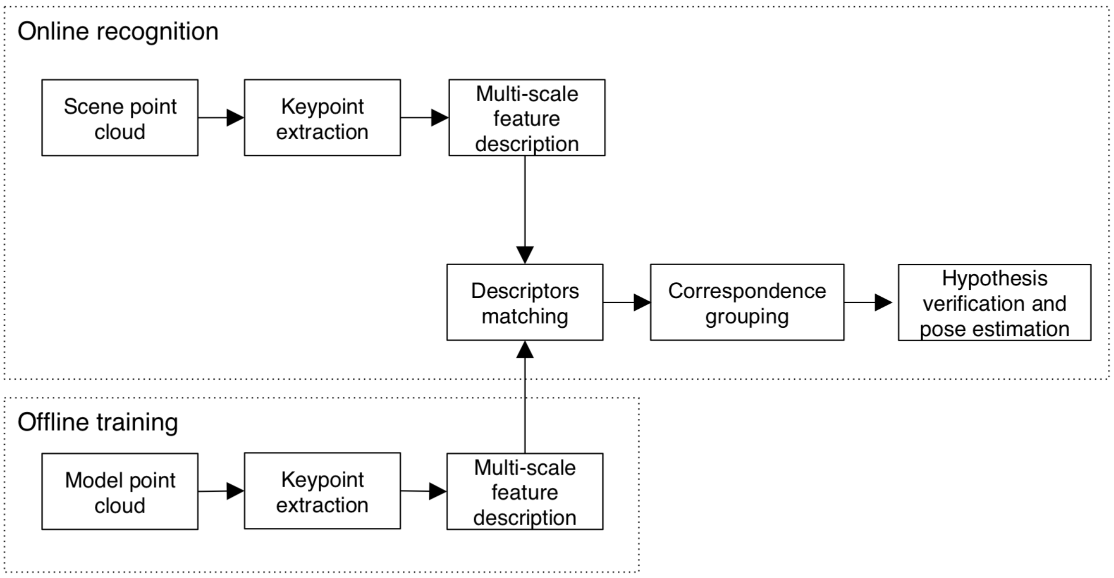


Figure 4.16: Commonly used local recognition pipeline.

First of all, a training of the system is performed. Training means, in this case, creating a database with all the objects we need to be able to recognize, and calculating the descriptors for them. Only after that, we can go through the recognition pipeline. In our case, when the warehouse automation robot receives a new task from a WMS it also receives the dimensions of the box to be picked. With this data, we generate a model of the box (see Figure 4.17) and compute local descriptors in order to find correspondences with the scene descriptors in the following processing steps.

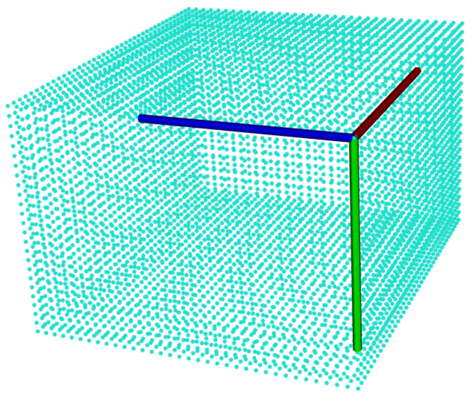


Figure 4.17: Generated box model.

There are some postprocessing steps that are not mandatory to perform but will yield better results if done, like pose refinement and hypothesis verification. The following sections describe in more details each step of the pipeline revealing some details about our particular implementation.

Keypoint extraction

Since computing descriptors for each point of the point cloud is a really costly operation that can slow down the whole recognition system tremendously it makes sense to perform descriptors computation only for a particular set of points. Such points are called keypoints. According to the [42] a good 3D keypoint:

– **Repeatable** with respect to point-of-view variations, noise, *etc.* (locally definable)

– **Distinctive**, *i.e.* suitable for effective description and matching (globally definable).

Because the second one depends on the local descriptor being used (and how the feature is computed), a different keypoint detection technique would have to be implemented for each. Another simple alternative is to perform downsampling on the cloud and use all remaining points. Intrinsic Shape Signatures (ISS) keypoint detector was used in this work. According to [43] it provides perfect results for recognition and pose estimation of objects of simple geometry. ISS keypoint detector scans the surfaces and chooses only points with large variations in the principal direction (*i.e.* the shape of the surface), which is ideal for keypoints. An example of detection is presented on Figure 4.18. Keypoints are colored in green.

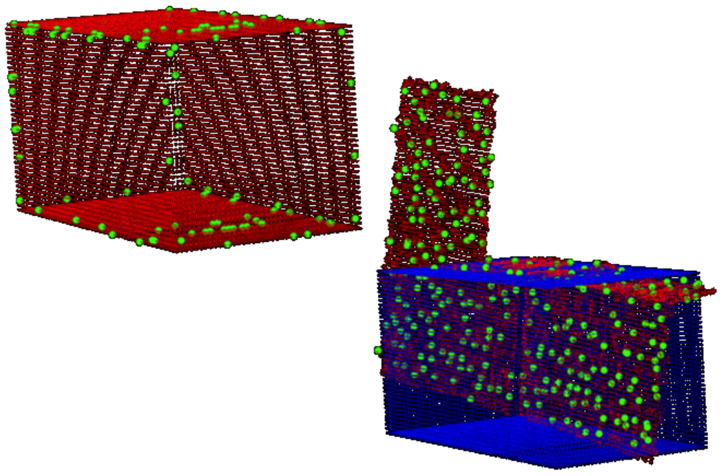


Figure 4.18: Result of keypoints detection and preliminary model alignment.

Computing descriptors

As it was previously discussed there is a big variety of descriptors available. For the purpose of this research, Fast Point Feature Histogram (FPFH) descriptor was chosen, since it provides comparably good recognition quality, fast to compute and requires a small amount of memory [44].

In order to deal with FPFH, we first need to understand how Point Feature Histogram (PFH) works since FPFH originates from it [45]. The core idea is quite simple. The PFH captures information about the geometry surrounding the point by analyzing the difference between the directions of the normals in the point neighborhood.It is important to note, that imprecise normals estimation might lead to the low-quality of descriptors estimation.

As a first step, the algorithm pairs all points in the vicinity (Figure 4.19). It uses not just the chosen keypoint with its neighbors, but also the neighbors with themselves. Then, a fixed coordinate frame is computed from their normals, for each pair. Having this frame, the difference between the normals is encoded with 3 angular variables. These variables, together with the euclidean distance between the points, are stored, and then binned to a histogram when all pairs have been computed. The final descriptor is the concatenation of the histograms of each variable (4 in total) (Figure 4.20).

PFH gives accurate results, but it has a serious drawback: it is too computationally expensive to be computed at the real time. For a cloud of n keypoints with k neighbors considered, it has a complexity of $O(nk^2)$. In order to eliminate this problem, FPFH was created. The FPFH considers only the direct connections between the current keypoint and its neighbors, not taking into account

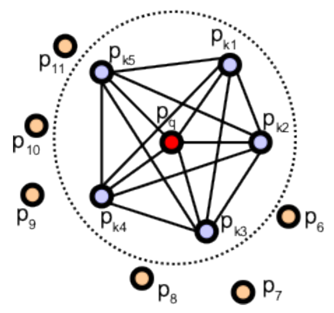
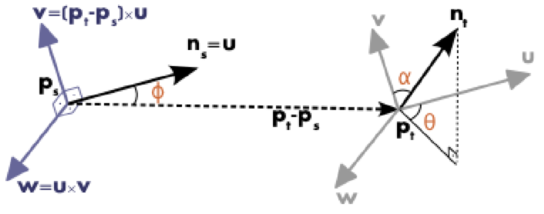


Figure 4.19: Point pairs produced while computing the PFH for a point.



$$u = n_i \qquad v = u \times \frac{(p_j - p_i)}{\|p_j - p_i\|_2} \qquad w = u \times v$$

$$\alpha = v \cdot n_j \qquad \phi = u \cdot \frac{(p_j - p_i)}{d} \qquad \theta = \arctan\left(w \cdot n_j, u \cdot n_j\right)$$

Figure 4.20: Fixed coordinate frame and angular features computed for one of the pairs.

additional links between neighbors (Figure 4.21). This decreases the complexity down to $O(nk)$. Because of this, the resulting histogram is referred to as Simplified Point Feature Histogram (SPFH). The reference frame and the angular variables are computed in the same manner as for PFH.

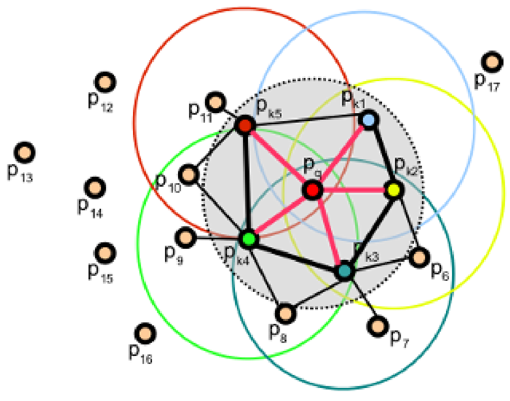


Figure 4.21: Point pairs established while computing the FPFH for a point.

To account for the lost connections, an extra step is introduced after computation of all histograms: the SPFHs of a point's neighbors are concatenated with its own, weighted according to the distance. This has the effect of giving a point surface information of points as far away as 2 times the radius used. As the last step, the 3 histograms (distance is not used) are merged to produce the final descriptor [46].

Descriptors matching

After the local descriptor has been computed for all keypoints in the cloud, the next step is to match them, finding correspondences between the scene and the object stored in our database (generated model of the box in our case). To do this, a search structure like a $k - d$ tree can be exploited to perform a nearest neighbor search, retrieving the Euclidean distances between descriptors. Optionally, a maximum distance value can be set as a threshold to consider two descriptors as corresponding ones. Every descriptor in the scene should be matched against the descriptors of the model. In general, if there are different objects to be recognized (different objects stored in objects data) every descriptor in the scene should be matched against the descriptors of all objects in order to account for the presence of multiple objects in the scene. We assume that in our case boxes of only one type are stored on one pallet, that is usually the case for the typical warehouse.

Correspondence grouping

At this point, we have a list of correspondences between keypoints in the scene and keypoints from the object (box). This does not necessarily mean that a given object is present in the scene. We need an additional check to discard correspondences that are not geometrically consistent. This check can be implemented with an additional step called correspondence grouping. It groups correspondences that are geometrically consistent (for the given object model) in clusters and discards the ones that do not. Rotations and translations are permitted, but anything other than that will not meet the criteria. Also, as a minimum of 3 correspondences is required for retrieving the 6 degrees of freedom (DoF) pose, groups with less than that can be ignored.
Several classes to perform correspondence grouping are offered by PCL. The simple one: *pcl::GeometricConsistencyGrouping* was chosen for the purpose of this work since our model (box) is geometrically simple. This class iterates over all feature correspondences which are not yet grouped, and adds them to the current subset if they are geometrically consistent. For every subset is also computes the transformation (rotation and translation), making the next step (pose estimation) unnecessary.

Pose estimation

Despite the fact, that we can already obtain pose estimate at the previous step we need another step to verify the results (remove correspondences that are not compatible with the same 6 DoF pose). Methods like RANSAC can be used, to get the rotation and translation of the rigid transformation that best fits the correspondences of a model instance. PCL has some classes for this. The one we are using here adds a prerejection step to the RANSAC loop, in order to discard hypotheses that are probably wrong. For that, the algorithm forms polygons with the points of the input subsets and then checks pose-invariant geometrical constraints. In other words, it makes sure that the polygon edges are of similar length. The number of iterations can be specified or a precision threshold can be set. The *pcl::SampleConsensusPrerejective* class is used in this research. It aligns model to the scene and returns its pose. In the form described in **Affine transformation** section of this chapter. Due to the complex scene layout in or testing data (high level of occlusion, lack of geometrical information), this method shows good results only on some images and in general loses to other methods described in this chapter (see **Implementation and experimental results** chapter).

Chapter 5

SYSTEM IMPLEMENTATION

This chapter discusses practical aspects of the implementation of computer vision algorithm developed in this work. This includes a software packages architecture, integration with ROS and other subsystems of the warehouse automation robot. Industrial manipulator (Universal Robot UR-10 in our case) control and its integration with the computer vision subsystem is discussed as well in this chapter.

5.1 Implementation of proposed methods

As it was already mentioned a PCL library was used for the development of algorithms, proposed in this work. The Point Cloud Library is an open-source library of algorithms for the tasks, related to the processing of point clouds and 3D geometry processing. The library contains algorithms for feature estimation, registration, model fitting, surface reconstruction, and segmentation. PCL is written in C++ and released under the BSD license. The algorithms implemented in PCL have been widely used in such areas as robotics perception, architecture, 3D scanning, and others.
The main practical result of this work is developed ***box_detector_pkg*** software package, which implements all described in the previous chapter algorithms as well as additional methods for point cloud processing and processing parameters tuning.

5.2 Integrations with ROS

Robot Operating System (ROS) is a collection of software packages, frameworks and standalone programs for robot software development. ROS provides standard operating system services such as hardware abstraction, low-level device control, implementation of commonly used functionality, message-passing between processes, and package management. Running sets of ROS-based processes are represented in a graph architecture where processing takes place in nodes that may receive, post and multiplex messages: sensor, control, state, planning, actuator and other types.
All parts of ROS software can be divided into 3 groups:

– Language-and platform-independent tools used for building and distributing ROS-based software.

– ROS client library implementations (*roscpp*, *rospy*, and *roslisp*).

– Packages containing application-related code which uses one or more ROS client libraries.

All software included in ROS is released under the BSD license, and as such are open source software and free for both commercial and research use.

Three layered architecture was used in warehouse automation robot design [47]. According to this approach, there are three layers of the robot architecture distinguished:

- Tasks - high-level goals and objectives that robot tries to achieve.
- Actions - atomic simple actions performed by the robot in order to complete tasks set for a robot.
- Actuators/Sensors - set of tools robot uses in order to perform useful actions and achieve its goals.

The simplified architecture developed for our particular robot is presented on Figure 5.1.

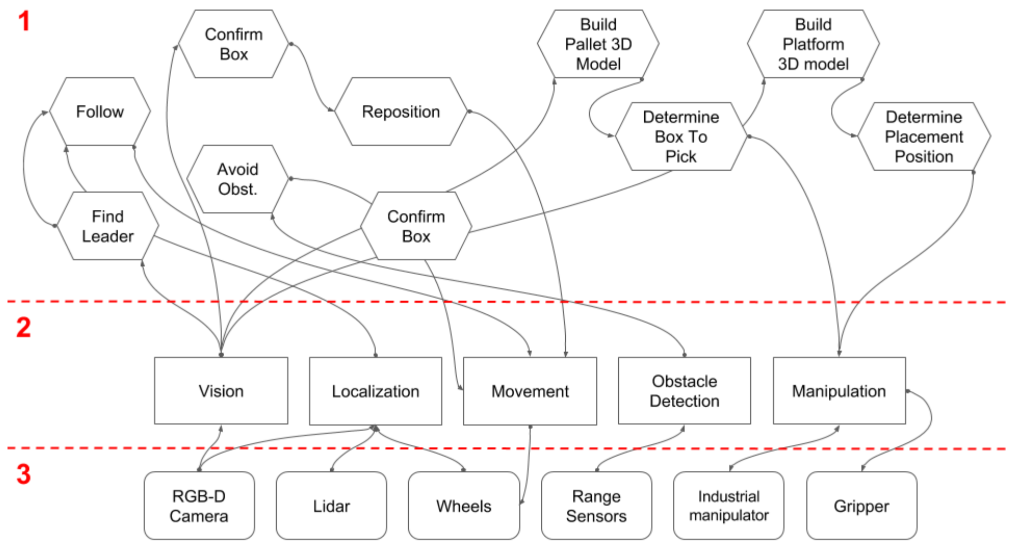


Figure 5.1: Three layered architecture developed for automation warehouse robot.

ROS allows an easy mapping of such architecture to the actual software. Further, we will discuss how computer vision subsystem developed in this work is implemented in ROS.

To communicate with the camera *openni2_launch* ROS package was used. This package contains launch files for using OpenNI-compliant devices in ROS. It supports the Asus Xtion, Xtion Pro, multiple version of the Primesense cameras. *OpenNI* stands for *Open Natural Interaction*. It is an open source software project focused on certifying and improving interoperability of natural user interfaces and organic user interfaces for Natural Interaction (NI) devices, applications that use those devices and middleware that facilitates access and use of such devices [48]. *Openni2_launch* launches an ROS-node which takes a row data from the sensor, does preprocessing and publishes the data to specific topics from where any subscriber can read it. Since the depth and color images come from two separate, slightly offset, cameras, they do not perfectly overlap. Therefore, for each pixel in the depth image, its position in 3D space has to be calculated and reprojected into the image plane of the RGB camera. This date is then published to "$/camera/depth\_registered/points$" topic which was used in this work.

Developed *box_detector_pkg* subscribes to discussed above "$/camera/depth_registered/points$" topic in order to receive the latest images captured by the sensor(figure form rqt graph). Since we do not need to continuously process images but do it only when picking is to be performed there should be a specific mechanism of launching the processing. Luckily ROS provides such capabilities through the use of ROS *actionlib* package which provides tools to create servers that execute long-running goals that can be preempted. It also provides a client interface in order to send requests to the server. In our case *box_detection* action is called each time when robot arrives to the pallet and needs to perform picking.

5.3 Integration with industrial manipulator

UR-10 industrial manipulator from Universal Robots was incorporated in the design of warehouse automation robot [49]. ROS-Industrial - an open-source project that extends the advanced capabilities of ROS software to industrial applications(*i.e.* manufacturing) - provides an easy to use interface to simulation and programming of UR-10. The *universal_robot* package contains packages that provide nodes for communication with Universal's industrial robot controllers, Unified Robot Description Format (URDF) models for various robot arms and associated *MoveIt* packages for manipulator control and kinematics.

The *ur_control_node* was developed in this work. It subscribes to the topic (published by *box_detector_package*) containing 3D pose of the detected boxes and sends commands to perform picking to the arm. The *ur_control_node* can control the real arm, as well as a manipulator, simulated in Gazebo [50], which simplifies the development process. The simulation process is shown on Figure 5.2.

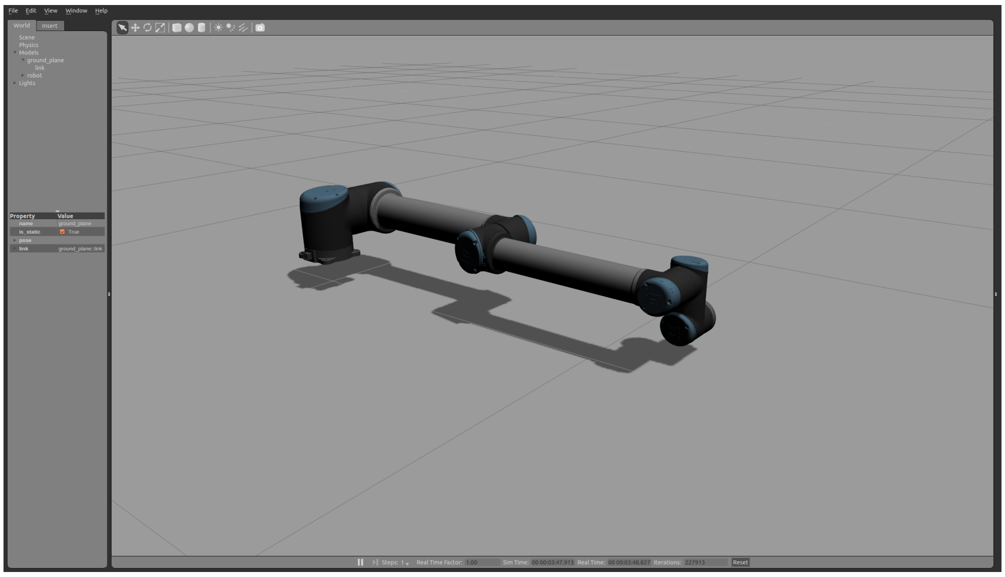


Figure 5.2: Simulation of UR-10 robot in Gazebo simulator.

Chapter 6

RESULTS AND EVALUATION

This chapter evaluates the performance of the [roposed pose estimation algorithm and its parts.

6.1 Testing environment

Framework

The entire system was implemented in C++ within ROS ecosystem, making use of highly optimized libraries for 3D point cloud processing (PCL) and tools (OpenNI) to create robot applications, with the state of art algorithms.

Dataset

For the performance evaluation, a dataset of scenes was collected. It contains 20 point clouds wich can be classified as:

- *Synthetic point clouds.* Gazebo simulator was used in order to capture modeled scenes.
- *Real point clouds, captured in a synthetic environment.* Pallets with boxes were assembled and captured in the Skoltech Robotics laboratory.
- *Real point clouds, captured in real environment.* This part of the dataset was collected in a real warehouse.

Hardware

All tests for which we report the results have been made using a desktop with an Intel $Core^{TM}$ i7-4790 CPU 3.60GHz x 8, 12GB RAM DDR3, Gallium 0.4 on NVE4, Ubuntu 14.04 64bit OS and ROS Indigo release.

Performance metrics

There are several standard metrics used in this work to evaluate the performance of developed algorithms.

Precision:

$$precision = \frac{correct\ matches\ found}{total\ matches\ found} \tag{6.1}$$

Recall:

$$recall = \frac{correct\ matches\ found}{total\ matches} \tag{6.2}$$

Another important performance measurement is *Computational time*.
Since there are many parameters to tune in each individual algorithm we also evaluate them in order to identify the optimal set.

Testing results

The main results are presented in Table 6.1.

Table 6.1: Testing results

Algorithm	**Precision (%)**	**Recall (%)**	**Average computational time (s)**
Graph-based	76	69	1.81
Region growing-based	79	86	0.93
Local descriptors-based (FPFH)	83	54	3.74
Combination	81	84	4.53

As it might be seen from testing results, *Region growing-based* approach shows the best overall performance. The deviation from the 100% precision is related to the fact that the method sometimes generates false clusters due to changing lighting conditions or loss/blurring of RGB edges on filtering stage.
The approach based on matching local descriptors of the model and scene has shown the best precision. This can be explained by the fact that by the principle of this method and a specific of problem solved in this work the method has a tiny probability to generate a false positive. In other words if this algorithm detects box in a scene, there is a big probability that in reality there is a box. Meanwhile due to partiality of data (some boxes are occluded by other, some sides of boxes are not visible in the images) the approach based on local descriptors often can not recognize the box and therefore has a very low "Recall" metric.
As for *Graph-based* method it showed to be the less reliable one. Since the presence of noise and errors in fitting line models the method generates big amount of false positives and false negatives. Although the method is can not be considered as a reliable one, there are ways simple to improve the quality of recognition and pose estimation (see **Future work** section).
It might be seen from the experimental results that the combination of methods losses to individual algorithms in one of the metrics but has the best performance in term of metrics in total. That allows us to say that the combination of individual methods outperform each individual one and with some improvements can be used in a real-life application.

Region-growing

One of the most crucial parameters for the *Region growing* approach is *RegionColorThreshold* which sets the threshold in the difference of RGB values to distinguish clusters. The experiment to identify the most appropriate value was run. The precision and recall of box recognition were used as a performance metric, since they allow us to understand whether all cluster/boxes in the scene were detected or not and how what is a fraction of false positives was. The results of the experiment are presented on Figure 6.1.

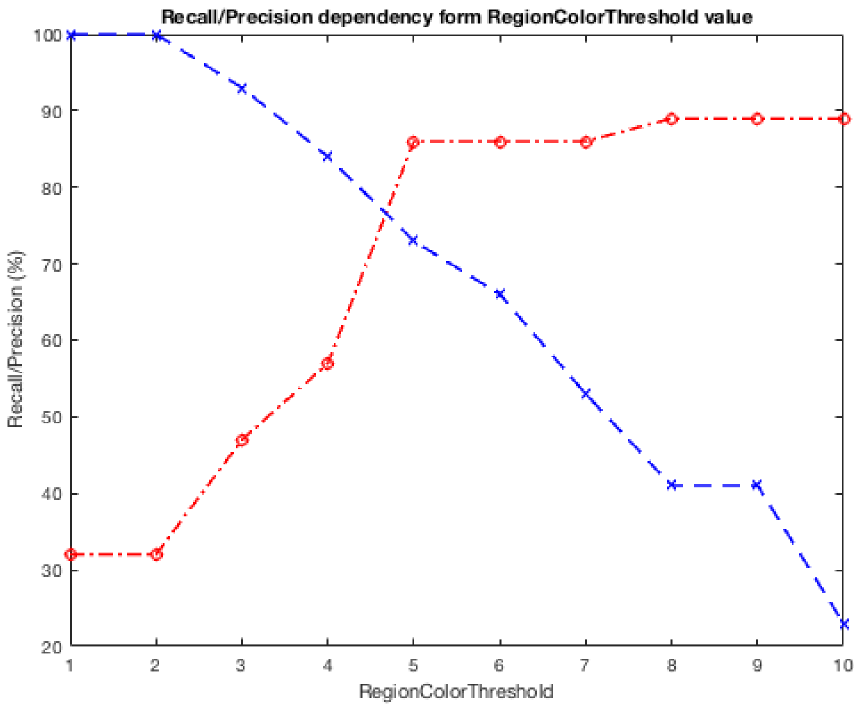


Figure 6.1: Dependency of region growing-based approach precision/recall from *RegionColorThreshold* algorithm parameter.

The experimental results presented above demonstrate the trade-off between *precision* and *recall* metrics. The optimum of *RegionColorThreshold* is identified to 5 as these two metrics meet in this point. It is interesting to note, that in none of the points of discovered dependencies we achieve 100% *precision* or *recall*. This proves again that none individual method can be applied to solve reliably box pose estimation problem and only their combination can be used in a real-world application.

Local descriptors

There is a huge variety of local descriptors proposed for 3D recognition problem (see **Local point descriptors-based approach**). Several of them were tested in the scope of this work. The results are presented in Table 6.2.

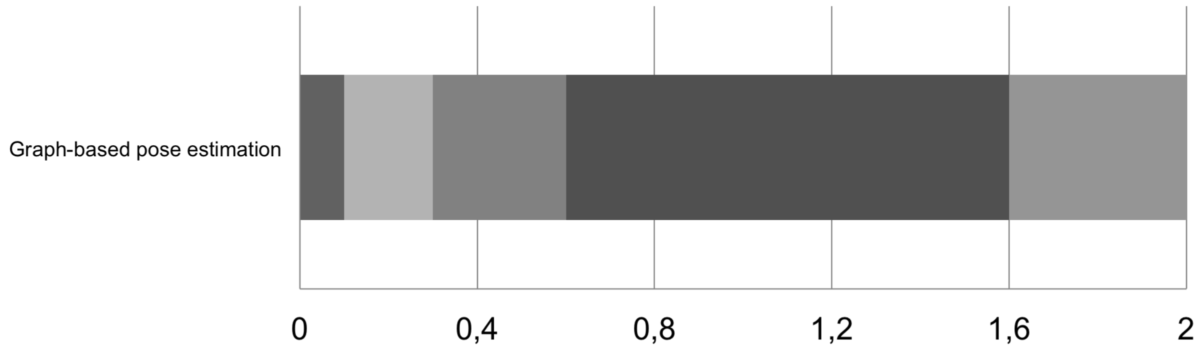
Table 6.2: Local-descriptors pose estimation testing results

Descriptor	**Translation (mm)**	**Rotation (rad)**
RSD	30	0.0052
PFH	7	0.0037
FPFH	9	0.044
USC	13	0.0037

As it might be seen from experimental results FPFH loses in accuracy to PFH, however, it takes much less to be computed and therefore was chosen to be used in this work.

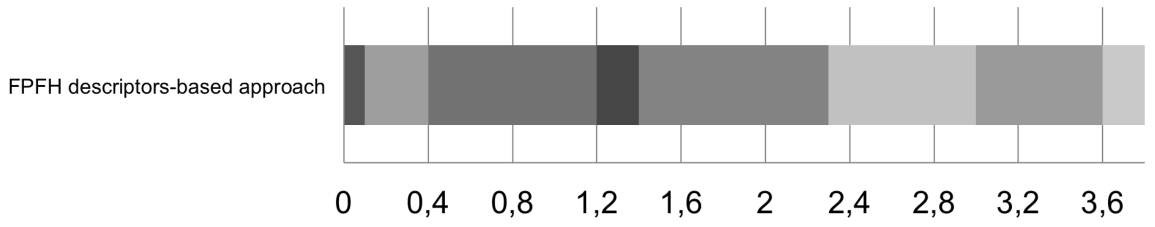
Time profiling

One of the most important metrics for any algorithm is computational time. All proposed methods were studied in terms of timings. 100 runs of each algorithm were performed. An overview of the total computation time of the main steps are reported in Figures 6.2a, 6.2b, 6.3.
The overall time of the hybrid algorithm equals in average to 4.53 s which allows us to draw a conclusion that the proposed approach can be used in a real-world application.



	Graph-based pose estimation
■ Affine transform	0,1
■ Edge detection	0,2
■ Filtration pipeline	0,3
■ Lines detection and intersections check	1
■ Cycles detection	0,4

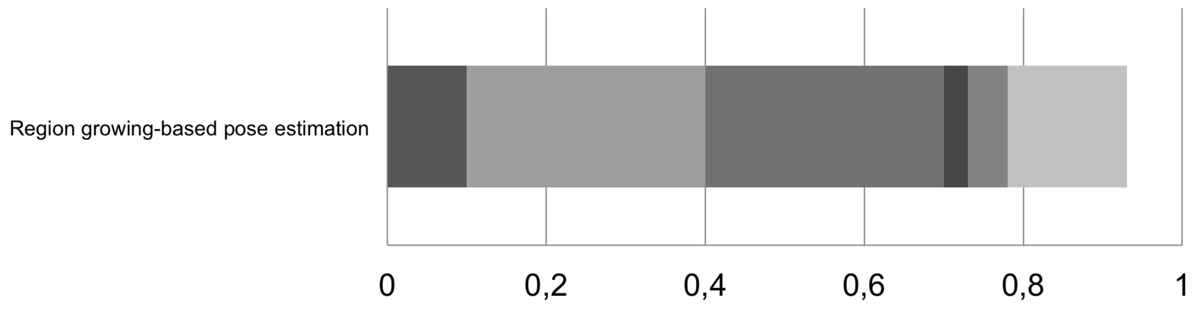
(a) Graph-based approach.



	FPFH descriptors-based approach
■ Affine transform	0,1
■ Filtration pipeline	0,3
■ Normals computation	0,8
■ ISS keypoints detection	0,2
■ FPFH descriptors computation	0,9
■ Descriptors matching	0,7
■ Correspondence grouping	0,6
■ Pose estimation	0,2

(b) Local descriptors-based approach.

Figure 6.2: Time profiling of used algorithms.



	Region growing-based pose estimation
■ Affine transform	0,1
■ Filtration pipeline	0,3
■ Region growing clustering	0,3
■ Statistical Outlier Removal	0,03
■ Concave Hull estimation	0,05
■ Bounding rectangle estimation	0,15

Figure 6.3: Region-growing-based approach time profiling.

Chapter 7

CONCLUSIONS AND FUTURE WORK

7.1 Conclusion

The pose estimation pipeline for detection of boxes lying on the pallet was proposed in this work. In order to evaluate proposed technique *box_detector_pkg* ROS software package was developed. It takes as an input the raw scene point cloud from RGB-D camera as well as camera 3D pose and dimensions of boxes to be detected. The package performs preprocessing of the input point cloud and applies the combination of three discussed pose estimation approaches: local descriptors-based, region-growing based and graph-based. As a result on the output of the program we obtain estimated 6 DoF pose of each visible box in the scene which is sent to the manipulator in order to perform picking. Developed algorithm does not require human intervention and preloaded database of models of the boxes.

The ROS package for receiving and interpreting the output of the *box_detector_pkg* was also implemented (*ur_control_node*). It receives coordinates of boxes, knowing the dimensions of the box calculates the coordinate for picking, and sends this coordinates to the manipulator. *box_detector_pkg* also was successfully integrated with the rest of robot's subsystems by introducing an *action_server* which allows other subsystems to make calls to the recognition system in order to initialize pose estimation procedure.

In the **Results and Evaluation** chapter, the quality of each of the estimation algorithms as well as their combination was evaluated by testing on a collected dataset of images. The test dataset contained both synthetic scenes, and scenes captured on a real warehouse. The results obtained in the tests have proved that proposed estimation pipeline can be used in real life applications. Even though the final has shown good recognition and pose estimation results, there are many improvements and optimizations to be done. In order to achieve at least the human level of efficiency, warehouse automation robot has to have a perception system which has a recognition quality close to 100% and be fast enough. Thereby, some further improvements will be mentioned in the **Future Work** section with the purpose of optimizing the speed and the reliability of the algorithm.

7.2 Future work

There are many ways to improve results of recognition and pose estimation algorithms presented in this work. Region growing-based approach highly depends on color and therefore has all drawbacks typical for algorithms working with 2D images. To eliminate errors related to changing lighting conditions adaptive threshold for assigning cluster to a point needs to be developed. It would tremendously decrease amount of false positives and therefore improve precision of the algorithm. Talking about local descriptors pipeline - Iterative Closes Point (ICP) refinement can be applied as the last step to increase pose estimation accuracy.

In a graph-based approach in order to decrease false negatives and positives rate another approach to finding intersections can be applied. Even a small inaccuracy in line model fitting leads to missing intersection between lines and detection of false intersections. Usage of cylinders of small radius (exact radius has to be identified experimentally) could help to overcome this issue.

The technique used in this work to combine pose estimation algorithms is quiet simple. Other techniques (*e.g.* neural network) could be applied to make pose estimation more reliable. Neural networks might be developed to recognize objects in a point cloud. This is currently a big topic of research and proposed architectures show promising results [51].

Finally, the speed of all algorithm can be improved with application of parallel computing which suits well for many used here approaches. PCL provides special classes substitutions (*e.g. NormalEstimationOMP< PointInT, PointOutT >* instead of *NormalEstimation< PointInT, PointOutT >*) which use *OpenMP* - programming interface (API) that supports multi-platform shared memory multiprocessing programming [52].

BIBLIOGRAPHY

[1] *Warehouses review in Moscow and Moscow Region from 05.26.2015.* URL: http://www.arendator.ru/articles/140103-obzor_rynka_skladskih_pomecshenij_moskvy_za_1_kvartal_2015_goda/.

[2] *Zebra Technologies Warehouse Vision Study.* URL: https://www.zebra.com/content/dam/zebra_new_ia/en-us/solutions-verticals/vertical-solutions/warehouse-management/global-summary/warehouse-vision-2020-study-global-summary.pdf.

[3] *Industrial Logistics Robots Market Shares, Strategies, and Forecasts, Worldwide, 2014 to 2020.* URL: http://www.marketresearchstore.com/report/industrial-logistics-robots-market-shares-strategies-and-forecasts-1396.

[4] *Design and Control of Warehouse Order Picking: a literature review.* URL: https://ideas.repec.org/p/ems/eureri/7322.html.

[5] *Amazon robotics official website.* URL: https://www.amazonrobotics.com/.

[6] *Locus robotics official website.* URL: http://www.locusrobotics.com/.

[7] *Fetch Robotics official website.* URL: http://fetchrobotics.com/.

[8] *Amazon is now using a whole lot more of the robots from the company it bought for 775 million.* URL: http://www.businessinsider.com/amazon-doubled-the-number-of-kiva-robots-2015-10.

[9] *Fetch Robotics Secures Massive 20 Million Investment from SoftBank.* URL: http://spectrum.ieee.org/automaton/robotics/industrial-robots/fetch-robotics-secures-massive-20-million-investment-from-softbank.

[10] P. Kormushev and S. R. Ahmadzadeh. *Robot learning for persistent autonomy.* Eds. Springer International Publishing, 2016.

[11] A. X. Lee, S. Levine, and P. Abbeel. "Learning Visual Servoing with Deep Features and Fitted Q-Iteration". In: *ArXiv e-prints* (Mar. 2017). arXiv: 1703.11000 [cs.LG].

[12] S. Bansal et al. "Goal-Driven Dynamics Learning via Bayesian Optimization". In: *ArXiv e-prints* (Mar. 2017). arXiv: 1703.09260.

[13] Ajmal S. Mian, Mohammed Bennamoun, and Robyn Owens. "Three-Dimensional Model-Based Object Recognition and Segmentation in Cluttered Scenes". In: *IEEE Trans. Pattern Anal. Mach. Intell.* 28.10 (Oct. 2006), pp. 1584–1601. ISSN: 0162-8828. DOI: 10.1109/TPAMI.2006.213. URL: http://dx.doi.org/10.1109/TPAMI.2006.213.

[14] Bastian Steder et al. "NARF: 3D Range Image Features for Object Recognition". In: *Workshop on Defining and Solving Realistic Perception Problems in Personal Robotics at the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS).* Taipei, Taiwan, Oct. 2010.

[15] Andrew E. Johnson and Martial Hebert. "Using Spin Images for Efficient Object Recognition in Cluttered 3D Scenes". In: *IEEE Trans. Pattern Anal. Mach. Intell.* 21.5 (May 1999), pp. 433–449. ISSN: 0162-8828. DOI: 10.1109/34.765655. URL: https://doi.org/10.1109/34.765655.

[16] Kevin Lai et al. "A large-scale hierarchical multi-view RGB-D object dataset." In: *ICRA*. IEEE, 2011, pp. 1817–1824. URL: http://dblp.uni-trier.de/db/conf/icra/icra2011.html#LaiBRF11.

[17] Marius Muja et al. "REIN - A fast, robust, scalable REcognition INfrastructure." In: *ICRA*. IEEE, 2011, pp. 2939–2946. URL: http://dblp.uni-trier.de/db/conf/icra/icra2011.html#MujaRBL11.

[18] Luis A. Alexandre. "3D Descriptors for Object and Category Recognition: a Comparative Evaluation". In: 2012.

[19] Radu Bogdan Rusu and Steve Cousins. "3D is here: Point Cloud Library (PCL)". In: *IEEE International Conference on Robotics and Automation (ICRA)*. Shanghai, China, May 2011.

[20] Stephan Zapotocky. "Image-Based Modeling with Polyhedral Primitives". MA thesis. Favoritenstrasse 9-11/186, A-1040 Vienna, Austria: Institute of Computer Graphics and Algorithms, Vienna University of Technology, 2013. URL: https://www.cg.tuwien.ac.at/research/publications/2013/zapotocky_2013_ma/.

[21] Aitor Aldoma et al. "CAD-model recognition and 6DOF pose estimation using 3D cues." In: *ICCV Workshops*. IEEE, 2011, pp. 585–592. ISBN: 978-1-4673-0062-9. URL: http://dblp.uni-trier.de/db/conf/iccvw/iccvw2011.html#AldomaVBGGRB11.

[22] Sergi Foix, Guillem Alenya, and Carme Torras. "Lock-in Time-of-Flight (ToF) Cameras: A Survey". In: *IEEE Sensors Journal* 11.9 (Sept. 2011), pp. 1917–1926. DOI: 10.1109/jsen.2010.2101060. URL: https://doi.org/10.1109%2Fjsen.2010.2101060.

[23] Don Murray and James J. Little. "Using Real-Time Stereo Vision for Mobile Robot Navigation". In: *Auton. Robots* 8.2 (Apr. 2000), pp. 161–171. ISSN: 0929-5593. DOI: 10.1023/A:1008987612352. URL: http://dx.doi.org/10.1023/A:1008987612352.

[24] Zhengyou Zhang. "Microsoft Kinect Sensor and Its Effect". In: *IEEE MultiMedia* 19.2 (Apr. 2012), pp. 4–10. ISSN: 1070-986X. DOI: 10.1109/MMUL.2012.24. URL: http://dx.doi.org/10.1109/MMUL.2012.24.

[25] Michael Riis Andersen et al. *Kinect Depth Sensor Evaluation for Computer Vision Applications*. Aarhus University, Department of Engineering, 2012.

[26] Michael Arens Timo Breuer Christoph Bodensteiner. "Low-cost commodity depth sensor comparison and accuracy analysis". In: *Proceedings of SPIE - The International Society for Optical Engineering 9250* (2014).

[27] Benjamin Langmann, Klaus Hartmann, and Otmar Loffeld. "Depth Camera Technology Comparison and Performance Evaluation." In: *ICPRAM (2)*. Ed. by Pedro Latorre Carmona, J. Salvador Sánchez, and Ana L. N. Fred. SciTePress, 2012, pp. 438–444. ISBN: 978-989-8425-99-7. URL: http://dblp.uni-trier.de/db/conf/icpram/icpram2012-2.html#LangmannHL12.

[28] *Done Deal: Apple Confirms It Acquired Israeli 3D Sensor Company PrimeSense*. URL: https://techcrunch.com/2013/11/24/apple-primesense-acquisition-confirmed/.

[29] *Microsoft to discontinue Kinect for Windows V1*. URL: http://3rd-strike.com/microsoft-to-discontinue-kinect-for-windows-v1/.

[30] Xuanwu Yin et al. "Efficient active depth sensing by laser speckle projection system". In: *Optical Engineering* 53.1 (2014), p. 013105. DOI: 10.1117/1.OE.53.1.013105. URL: http://dx.doi.org/10.1117/1.OE.53.1.013105.

[31] *Xtion PRO LIVE*. URL: https://www.asus.com/us/3D-Sensor/Xtion_PRO_LIVE/.

[32] *Orbbec Persee.* URL: https://orbbec3d.com/product-persee/.

[33] *Intel® RealSense™ Developer Kit (SR300).* URL: https://click.intel.com/intelrealsense-developer-kit-featuring-sr300.html.

[34] J. Y. Bouguet. “amera Calibration Toolbox for Matlab.” In: ().

[35] *What Is Camera Calibration?* URL: https://www.mathworks.com/help/vision/ug/camera-calibration.html.

[36] *Depth Camera Calibration(Kinect,Xtion,Primesense).* URL: http://jsk-recognition.readthedocs.io/en/latest/jsk_pcl_ros/calibration.html.

[37] *ROS tf package.* URL: http://wiki.ros.org/tf.

[38] Jacob H. Palnick. “Plane Detection and Segmentation For DARPA Robotics Challenge.” MA thesis. Worcester Polytechnic Institute, 2014.

[39] Robert C. Bolles. “Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography”. In: *Communications of the ACM* (1981).

[40] Yinghui Xiao Qingming Zhan Yubin Liang. “Color-based segmentation of point clouds”. In:

[41] Sherif Barakat Khaled M. Alhamzi Mohammed Elmogy. “3D Object Recognition Based on Local and Global Features Using Point Cloud Library”. In: ().

[42] Bastian Steder et al. “Point Feature Extraction on 3D Range Scans Taking into Account Object Boundaries”. In: *In Proc. of IEEE International Conference on Robotics and Automation (ICRA*. 2011.

[43] Yu Zhong. *Intrinsic shape signatures: A shape descriptor for 3D object recognition*. Undetermined. DOI: 10.1109/ICCVW.2009.5457637.

[44] Radu Bogdan Rusu, Nico Blodow, and Michael Beetz. “Fast Point Feature Histograms (FPFH) for 3D Registration”. In: *Proceedings of the 2009 IEEE International Conference on Robotics and Automation*. ICRA’09. Kobe, Japan: IEEE Press, 2009, pp. 1848–1853. ISBN: 978-1-4244-2788-8. URL: http://dl.acm.org/citation.cfm?id=1703435.1703733.

[45] Radu Bogdan Rusu et al. “Persistent point feature histograms for 3d point clouds”. In: *In Proceedings of the 10th International Conference on Intelligent Autonomous Systems (IAS-10)*. 2008.

[46] *The official wiki of the Robotics Group of the University of Leon.* URL: http://robotica.unileon.es/.

[47] *A Layered Approach to Designing Robot Software.* URL: http://www.ni.com/white-paper/13929/en/.

[48] *OpenNI project.* URL: https://structure.io/openni.

[49] *Universal Robots official website.* URL: https://www.universal-robots.com/.

[50] *Gazebo simulator.Official website.* URL: http://gazebosim.org/.

[51] A. Balu et al. “A Deep 3D Convolutional Neural Network Based Design for Manufacturability Framework”. In: *ArXiv e-prints* (Dec. 2016). arXiv: 1612.02141 `[cs.CV]`.

[52] *OpenMP official website.* URL: http://www.openmp.org/.