Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Engineering Cybernetics

Mikael Andreas Medina

Vision-Based Grasp Pose Detection for Autonomous Underwater Vehicles with 0-DoF Manipulators

Master's thesis in Cybernetics and Robotics
Supervisor: Damiano Varagnolo
Co-supervisor: Simon Andreas Hagen Hoff & Erlend Andreas Basso
June 2023

NTNU
Norwegian University of
Science and Technology

Mikael Andreas Medina

Vision-Based Grasp Pose Detection for Autonomous Underwater Vehicles with 0-DoF Manipulators

Master's thesis in Cybernetics and Robotics
Supervisor: Damiano Varagnolo
Co-supervisor: Simon Andreas Hagen Hoff & Erlend Andreas Basso
June 2023

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Engineering Cybernetics

NTNU
Norwegian University of
Science and Technology

Abstract

As the level of exploitation of ocean resources and demand for sustainable offshore solutions rises, so too does the need for autonomous underwater vehicles (AUVs). AUVs can help perform routine inspection, maintenance, and repair (IMR) tasks, reducing the need for costly human expeditions to rough and challenging environments.

An AUV requires an understanding of the scene in order to perform intervention tasks. This thesis presents a framework for understanding the scene as observed by an optical camera to generate suggested positions for an AUV to grasp an object. The suggested framework consists of four main components that are put together and evaluated through simulation. With the aim of giving the AUV a semantic understanding of the scene, the theory behind creating a three-dimensional reconstruction of the observed scene is presented. The reconstructed scene is used as a basis for generating candidate grasp poses based on geometric considerations of the scene and the AUV's gripper. A grasp pose evaluation framework is suggested to evaluate the quality of the generated grasp candidates to extract the pose most likely to be successful. Several geometric quality metrics are proposed for the purpose of this evaluation, each one emphasizing a separate metric considered to indicate a better grasp. For this thesis, the gripper is considered rigidly fastened to the AUV and unable to move on its own. This imposes constraints on how the AUV has to be positioned in order to grasp objects. A scoring function is introduced as a method of evaluating grasp pose candidates based on their orientation.

The proposed grasp pose detection framework is shown to produce viable grasps through simulation. However, no experimental results from a field situation are provided, which leaves the suggested framework with some way to go before it can be verified.

Sammendrag

Med en stor økning i utnyttelsen av havressurser og en økende etterspørsel etter bærekraftige offshore-løsninger, øker også behovet for autonome undervannsfarkoster (AUV-er). AUV-er kan brukes til å utføre rutinemessige inspeksjons-, vedlikeholds- og reparasjonsoppdrag (IMR), og reduserer behovet for kostbare menneskelige ekspedisjoner til tøffe og utfordrende miljøer.

For å kunne utføre slike intervensjonsoppgaver krever AUV-er en situasjonsforståelse. Denne avhandlingen presenterer et rammeverk for at en AUV skal tilegne seg en forståelse om omgivelsene ved bruk av et optisk kamera for å generere foreslåtte posisjoner der AUV-en kan gripe observerte objekter. Det foreslåtte rammeverket består av fire hovedkomponenter som til slutt settes sammen og evalueres gjennom simulasjon. Med mål om å gi AUV-en en semantisk forståelse av den observerte situasjonen, presenteres teorien bak hvordan en tredimensjonal rekonstruksjon av scenen kan gjøres. Den rekonstruerte scenen brukes deretter som grunnlag til å generere kandidater til gripeposisjoner basert på geometriske betraktninger av den observerte scenen og griperen til AUV-en. Videre foreslås det en ramme for evaluering av gripeposisjoner, for å evaluere de genererte kandidatposisjonene og finne den posisjonen som mest sannsynlig vil lykkes. Til dette formålet foreslås flere geometriske kvalitetsmål, hver med vekt på en separat metrikk som anses å indikere et bedre grep. I denne oppgaven betraktes det en griper som sitter fastmontert til AUV-en og som ikke kan bevege seg på egenhånd, noe som legger begrensninger på hvordan AUV-en må plassere seg for å gripe objekter. En poengfunksjon introduseres som en metode for å evaluere gripeposisjonskandidater basert på orienteringen deres.

Det foreslåtte rammeverket for å detektere gripeposisjoner blir gjennom simulasjon vist til å være kapabelt til å generere brukbare grep. Imidlertid er det ikke presentert eksperimentelle resultater fra en felt-situasjon, noe som etterlater det foreslåtte rammeverket med en vei å gå før det kan verifiseres.

Preface

This master's thesis concludes a five-year journey toward the degree of Master of Science in Cybernetics and Robotics at the Norwegian University of Science and Technology (NTNU). Thesis work has been carried out during the winter and spring of 2023 at NTNU, under the supervision of Professor Damiano Varagnolo, Ph.D. candidate Simon Andreas Hagen Hoff, and Postdoctoral Fellow Erlend Andreas Basso, at the Department of Engineering Cybernetics at NTNU.

Parts of this thesis use the results produced during a specialization project conducted in the autumn of 2022. As the project report is not published, some of the work has been restated in this thesis, this includes the sections about the simulator and software setup, Sections 7.1 and 7.2. The presented method has been implemented and tested using the Robot Operating System and the Gazebo simulator. C++ has been the main language of implementation, using the Point Cloud Library for point cloud processing and the linear algebra library Eigen for matrix and vector operations.

First of all, I would like to thank Simon and Erlend, who have been a huge source of help and inspiration when I needed it, and also for being great motivators in times of hardship. I would like to thank Damiano for giving me this much freedom with my thesis and for leaving me in the more than capable hands of Simon and Erlend.

I would also like to give thanks to my family for all their unrelenting support, even through the challenges they have endured whilst I have been studying, and to my girlfriend who encouraged me to spend more time at my workplace than with her. Finally, my time at NTNU would never have been the same without the friends I made along the way. From the unwinding board game nights to the early wake-up motivational coffee meetings, every morning through the final semester. I could probably have done it without you, but I would definitely have chosen to do it with you all again a million times if given the chance.

Trondheim, June 2023
Mikael Andreas Medina

Contents

Figures

Chapter 1

Introduction

This introductory chapter gives an insight into the motivation behind this thesis in Section 1.1 before the problem statement is presented in Section 1.2. Thereafter, Section 1.3 presents the main contributions, followed by an outline of the thesis in Section 1.4.

1.1 Motivation

Human beings are curious and explorers by nature. Throughout history, the world has been extensively explored and mapped out, and with the current technology, anyone with a handheld device is able to instantly look up directions to practically anywhere on Earth. Since the early ages people have looked to the stars, and — as soon as technology allowed for it — explorers went to the moon. Yet, the vast oceans of the Earth remain mystical and largely uncharted.

Covering more of the Earth's surface than land, the oceans provide an intriguing and largely untapped potential for explorers and entrepreneurs. Increasing resource demands and environmental challenges push the human race to reach into the dark depths below. The seafloor is thought to have large amounts of mineral resources [1]. Demand for sustainable nutrition drives researchers to utilize the ocean for large-scale renewable resources such as fish and kelp [2]. Furthermore, some of the largest man-made structures dwell in the rough conditions posed by the deep oceans, such as the Petronius oil platform towering more than 530 meters above the ocean floor [3].

In common for these fields is the fact that the ocean is a dangerous place to work. Remotely Operated Vehicles (ROVs) reduce the need for direct human interaction, allowing operators to remotely perform tasks such as inspection, maintenance, and repair (IMR). ROV operations reduce the risk of human injury but require large resources in the form of experienced operators, surface support vessels, and time, all expensive resources [4]. Autonomous Underwater Vehicles (AUVs) capable of performing IMR operations alleviate the need for large investments in time and resources, further enabling safe access to the ocean, and making AUV

solutions increasingly desirable for offshore operators.

In order to effectively perform IMR and intervention tasks, AUVs need situational awareness. While visual sensors such as cameras achieve high-resolution representations for unmanned solutions on land and in the air, subsea solutions are limited by rough underwater conditions. Image enhancement algorithms use physical properties of light in order to improve the clarity in captured images [5]. However, for intervention tasks, objects have to be recognized as well, as opposed to just being captured in images. Underwater solutions such as the object detection algorithm explored by Bazeille et al. exploit phenomena of underwater lighting in order to detect objects of known color [6]. Rizzini et al. expand this methodology further with assumptions about the shape and color of man-made objects for easier recognition in underwater settings [7].

Machine learning solutions for object detection are rapidly gaining traction, among others, methods such as You Only Look Once [8] and Single Shot MultiBox Detector [9] can be said to have popularized this approach. Mandal et al. propose an automatic identification algorithm for fish species [10], while the method presented by Katayama et al. attempts to address color correction and object detection simultaneously [11]. In a recent survey, Xu et al. conclude that while underwater object detection solutions achieve great results, it remains a vital and challenging research topic [12].

In addition to object detection, an AUV might have to interact with the environment. Autonomous subsea intervention is still in its early days of development, with few commercially employed solutions, to the author's best knowledge. A proposed solution by Faria et al. uses LEDs mounted on a valve to indicate its position and two 7 Degrees of Freedom (DoF) arms for manipulation [13]. However, control of the arms requires solving the inverse kinematics, which is prone to errors and inaccuracies. Palomeras et al. faced similar challenges with the joint positions of the manipulators on their AUV, as well as concluding that visibility and illumination poses a great challenge [14].

Interacting with a detected Object of Interest (OOI) requires knowledge of where to place the gripper. Given the geometry of the gripper, determining where to place it in order to grasp an object is referred to as grasp pose detection. Zapata-Impata et al. propose a method using principal component analysis and geometric considerations of observed objects to determine grasping positions [15]. In their approach ten Pas et al. also use geometrical considerations for grasp pose candidates, which are then classified using a machine learning algorithm to determine how likely the grasp is to work [16].

The use of multi-DoF manipulators increases the flexibility in AUV positioning, as the arm can correct for positioning errors. However, as previously discussed, this comes at the cost of an increase in complexity of control. A trade-off thus has

to be made between the complexity of the ROV and its manipulator. Furthermore, increasing the number of movable parts puts the AUV at risk of itself needing maintenance, possibly reducing its efficiency.

1.2 Problem Statement

Commercial ROVs are large and require professionally trained crew to pilot and control their manipulators. Using an AUV for intervention tasks would lower operational costs and increase the up-time of maintenance operations. In order for an AUV to interact with the environment, situational awareness is needed along with a method of figuring out where to grasp perceived objects. For reduced complexity of the AUV, this thesis aims to develop a grasp pose detection framework for a rigidly mounted zero DoF gripper, using geometrical considerations of the scene and the gripper.

1.3 Contributions

This thesis investigates a method of autonomously detecting grasping poses on underwater objects, intended to be used as a reference for a control system for autonomous intervention. The suggested method is based on using an AUV rigidly mounted with a 0 DoF gripper. This imposes the problem of having to maneuver the ROV correctly in order to position the gripper, as the arm is unable to correct any positioning errors. The main contributions of this thesis are twofold; firstly, two sampling methods boasting different qualities emphasizing various aspects of the sampled set are presented. Secondly, four quality metrics used for the evaluation of candidate grasping poses are presented, each highlighting a different aspect of what constitutes a good grasp.

1.4 Thesis Structure

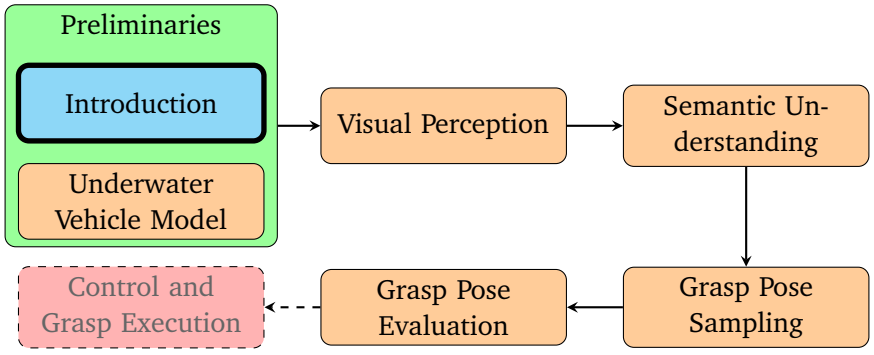


Figure 1.1: Pipeline for autonomous underwater grasping of unknown objects. This chapter has motivated the thesis and introduced its structure. The red faded node represents the next step, which is outside the scope of this thesis.

Grasping an object is the final step in a series of processes, each step representing highly prevalent research problems. This thesis is structured such that each chapter represents a piece of the puzzle, introducing relevant theories and methodology for performing each task, leading up to the results from a simulation of the presented solution. This chapter has presented the motivation behind the thesis, along with the main contributions and structure. In Chapter 2, some preliminary theory regarding modeling of underwater vehicles is presented. This theory is applied to the implementation and simulation setup introduced in Chapter 7. Chapter 3 introduces camera modeling and concepts used for robotic vision and lays the foundation for the pipeline suggested in this thesis.

A general overview of the pipeline is shown in Figure 1.1, where a cyan highlight indicates that the current chapter covers the highlighted topic. Subsequent chapters will introduce and further emphasize their own fields, while the red node in Figure 1.1 denotes future work for the implementation of a control scheme using the generated grasp pose. Chapter 4 outlines processing techniques for point cloud processing. Chapter 5 introduces two methods of sampling grasp pose candidates before quality metrics used to evaluate the poses are introduced in Chapter 6. The simulation setup used for testing the proposed methods is presented in Chapter 7. Thereafter, Chapters 8 and 9 present the results from the simulation and discuss the performance before Chapter 10 concludes the thesis.

Chapter 2

Modeling of Underwater Vehicles

Research on dynamics for aquatic vehicles has been ongoing for many years and has produced accurate equations for guidance, navigation, and control. This chapter acts as a brief introduction to the dynamics of an underwater vehicle, specifically as presented by Fossen in [17]. Section 2.1 covers some preliminary details such as reference frames and transformations before the dynamics of an underwater vehicle are presented in Section 2.2.

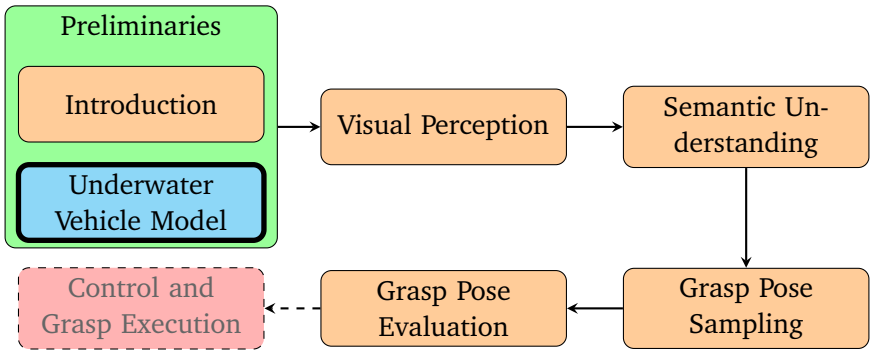


Figure 2.1: In this chapter preliminaries on the modeling of underwater vehicles are presented.

2.1 Preliminaries

This section serves as an overview of some notation used throughout this thesis, as well as an introduction to preliminary concepts such as frames and rotations.

Throughout this thesis vectors are denoted as bold lowercase letters $\mathbf{v}$ and matrices with bold capital letters $\mathbf{M}$. Superscript on a vector denotes the reference frame the vector is relative to, the vector $\mathbf{v}^n$ is a vector in the north-east-down (NED) frame. A vector from one point to another is denoted $\mathbf{v}_{ab}^b$, meaning the vector from a to b represented in frame b. For the purpose of this thesis, NED is treated as a world-fixed inertial frame, while the BODY frame b is a body-fixed frame with axes pointing front, right, and down relative to a body.

A rotation matrix $\mathbf{R}_a^b$ is a matrix in the special orthogonal group of all rotations in Euclidean space, i.e. $\mathbf{R}_a^b \in$SO(3). Notation-wise $\mathbf{R}_a^b$ reads as the rotation from frame a to b. Rotation matrices are orthogonal and satisfy the properties

$$\mathbf{R}\mathbf{R}^\top = \mathbf{R}\mathbf{R}^{-1} = \mathbf{I}, \quad |\det(\mathbf{R})| = 1 \tag{2.1}$$

, i.e. the transpose of the matrix is equal to its inverse. An inverse of a rotation matrix represents the inverse of the rotation, i.e. $\mathbf{R}_a^b = \text{inv}(\mathbf{R}_b^a)$. Euler angle representation of rotation denotes the rotation from one frame to another as a series of rotations around the x-, y- and z-axis by the angles ϕ, θ, ψ respectively, where ϕ, θ, ψ are commonly referred to as roll, pitch, and yaw. The individual rotation matrices are given in Equations (2.2) to (2.4), while the full rotation is the product $\mathbf{R}_{zyx}(\mathbf{\Theta}) = \mathbf{R}_z(\psi)\mathbf{R}_y(\theta)\mathbf{R}_x(\phi)$, where $\mathbf{\Theta} = [\phi, \theta, \psi]^\top$. Consequently, a rotation from BODY to NED is written as $\mathbf{R}_b^n(\mathbf{\Theta}_{nb})$, where $\mathbf{\Theta}_{nb}$ represents the Euler angles.

$$\mathbf{R}_x(\phi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\phi) & -\sin(\phi) \\ 0 & \sin(\phi) & \cos(\phi) \end{bmatrix} \tag{2.2}$$

$$\mathbf{R}_y(\theta) = \begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) \\ 0 & 1 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) \end{bmatrix} \tag{2.3}$$

$$\mathbf{R}_z(\psi) = \begin{bmatrix} \cos(\psi) & -\sin(\psi) & 0 \\ \sin(\psi) & \cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.4}$$

The Euler angle representation of a rotation matrix is susceptible to gimbal lock. Gimbal lock is a phenomenon where two of the three rotational axes align, causing the rotation to lose a degree of freedom. When using the Euler angle representation, gimbal lock happens at $\theta = \pm 90°$, making it so a change in ρ or ψ causes the same rotation. Alternative representations of rotations that avoid the problem of gimbal locking are for instance quaternions and Rodrigues' rotation formula.

Rodrigues' rotation formula describes the rotation of a vector $\mathbf{v}$ around a unit rotation axis $\mathbf{k}$ by the angle θ as in Equation (2.5).

$$\mathbf{v}_{rot} = \mathbf{v}\cos(\theta) + (\mathbf{k} \times \mathbf{v})\sin(\theta) + \mathbf{k}(\mathbf{k} \cdot \mathbf{v})(1 - \cos(\theta)) \tag{2.5}$$

Rotation quaternions are unit vectors of four elements, where three elements constitute the imaginary part and one element the real part. While quaternions are less intuitive than Euler angles, they avoid gimbal lock and allow rotations to be represented by a single angle around an axis, similar to Rodrigues' formula.

Conversion from a rotation using Rodrigues' formula to a rotation using quaternions is done simply by halving the angle and defining the real and imaginary parts of the quaternion. This is shown in Equation (2.6), where w is the real part of the quaternion and x, y, z the imaginary.

$$\mathbf{q} = \begin{bmatrix} w \\ x \\ y \\ z \end{bmatrix} = \begin{bmatrix} \cos\left(\frac{\theta}{2}\right) \\ \mathbf{k}_{3\times 1} \sin\left(\frac{\theta}{2}\right) \end{bmatrix} \tag{2.6}$$

2.2 Underwater Vehicle Dynamics

In this section a 6 degrees of freedom (DoF) model for underwater vehicles is presented. The model is based on the work presented by Fossen in [17], where the dynamics for an underwater vehicle are presented as

$$\dot{\boldsymbol{\eta}} = \mathbf{J}_k(\boldsymbol{\eta})(\boldsymbol{\nu}_r + \boldsymbol{\nu}_c), \quad k \in \{\boldsymbol{\Theta}, \mathbf{q}\} \tag{2.7}$$

$$\mathbf{M}\dot{\boldsymbol{\nu}}_r + \mathbf{C}(\boldsymbol{\nu}_r)\boldsymbol{\nu}_r + \mathbf{D}(\boldsymbol{\nu}_r)\boldsymbol{\nu}_r + \mathbf{g}(\boldsymbol{\eta}) + \mathbf{g}_0 = \boldsymbol{\tau} \tag{2.8}$$

where k denotes Euler angles or unit quaternions. $\boldsymbol{\eta}$ is chosen accordingly and represents the pose — position and orientation — in the inertial frame. Assuming constant current velocity, $\boldsymbol{\nu}_c$ is the linear ocean current velocity, $\boldsymbol{\nu}_r$ is the relative velocity given in BODY, $\boldsymbol{\nu}_r = \boldsymbol{\nu} - \boldsymbol{\nu}_c$. $\mathbf{J}_k(\boldsymbol{\eta})$ is the transformation from BODY to NED and also changes depending on angle representation, further details in Section 2.2.1. The matrices $\mathbf{M}$, $\mathbf{C}(\boldsymbol{\nu}_r)$ and $\mathbf{D}(\boldsymbol{\nu}_r)$ consist of rigid body dynamics and hydrodynamics, and are known as the system inertia, Coriolis and centripetal, and damping matrices respectively.

$$\mathbf{M} = \mathbf{M}_{RB} + \mathbf{M}_A \tag{2.9}$$

$$\mathbf{C}(\boldsymbol{\nu}_r) = \mathbf{C}_{RB}(\boldsymbol{\nu}_r) + \mathbf{C}_A(\boldsymbol{\nu}_r) \tag{2.10}$$

$$\mathbf{D}(\boldsymbol{\nu}_r) = \mathbf{D} + \mathbf{D}_n(\boldsymbol{\nu}_r) \tag{2.11}$$

In the following sections a selection of components of Equation (2.7) to (2.11) will be discussed further.

2.2.1 Kinematics

Kinematics describes the geometrical aspect of motion of the underwater vehicle. As the underwater vehicle moves, the position and orientation relative to the world-fixed inertial NED frame is given by

$$\boldsymbol{\eta} = [x^n, y^n, z^n, \phi, \theta, \psi]^\top \tag{2.12}$$

for Euler angles and

$$\boldsymbol{\eta} = [x^n, y^n, z^n, \eta, \epsilon_1, \epsilon_2, \epsilon_3]^\top \tag{2.13}$$

using quaternions. For convenience, the velocity of the vehicle relative to the origin of NED is given in the BODY frame. As such the linear velocity $\mathbf{v}_{nb}^{b} = [u, v, w]^{\top}$ is expressed along the x^{b}, y^{b}, z^{b}-directions of BODY respectively, and the angular velocity of the axes relative to NED is $\boldsymbol{\omega}_{nb}^{b} = [p, q, r]^{\top}$. $\mathbf{J}_{k}(\boldsymbol{\eta})$ relates the velocity $\boldsymbol{\nu} = \begin{bmatrix} \mathbf{v}_{nb}^{b} \\ \boldsymbol{\omega}_{nb}^{b} \end{bmatrix}$ in BODY to the velocity in NED. For conciseness, the details of the matrices $\mathbf{R}$ and $\mathbf{T}$ are left out here, but the transformation has the form

$$\mathbf{J}_{\boldsymbol{\Theta}}(\boldsymbol{\eta}) = \begin{bmatrix} \mathbf{R}_{b}^{n}(\boldsymbol{\Theta}_{nb}) & \mathbf{0}_{3\times 3} \\ \mathbf{0}_{3\times 3} & \mathbf{T}_{b}^{n}(\boldsymbol{\Theta}_{nb}) \end{bmatrix} \tag{2.14}$$

for Euler angles, and

$$\mathbf{J}_{\mathbf{q}}(\boldsymbol{\eta}) = \begin{bmatrix} \mathbf{R}_{b}^{n}(\mathbf{q}_{b}^{n}) & \mathbf{0}_{3\times 3} \\ \mathbf{0}_{4\times 3} & \mathbf{T}_{b}^{n}(\mathbf{q}_{b}^{n}) \end{bmatrix} \tag{2.15}$$

for quaternions.

2.2.2 Kinetics

Kinetics relates forces to the motion of a body. Kinetic forces include forces due to the shape of the body and the relationship between the body and the environment. Sticking to the framework presented in [17], kinetics are divided into rigid-body kinetics, hydrodynamics, and hydrostatics. For the purpose of this thesis, the details behind the Coriolis and centripetal matrix $\mathbf{C}(\boldsymbol{\nu}_{r})$ have been left out, however, it is noted that these terms arise from the motion of the rotating BODY frame with respect to NED. For all intents and purposes, the Coriolis and centripetal matrices can be calculated from the rigid-body and hydrodynamic added mass matrices $\mathbf{M}_{RB}$ and $\mathbf{M}_{A}$ [17].

Rigid-body

In order to take advantage of a vehicle's geometric properties, it is convenient to express the equations of motion about an arbitrary Coordinate Origin (CO). To do so, the equations are first derived around the Center of Gravity (CG), then they are transformed to the CO using a coordinate transform. The transformation is calculated using the vector $\mathbf{r}_{bg}^{b}$ from CO to CG. The rigid body inertia matrix can then be expressed around the CO as

$$\mathbf{M}_{RB} = \begin{bmatrix} m\mathbf{I}_{3} & -m\mathbf{S}(\mathbf{r}_{bg}^{b}) \\ m\mathbf{S}(\mathbf{r}_{bg}^{b}) & \mathbf{I}_{g}^{b} - m\mathbf{S}^{2}(\mathbf{r}_{bg}^{b}) \end{bmatrix} \tag{2.16}$$

where m is the mass of the vehicle, $\mathbf{I}_{g}^{b}$ the inertia matrix about CG, and $\mathbf{S}(\mathbf{r}_{bg}^{b})$ the skew-symmetric matrix representation of $\mathbf{r}_{bg}^{b}$, i.e.

$$\mathbf{S}(\mathbf{r}_{bg}^{b}) = \begin{bmatrix} 0 & -\mathbf{r}_{3} & \mathbf{r}_{2} \\ \mathbf{r}_{3} & 0 & -\mathbf{r}_{1} \\ -\mathbf{r}_{2} & \mathbf{r}_{1} & 0 \end{bmatrix} \tag{2.17}$$

Assuming that the CO is defined close to the CG, $\mathbf{r}_{bg}^{b} \approx \mathbf{0}$, and the rigid body inertia matrix is reduced to

$$\mathbf{M}_{RB} = \begin{bmatrix} m\mathbf{I}_3 & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{I}_g^b \end{bmatrix} \tag{2.18}$$

Hydrodynamics

Hydrodynamics are forces imposed on the body as it moves through liquids. The added mass matrix $\mathbf{M}_A$ describes the mass displaced by the vehicle as it moves and is highly dependent on the shape of the vehicle as well as its velocity. Assuming symmetry of the vehicle about the xz and yz planes yields an added mass matrix with mostly terms on the diagonal and a couple of off-diagonal terms. Either assuming that the off-diagonal terms are much smaller than the diagonal terms, or through xy symmetry, the added mass matrix is reduced to a diagonal matrix

$$\mathbf{M}_A = -diag\{X_{\dot{u}}, Y_{\dot{v}}, Z_{\dot{w}}, K_{\dot{p}}, M_{\dot{q}}, N_{\dot{r}}\} \tag{2.19}$$

As the motion of an underwater vehicle is highly nonlinear and coupled at high speeds, modeling the damping, as well as the previously mentioned terms, is a complicated task. In Equation (2.11) damping has been divided into a linear damping term $\mathbf{D}$ and a nonlinear term $\mathbf{D}_n(\boldsymbol{\nu}_r)$. By assuming operations at low speed, the nonlinear damping term can be neglected. Furthermore, assuming symmetry over multiple planes and that off-diagonal terms are low, the linear damping matrix is also reduced to a diagonal matrix

$$\mathbf{D} = -diag\{X_u, Y_v, Z_w, K_p, M_q, N_r\} \tag{2.20}$$

Hydrostatics

A submerged body displaces a volume of liquid equal to the volume of the body. If the density of the body displacing the liquid is lower than the density of the liquid, the liquid exerts an upwards force on the body, this is known as buoyancy. Buoyancy is exerted on the center of the volume that displaces the liquid, the Center of Buoyancy (CB). For a fully submerged body, the CB coincides with the centroid of the body. The submerged body is also affected by gravity in CG. Gravity attempts to pull the body down, while buoyancy pushes upwards, giving rise to the name restoring forces. Furthermore, if the weight of the body is equal to the weight of the liquid it displaces, the body stays stationary. A body in which buoyancy and gravity are equal is called neutrally buoyant.

Buoyancy and gravity enter the model in Equation (2.8) as the term $\mathbf{g}(\boldsymbol{\eta})$, while the term $\mathbf{g}_0$ relates to ballast, i.e. weight that can be loaded to change the properties of buoyancy. As the restoring forces act along the vertical axis of the

inertial NED frame, they can be represented in NED as

$$\mathbf{f}_b^n = \begin{bmatrix} 0 \\ 0 \\ B \end{bmatrix}, \quad \mathbf{f}_g^n = \begin{bmatrix} 0 \\ 0 \\ W \end{bmatrix} \tag{2.21}$$

where $W = mg$ is the gravity and $B = \rho g \nabla$ is the buoyancy. Assuming that the vehicle is neutrally buoyant the restoring forces are

$$\mathbf{g}(\boldsymbol{\eta}) = \begin{bmatrix} 0 \\ 0 \\ 0 \\ -BG_y W \cos(\theta)\cos(\phi) + BG_z W \cos(\theta)\sin(\phi) \\ BG_z W \sin(\theta) + BG_x W \cos(\theta)\cos(\phi) \\ -BG_x W \cos(\theta)\sin(\phi) - BG_y W \sin(\theta) \end{bmatrix} \tag{2.22}$$

where $BG_{x,y,z}$ are the components of the vector defining the distance between CG and CB, $[BG_x, BG_y, BG_z]^\top$. In practice, most AUVs are designed such that the buoyancy is slightly larger than the force of gravity such that the AUV slowly rises to the surface in the event of a malfunction.

As previously stated, the restoring forces act opposite to each other. If the CB is below the CG, the buoyant forces will push the CB upwards, while the gravitational forces push the CG downwards. This results in both restoring forces working to drive the CG below the CB and the vehicle will spin. Thus, for a vehicle where the CG is located far below the CB, achieving poses with a high degree of roll or pitch requires the vehicle to combat the restoring forces.

Chapter 3

Robotic Vision

For a robot to get an understanding of a scene, it uses sensors to measure what the environment around it is like. This chapter introduces robotic perception through the means of optical cameras, starting with introducing the basic mathematical principles behind the pinhole camera model in Section 3.1. Further, Section 3.2 hints at why a camera modeled and calibrated for in-air operations might struggle underwater, before Section 3.2.1 introduces a camera model designed for more accurate underwater imaging. Sections 3.3 to 3.5 introduces feature detection and matching, which is the process of describing regions of interest in one image in order to recognize the same region in other images. When a point is recognized in two images, knowledge of the camera positions can be used to estimate the three-dimensional (3D) coordinates of the point. This is one of the core motivators behind stereo imaging, a topic presented in Section 3.6. Section 3.7 expands on this with some other methods of acquiring the depth of a point in an image.

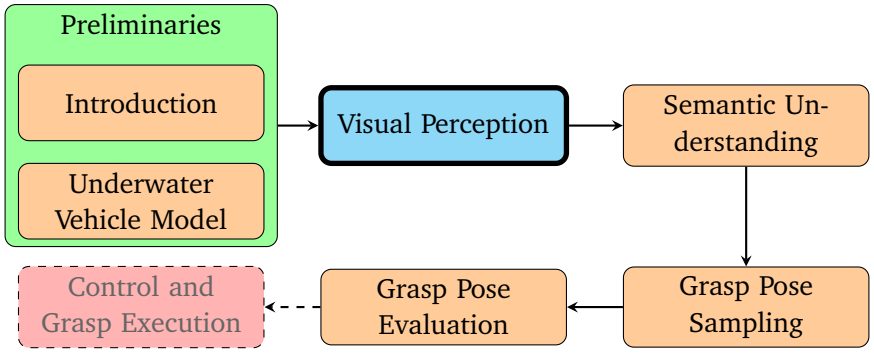


Figure 3.1: In this chapter theory and methods related to the visual perception part of the pipeline is presented.

3.1 Camera Modeling

In its most basic form, a camera is a device that projects the light of a 3D scene onto a 2D imaging plane. The first cameras were based on the principle that light traveling from a 3D scene through a small opening in a wall depicts the scene

inverted on another wall behind the opening. Early in the 1600s Johannes Kepler, among others, termed this the camera obscura [18]. Modern-day cameras use the same principles but capture the projected image by using a grid of photosensitive sensors as the 2D imaging plane. Digital representations of images consist of matrices holding the intensity value of each individual pixel. An image with one such matrix represents the intensity of one color, popularly black and white. Colored images are usually represented using three channels, one channel each for the colors red, green, and blue (RGB).

3.1.1 Pinhole Camera

The pinhole camera model is a mathematical model that describes the relation between the 3D real-world coordinates and the 2D pixel coordinates, illustrated in Figure 3.2. As all the rays converge through a singular point in the aperture — i.e. the origin o in Figure 3.2 — this kind of camera model is referred to as a singular viewpoint (SVP) model. Using the similar triangles illustrated in Figure 3.2, the relation between camera coordinates (X, Y, Z) and image coordinates (x, y) can now be expressed as in Equation (3.1). The ray is captured by the imaging sensor, indicated by the leftmost frame, however, a virtual imaging plane is defined at a positive distance f in front of the aperture to avoid the image appearing flipped.

$$\begin{aligned} \frac{x}{f} = \frac{X}{Z} &\longrightarrow x = f\frac{X}{Z} \\ \frac{y}{f} = \frac{Y}{Z} &\longrightarrow y = f\frac{Y}{Z} \end{aligned} \tag{3.1}$$

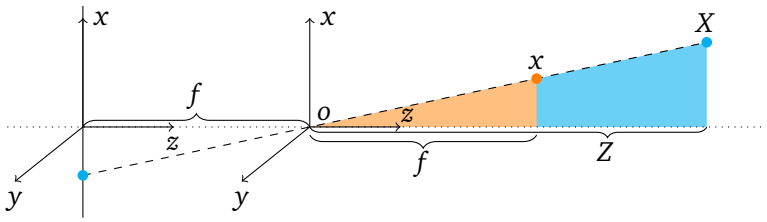


Figure 3.2: Similar triangles form the basis of the projective pinhole camera model. The leftmost frame is the imaging plane, where the observed point has been inverted, as in the camera obscura. The distance f is the focal length.

From real-world coordinates to pixel coordinates, there are two main transformations happening. The first transformation is from world coordinates to camera coordinates, the parameters of this transformation are called extrinsic. Going from camera coordinates to pixel coordinates is the second transformation, whose parameters are referred to as intrinsics [19].

Homogeneous Coordinates

Homogeneous coordinates are used to allow for more convenient calculations than their counterpart Cartesian. With homogeneous coordinates, any affine transformation is reduced to a matrix-vector product. This allows for a more complex series of transformations to be calculated into one transformation matrix, rather than having to do all operations individually.

To transform a Cartesian vector to a homogeneous one, simply add a 1 at the end, see Equation (3.2). Extracting the Cartesian vector from a homogeneous one is done by first dividing the vector by the last element and then removing it, see Equation (3.3).

$$\mathbf{x} = \begin{bmatrix} x \\ y \\ z \end{bmatrix} \longrightarrow \tilde{\mathbf{x}} = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{3.2}$$

$$\tilde{\mathbf{x}} = \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} \longrightarrow \mathbf{x} = \begin{bmatrix} x/w \\ y/w \\ z/w \end{bmatrix} \tag{3.3}$$

Extrinsics

Given that the camera is placed with a known relation to a world frame, the extrinsics consist of the rigid-body transformation between the camera frame and the world frame. A homogeneous rigid-body transformation consists of the rotation and translation required to go from one frame to another. Equation (3.4) shows a homogeneous transform from frame b to frame a, where $\mathbf{t}_{ab}^{a}$ indicates the position of frame b, relative to a. Homogeneous world coordinates are then transformed to homogeneous camera coordinates according to Equation (3.5), where $\mathbf{T}_{w}^{c}$ is the extrinsic matrix.

$$\mathbf{T}_{b}^{a} = \begin{bmatrix} \mathbf{R}_{b}^{a} & \mathbf{t}_{ab}^{a} \\ \mathbf{0}_{1\times 3} & 1 \end{bmatrix} \tag{3.4}$$

$$\tilde{\mathbf{x}}^{c} = \mathbf{T}_{w}^{c} \cdot \tilde{\mathbf{x}}^{w} \tag{3.5}$$

Intrinsics

The intrinsic camera parameters describe the relation between 3D camera coordinates and the 2D image coordinates. Converting the pinhole model described in Equation (3.1) to homogeneous coordinates, expanding and rearranging the

terms, yields the matrix-vector relation in Equation (3.6). The 3×4 identity matrix, $\mathbf{P}_{3\times4}$, represents projection from 3D homogeneous camera coordinates to 2D image coordinates, while the matrix containing the focal length f is scaling.

$$Z^c \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X^c \\ Y^c \\ Z^c \\ 1 \end{bmatrix} \tag{3.6}$$

Pixels are commonly given with integer values, contrary to image coordinates. To address this, the image coordinates are scaled with a factor related to the physical configuration of the sensor

$$\begin{bmatrix} x_s \\ y_s \end{bmatrix} = \begin{bmatrix} s_x & s_\theta \\ 0 & s_y \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \tag{3.7}$$

where s_θ is the skew factor, which is 0 if the pixels are rectangular.

The model in Equation (3.6) is defined for an image plane where the center of the image is the origin, however, it is common to denote the top left pixel as the origin for computations. Correcting the origin is done with a simple translation

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x_s + o_x \\ y_s + o_y \end{bmatrix} \tag{3.8}$$

where the point (o_x, o_y) is the center of the image sensor relative to the top right. Finally, these transformations are put together in their homogeneous form to compose the intrinsic matrix $\mathbf{K}$.

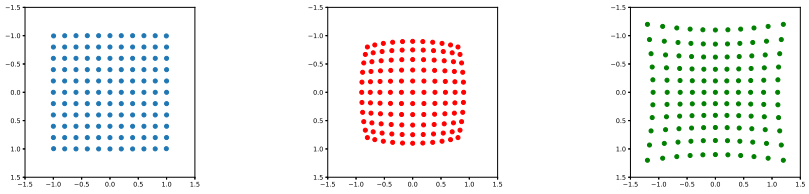
$$\mathbf{K} = \begin{bmatrix} s_x & s_\theta & o_x \\ 0 & s_y & o_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} fs_x & s_\theta & o_x \\ 0 & fs_y & o_y \\ 0 & 0 & 1 \end{bmatrix} \tag{3.9}$$

Replacing the left-hand Z^c with a possibly unknown scaling factor λ, the full pinhole camera model can now be expressed as

$$\begin{gathered} \lambda \tilde{\mathbf{x}}' = \mathbf{K}_s \mathbf{K}_f \mathbf{P}_{3\times4} \mathbf{T}_w^c \tilde{\mathbf{x}}^w \\ \lambda \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} fs_x & s_\theta & o_x & 0 \\ 0 & fs_y & o_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{R}_w^c & \mathbf{t}_{cw}^c \\ \mathbf{0}_{1\times3} & 1 \end{bmatrix} \begin{bmatrix} x^w \\ y^w \\ z^w \\ 1 \end{bmatrix} \end{gathered} \tag{3.10}$$

3.1.2 Distortion

In practice a camera is not just a hole in a surface, but rather one or more lenses that alter the path of the light rays. The arrangement of the lenses relative to the imaging sensor can cause distortion of pixels that make straight lines appear curved, this is known as radial distortion, see Figure 3.3. If the imaging sensor is not correctly aligned with the lenses the image may appear tilted, known as tangential distortion, see Figure 3.4.



(a) No distortion. **(b)** Barrel distortion. **(c)** Pincushion distortion.

Figure 3.3: Comparison of different radial distortions.

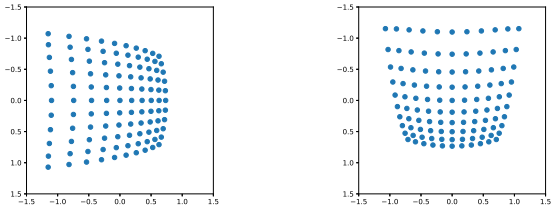


Figure 3.4: Examples of tangential distortion.

Distortion can be corrected using models that mathematically describe the effect of the distortion on the pixels. One of the most common distortion models is the Brown-Conrady model, which models radial distortion as

$$\begin{aligned} \delta x_r &= x\left[k_1(x^2+y^2)+k_2(x^2+y^2)^2+k_3(x^2+y^2)^3+\ldots\right] \\ \delta y_r &= y\left[k_1(x^2+y^2)+k_2(x^2+y^2)^2+k_3(x^2+y^2)^3+\ldots\right] \end{aligned} \tag{3.11}$$

and tangential distortion as

$$\begin{aligned} r &= \sqrt{x^2+y^2} \\ \delta x_t &= \left[p_1(r^2+2x^2)+2p_2xy\right](1+p_3r^2+p_4r^4+p_5r^6+\ldots) \\ \delta y_t &= \left[p_2(r^2+2y^2)+2p_1xy\right](1+p_3r^2+p_4r^4+p_5r^6+\ldots) \end{aligned} \tag{3.12}$$

where k_n are the radial, and p_n the tangential, distortion coefficients of the lens [20]. It is usually sufficient to only use lower order terms for both types of distortion, as the higher order terms give a diminishing effect.

The distortion models use coefficients specific to the lens, which need to be found by calibrating the camera before use. Camera calibration can also be used to find the intrinsic and extrinsic parameters. One method of camera calibration that is able to estimate all of these parameters through a maximum-likelihood problem was established in 2000 by Zhengyou Zhang [21]. Zhang's method proposes that calibration can be done by taking a series of images of a pattern of known size, then posing the pinhole model equations as given in Equation (3.10) as a maximum-likelihood estimation problem. This method estimates the extrinsics and intrinsics. Estimating the distortion parameters is done in a similar manner, where the ideal image coordinates are calculated with the pinhole model, then compared to the calibrated coordinates giving an error known as the reprojection error. The reprojection error quantifies how accurate the camera calibration is, and can be minimized through optimization techniques.

3.2 Underwater Camera Modeling

If the camera model is inaccurate and produces images with many deviations from the truth, algorithms depending on the image might perform worse. Viz., localization indicating the AUV is closer to an object than it is, or object detection not being able to detect misshaped familiar objects. As such, it is important to use the proper equipment with proper calibration for the environment. This section aims to familiarize the reader with the motivations behind why underwater camera models are needed, before introducing the Pinax-model for underwater imaging.

Submerging a camera in liquid presents a new challenge, as the light behaves differently than in air. Physical properties of liquids degrade the light exponentially as it travels, resulting in hazy-looking images [5]. Turbidity and floating particles further worsen image quality underwater. Furthermore, to protect the camera, it is commonly placed in a watertight enclosure which can have more implications on light rays. A camera model that aims at resolving the latter issue is presented in Section 3.2.1.

Light changes speed when traveling from one medium to another. If the angle of attack is different than 90°, the change of speed causes the light to bend, known as refraction. The blue rays in Figure 3.5 visualize such refractions. Camera lenses are commonly placed behind flat viewports made of transparent materials such as glass. As the light from a scene hits the glass from different directions, the amount of refraction varies. If the variation in refraction is significant enough, the singular viewpoint assumption of the pinhole camera model is invalidated, as the light no longer converges to one point in the aperture. The underlying geometry of rays in such a system has been shown to correspond to an axial camera rather than an SVP pinhole camera [22, 23]. An axial camera is a camera where all light rays converge along a stretched-out line of points in the camera's normal direction, rather than in one singular point such as the focal point in the pinhole model.

3.2.1 The Pinax-model

Previous methods of modeling underwater cameras predominantly use the pinhole model, then calibrate the camera in-situ underwater [24–26]. This is a tedious process that requires precise calibration to be done in still and clear water. The Pinax-model is a camera model that provides accurate and efficient refraction correction for underwater imaging, without using in-situ underwater calibration [24]. While it can be shown that using an SVP calibration for an underwater camera can be sufficient at a fixed distance, the fall-off in performance increases rapidly as distance changes [24].

To formulate the Pinax-model a set of assumptions are made, mainly related to the physical configuration of the camera and the conditions of the environment. Assumption number one states that the camera should be placed such that the distance from the camera to the encapsulating flat port is small and near the optimal distance. In short, this assumption is sufficiently satisfied by placing the camera as close as possible to the glass. Secondly, it is assumed that the axis of the camera aligns with the normal of the glass surface, or that a correcting transform can be applied. Again, this assumption is satisfied by taking careful note of the placement of the camera. The third and fourth assumptions relate to knowing the refraction index of the glass and the water respectively. These indices can for instance be found in the product specification for the glass, and by using table values for water refraction.

Using the result that the camera satisfies an SVP pinhole model at the calibrated distance, the Pinax-model defines an offline method for generating a map of the refracted rays. A virtual pinhole camera $\mathcal{C}^v$ is defined for the camera, whose images correspond to the distorted images produced. Each point $\mathbf{p}_v$ of the virtual camera is then projected to points $\mathbf{m}_w$ on the Pinax plane in front of the camera, using the inverse projection of the pinhole model. Using the results presented in [22], a 12^{th} degree polynomial defines the forward projection from the Pinax plane to a point $\mathbf{m}_a$ on the inside of the flat port encasing the camera. Now all the points $\mathbf{m}_a$ correspond to the scene as it appears after refractions through the flat port, using the in-air calibrated pinhole model for the physical camera $\mathcal{C}^p$ now yields the refraction-corrected image. This procedure is illustrated in Figure 3.5.

Generating the mapping of the refracted rays is shown in [24] to be a time-consuming task. However, as this only needs to be done once for the setup, this procedure can be done offline and the full camera system can be deployed with the static map for real-time use. Calibrating the camera is also as simple as calibrating it as an SVP pinhole camera in the air, as the projection for the physical camera is only from the inner surface of the viewport to the camera. Thus, calibration can be done using popular and widely implemented methods such as Zhang's method, briefly covered in Section 3.1.2.

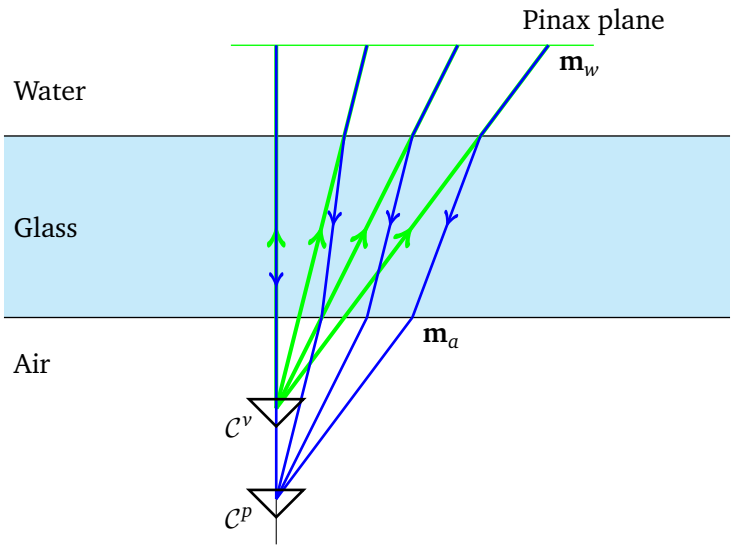


Figure 3.5: Projection from the virtual camera $\mathcal{C}^v$ to the Pinax plane, then back to the physical camera $\mathcal{C}^p$ by solving the 12^{th} degree polynomial presented in [22]. Note that the Pinax plane is placed further away from the glass, this distance is purely for visualization.

3.3 Feature Detection

When looking at a series of images, humans can easily recognize it as the same scene given that enough identifiers are present. These identifiers can range from the color of a wall to the shape of a rock. Recognizing an identifier depends on how well the person knows the scene. A major strength in state-of-the-art computer vision is that this process can be automated, fast, and even able to make matches between scenes a human would struggle with. In order to do so, algorithms known as feature detectors process images and produce feature descriptors that can uniquely identify parts of the image.

3.3.1 Properties of a Feature Detector

Due to inconsistency in lightning levels and hue, a scene undergoes multiple visual changes over time. This and other factors such as geometric changes in camera position give rise to the need of some desired properties of a feature detector. As stated by Tuytelaars et al. in [27] it would be ideal if the detector could use semantically meaningful parts of objects in the scene, however, this is infeasible as it would require high-level knowledge of the scene. With modern-day machine learning solutions this might become more feasible in time [28], but that discussion is left out here.

One can easily reason that an important property of a feature detector is its ability to repetitively produce the same results, even under changes in viewing

conditions. Tuytelaars states that repeatability is arguably the most important property and that it is mainly achieved in two ways [27]. Repeatability can be improved by making the detector invariant to transformations, or by increasing robustness to smaller changes such as image noise, for instance in the form of blur. Other ideal properties of a feature detector mentioned by Tuytelaars include distinctiveness, meaning that the underlying descriptors of a feature have high variance such that features are easily distinguished. A feature detector should inhibit the locality property, a feature should be represented by a small neighborhood in order to reduce the probability of occlusion, and allow for transformative effects between different views. As distinctiveness and locality are competing factors, a trade-off has to be made. Tuytelaars mentions other properties, but the final property discussed in this thesis is the number of features. Sufficiently many features should be detected such that even smaller objects have multiple features, meaning that the information of the image is more or less reflected by the extracted features.

3.4 A Selection of Feature Detectors

Humans tend to recognize scenes through a high level of semantic understanding, using global features such as specifically shaped objects or different vegetation. However, as these descriptors carry such a high level of semantic understanding, using them in a general algorithm would require the algorithm to have a similar understanding which is understandably complex. Feature detection algorithms rather use local features — commonly based on the pixel intensity of the image — such as corners and edges. Different approaches can be employed to detect such local features, in this section two feature detectors are introduced.

3.4.1 Harris Corner Detector

A simple yet efficient algorithm for detecting edge and corner features in images is the Harris-Stephens corner detector [29]. Based on the sum of squared differences (SSD) between image patches, the algorithm calculates a corner response function that indicates whether or not the region contains a corner.

Given an image $\mathbf{I}$ the intensity of the pixel at coordinates (x, y) is given by $\mathbf{I}(x, y)$. A local view into a region of an image, denoted a window, is given by $(x, y) \in \mathbf{W}$. Thus, the sum of intensities in the window $\mathbf{W}$ is

$$\sum_{(x_k, y_k) \in \mathbf{W}} \mathbf{I}(x_k, y_k) \tag{3.13}$$

Displacing the window by an amount $(\Delta x, \Delta y)$ the sum of intensities in the window is simply

$$\sum_{(x_k, y_k) \in \mathbf{W}} \mathbf{I}(x_k + \Delta x, y_k + \Delta y) \tag{3.14}$$

Subtracting the intensity of the displaced window from the normal window and squaring gives the change in intensity denoted as E_{SSD}, shown in Equation (3.15). Assuming the displacement is small, the intensity at each pixel in (3.14) can be approximated with a first-order Taylor expansion, (3.16).

$$E_{SSD} = \sum_{(x_k,y_k)\in\mathbf{W}} (\mathbf{I}(x_k,y_k) - \mathbf{I}(x_k+\Delta x, y_k+\Delta y))^2 \tag{3.15}$$

$$\mathbf{I}(x_k+\Delta x, y_k+\Delta y) \approx \mathbf{I}(x_k,y_k) + \frac{\delta \mathbf{I}(x_k,y_k)}{\delta x}\Delta x + \frac{\delta \mathbf{I}(x_k,y_k)}{\delta y}\Delta y \tag{3.16}$$

Substituting (3.16) in (3.15), leaving out (x_k, y_k) for conciseness and abusing that the approximation is accurate with small displacements, the SSD is now given by

$$\begin{aligned} E_{SSD} &= \sum_{(x_k,y_k)\in\mathbf{W}} \left(\mathbf{I}(x_k,y_k) - \mathbf{I}(x_k,y_k) + \mathbf{I}_x\Delta x + \mathbf{I}_y\Delta y\right)^2 \\ &= \sum_{(x_k,y_k)\in\mathbf{W}} \left(\mathbf{I}_x\Delta x + \mathbf{I}_y\Delta y\right)^2 \end{aligned} \tag{3.17}$$

Where $\mathbf{I}_x$ denotes the partial derivative of $\mathbf{I}$ with respect to x, see Appendix A.3. Equation (3.17) can be written in matrix form

$$\begin{aligned} E_{SSD} &= \sum_{(x_k,y_k)\in\mathbf{W}} \begin{bmatrix}\Delta x & \Delta y\end{bmatrix} \begin{bmatrix}\mathbf{I}_x \\ \mathbf{I}_y\end{bmatrix} \begin{bmatrix}\mathbf{I}_x & \mathbf{I}_y\end{bmatrix} \begin{bmatrix}\Delta x \\ \Delta y\end{bmatrix} \\ &= \sum_{(x_k,y_k)\in\mathbf{W}} \begin{bmatrix}\Delta x & \Delta y\end{bmatrix} \begin{bmatrix}\mathbf{I}_x^2 & \mathbf{I}_x\mathbf{I}_y \\ \mathbf{I}_y\mathbf{I}_x & \mathbf{I}_y^2\end{bmatrix} \begin{bmatrix}\Delta x \\ \Delta y\end{bmatrix} \\ &= \Delta\mathbf{x}\cdot\mathbf{M}\cdot\Delta\mathbf{x} \end{aligned} \tag{3.18}$$

$$\mathbf{M} = \begin{bmatrix} \sum\limits_{(x_k,y_k)\in\mathbf{W}} \mathbf{I}_x^2 & \sum\limits_{(x_k,y_k)\in\mathbf{W}} \mathbf{I}_x\mathbf{I}_y \\ \sum\limits_{(x_k,y_k)\in\mathbf{W}} \mathbf{I}_y\mathbf{I}_x & \sum\limits_{(x_k,y_k)\in\mathbf{W}} \mathbf{I}_y^2 \end{bmatrix} \tag{3.19}$$

where $\mathbf{M}$ is known as the structure tensor, or the second-moment matrix, describing the gradient in the neighborhood of a point. The gradient information is exactly what provides information about the composition of the image, i.e. in what direction there is a rapid change, as this indicates an edge.

$$R = \det(\mathbf{M}) - k\cdot\text{trace}^2(\mathbf{M}), \quad k\in[0.04, 0.15] \tag{3.20}$$

In [29] Harris suggests the corner response function in Equation (3.20), where k is found empirically. This representation avoids explicit calculation of the eigenvalues of M, however, it is worth noting that the eigenvalues represent the amount

of change in the direction of the eigenvectors. By inspection, the eigenvalues indicate that there is a corner if both values are large, an edge if one eigenvalue is close to zero, and a flat region if both eigenvalues are close to zero.

As the corner response function essentially uses the eigenvalues and vectors to indicate corners, rotating the image simply rotates the eigenvectors, which indicates that the Harris corner detector is invariant under rotation. The image derivatives are shift-invariant, meaning the corner detector is translation invariant. As long as the intensity change is constant over the entire image, the corner detector is also invariant to intensity changes. Scale changes of the image will change how much information each window contains and as such the corner detector is not invariant to scale, shown in Figure 3.6.

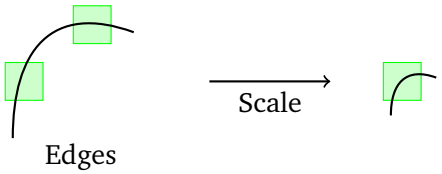


Figure 3.6: The curve on the left is detected as multiple edges, while the same curve in a down-scaled image is detected as a corner.

3.4.2 Scale-Invariant Feature Transform

In order to achieve a scale-invariant feature detector, some more convoluted methods need to be employed. The Scale-Invariant feature transform (SIFT) developed by Lowe in 1999 generates a scale space representation of an input image to detect features [30]. This section aims to give a brief introduction to SIFT and the concepts behind it.

A scale space representation of an image is simply a collection of the image at different scales. For each scale, the method generates multiple blurred images at increasing degrees of blur. All the different blurred images at one scale are denoted as an octave. Blurring is achieved by convolving the image with a Gaussian kernel of increasing σ, see Appendices A.1 and A.2. A resulting scale space for an image is shown in Figure 3.7.

Once the scale space for the image has been generated, the difference between images of increasing blur is calculated, known as the Difference of Gaussians (DoG). The DoG produces multiple layers of differences for each scale, where the highlighted areas correspond to regions of interest, see Figure 3.8. In essence, the DoG is a feature enhancement algorithm that accentuates regions of moderate frequency changes while attenuating low- and high-frequency regions, like a band-pass filter.

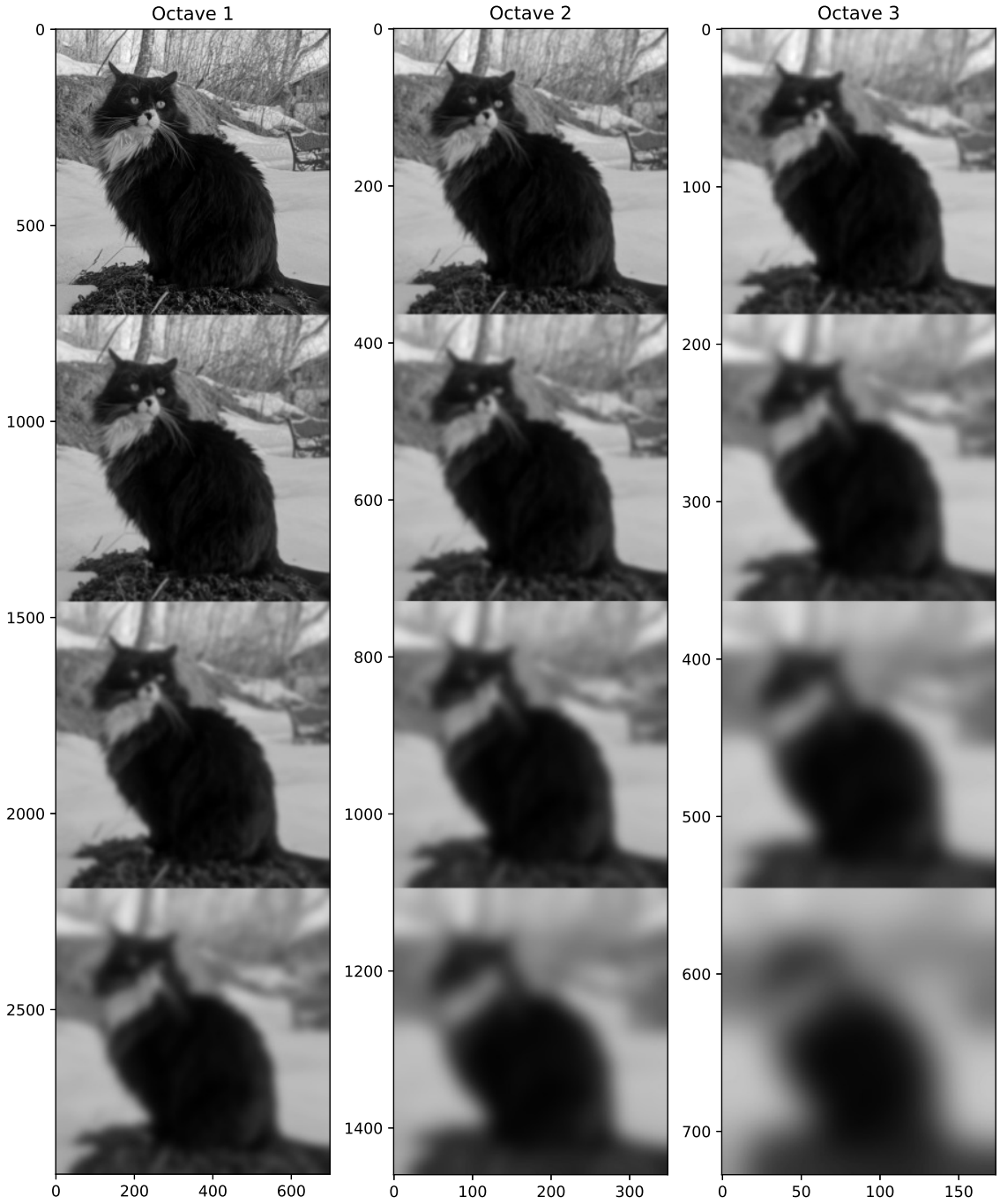


Figure 3.7: Three octaves of an image with varying scale and blur. Note the magnitude of the axes.

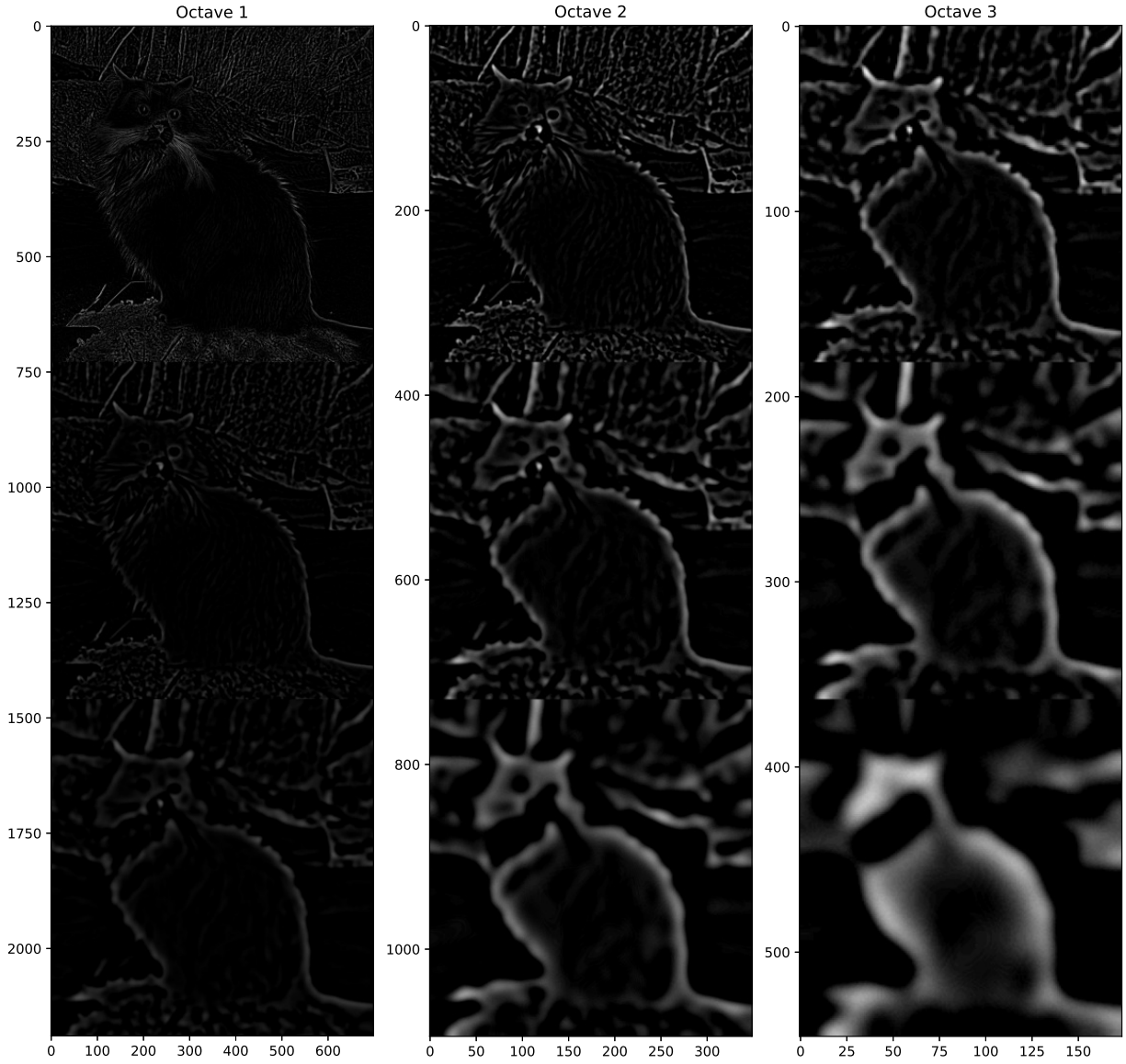


Figure 3.8: Difference of Gaussians for the scale space images in Figure 3.7. Each individual DoG is the difference between two neighboring images within the same octave.

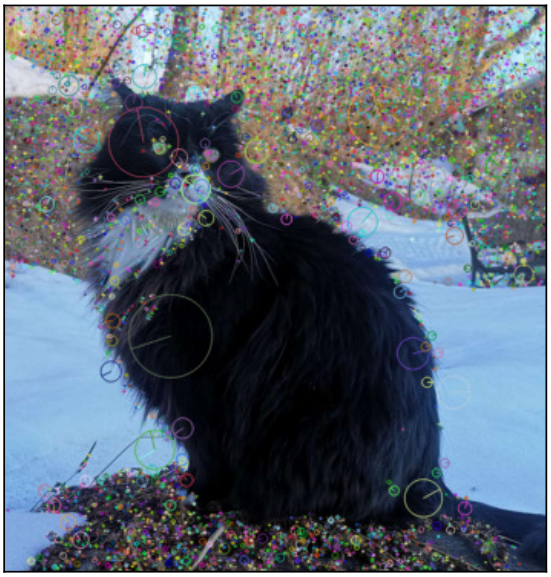


Figure 3.9: Detected SIFT keypoints. The size of the keypoint determines the scale at which it was found, and the small line indicates the canonical orientation.

Once the DoG has been calculated for all blurred images at each scale, a comparison is done to find the keypoints. By comparing each point with its local neighborhood the local maxima of a single image is found, this point is then compared to the corresponding neighborhood in adjacent scales. If the point remains the local maxima, the scale of the keypoint has been determined, and as such, information about the scale can be saved with the keypoint for future reference. Detected keypoints are shown in Figure 3.9 as circles, where the inner line depicts the orientation of the keypoint, while the size corresponds to what scale the keypoint was found at.

3.4.3 Feature Descriptors

Once a feature has been found in an image, a description of the feature is stored for later use. A feature descriptor might be as simple as storing the image patch that constitutes the neighborhood of the detected feature. Having detected a corner using e.g. the Harris Corner Detector described in Section 3.4.1, the feature is stored as the image patch centered at the corner, resulting in something like the highlighted patch in Figure 3.10.

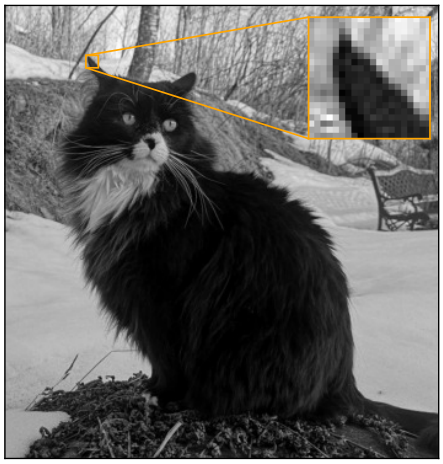


Figure 3.10: Example of an image patch that could correspond to a corner feature.

In Section 3.4.1 it is briefly stated that the Harris Corner Detector is invariant to rotation, as corners detected by this method come with corresponding eigenvectors indicating the direction of change. Normalizing the orientation of the descriptor patch can then be done by rotating the patch so the eigenvector corresponding with the greatest change aligns with i.e. the x-axis. Detected corners in the new image can then be rotated according to their eigenvectors and then compared to

the saved feature descriptor to find potential matches. This is a simplified version of the Multi-scale Oriented PatcheS (MOPS) descriptor as described in [31].

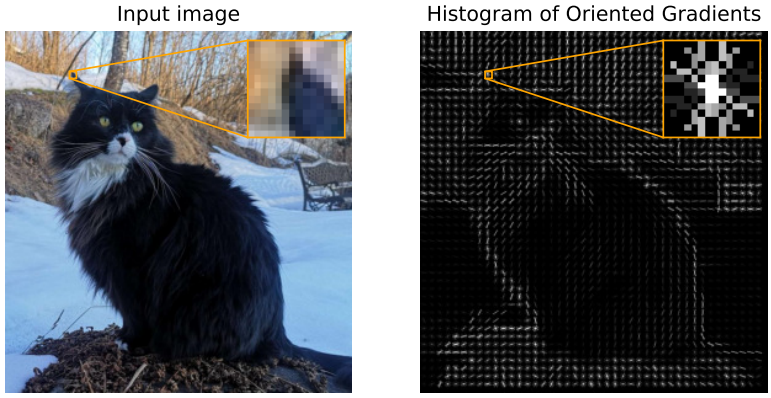


Figure 3.11: Histogram of gradients of an image, generated using Scikit-Image *hog()* function [32].

The aforementioned feature descriptor, MOPS, is invariant to rotation and translation, however, much like the corner detector it is based upon, it is not invariant to scale. In Section 3.4.2 the scale-invariant feature detector SIFT was introduced. Features detected through SIFT are stored using corresponding SIFT feature descriptors, which give rotation, translation, and scale invariance. At each pixel, both gradient orientation and magnitude are calculated by means of local pixel differences. The gradient directions for all pixels in the neighborhood are compared in a histogram, and the peak value determines the keypoints canonical orientation. A histogram of gradients for an image is visualized in Figure 3.11. The final SIFT feature descriptor thus consists of the position of the keypoint, the scale at which it was found, the canonical angle from the histogram of gradients, and the size of the neighborhood.

3.5 Feature Matching

Once features have been extracted and stored using feature descriptors, finding pairings of which features might be related to each other is the next step. A naive approach would be taking the patch around a detected keypoint, to then scan a second image for that same patch. This method could work in the absolutely most basic cases but is not advisable. A search in this manner would require the exact patch to be present in order to be detected, which is a highly unlikely situation. Thus, feature matching is formulated as a similarity problem, where features are matched as long as they are similar enough, rather than just identical.

One approach to feature matching is to use a distance measure. Assuming that the feature descriptors are designed such that they can be used in distance measurements, finding the closest match is equivalent to finding the two describing vectors with the shortest distance between them [33]. Let $\mathbf{d}_i$ denote the descriptor of the i-th feature, then the Euclidean distance between two features is given by the Euclidean norm

$$\text{distance}(\mathbf{d}_1, \mathbf{d}_2) = ||\mathbf{d}_1 - \mathbf{d}_2||_2 \tag{3.21}$$

where a feature is accepted if the distance is less than a chosen threshold. For large multidimensional features, finding the closest match through a nearest neighbor search is an efficient alternative. This is described for SIFT features in [30], and a nearest neighbor algorithm is described in more detail in Section 4.1.2.

3.6 Stereo Cameras

A single camera will capture 2D projections of 3D scenes, which leads to a loss of depth information. By using two or more imaging sensors with a known relation to each other, geometrical considerations can be used to extrapolate depth information of the scene. The methods described in this section can be done manually with two separate cameras, however, a multitude of fully integrated solutions exist, such as the Intel Realsense series of cameras [34].

3.6.1 Triangulation

Given a set of corresponding points in two images, a linear approximation can be made to estimate the 3D coordinates of the point P, known as triangulation. Using the camera's projection model, rays from the projected points $\mathbf{x}_L$ and $\mathbf{x}_R$ are available. Intuitively the original point P is then located at the intersection between the rays, as shown in Figure 3.12. In an ideal world, this would be the case, however, due to inaccuracies and noise in the real world, an approximation has to be sufficient. This section covers a triangulation algorithm presented by Longuet-Higgins in [35] that imposes constraints on the projected rays to approximate the coordinates of P.

Let $\tilde{\mathbf{X}}$ be the homogeneous coordinates of P as seen from the left camera, assumed to align with the world frame. The projected point $\tilde{\mathbf{x}}_L$ is then given by (3.10), with intrinsics $\mathbf{K}_L$ and extrinsics equal to the identity, due to no rotation nor translation. $\mathbf{P}_L$ denotes the full projection matrix of the left-hand camera.

$$\lambda\tilde{\mathbf{x}}_L = \mathbf{K}_L\mathbf{I}_{4\times4}\tilde{\mathbf{X}} = \mathbf{P}_L\tilde{\mathbf{x}}^w \tag{3.22}$$

The projected point $\tilde{\mathbf{x}}_R$ as observed by the right camera, placed with a known transformation $\mathbf{T}$ relative to the left camera, is then given by

$$\lambda\tilde{\mathbf{x}}_R = \mathbf{K}_R\mathbf{T}\tilde{\mathbf{x}}^w = \mathbf{P}_R\tilde{\mathbf{x}}^w \tag{3.23}$$

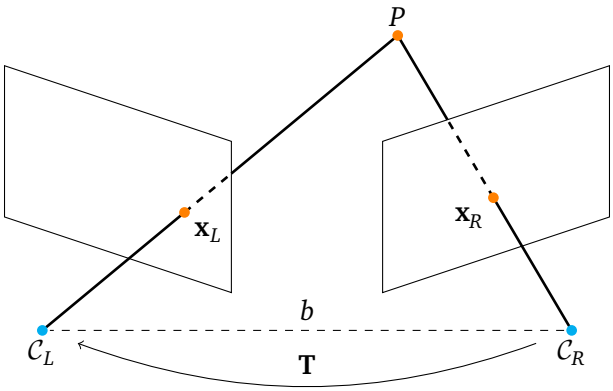


Figure 3.12: Visualization of the triangulation problem. The rays going from the cameras $\mathcal{C}_{L,R}$ might not exactly intersect at P. The distance b between the two cameras is commonly denoted as the baseline.

The following derivation is equal for both left and right, hence the subscript L,R is left out. Denoting the i-th row vector of projection matrix $\mathbf{P}$ as $\mathbf{p}_i$ where $i \in [1,2,3]$, equations (3.22) and (3.23) may be written as

$$\lambda x' = \mathbf{p}_1 \tilde{\mathbf{x}}^w, \quad \lambda y' = \mathbf{p}_2 \tilde{\mathbf{x}}^w, \quad \lambda = \mathbf{p}_3 \tilde{\mathbf{x}}^w \tag{3.24}$$

Eliminating the unknown scaling factor λ by inserting the third equality in the first two, results in the two linear equations

$$\begin{aligned} x' \mathbf{p}_3 \tilde{\mathbf{x}}^w &= \mathbf{p}_1 \tilde{\mathbf{x}}^w \\ y' \mathbf{p}_3 \tilde{\mathbf{x}}^w &= \mathbf{p}_2 \tilde{\mathbf{x}}^w \end{aligned} \tag{3.25}$$

Rearranging (3.25) for both cameras and grouping them together results in the system of linear equations given by

$$\mathbf{A}\tilde{\mathbf{x}}^w = \mathbf{0} \tag{3.26}$$

where $\mathbf{A}$ is the 4×4 matrix

$$\mathbf{A} = \begin{bmatrix} x'_L \mathbf{p}_3 - \mathbf{p}_1 \\ y'_L \mathbf{p}_3 - \mathbf{p}_2 \\ x'_R \mathbf{p}_3 - \mathbf{p}_1 \\ y'_R \mathbf{p}_3 - \mathbf{p}_2 \end{bmatrix} \tag{3.27}$$

In the ideal case, this system can be solved for $\tilde{\mathbf{x}}^w$ up to the unknown scaling factor λ. However, as previously stated, the relation presented in Equation (3.26) does not hold precisely with noisy real-world cameras, an approximate solution will suffice [36]. This can be achieved for instance using a linear least-squares method such as Singular Value Decomposition (SVD). As this section only intends to give intuition on the main ideas behind triangulation, the SVD solution is left to the reader.

As a final note on the linear approximation to triangulation, it is worth noting that the linear method is heavily affected by the initial pairing of point correspondences and their levels of noise. As such it might be advisable to use the result of Equation (3.26) as an initial guess for a nonlinear iterative estimation method, such as Levenberg-Marquardt [36].

3.7 RGBD Cameras

An RGBD camera is the combination of an RGB camera and a depth sensor in one package. The camera captures a colored image of the scene, while the depth sensor measures distances. Given that both sensors have similar resolutions, the sensor returns an image along with corresponding depths at each pixel. As the pinhole camera has been covered somewhat extensively, this section will briefly introduce a selection of depth-sensing methods.

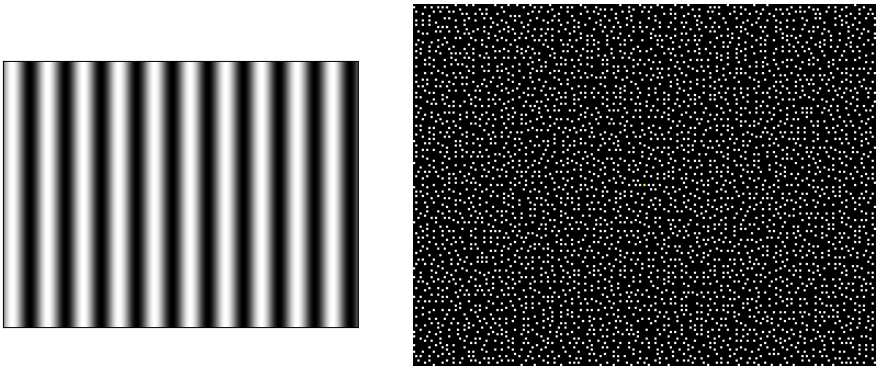
3.7.1 Structured Light

One category of depth sensors uses structured light to extract depth information from a scene. Structured light sensors are set up like stereo cameras and contain one projector and one detector. The projector projects light in a structured pattern onto the scene, which is then observed from a different viewpoint by the observing camera. Patterns used for structured light sensors are typically straight lines or repeating dot patterns as seen in Figure 3.13. When the known pattern of light is projected onto a 3D object, the pattern appears deformed to the observing camera. Comparing the deformed pattern to the original gives a disparity map which is used to compute the depth.

Under the reasonable assumption that the camera and projector are only displaced horizontally with baseline distance b, the disparity is reduced to a difference in horizontal values. A full disparity map for the projected pattern can be computed for each pixel, giving the disparity $m(x, y)$ for the pixel located at (x, y). As disparity is given in pixel coordinates, the focal length is also given in pixel units, i.e. $f_p = \frac{f}{s}$. Depth for the pixel at (x, y) can now be computed as $d = \frac{b \cdot f_p}{m(x,y)}$. A popular structured light sensor is the Kinect developed by Microsoft [37].

3.7.2 Time Of Flight

LIght Detection And Ranging (LIDAR) is another well-known method of measuring depth in an environment by using the time of flight for light. A LIDAR sensor works by emitting a pulse of light toward the scene. The light travels through the air at speed c and is reflected back to the sensor when hitting objects. When the reflected light hits the LIDAR sensor, the traveled distance l is given by the time it took for the reflected light to hit the sensor, i.e. $l = c \cdot t$. As the light had to travel



(a) Structured light pattern of vertical lines with varying intensity according to a sinusoidal signal.

(b) Parts of the structured light pattern of a Microsoft Kinect sensor, uncovered by Reichinger in [38].

Figure 3.13: Examples of structured light patterns.

this distance twice, once from the sensor to the object and once from the object to the sensor, the distance to the scene is simply given by $d = \frac{l}{2}$.

3.8 Scene Reconstruction

When depth information has been gathered for a large portion of the scene as observed by a camera, a 3D representation can be generated. This is done by augmenting the image coordinates with their corresponding depth value, resulting in each triangulated point being represented in 3D. Storing all the triangulated points in a structured manner is referred to as a point cloud. Plotting each point in a 3D-coordinate system will then give a sparse representation of the captured scene's surfaces.

A point cloud can be enhanced further with more information about each point. While getting a 3D representation of the scene in itself is useful enough, appending information such as color to the points is a natural choice. Estimates of scene semantics can also improve the information yielded by each point, such as an estimate of the surface normal in the neighborhood of each point, this is further explored in the following chapter.

Chapter 4

Point Clouds

A point cloud produced by triangulation with an optical sensor is a sparse representation of an observed scene. The higher the density of the points, the more accurate the scene appears. However, imaging sensors and reconstruction methods are prone to noise and faulty measurements. In order to provide semantically meaningful information the point cloud has to be processed.

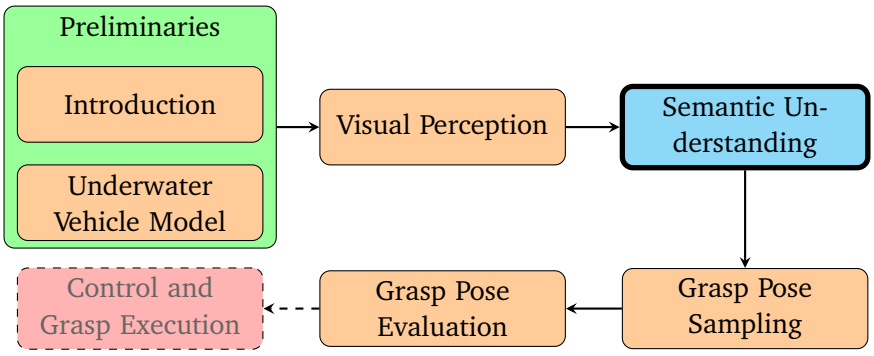


Figure 4.1: This chapter marks the step of generating semantically meaningful information for the autonomous system.

This chapter introduces point cloud processing at different stages. Starting with preprocessing in Section 4.1 a quick method of removing invalid values is presented in Section 4.1.1. Thereafter, a data structure for efficiently representing a point cloud in a search tree is introduced in Section 4.1.2, which is followed by a brief overview of a statistical outlier removal method for denoising in Section 4.1.3. Filtering methods for reducing the complexity of a point cloud are presented in Sections 4.1.4 and 4.1.5. Section 4.2 introduces two methods of extracting semantic information from a scene represented by a point cloud, namely, a surface normal estimation algorithm in Section 4.2.1 and a segmentation algorithm in Section 4.2.2.

4.1 Preprocessing

A raw point cloud generated from stereo camera triangulation as described in Section 3.8 can be full of artifacts, noise, and wrongly triangulated pixels. To combat this issue some preprocessing methods are popularly employed. This section covers the preprocessing steps used for this thesis, as well as some methods that should be considered when implementing for field testing.

4.1.1 Removing Invalid Values

Erroneous sensor measurements could lead to invalid point values in point clouds. Such measurements can occur if regions of the scene/image are outside the range of a depth sensor, where points commonly get placed at infinite depth. This can also occur if a point is erroneously triangulated. When using an imaging sensor paired with a depth sensor to produce depth information, the resolution of the two sensors might not always be the same, and in such instances, depth information is only available for parts of the scene. This section emphasizes that albeit simple and fast, removing the invalid values is an important step in the process.

A point with an invalid value is typically denoted with one or more coordinates being infinite or not-a-number (NaN). As a point cloud is essentially a list of points that contain their individual coordinates, removing invalid values is as simple as iterating over the list and then removing the points with invalid values.

4.1.2 k-Dimensional Tree

Semantic understanding of a point cloud often requires knowledge about neighboring points, as the geometry of a singular point means little to nothing alone. Viz., estimating a surface can not accurately be done with just one point. Hence, it is desirable to have a search method that is able to quickly find the k nearest neighbors (kNN) of a given query point.

As the nearest neighbor search is used for multiple purposes, and might therefore be repeated often, it is desirable that when querying for kNNs a re-computation of distances is avoided. Thus the need for a data structure able to convey distance information in a fast way is prudent. The k-Dimensional tree (kD tree) is one such data structure, presented in 1975 by Jon Louis Bentely [39]. A kD tree is a space-partitioning data structure that organizes k-dimensional data points in a search tree, where nodes close to each other in the tree correspond to points close in the k-dimensional space.

Binary Tree

To understand the structure of a kD tree, a brief introduction to the binary tree is included. A binary tree is a data structure consisting of nodes, represented by a

value, for instance integers. Each node has a maximum of two children, denoted the left or right child. When constructing a binary tree the value of the new node is compared to the existing one, if it is greater, the new node becomes the right child, and if it is lower it becomes the left child. Thus, one simply follows the path to the left or right until the desired value is reached. If a dead end is reached before the desired value is reached it is indicative of the value being absent from the tree.

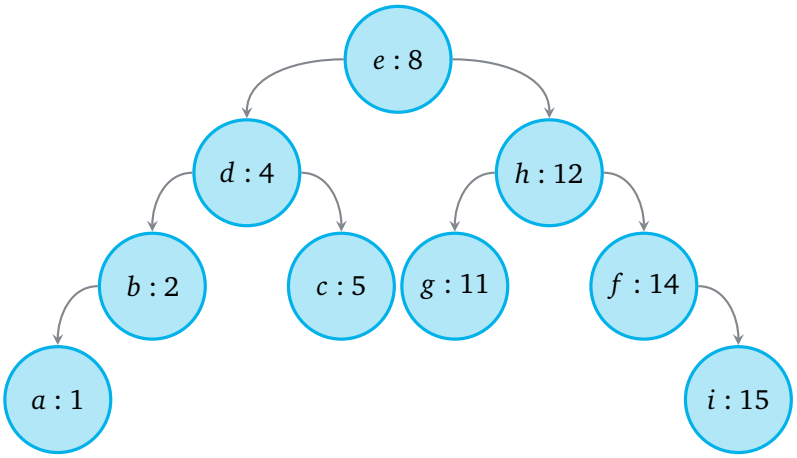


Figure 4.2: An example of a balanced binary tree.

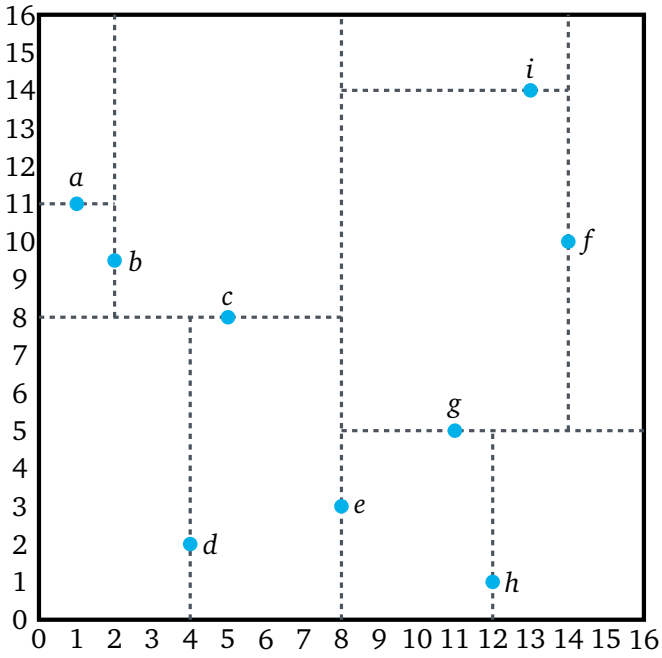
The binary tree in Figure 4.2 has been balanced, meaning the depth has been reduced to the minimum. A worst-case scenario of the binary tree in Figure 4.2 would be starting with node a, then having each new node being inserted in incrementing fashion, giving the tree a-b-d-c-e-g-h-f-i, which would have a depth of 8 rather than 3 in the balanced case.

Building a k-Dimensional Tree

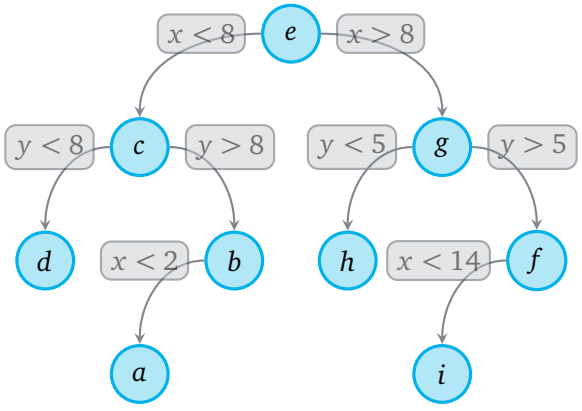
A kD tree takes the idea of a binary tree further. In a kD tree, each node is represented by a multidimensional point. The root node generates a separating hyperplane on the first dimension where the left subtree represents points that have lower values, and the right subtree has higher values. For each subtree, the dimension of comparison is changed, so if the root node does comparisons in the first dimension, children of the root node perform comparisons in the second dimension, and so on. Each subtree then generates its separating hyperplane in the dimension of comparison, and the full tree is generated [40].

Searching a k-Dimensional Tree

The process of querying a kD tree is similar to the process of building it. Starting at the root node, the query point makes comparisons in the dimension corresponding to the one used to separate at that level, iterating all the way to a leaf node. Once



(a) Two dimensional point cloud with separating hyperplanes visualized.



(b) Tree representation of the kD tree.

Figure 4.3: kD tree for two dimensions. First separating hyperplane separates on x-axis, then separation alternates between y- and x-axis respectively. ©2021 Jonathan Viquerat, with modifications [41].

a leaf node is reached a comparison is done, if the distance is shorter than the current shortest distance, it is updated. The algorithm then recursively iterates backward to the root of the tree, checking whether the opposing hyperplane has points closer to the query point or not.

The distance calculations done when querying a point can be saved during a search. This allows for the values to be looked up when performing another search later, reducing the need for re-computations. The algorithm can also be expanded to computing the kNNs by keeping a record of the set of closest matches, not just the single closest match [42].

4.1.3 Denoising

In the ideal case, all points in the point cloud are perfectly triangulated, and their position is accurately represented in the 3D plot. However, this is most likely never the case outside of simulation, and as such, methods for removing noise are needed.

A statistical outlier removal algorithm removes points that do not fit within a statistical measure of the point cloud. One such method described by Rusu in [43] iterates over the point cloud twice. In the first iteration, the average distance for each point to its kNN is calculated. After the first iteration, the algorithm calculates the mean μ and standard deviation σ of the nearest neighbor distances. A distance threshold is then formulated as $\mu + \sigma$, then in the second iteration over the point cloud, points that fall outside the threshold are removed. In [43] experimental data shows that this algorithm performs well in datasets where approximately 1% of points are considered noise.

4.1.4 Random Sample Consensus

Not only do point clouds potentially contain noisy and faulty values, but they can hold a lot of information that is strictly unwanted for the desired application. Say a robot wants to pick up an object sitting on a table. Given a point cloud of the scene, only a subset of the points belongs to the object, while a majority belongs to the table. An algorithm for grasping the object need only consider grasping poses for the points belonging to the object, and not the entire table surface.

One method for filtering out unwanted data is through Random Sample Consensus (RANSAC) developed by Fischler and Bolles in 1981 [44]. RANSAC is an iterative method of estimating a mathematical model fitting a provided dataset. The core assumption behind RANSAC is that the provided dataset consists of points that can fit a parametric model (inliers) and points that can be considered noise (outliers). RANSAC roughly consists of two main steps, which are iteratively repeated until either a maximum amount of iterations has passed, or a model with sufficiently good fit has been found.

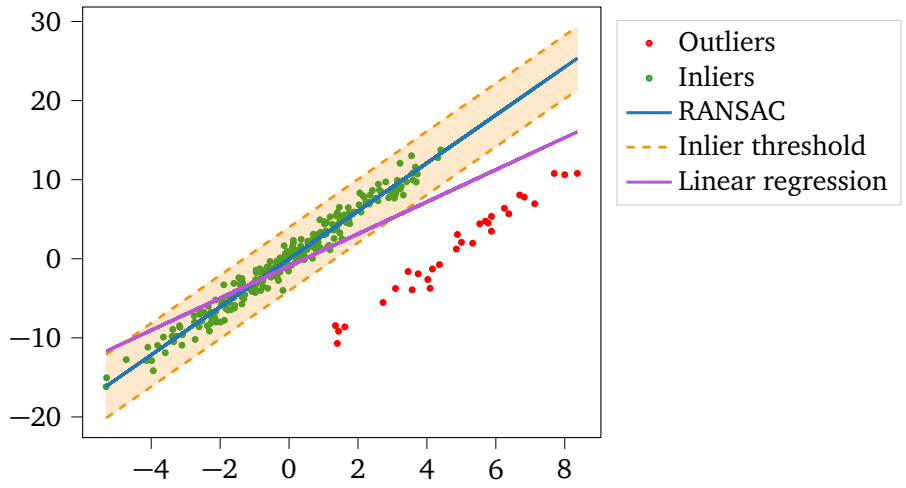


Figure 4.4: Comparison of RANSAC and normal linear regression on a dataset containing a noisy line with outliers. Observe how the linear regression model heavily skews towards the outliers in an attempt to fit the model to all the points.

The first step of RANSAC is to randomly select a subset of points from the provided dataset. A parametric model is then fit to the selected subset, which constitutes the second step. The order of the fitted model is in general selected from the size of the selected sample, but expansions to the algorithm allow for a prior model order to be provided. In 1999 Torr and Zissermann presented the Maximum Likelihood SAC (MLESAC) variation of RANSAC, which selects a model based on a maximum likelihood [45], allowing for a prior to be provided for estimation of datasets with some predetermined known structure.

Fischler and Bolles identify three main parameters that define different thresholds in the RANSAC algorithm. The first threshold parameter is the error tolerance used to decide whether or not a point is compatible with the estimated model, i.e. the maximum distance a point can lie from the model to be considered an inlier in the case of Figure 4.4. A check is performed using this threshold to identify all the inliers of that model. The subset of points that define a model and the points that are inliers for that model make up the consensus set. Secondly, a parameter to limit the amount of model parameter sets to try is defined. This ensures that the algorithm is guaranteed to terminate if a sufficiently large consensus set has not been found in the specified amount of iterations. Thirdly, a threshold that defines the desired minimum amount of inliers in a consensus set. In this way, a model is accepted as correct if the consensus set is bigger than the threshold.

There is no maximum number of iterations guaranteed to give the desired result. However, an estimate can be found from the desired probability of the RANSAC algorithm giving at least one good result, the steps are as follows: (1) pick a desired probability p that at least one useful result is found. For simplicity

let a useful result be a result in which all n randomly sampled points are inliers. (2) calculate the inlier fraction using Equation (4.1). In situations where w is not known exactly, a rough estimate will suffice.

$$w = \frac{\text{Inliers}}{\text{Total points}} \tag{4.1}$$

(3) the probability that n points are inliers is then given by w^n, consequently $1-w^n$ is the probability that at least one of the points is an outlier. Thus, the probability of the algorithm not returning a useful result in k iterations is given by

$$1 - p = (1 - w^n)^k \tag{4.2}$$

Furthermore, by taking the logarithm of both sides in Equation (4.2), the estimated maximum iterations needed for the probability p of picking a useful set is

$$k = \frac{\log(1-p)}{\log(1-w^n)} \tag{4.3}$$

RANSAC is robust, computationally efficient, and capable of finding models even in datasets heavily affected by outliers. In Figure 4.4 a RANSAC estimate is compared to a normal linear regression, and it is clear that the linear regression estimate is heavily influenced by the outliers. As RANSAC samples only a random subset to perform thresholding checks on, the influence of the outliers is heavily reduced. However, it is worth noting that the algorithm is heavily affected by the threshold parameters. There is no upper bound on iterations unless specified, and there is no guarantee that the sampled points will lead to an optimal model. If the inlier threshold was halved in Figure 4.4, a lot of the inliers would fall outside the threshold.

4.1.5 Downsampling

Depending on the resolution of the sensor or method used to generate a point cloud, the density varies. This is sort of analogous to the variation in scale seen in Figure 3.6, Naturally, computation time complexity increases with the number of points in the cloud, thus it is desirable to reduce the number of points without loss of defining features to speed up computations. This process is known as downsampling.

VoxelGrid filtering is a technique used for point cloud downsampling. To perform VoxelGrid filtering the 3D space holding the point cloud is divided into cubic cells called voxels. A voxel thus holds all the points in the cloud that lies within that region of the 3D space. Downsampling is then performed by calculating the centroid of all points contained in each voxel, then replacing those points with the centroid. The larger the voxels, the greater the magnitude of downsampling, as each voxel will hold more points. Reducing the number of points in this way

generally preserves the structure and shape of the point cloud, as each region is represented by its centroid. However, a loss of resolution naturally leads to some loss of information. Thus, a trade-off has to be made on information loss versus the grade of downsampling by specifying the voxel size.

4.2 Understanding the Scene

Once the point cloud has been properly pre-processed, it is ready to be processed further for the extraction of semantic information. In this section, two methods of capturing the underlying geometric properties of the point cloud are presented. Firstly, a method of estimating the surface normals is introduced. Then, a region-growing-based segmentation method is presented.

4.2.1 Surface Normal Estimation

The full point cloud gives a rough representation of the observed scene but is lacking in local describing features. A safe assumption for the point cloud is that points very close to each other form a surface, these points might not belong to the same object, but they roughly form a surface nonetheless. In order to accurately describe the geometry of an underlying surface it is important to estimate its orientation in space. The k points $\mathcal{P}^k$ closest to a query point $\mathbf{p}_q$ can be found using a kNN algorithm such as the one described in Section 4.1.2. Once the neighborhood $\mathcal{P}^k$ has been determined, algorithms can be used to determine the underlying structure of $\mathbf{p}_q$.

Estimating the surface normal of a point can be done by looking at characteristics of its local neighborhood $\mathcal{P}^k$. The 3D tangent plane fitting algorithm, presented by Berkmann et al. in [46] uses such local considerations to estimate a plane tangent to the centroid of $\mathcal{P}^k$. A tangent plane consists of three basis vectors, two pointing in the directions of the neighborhood, and a third that is perpendicular to the two others, this is the normal vector. The algorithm defines the local first-order surface covariance matrix $\mathbf{C}$ of the neighborhood $\mathcal{P}^k$ as

$$\mathbf{C} = \frac{1}{k}\sum_{i=1}^{k}(\mathbf{p}_i - \bar{\mathbf{p}}) \cdot (\mathbf{p}_i - \bar{\mathbf{p}})^\top \tag{4.4}$$

where $\bar{\mathbf{p}}$ is the centroid of $\mathcal{P}^k$ given by

$$\bar{\mathbf{p}} = \frac{1}{k}\sum_{i=1}^{k}\mathbf{p}_i \tag{4.5}$$

$\mathbf{C}$ is a symmetric positive semi-definite matrix, hence all the eigenvalues are real and non-negative, i.e. $\lambda_i \in \{\mathbb{R}|\lambda_i \geq 0\}, i = 1,2,3$. The eigenvectors $\mathbf{v}_i$ form an orthogonal frame, where the two eigenvectors $\mathbf{v}_{1,2}$ corresponding to the largest

eigenvalues $\lambda_{1,2}$ form a plane tangent to $\mathcal{P}^k$ at $\mathbf{p}_q$. The frame spanned by the eigenvectors corresponds to the principal components of $\mathcal{C}$, and as such the eigenvector $\mathbf{v}_0$ corresponding to the smallest eigenvalue λ_0 is an approximation of the normal $\mathbf{n}$ to the tangent plane, i.e. the surface normal at $\mathbf{p}_q$ [47].

The frame formed by the principal components of the neighborhood $\mathcal{P}^k$, i.e. the eigenvectors, is ambiguous in the sense that there is no general way of solving for the sign of $\mathbf{n}$. As such all the estimated surface normals for the full point cloud might not be consistently oriented. For a full point cloud, made by a 3D scan from all directions of an object, rectifying this is difficult [47]. However, as the point clouds generated by a stereo camera generally have a known viewpoint position $\mathbf{v}_p$, surface normals can be corrected towards the viewpoint. As any surface normal pointing away from the viewpoint would be considered as being behind an observed object, this is a non-feasible normal direction. Correction is done by making sure the surface normals satisfy the equation

$$\mathbf{n} \cdot (\mathbf{v}_p - \mathbf{p}_i) > 0 \tag{4.6}$$

4.2.2 Segmentation

In the context of 3D point clouds, segmentation is the act of defining which points belong to which objects/regions in the scene. The clustering of a point cloud into the different segments that constitute the scene provides vital contextual information. This section aims to inform the reader why segmentation is a powerful tool, then provide an example of a segmentation algorithm.

Take for instance a point cloud representation of a cup on a table. Given that the segmentation algorithm properly manages to separate the two, detecting the cup would be reduced to comparing both segments of the cloud to an already stored cloud representation of the cup. Comparing two point clouds and measuring their similarity is a well-researched topic known as point cloud registration, however, as it is not covered in this thesis, the reader is referred to a thorough survey on the state of point cloud algorithms by Huang et al. [48]. When lacking prior knowledge of the object — or in cases where it is poorly represented by the point cloud — registration algorithms are limited.

Without prior knowledge of the present objects, segmentation nevertheless provides valuable information for autonomous systems. While registration algorithms might have limited potential, object detection/recognition algorithms can provide beneficial information. As 3D scanners such as LIDAR and Kinect have been widely employed in the last decade, a multitude of segmentation approaches have been made. Nguyen et al. performed a survey of methods in 2013 [49] and distinguished segmentation algorithms into multiple categories such as edge-based methods, region-based methods, and model-based methods. For the purpose of this thesis, only region-based methods have been considered, but it is

worth noting that model-based machine learning methods are rapidly increasing in popularity, such as Microsoft's newly published and popular Segment Anything model for images [28].

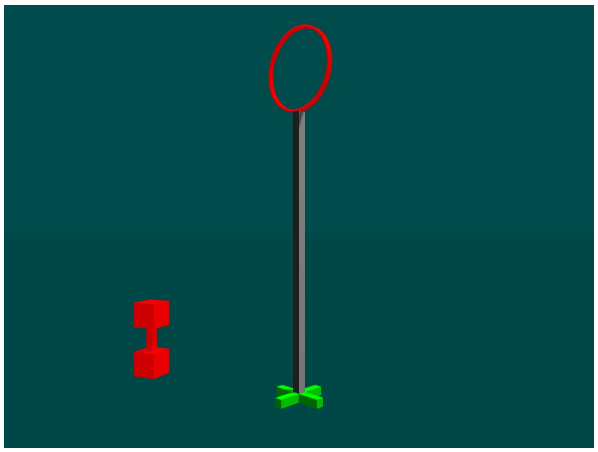


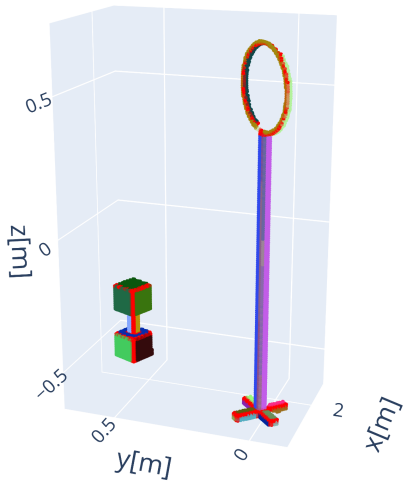
Figure 4.5: Test scene used for comparison of region-growing segmentation.

Region-growing segmentation methods start by selecting a point, then based on certain criteria iteratively adds neighboring points to the current set. When no more points pass the criteria for being added to the set, the region is considered fully grown, and a new point is chosen as starting point for the next region. This is repeated until all points have been added to their corresponding regions. Depending on the algorithm, a thresholding can be done on the regions to determine if it sufficiently big to be considered a proper segment.

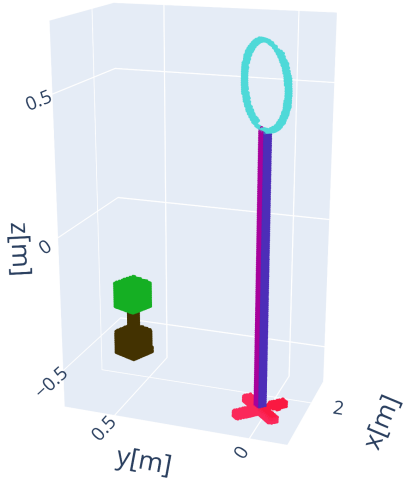
There are a wide variety of criteria that can be used for defining whether a point belongs to a region or not. The criteria used in the method implemented for this thesis are surface normal direction, curvature, distance, and optionally color. Most of these criteria are self-explanatory, but curvature is a measure that requires some explanation. As was the case with the surface normal, one method of inferring curvature is by estimating it from the local neighborhood of points. As it happens, this information is held by the same covariance matrix $\mathcal{C}$ as in Equation (4.4). Recall that the eigenvalue corresponding to the surface normal, λ_0, indicates the rate of change in the direction of minimum variance. The change of curvature in the neighborhood $\mathcal{P}_k$ can then be estimated as the ratio between the minimum eigenvalue and the sum of the eigenvalues

$$\sigma_{\mathbf{p}} = \frac{\lambda_0}{\lambda_0 + \lambda_1 + \lambda_2} \tag{4.7}$$

This ratio can be understood as the variation in the direction of $\mathbf{n}$ relative to the total variation in the direction of all eigenvectors. High values of $\sigma_{\mathbf{p}}$ indicate a large variance in the normal direction, while a low value of $\sigma_{\mathbf{p}}$ indicates that the



(a) Region-growing segmentation. Regions are color-coded and red points correspond to points that do not belong to a specific region.



(b) RGB region-growing segmentation. Regions sufficiently close in color and space have the same color. Note that while the long pole is black, reflected lightning makes the color difference too big, as seen in Figure 4.5. For the left object, the distance threshold is not satisfied, hence it is split into two segments despite the color being sufficiently close.

Figure 4.6: Comparison of region-growing segmentation with and without color comparison.

points in the neighborhood lie in the tangent plane of point **p**, indicating low curvature [47].

To summarize, the region-growing segmentation algorithm starts with selecting the point with minimum curvature, which according to [47] helps reduce the number of segments, and adds it to a set of seeds. The neighborhood $\mathcal{P}^k$ of the starting point is found and a test is done on the normal at each point. If the angle between the normals is lower than a threshold value, the point is added to the current region. If RGB segmentation is specified, a color check is performed here as well. For the point to be added to the region the difference in color needs to be below a certain threshold. Next, each point in the neighborhood is tested on curvature, if the curvature is below a threshold, the point is added to the seeds. This concludes the checks for the starting point, which is then removed from the set of seeds. The next seed is picked from the set and the same checks are performed. When the set is empty it means the region has been fully grown, a new starting point is found and the process is repeated. In the case of RGB segmentation, a merging step takes place after all the segments have been classified. A distance threshold determines if two segments are close enough to be merged, if they are, the difference in average color for the two segments has to pass a last threshold to merge. An example of a cloud segmented using regular region-growing segmentation is shown in Figure 4.6a, while the color criteria have been added in Figure 4.6b.

Chapter 5

Grasp Pose Sampling

When a human is tasked with picking up an object, a decision about how to pick it up is made just by taking a quick gaze at it. This intuition is built up over years of picking up objects, all the way from being a toddler. However, putting human intuition into an algorithm for use by a robot is a complicated — nigh impossible — process, a generalized grasping algorithm that performs just as well as a human has yet to be made. Still, researchers keep working towards it, as it would change the robotics field drastically [50].

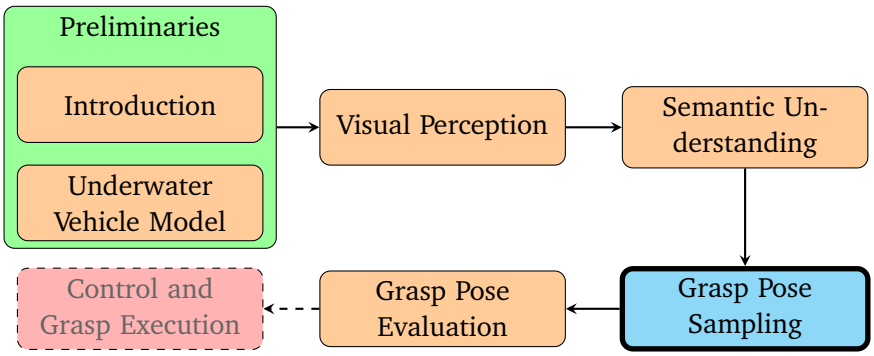


Figure 5.1: This chapter marks the step in the process where grasp pose candidates are sampled. This chapter presents two methods of doing so, along with some grasp pose semantics used.

Multiple strategies can be utilized when generating candidate poses for a grasping algorithm. This chapter will cover two slightly different approaches for sample pose generation. Before delving into specifics about sampling, Section 5.1 introduces the different terms used when talking about grasping, as well as the gripper itself. Then, Section 5.2 presents the first sampling method based on uniform sampling of points in the point cloud, before another sampling method based upon point cloud characteristics is presented in Section 5.3

5.1 Grasp Pose Semantics

This section aims at giving the reader an understanding of the terminology used for the gripper and subjects related to it in this thesis. Starting with the basic term grasp pose — or simply *grasp* — which essentially means the position and orientation of a gripper. In the context of this thesis, the discussion is limited to two-finger grippers. Anatomical terms such as finger and palm are used in the same context as on a human hand, note that the surface normal of the palm points in the direction of the closing region of the fingers. As such, the word hand might be used interchangeably with either gripper or grasp pose, based on context. When referring to the normal direction — or just *normal* — of a grasp, it is implied that this is the surface normal of the palm of the hand.

A candidate grasp pose is a grasp pose that has been sampled, but not yet evaluated. As such, a candidate grasp pose can be in a configuration that ultimately wouldn't lead to a grasp, such as a pose where the gripper would intersect or collide with the object. The closing region is the region enclosed between the fingers of the gripper and the palm, i.e. the area spanned by moving the fingers from a fully open to a fully closed position. If a candidate pose has no points from the object in its closing region, the grasp would unsurprisingly not be able to pick up the object.

A point being sampled is considered as centered in the closing region of the gripper, as illustrated in Section 5.1. In this manner, if the object geometry allows for it, simply closing the gripper would be sufficient to grasp it as there are no collisions. For sampled points in surfaces that are larger than the max aperture of the gripper, a collision check would have to be made, which is further discussed in Section 6.1.

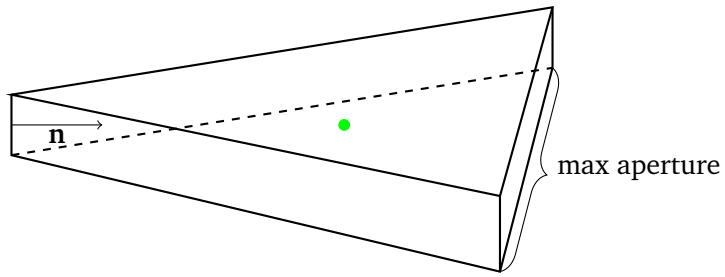


Figure 5.2: Sampled point inside closing region of a gripper with simplified geometry. **n** is the normal pointing out of the grippers palm.

5.2 Uniform With Local Variation

Sampling points uniformly from the object point cloud is a straight forward method of generating candidate poses. Given that segmentation has output a point cloud consisting of the OOI, then each point could be a potential basis for a grasp. The

initial sample gives points belonging to the object, with an orientation aligned with the direction the camera is facing. It is easy to see that only considering poses with this alignment would drastically reduce the chances of finding a grasp.

To widen the sample size and introduce variation in orientations, the uniformly sampled poses are augmented. Augmentations are done by pulling from a normal distribution then adding the result to the sampled pose, the standard deviation in orientation and position can be tuned according to the desired variation. Having augmented all the uniformly sampled poses, the resulting cloud of grasping candidates will be reminiscent of star clusters, see Figure 5.3. Pseudocode for the uniform sampling algorithm is given in Algorithm 1.

While using a uniform distribution with enough samples is sure to cover the entire object, this method generates multiple candidates closely clustered together. The tight clustering in itself is not necessarily a problem as the standard deviation could be tuned, but it could lead to detrimental performance issues. If the initially sampled point is objectively a bad place to grasp, the newly sampled points with variations will most likely also just be bad candidates. In order to extract good candidates, the sampling algorithm would have to generate many candidates, just to discard a large chunk of the bad ones at later stages. It is worth repeating that each of the candidates, i.e. the blue points in Figure 5.3, would also have individual orientations.

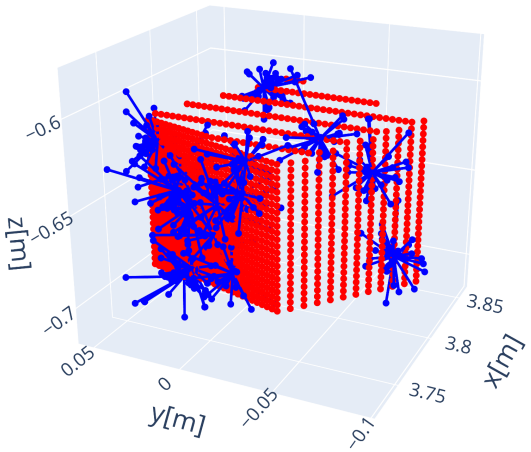


Figure 5.3: Uniformly sampled points with local variations looking like star clusters. Red points belong to the object that is being sampled from and blue points are sampled positions. Lines are drawn from the uniformly sampled point to their corresponding local variations.

Algorithm 1 Uniform grasp pose sampling with local variations

Input:
$\mathcal{P} \leftarrow$ Point cloud
$\boldsymbol{\sigma}_p \leftarrow$ Standard deviation position
$\boldsymbol{\sigma}_\theta \leftarrow$ Standard deviation orientation
$k \leftarrow$ Number of points to sample uniformly
$n \leftarrow$ Number of local variations for each uniformly sampled point

procedure UNIFORMSAMPLING($\mathcal{P}, \boldsymbol{\sigma}_p, \boldsymbol{\sigma}_\theta, k, n$)
  $\mathcal{S} \leftarrow \emptyset$ ▷ Initialize sampled poses as empty set
  $\mathcal{P}^k$ sampled from $\mathcal{U}(\mathcal{P}, k)$ ▷ Sample k points uniformly from $\mathcal{P}$
  for all points $\mathbf{p}$ in $\mathcal{P}^k$ **do**
    for $i \leftarrow 1, n$ **do** ▷ Sample n variations per uniformly sampled point
      $\mathbf{v}$ sampled from $\mathcal{N}(\mathbf{0}, \boldsymbol{\sigma}_p)$ ▷ Sample local variation
      $\mathbf{p}_i \leftarrow \mathbf{p} + \mathbf{v}$ ▷ Add local variation to each point
      $\boldsymbol{\theta}_i$ sampled from $\mathcal{N}(\mathbf{0}, \boldsymbol{\sigma}_\theta)$ ▷ Sample 3DoF orientation
      $\boldsymbol{\eta} \leftarrow [\mathbf{p}_i^\top, \boldsymbol{\theta}_i^\top]^\top$ ▷ Sampled pose is position $\mathbf{p}_i$ and orientation $\boldsymbol{\theta}_i$
      $\mathcal{S} \cup \{\boldsymbol{\eta}\}$ ▷ Add pose to set of sampled poses
    end for
  end for
  return $\mathcal{S}$
end procedure

5.3 Using Point Cloud Characteristics

Hand placement when picking up objects is far from arbitrary. Picking up an object far from the center of mass leads to gravity acting as a wrench, causing the object to pivot and possibly slip from the grasp. Through many years of intuition behind holding things, humans can adapt to this challenge when sensing movement in the object. Counteractions might include rapidly moving the grip to another location, supporting the object with another hand, tightening the grip, or simply letting the object fall while attempting to catch it with another extremity, popularly a foot. These counteractions can have varying degrees of success, hence it is important to consider the hand placement beforehand. To give a robot the capabilities of all these counteractions is a hard task, hence an approach to combat it is to perform the grasp closer to the center of mass to minimize the wrench produced by gravity.

Estimating the center of mass for an unknown object is a thesis, or even an entire research field, in its own right, a simpler approach can be made with some assumptions. Under the realistic assumption that the observed point cloud has a uniform mass distribution, the center of mass coincides with the centroid of the point cloud. As such, if a sufficiently accurate segmentation has been done the OOIs center of mass can be estimated as the centroid of the point cloud segment.

A point cloud as observed by a stereo camera only has partial information about a scene. Any part of the scene occluded by an object would not be included in the point cloud. Without performing a full survey of the scene this limits the knowledge of full 3D information about object shapes and depth in the scene. Hence, an assumption that the object cloud as a result of segmentation carries sufficient information about the object is made. In essence, this assumption means that estimating the center of mass for the observed part of the object is sufficiently close to estimating the center of mass for the entire object.

In light of the previous assumptions, estimating the segment's center of mass is reduced to estimating its centroid. Recall that the centroid can be estimated using Equation (4.5), but rather than summing over a neighborhood the whole cloud is used. The estimated center of mass can then be used as the mean in a distribution of candidate grasping poses. There are some considerations to be taken when picking the standard deviation, and type of distribution, which depend on the desired properties of the sampling algorithm. Namely, using a distribution with heavy tails would give a higher probability of sampling poses at the extremities, while a lighter-tailed distribution has a higher density at the mean. Under the assumption that the center of mass is a good point to grasp, a high density at the mean is desirable. However, for objects that are hard — or even impossible — to grasp at the centroid, sampling around the mean would have a low probability of finding a feasible grasp, such as a large ring.

The normal distribution is a natural first choice for a roboticist, satisfying the

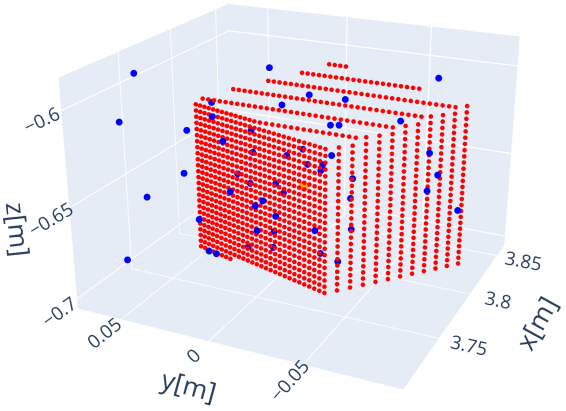
property of having a sufficiently high and wide peak and decently weighted tails. The final consideration for the sampling distribution then falls on the standard deviation. Again the dilemma of picking a sufficiently wide distribution arises, as a low standard deviation implies a lower variation in gripper poses. As the sampled poses are in 3D, the distribution forms an ellipsoid around the mean, furthermore, if the standard deviation for x, y and z is the same the distribution has a spherical shape. For an object that is significantly smaller along one axis than the other it is undesirable to sample along both axes with the same standard deviation, as this would likely lead to more infeasible poses.

Point cloud characteristics such as the variation along the x-, y- and z-axis can aid in picking standard deviations for sampling. In similar fashion as finding the covariance matrix of the neighborhood $\mathcal{P}^k$ in Section 4.2.1, the covariance of the entire cloud can be estimated using the estimated centroid then summing over the entire cloud with Equation (4.4). A good pick for standard deviation in x, y and z is then given by the square root of the diagonal elements of $\mathbf{C}$ respectively. Finally, pseudocode for the algorithm is given in Algorithm 2

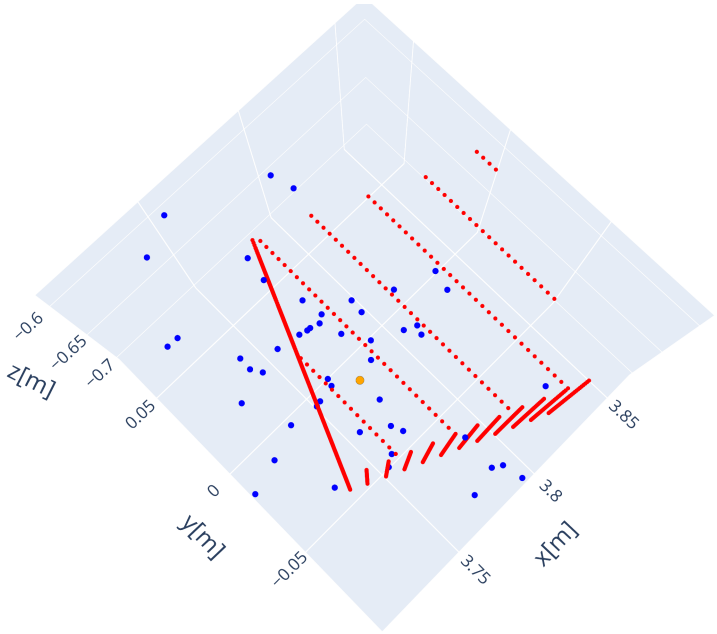
Algorithm 2 Normal grasp pose sampling around centroid

Input:
$\mathcal{P} \leftarrow$ Point cloud
$\boldsymbol{\alpha}_p \leftarrow$ Scaling factor for estimated standard deviation position
$\boldsymbol{\sigma}_\theta \leftarrow$ Standard deviation orientation
$k \leftarrow$ Number of points to sample

procedure NORMALSAMPLING($\mathcal{P}, \boldsymbol{\alpha}_p, \boldsymbol{\sigma}_\theta, k$)
  $\mathcal{S} \leftarrow \emptyset$ ▷ Initialize sampled poses as empty set
  $\bar{\mathbf{p}} \leftarrow$ESTIMATECENTROID($\mathcal{P}$) ▷ Equation (4.5)
  $\mathbf{C} \leftarrow$ESTIMATECOVARIANCE($\mathcal{P}$,$\bar{\mathbf{p}}$) ▷ Equation (4.4)
  $\boldsymbol{\sigma}_p^2 \leftarrow \text{diag}(\mathbf{C})$ ▷ Standard deviation is $\sqrt{\text{var.}}$
  for $i \leftarrow 1, k$ **do**
    $\mathbf{p}_i$ sampled from $\mathcal{N}(\bar{\mathbf{p}}, \boldsymbol{\alpha}_p^\top \boldsymbol{\sigma}_p)$ ▷ Sample position with scaling on std.dev.
    $\boldsymbol{\theta}_i$ sampled from $\mathcal{N}(\mathbf{0}, \boldsymbol{\sigma}_\theta)$ ▷ Sample 3DoF orientation
    $\boldsymbol{\eta} \leftarrow [\mathbf{p}_i^\top, \boldsymbol{\theta}_i^\top]^\top$ ▷ Sampled pose is position $\mathbf{p}_i$ and orientation $\boldsymbol{\theta}_i$
    $\mathcal{S} \cup \{\boldsymbol{\eta}\}$ ▷ Add pose to set of sampled poses
  end for
  return $\mathcal{S}$
end procedure



(a) Frontal view at a slight angle.



(b) Top-down view.

Figure 5.4: Example of sampled positions using a normal distribution with mean at the centroid. Centroid is drawn in orange, sampled positions are blue and the object cloud is red.

Chapter 6

Grasp Pose Evaluation

To decide what grasp pose candidate is best suited for grasping the object of interest, quality measures have to be calculated. Optimally these measures have a high correlation with experimental results such that high quality in the algorithm is indicative of a high-quality grasp. While using machine learning frameworks to classify grasps is growing rapidly in popularity [16], the quality measures used for this thesis are purely geometry based. In Section 5.3 it is mentioned that gripping near the center of mass is indicative of a more stable grasp, it is assumed that this quality is baked into the sampling algorithm.

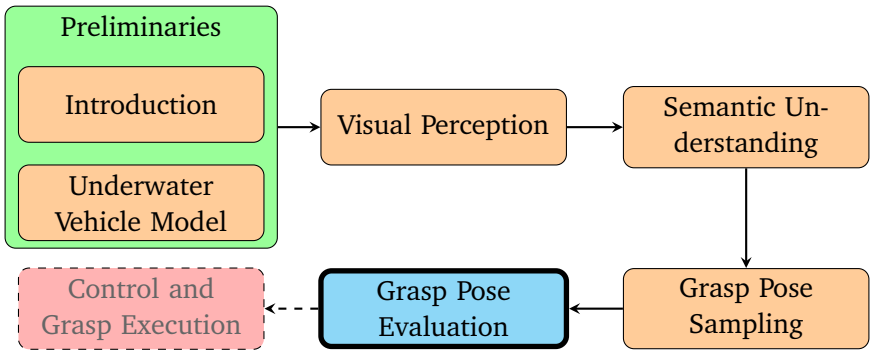


Figure 6.1: This chapter wraps up the presented pipeline with reasoning behind some quality metrics for evaluating grasp poses candidates.

Grasp pose evaluation is formulated as a score, where each component adds to the score according to how well the pose satisfies the metric. At the end, the pose with the highest score is labeled as the best pose, as this is the pose that aligned best with the metrics presented in this chapter.

This chapter aims to introduce further quality measures for a grasp candidate and the reasoning behind them. Section 6.1 discusses the removal of candidate grasps that have invalid configurations before a metric describing hand alignment is introduced in Section 6.2. Further, Section 6.3 presents a discussion about punishing demanding poses due to the restoring forces on a submerged vehicle. A secondary measure of hand alignment relative to the estimated object surface is

provided in Section 6.4, before Section 6.5 provides a qualitative measure of how well the gripper encloses the object. Finally, the full metric used for scoring a grasp candidate is summarized in Section 6.6.

6.1 Collisions

As the presented sampling algorithms sample poses with little to no regard for the gripper geometry, some grasp candidates are bound to be infeasible. A pose is given by the position of the centroid of the closing region and an orientation that dictates the normal direction of the hand, as shown in Section 5.1. Depending on the gripper configuration, the aperture dictates the maximum width of an object it can grip. Thus, if the closing region of the gripper intersects a region in the point cloud exceeding this width, a collision would occur.

To detect possible collisions an outer collision box for the gripper is defined. The collision box encloses the closing region and is large enough to enclose the physical geometry of the gripper in a fully open configuration. To avoid poses where the gripper would fit at the object, but be too wide to get there, the collision box stretches backward along the arm. A collision is indicated if a point contained in the collision box does not simultaneously lie within the closing region. This relationship is visualized in Figure 6.2.

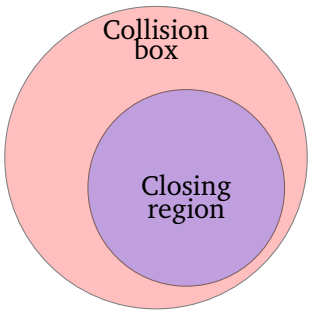


Figure 6.2: Diagram showing that the closing region is a subset of the collision box. If a point lies in the collision box and not the closing region it indicates a collision.

The algorithm creates convex hulls for the collision box and the closing region. The convex hulls are defined as the geometry of the gripper around the center point of the closing region. The next step is to filter the points that lie within the collision box, the points that do are added to a temporary cloud. The temporary cloud and the closing region are similarly filtered. If any point in the temporary cloud is outside of the closing region, it is considered at risk of collision, and the grasp is classified as a collision. Any grasps marked as collisions are removed from the sample set $\mathcal{S}$. Grasps with no points in the closing region are also filtered out at this stage.

6.2 Normal Angle Alignment

Through experiments, Balasubramanian et al. discovered that humans tend to align their wrists with the principal axis or its perpendiculars when grasping objects [50]. In order to measure how well a grasp pose candidate aligns with the point cloud, a comparison is done with the grasp normal and the estimated surface normal of the closest point. The point in the closing region closest to the gripper position is found using kNN as described in Section 4.1.2, and the normal corresponding to that point is readily available from the result of the method explained in Section 4.2.1. The grasps alignment with the surface normal is calculated directly from the vector cross-product as

$$\cos(\theta) = \frac{\mathbf{n}_g \times \mathbf{n}_s}{||\mathbf{n}_g|| \cdot ||\mathbf{n}_s||} \tag{6.1}$$

where $\mathbf{n}_g$ is the grasp normal and $\mathbf{n}_s$ is the estimated surface normal of the closest point.

The angle between the normals could be computed as $\arccos(\cdot)$ of Equation (6.1), however, in order to use it directly as a score it is left as is. Cosine provides values in the range $[-1, 1]$, where -1 means the normals are pointing in opposite directions, 0 means they are perpendicular, and 1 means they point in the same direction. Hence, taking the absolute value of (6.1), i.e.

$$q_n = |\cos(\theta)| = \left| \frac{\mathbf{n}_g \times \mathbf{n}_s}{||\mathbf{n}_g|| \cdot ||\mathbf{n}_s||} \right| \tag{6.2}$$

provides a score in the range $[0, 1]$ of how well the normals align. A score of 1 indicates that the angle between the normals is 0, which is analogous to the grasp being perpendicular to the surface.

6.3 Orientation Constraints

As the camera is mounted rigidly on the AUV, the images it produces, and the point clouds, are relative to the AUV's pose. Recall that the gripper is rigidly fixed to the AUV and has no DoF once fastened. Hence, if a grasp candidate is sampled at an orientation far from $\boldsymbol{\theta} = [0, 0, 0]^\top$, performing that grasp requires the AUV to achieve that pose. For a neutrally buoyant vehicle, the restoring forces will always try to keep CG below CB, consequently, an AUV at 90° roll produces restoring forces pushing roll back to 0. Achieving a grasp at 90° roll would then require thrust to compensate for the restoring forces, making the grasp harder to perform.

Assuming that the AUV is in a neutral orientation, the point cloud generated is aligned with NED. This also means that all sampled poses are relative to the neutral NED orientation. A scoring function is introduced in order to accommodate

for low-impact orientations, giving higher scores closer to a neutral orientation. The suggested scoring function is

$$q_{o,\alpha} = 1 - |\sin(\alpha)|, \quad \alpha = \{\phi, \theta, \psi\} \tag{6.3}$$

which gives a decreasing value as the angle deviates from 0, as shown in Figure 6.3. Note that when passing $\frac{\pi}{2}$ the value increases again. In practice this would for instance mean a roll of more than 90°, however, this can also be solved by rolling the appropriate amount in the opposite direction due to gripper symmetry. For pitching this solution can not be applied, hence, situations where the sampled orientation has a component greater than $\frac{\pi}{2}$ should be avoided e.g. by using a significantly low standard deviation when sampling.

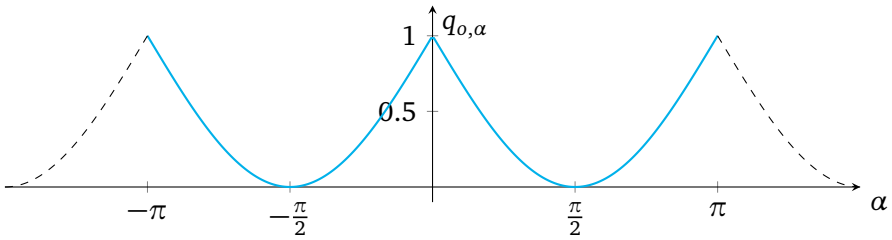


Figure 6.3: Scoring function for orientations.

Orientation is inherently a three DoF property, and as such the scoring function gives a score $q_{o,\alpha} \in [0,1]$ for each degree of freedom. With the purpose of reducing the impact of orientation scores, a weighting is done such that the total contribution of $q_o = q_{o,\phi} + q_{o,\theta} + q_{o,\psi}$ is in the range $[0,1]$, i.e.

$$q_o = w_1 q_{o,\phi} + w_2 q_{o,\theta} + w_3 q_{o,\psi}, \quad w_1 + w_2 + w_3 = 1 \tag{6.4}$$

Furthermore, it is worth noting that a deviation from 0 in yaw has little to no impact on the difficulty of reaching the pose, hence w_3 can be picked significantly lower than the other weights. A word of caution here is that with point clouds generated from only one viewpoint, knowledge of the scene decreases when yaw deviates from the viewpoint, as such, the standard deviation in yaw should also be tuned accordingly.

6.4 Contact Point Alignment

In order to grasp and hold an object the grasping hand needs to counteract a series of forces. Simply picking an object straight up requires the negation of gravitational forces. This can be achieved either by getting underneath the object and pushing it upwards, fastening something to the object and hoisting it like a crane, or by grabbing the object by the sides and lifting it. Depending on the configuration of the two-finger gripper, mainly the first or last options are applicable,

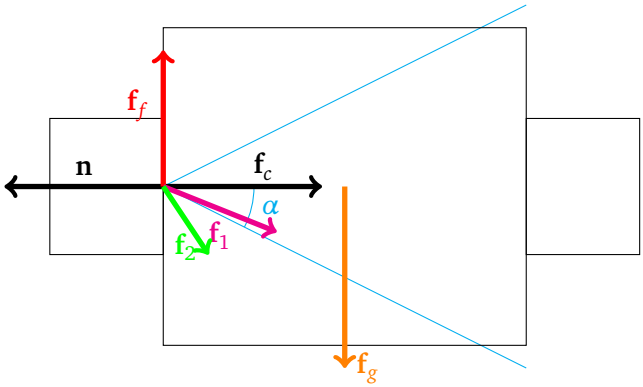


Figure 6.4: Forces acting in one contact point of a two-finger grasp, showing how the total force needs to be inside the friction cone in order for the object not to slip.

however, in both cases, the stability of the grasp depends on the friction at the contact points.

When contact is formed between two objects, a perpendicular force is exerted on both objects. The static object exerts a normal force on the object that comes in contact with it, which exerts a contact force on the static object. When a gripper closes on an object, the normal force stops the gripper from passing through the object. In order to cancel out the gripper movement, the normal force is equal in magnitude to the contact force, but opposite in direction.

Due to the contacting surfaces being uneven on a microscopic level, a friction force occurs when the surfaces move in opposite directions. The friction force is opposite to the force that moves the object. Using a Coulomb friction model, the friction force required to keep the object stationary is given by

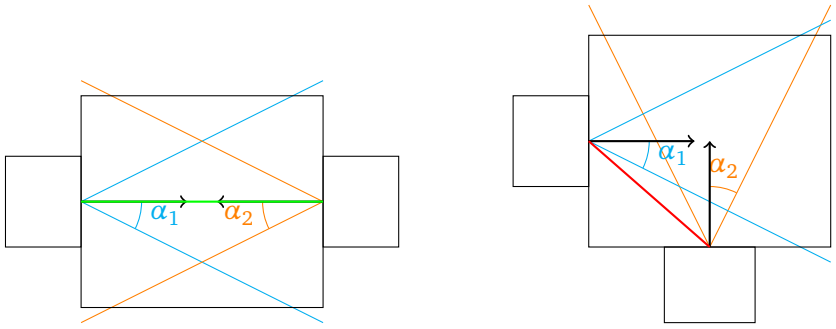
$$\mathbf{f}_f \leq \mu_s \mathbf{n} \tag{6.5}$$

where μ_s is the static friction coefficient for the contact between the two materials, and $\mathbf{f}_f, \mathbf{n}$ is the friction and normal force respectively. Thus, if the friction force increases above $\mu_s \mathbf{n}$ as a result of an external force, the object starts moving in the direction of the external force. The friction cone is defined as the region spanned by the set of forces that can be applied without causing the object to be moved. For one contact point, this gives the friction cone

$$\left\{ \mathbf{f} \in \mathbb{R}^3 : \sqrt{f_x^2 + f_y^2} \leq \mu_s f_z, f_z \geq 0 \right\} \tag{6.6}$$

where f_x and f_y are forces tangent to the contact surface and f_z is perpendicular to the surface.

For a grasp where two fingers are placed on opposite sides of an object, in an attempt to lift it, the gravitational force pulls the object down with a force



(a) Antipodal grasp **(b)** Not antipodal grasp.

Figure 6.5: Two finger grasps as seen from above, the grasp in **(a)** satisfies the antipodal condition, while the grasp in **(b)** does not. Further, the line connecting the contact points is also referred to as the connecting line.

perpendicular to the contact forces. In order to stop the object from slipping, the gravitational force needs to be canceled out sufficiently by the friction forces. In Figure 6.4 the forces are drawn for one contact point. The angle α comes from the right triangle spanned by the contact and friction forces, i.e. $\mu_s = \frac{\mathbf{f}_f}{\mathbf{f}_c} = \tan(\alpha)$. Let the angle α in Figure 6.4 be given by the friction coefficient μ_s. Then, given the contact force $\mathbf{f}_c$ and the friction $\mathbf{f}_f$ as a result of the gravitational force $\mathbf{f}_g$ in Figure 6.4, the resulting total force is $\mathbf{f}_1$. As $\mathbf{f}_1$ lies inside the cone spanned by α, no slip occurs. However, lowering the contact force, and thus also the friction force, the total force is $\mathbf{f}_2$, which lies outside the friction cone, and a slip occurs.

Murray et al. define any grasp able to resist external wrenches with sufficient contact forces as a force-closure grasp [51]. Furthermore, Murray et al. state that a grasp with two contact points with friction is a force-closure if and only if the lines connecting the contact points lie within both friction cones. A grasp satisfying this condition is termed an antipodal grasp. This adds the additional constraint that the two friction cones have to open towards each other, in essence meaning that the fingers should be placed sufficiently opposite to each other. This is shown in Figure 6.5, where Figure 6.5b creates a situation where the connecting line lies outside the friction cones, and Figure 6.5a is an antipodal grasp.

Thus, the antipodality of a grasp can be indicated by the line connecting the contact points. Given a contact force at a contact point, the friction cone around the normal component of the contact force indicates how robust that contact is to slipping. As the setup used for this thesis has no measurement of contact force, a comparison of the connecting line and the two contact point normals gives an indication of how well the antipodality constraint is satisfied. The contact point for each finger is estimated by finding the point in the closing region that is closest to the plane spanned by the corresponding finger. Let $\mathbf{n}_g$ be the line connecting the contact points and $\mathbf{n}_s$ be the estimated surface normal, the alignment of the

two vectors is then given by Equation (6.1). This alignment is calculated for each contact point, giving two scores, q_{cl} for the left finger and q_{cr} for the right finger, calculated the same way as normal alignment, Equation (6.2).

6.5 Inlier Count

A grasp pose candidate which only encompasses a small part of the point cloud is able to get a fairly high evaluation based on the previously mentioned metrics. However, if the grasp barely contains any part of the object the grasp is likely not satisfactory. Hence it is pertinent to give grasp candidates containing more of the object a better score.

In its simplest form, scoring based on inliers is simply an addition to the score proportional to the amount of points in the closing region. However, as the other metrics mentioned in this chapter provide scores within the region $[0, 1]$ it would be suitable for the inlier metric to do the same. It follows from the nature of the problem that the amount of inliers is highly dependent on the geometry of the gripper and the object, as well as the density of the point cloud.

As the gripper geometry is known and the downsampling algorithm described in Section 4.1.5 uses voxels of a given size, a theoretical maximum amount of inliers can be found. By calculating the volume of the closing region and knowing the volume of the voxels, the maximum amount of inliers is calculated as the amount of voxels fitting in the closing region. Thus, a score is given by the percentage of inliers compared to the maximum possible, Equation (6.7). However, the maximum as computed in this manner is not feasible, as it represents a fully occupied closing region, which is only possible given a dense point cloud. The surfaces constructed by the point cloud generation methods explained in this thesis would likely never fully populate the closing region. Hence, a percentage score of inliers based on the maximum found in this manner would produce scores in the lower percentages.

$$q_i = \frac{\text{Inliers}}{\text{Maximum inliers}} \tag{6.7}$$

For the purpose of this thesis, the value of maximum inliers was found empirically, in order to get an inlier score closer to the range $q_i \in [0, 1]$. Experiments gave an intuition of how many inliers were expected for the chosen gripper geometry, and the maximum inlier amount was changed accordingly. It is worth noting that lowering the maximum inlier amount based on empirical knowledge introduces cases where a grasp could get a percentage score of above 1, indicating that the grasp contains more inliers than the maximum limit. This quality measure is highly influenced by several geometrical factors with regards to both the gripper and the object, meaning it should probably be tuned for individual cases.

6.6 Final Scoring

In the preceding sections scores indicating geometric considerations for a robust grasp have been introduced. The final quality indicator of a grasp is the weighted sum of these scores, given by Equation (6.8). Weighting the individual scores to indicate priority of the grasp. For instance if the orientation is of low importance, setting the weight w_o close to zero reduces the impact.

$$q_{tot} = w_n q_n + w_o q_o + w_c q_{cl} + w_c q_{cr} + w_i q_i \tag{6.8}$$

Each individual quality measure as stated in this chapter gives a score $q \in [0, 1]$. Thus, the maximum achievable score — given that all weights are 1 and maximum inliers is picked sufficiently high — is a score of 5, which indicates a grasp that aligns perfectly with all the provided measures.

To summarize, q_n evaluates the alignment of the grasp normal and the estimated surface normal. q_o is the weighted sum of the orientation scores that evaluates the grasp pose based on its orientation. q_{cl} evaluates the alignment of the connecting line at the left contact point to the estimated surface normal and q_{cr} at the right contact point. Finally, q_i is the inlier score which evaluates the number of inliers within the closing region.

Chapter 7

Implementation

To test the solution explored in this thesis, a simulation environment has been used. The simulator was originally developed alongside a specialization project conducted during the autumn of 2022. As the specialization project is unpublished, some implementation details of the simulator environment are restated here. Some modifications were made to the simulator to facilitate stereo/RGBD vision to generate point clouds.

This chapter will cover the implementation framework used for simulating the method described in this thesis. Starting by presenting the Robot Operating System software framework in Section 7.1, before introducing the simulator Gazebo in Section 7.2, followed by a brief introduction to the Point Cloud Library in Section 7.3. As the simulator provides a scenario with optimal conditions, some of the inherent assumptions this brings to the system are covered in Section 7.4.

7.1 The Robot Operating System

The Robot Operating System (ROS) is an open-source framework for building and implementing robotic systems, maintained by the Open Robotics organization. ROS provides a message-passing framework for modular applications in the structure of nodes. A node is a process capable of performing computations, which communicates with other nodes through named streams called topics. Nodes can publish messages on topics, other nodes in turn can then access the message by subscribing to the same topic. In this manner, communication is anonymous between nodes in the sense that a node can freely publish to a topic without knowing if any other nodes subscribe. Topics specify their types of messages, such that a node publishing or subscribing knows what to expect when interfacing with the topic.

ROS applications become highly modular due to the nature of communication between nodes. New nodes can be developed on the fly, as the interface is known, allowing for rapid testing and integration of new modules. ROS also provides a framework for launching larger applications consisting of multiple nodes through

launch files, which allows the developer to specify which nodes are started and with what settings. The development of complex systems consisting of multiple modules is thus facilitated by the framework provided by ROS.

7.2 Gazebo

Gazebo is a 3D robotics simulation environment that provides a realistic physics engine supporting a wide range of sensors and actuators. Maintained by the Open Robotics organization, the software is open source and available under the Apache 2.0 license. The simulator grants the ability to simulate highly complex robot behavior, which makes it ideal for use before the field deployment of developed systems.

Gazebo supports an array of files used to describe the world and the robots in it and boasts an arsenal of different sensors it can simulate. This allows for specific robot designs to be provided for simulation with high customizability for accurate simulations. The simulator comes with a buoyancy system and is capable of simulating hydrodynamics such as fluid added mass, damping, and Coriolis forces, $\mathbf{M}$, $\mathbf{D}$ and $\mathbf{C}$ respectively, which makes it well suited for underwater simulations. $\mathbf{C}$, the Coriolis and centripetal matrix, is calculated internally by the simulator using $\mathbf{M}$ as stated in Section 2.2.2, as such, the simulator only needs to be provided with the added mass and damping matrices. For this thesis, a BlueROV2 with the heavy configuration has been simulated, fitted with an RGBD camera.

The BlueROV2, with its heavy configuration, is a small and affordable high-performance ROV produced by Blue Robotics [52]. The ROV is small at approximately $0.46\,\text{m} \times 0.34\,\text{m} \times 0.25\,\text{m}$ and weighs in at around 12 kg. The heavy configuration comes with eight thrusters and is capable of precise movement with full 6 DoF control. With its modular frame, the BlueROV2 is easily modifiable, for the purpose of grasping, a two-finger gripper is rigidly attached to the frame, meaning it can only grasp in the configuration it is attached in. Figure 7.1 shows the BlueROV in its simulated configuration.

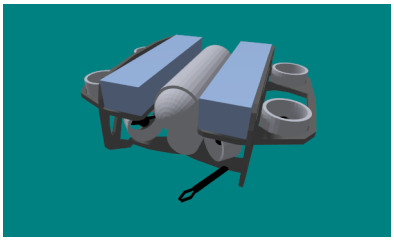


Figure 7.1: BlueROV in Gazebo.

The Open Robotics organization, which maintains both Gazebo and ROS, has made it possible to communicate between the two frameworks seamlessly. Gazebo is structured similarly to ROS with nodes, topics, and messages, allowing for a ROS

node to act as a bridge to Gazebo. In this manner, nodes can publish messages on topics in Gazebo, such as a camera sensor publishing an image, and then a node running in ROS can subscribe to the topic and receive the image.

7.3 The Point Cloud Library

The Point Cloud Library (PCL) is an open-source software framework for 2D/3D image and point cloud processing [53]. PCL carries a wide range of algorithms used for point cloud processing, such as the ones presented in Chapter 4. The solution implemented for this thesis uses PCL extensively for these purposes.

7.4 Inherent Assumptions

As the simulator provides an ideal environment without any disturbances, some assumptions are inherently made about the system. The assumptions have varying degrees of realism as the visual conditions in the simulator are a bad representation of rough underwater conditions. This section serves as a brief overview of considerations and assumptions done at the different steps of the pipeline.

7.4.1 Imaging and Point Cloud Construction

Underwater imaging is a tough task as underwater phenomena can heavily impact the captured images. The presence of marine snow and other suspended particles in water can cause turbidity, making the water hazy and cloudy and interfering with the penetration of light. Projected light from an AUV can also be reflected by the suspended particles and turbid water, known as backscattering, which impacts contrast in the captured scene. Furthermore, the light rapidly attenuates in water, making colors appear different and duller.

The severity of the aforementioned effects varies based on the conditions of the water body and depth. In the simulated solution these effects are not present, which poses the assumption that the underwater conditions are perfect, with low turbidity and good lighting. A light source in the simulator acts as a sun which casts a reflection on certain objects, such as the black bar in Figure 4.5. This effect is reminiscent of light changing the apparent color of perceived objects, but an assumption is nevertheless made that this effect is minimal. The effect of attenuation is also assumed negligible, i.e. colors appear as they are.

As the camera in the simulation produces perfect representations of the perceived scene, the camera is assumed to be properly calibrated with a suitable model for underwater imaging. This relates to the fact that there is no noise present in the images, nor are there any distortions. Furthermore, the point cloud produced by the RGBD camera in Gazebo produces exact representations. For this

to be feasible, assumptions are made that triangulation for 3D scene reconstruction is accurate, and that objects even with low-texture areas can be reconstructed using the chosen reconstruction method. To somewhat more accurately represent field conditions, the depth sensor in the simulator has been limited to a maximum distance of 6 m.

7.4.2 Point Cloud Processing

Given that a point cloud is produced, most processing can be done without assumptions. However, an assumption is made that the seafloor is sufficiently flat so that it can be estimated and removed with a RANSAC model to speed up segmentation. For the processing to produce accurate results, it is assumed that the point cloud produced by reconstruction has a sufficient density to accurately represent the scene. For this thesis, it has been assumed that the object of interest can be distinguished by its color, allowing the use of color segmentation algorithms to identify it. In the case of multiple objects of the same color, this assumption is extended to the object with the highest intensity of that color. It is further assumed that the point cloud's estimated surface normals represent the true surface.

7.4.3 Grasp Pose Sampling

As the point clouds generated mainly show objects from one side, it is assumed that this is sufficient information to get a decent idea of the shape of the object. In essence, this assumption means that a grasp is collision-free as long as it passed the collision check described in Section 6.1, and is not placed fully behind the object, i.e. the object geometry does not deviate considerably from the point cloud representation. For further implementation the simulator also provides accurate odometry, giving the position and orientation of the AUV. When control algorithms are implemented this means that full knowledge is available to guide the vehicle towards the target grasping pose.

Chapter 8

Results

The following chapter presents the experimental results of the grasp pose detection method explored in this thesis, tested by means of simulation in Gazebo. Section 8.1 introduces the simulation environment used for testing. Section 8.2 presents results related to the processing steps described in Section 4.1. Results from different sampling methods are presented in Section 8.3, followed by Section 8.4 which explores the metrics used for evaluating the grasp pose candidates before final results are displayed in Section 8.5.

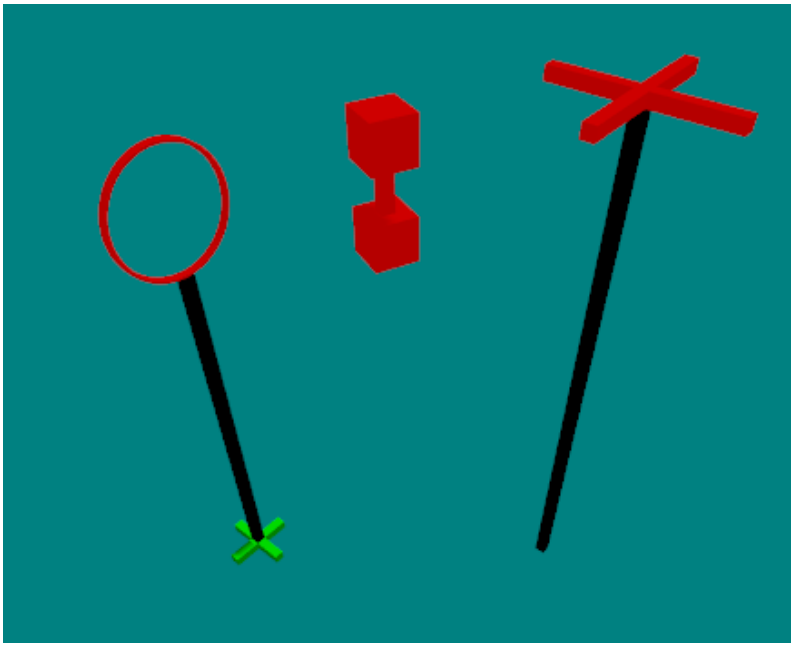


Figure 8.1: The three objects present in the simulation environment. For this figure, the barbell (middle object) has been raised to the level of the others and thus appears larger than it is.

8.1 Experimental Setup

In order to test the proposed framework for grasp pose detection, a simulation environment in Gazebo is used. A model of a BlueROV2 heavy fitted with an RGBD camera is placed in the Gazebo-simulated world. In the simulator three other objects of interest are present. The first object is shaped like a barbell, consisting of two squares connected by a narrow rectangle. This shape is used to represent an object only graspable in a limited region. The second object is a ring on top of a pole, which has the property that the centroid contains no viable grasps, while the ring itself is graspable from essentially any pose. Finally, a horizontal cross is present in the simulation, aimed at testing the orientation aspect of the implementation. Figure 8.1 shows the three objects in the simulator.

The RGBD camera publishes its image and corresponding point cloud to a Gazebo topic, which is then bridged to ROS for further processing. A rough outline of the node structure is shown in Figure 8.2. Note that all point clouds shown in this chapter show coordinates relative to the camera, i.e. $y = 0$ implies the object is straight in front, and negative z means that the object is below the ROV.

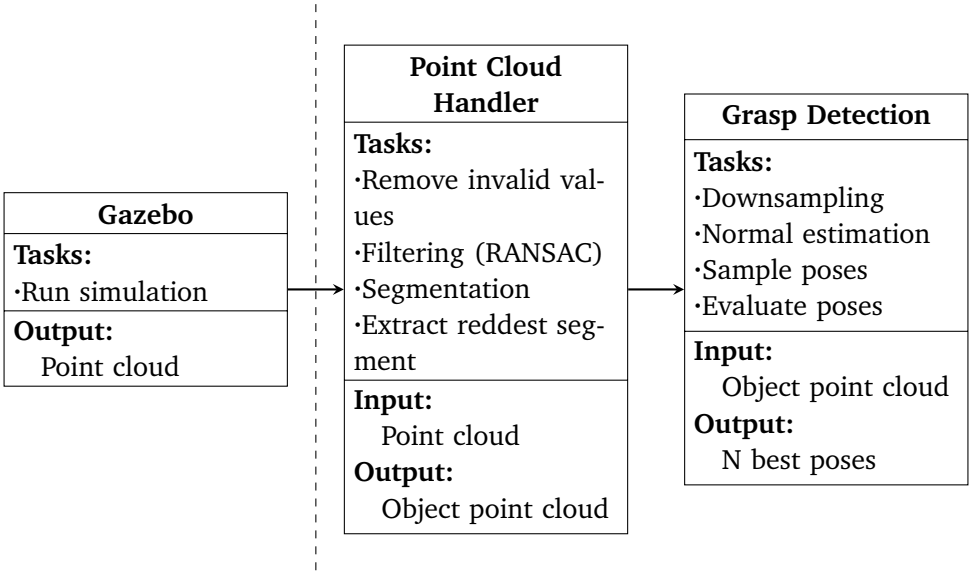


Figure 8.2: Overview of Gazebo and ROS node structure, along with their tasks. Dashed line indicates the separation between Gazebo(left) and ROS(right).

8.2 Point Cloud Processing

Different measures are used in order to evaluate the performance of the point cloud processing steps. A measurement of time indicates how effective filtering is, while the segmentation results are quantitatively inspected. As a baseline, the point cloud is first segmented without performing any filtering, other than remov-

ing the points with invalid values. Table 8.1 shows the average time for removing invalid values and performing segmentation on the unfiltered point cloud, shown in Figure 8.3a. Note that the color of the segments after segmentation — as in Figure 8.3b — are randomly picked by the segmentation algorithm, true colors are shown in Figure 8.3a.

Table 8.1: Average time for removing invalid values and performing segmentation on an unfiltered scene observed by a stationary ROV over 10 iterations.

	Average Time
Removing Invalid Values	10.3 ms
Segmentation	3650.7 ms

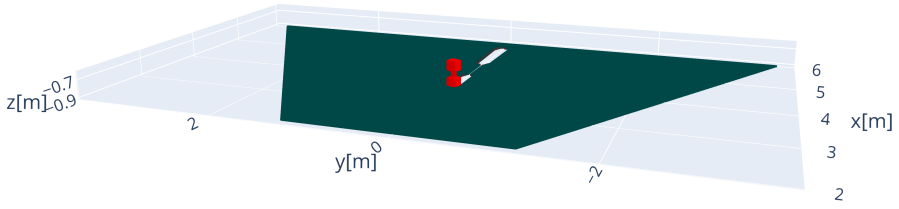
RANSAC is then used to filter out the points corresponding to the seafloor. As the point cloud is of a known size, the inlier fraction is found by looking at the size of the two segments in Figure 8.3b, the segment corresponding to the plane consists of 395245 points, while the barbell consists of 2835. Let the desired probability of finding a fitting model for the ground be $p = 99\%$. Only $n = 3$ points are needed for estimating a plane. These numbers are then plugged into Equation (4.3), giving the iterations needed $k = 1.19$. Thus, by setting the maximum iterations of RANSAC to $k = 20$, a model fitting the ground should be found with sufficient probability.

Table 8.2 shows the average time of segmenting the same scene as in Figure 8.3a after removing the ground with RANSAC. The resulting segmented point cloud can be seen in Figure 8.4.

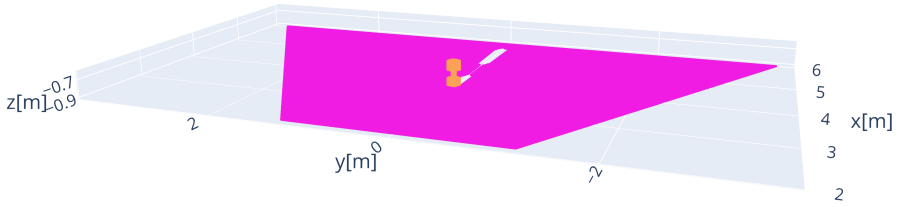
Table 8.2: Average time for removing invalid values and performing segmentation, after removing points corresponding to the seafloor using RANSAC, on an unfiltered scene observed by a stationary ROV over 10 iterations.

	Average Time
Removing Invalid Values	18.7 ms
Segmentation	25.6 ms
RANSAC	31.1 ms

In scenes with multiple objects, the object of interest is assumed to be recognized by color, see Section 7.4.2. Figure 8.5 shows the process of segmenting a scene with multiple red objects, where the final segment sent to the Grasp Detection node is shown in Figure 8.5c.



(a) Raw point cloud.



(b) Segmented point cloud.

Figure 8.3: Scene used for comparison of segmentation with and without filtering with RANSAC.

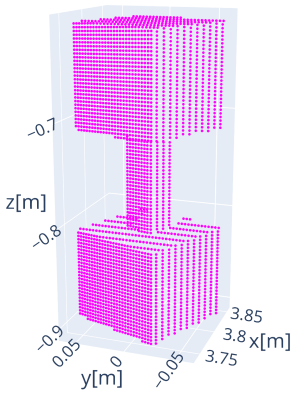
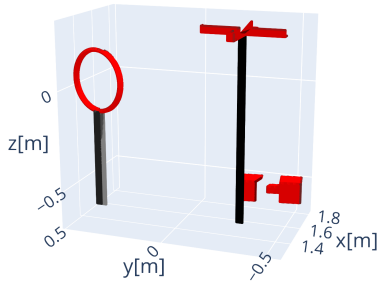
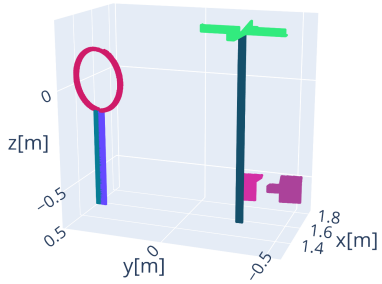


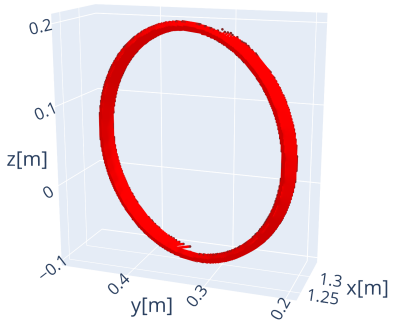
Figure 8.4: Result of running segmentation on the point cloud in Figure 8.3a after performing RANSAC and removing the points belonging to the ground.



(a) Raw point cloud.



(b) Segmented point cloud.



(c) Extracted object.

Figure 8.5: Process of segmentation and extraction of object from a scene containing multiple red objects.

8.3 Grasp Pose Sampling

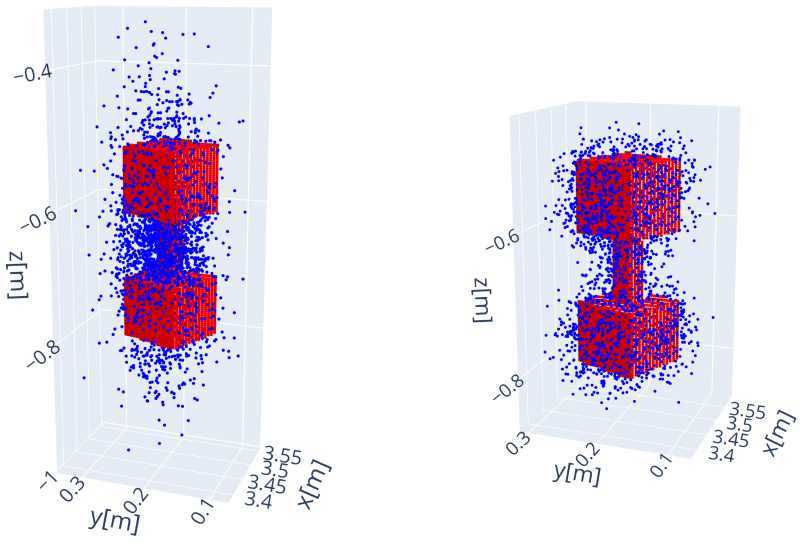
Once the point cloud has been segmented and the segment thought to contain the OOI has been found, it is passed to the grasping node. Two different sampling methods, as described in Sections 5.2 and 5.3, were tested. Of most interest when considering the sampling algorithms is their ability to sufficiently cover the object, in order to accommodate finding grasps on objects of all shapes. For this purpose two objects with different properties were tested, the barbell and the circle. For brevity the sampling method described in Section 5.2, sampling uniformly over the object with local variations, is referred to as the uniform method. Similarly, sampling normally over the point cloud using object characteristics, as described in Section 5.3, is referred to as the normal method.

Figure 8.6 compares sampling with the normal method and the uniform method. Normal sampling has a higher density around the centroid but has heavier tails in the direction of the largest variance. As the object is observed from the x direction, it has the lowest variance there as well, which in Figure 8.6c causes the sampled cloud to be thin. Uniform sampling generally covers the object uniformly, which ensures that even for objects that have a centroid outside of the object geometry, sampled grasps are decently close. In Figure 8.7 the assumption that the centroid is a good place to grasp is invalid, as such, sampling normally produces a lot of infeasible positions. Uniform sampling follows the object geometry and appears to be better suited in such a situation.

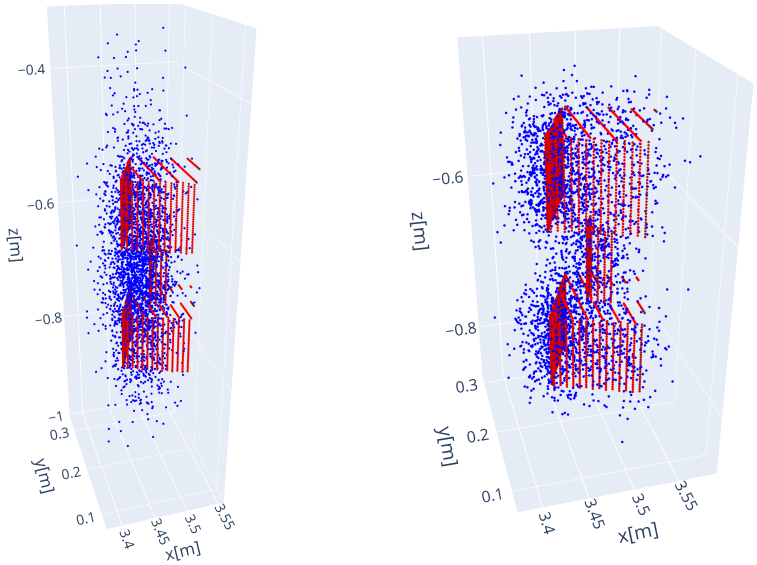
To evaluate the efficiency of the two sampling methods, a measurement of time spent on sampling is made. Furthermore, the amount of poses after collision checking is noted. Collision checking is evaluated further in Section 8.4, and the results presented in Table 8.3 purely represent the throughput of feasible sample poses generated by the sampling methods. For this purpose, both sampling methods are tested with 50 iterations on each object, sampling 5000 poses. The objects are observed at the same angle as in Figures 8.13 to 8.15.

Table 8.3: The average number of collision-free poses sampled over 50 iterations with 5000 poses sampled in each iteration.

Object	Method	Time	Collision-Free Poses	Percentage
Barbell	Normal	12.88 ms	25.56	0.51%
Barbell	Uniform	22.56 ms	37.24	0.75%
Circle	Normal	13.86 ms	43.14	0.86%
Circle	Uniform	21.92 ms	394.96	7.9%
Cross	Normal	17.52 ms	18.94	0.38%
Cross	Uniform	26.12 ms	58.52	1.17%
Total	Normal	14.75 ms	29.21	0.19%
Total	Uniform	23.53 ms	163.57	1.09%

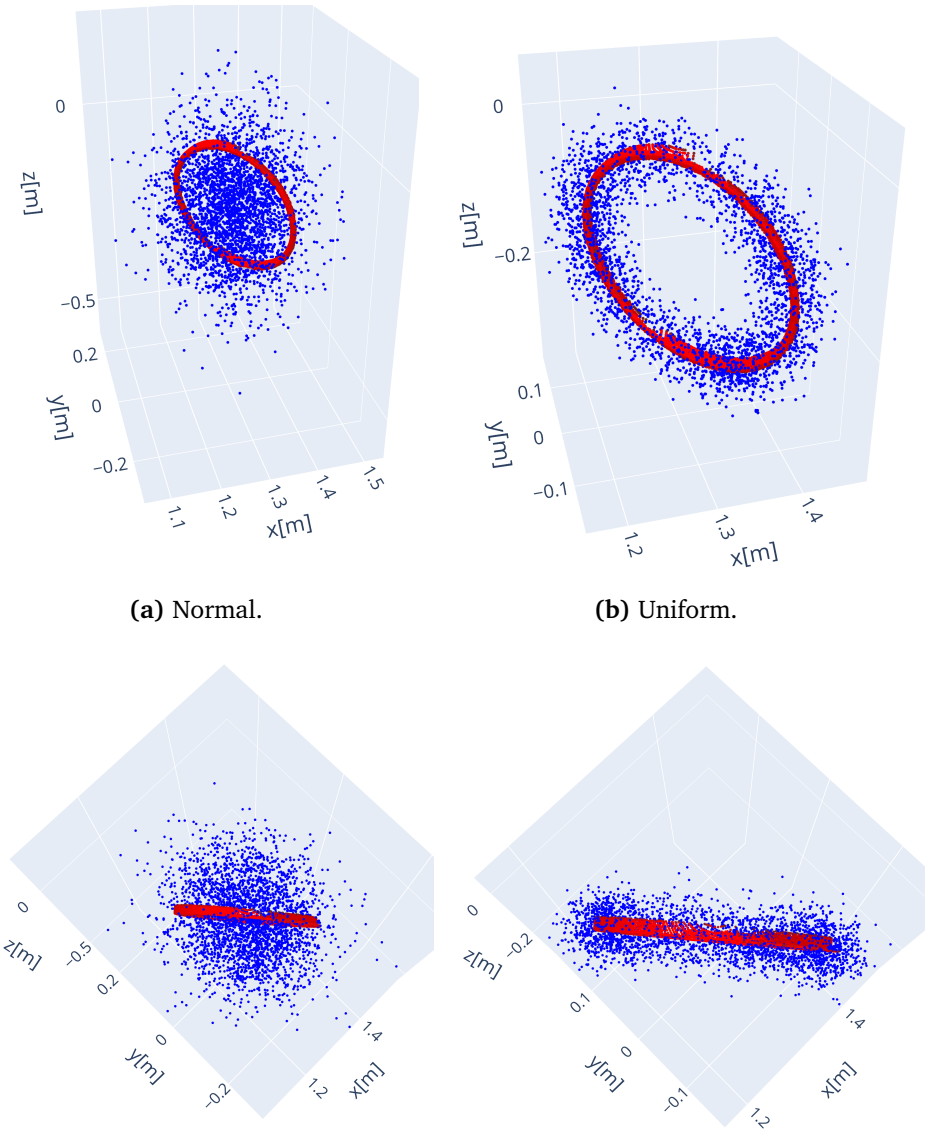


(a) Normal. **(b)** Uniform.



(c) Normal. **(d)** Uniform.

Figure 8.6: Comparison of the uniform and normal sampling on the barbell. (a), (b) Normal distribution has denser samples towards the centroid, uniform covers the object more. (c) Object shape causes high variation in z, lower in x. (d) Again, uniform samples closer to the object.



(a) Normal. **(b)** Uniform.

(c) Normal. **(d)** Uniform.

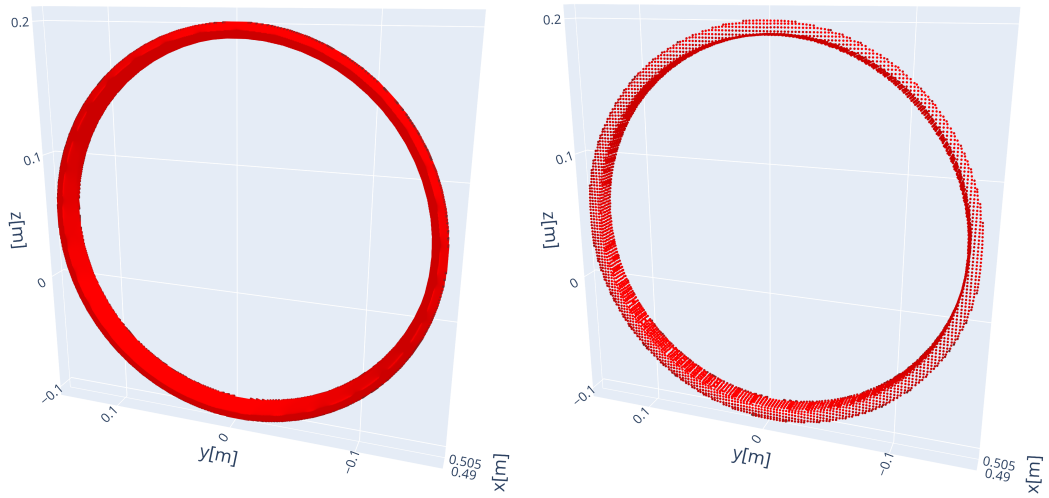
Figure 8.7: (a), (b), (c) and (d) Normal sampling uses characteristics of the point cloud, assuming that the centroid is a good region to grasp. Uniform sampling follows the object, which in this case produces more viable candidates.

8.4 Grasp Pose Evaluation

Before any grasp pose candidates are evaluated, poses that would collide with the object are discarded. This is done by checking if points lie in the collision box of the grasp, as described in Section 6.1. As this algorithm is proportional to the number of points in the point cloud, the effect of downsampling before collision checking is apparent. The point cloud in Figure 8.8a contains 64648 points, which is reduced to 4446 after downsampling using voxels with size 2.5 mm × 2.5 mm × 2.5 mm, Figure 8.8b. Table 8.4 shows the time spent on collision checking 3000 sampled poses with and without performing downsampling.

Table 8.4: Time spent on checking 3000 grasp pose candidates for collisions. Times are averaged over 10 iterations.

	Points	Average Time
Collision checking	64648	228.5 s
Downsampling	—	4 ms
Collision checking after downsampling	4446	16.9 s



(a) Raw. **(b)** Downsampled.

Figure 8.8: Point cloud used for testing downsampling. The downsampled point cloud carries roughly the same information as the unfiltered point cloud.

The internal quality measures described in Chapter 6 give an indication of how grasps are ranked against each other. In the following figures, each quality metric is tested on its own, starting with the alignment with the estimated closest surface normal, described in Section 6.2. Note that in the following figures, the quality of a grasp is indicated by its color. A brighter color indicates a higher-rated grasp, the color space is scaled such that the best grasp is equal to the maximum and the worst grasp is equal to the minimum.

Figure 8.9 shows five grasps scored exclusively based on their alignment with the surface normal. The dark blue grasp is parallel to the surface and has a score of nearly 0, while the bright yellow grasp appears to be aligned with the surface normal. It is worth pointing out that the visualized grasps are thin lines, while the closing region of the gripper is 1.5 cm tall. This means that grasps that have few inliers in simulation might appear as having no inliers in the figures.

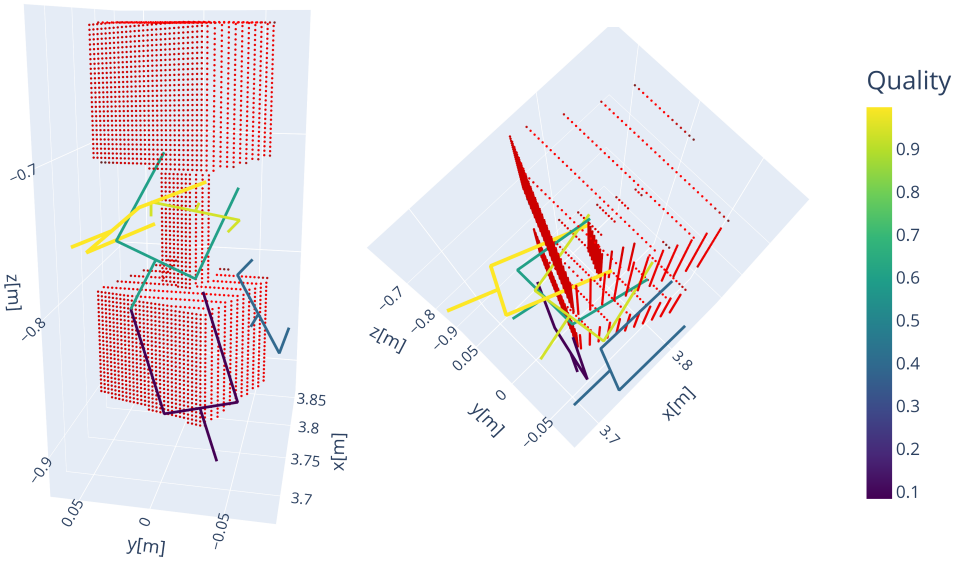


Figure 8.9: Five grasps evaluated using normal angle alignment. The best grasp appears normal on the point cloud, while the worst is close to parallel to the point cloud.

In Figure 8.10, six grasps are evaluated using the method described in Section 6.4. The line connecting the points closest to the fingers of the gripper is compared to the estimated surface normals at said points. The total score for each grasp is given by the sum of the two alignments, giving scores in the range $q_c \in [0, 2]$ in the unweighted case.

Grasps solely evaluated based on their orientation are shown in Figure 8.11. Orientations deviating from a neutral pose are punished using the loss function described in Equation (6.3). For the purpose of this figure, the weighting scheme introduced in Equation (6.4) has not been used, such that the scores in Figure 8.11 are in the range $q_o \in [0, 3]$.

Finally, evaluating grasps purely based on their amount of inliers gives results as in Figure 8.12. This is the only metric completely disregarding placement, only optimizing for the sheer volume of perceived points in the closing region. For the results shown in this figure, the maximum amount of inliers is set to 135 points, which approximately corresponds to 5% of the point cloud with 2725 points. As seen from the quality indicator, 135 maximum inliers appear to be hard to achieve

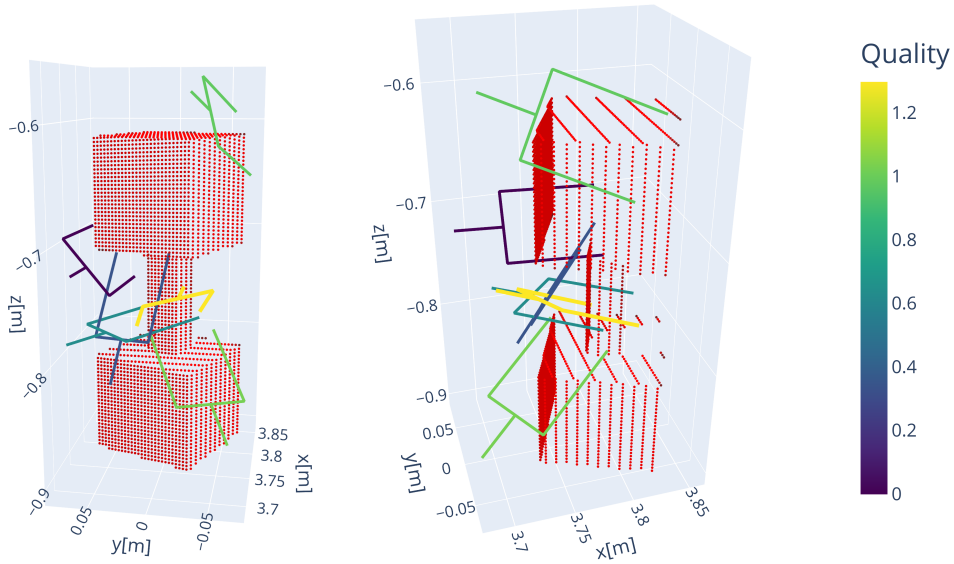


Figure 8.10: Six grasps evaluated using contact point alignment. Due to the view angle and shape of the object, no two surfaces where the gripper fits are parallel in the point cloud, which limits the accuracy of the quality measure in this case.

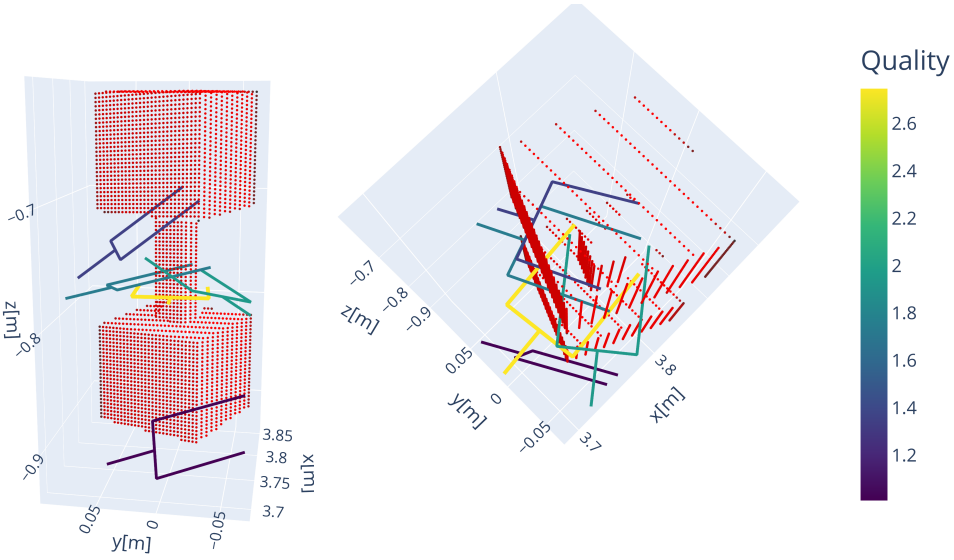


Figure 8.11: Grasp poses evaluated purely based on their orientation. No weighting has been done for the individual directions in this figure.

for this point cloud, and a weighting or readjustment might be warranted for inliers to have a greater impact on the final scoring.

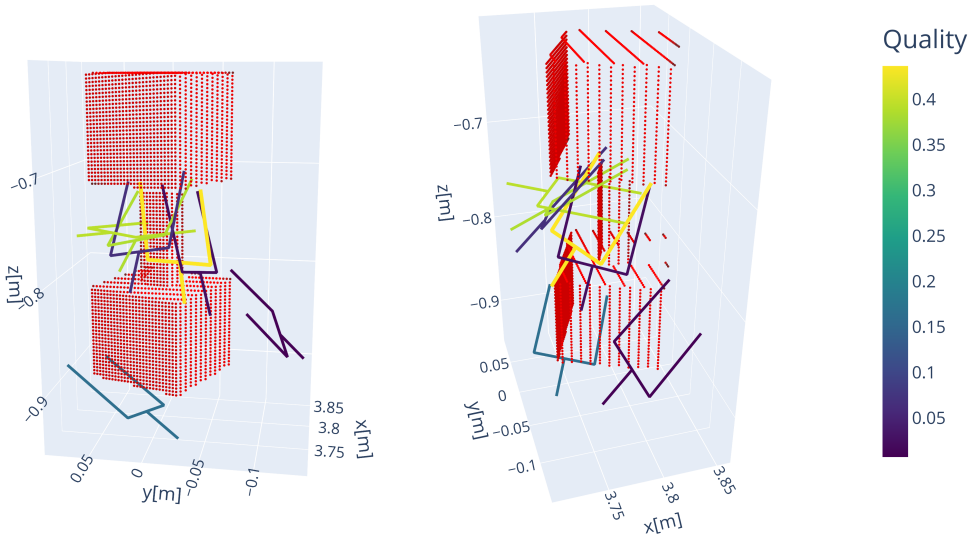


Figure 8.12: Grasp poses wholly evaluated based on their amount of inliers. Note how more tilted grasps are favored at the corners of the square pieces, as this maximizes the used volume of the closing region.

8.5 Final Results

Finally, all quality metrics are combined in order to evaluate the candidate grasp poses. However, as this thesis does not cover control and final interaction with the object, no experiments performing interaction have been conducted. Thus, the grasp poses that score the best are evaluated by inspection to confirm whether or not the grasp is a success.

The highest rated grasp pose from 50 iterations is kept for each of the objects, Figures 8.13 to 8.15 show these 50 poses. In each iteration, 3000 poses are sampled. For the results in Figures 8.13 to 8.15 the weights shown in Table 8.5 are used. The weighting scheme for orientation shown in Equation (6.4) is applied, with emphasis on the fact that achieving poses with large yaw is less problematic than roll or pitch. For the barbell, normal sampling is used with no scaling on standard deviation in position, as this produces more grasps close to the more graspable centroid. For all objects, the standard deviation in orientation is $\frac{\pi}{4}$ for all directions. Uniform sampling is applied for the cross and the circle, as this produces more feasible poses based on their geometries, ref. Table 8.3. The histograms in Figure 8.16 show the distribution of scores for the 150 total grasps in the three figures.

Table 8.5: The weights used for the final results.

	w_n	w_o	$w_{o,\phi}$	$w_{o,\theta}$	$w_{o,\psi}$	w_c	w_i	Maximum Inliers
Barbell	1.0	1.0	0.45	0.45	0.1	1.0	2.0	135
Circle	1.0	1.0	0.45	0.45	0.1	1.0	1.0	135
Cross	1.0	1.0	0.45	0.45	0.1	1.0	1.0	250

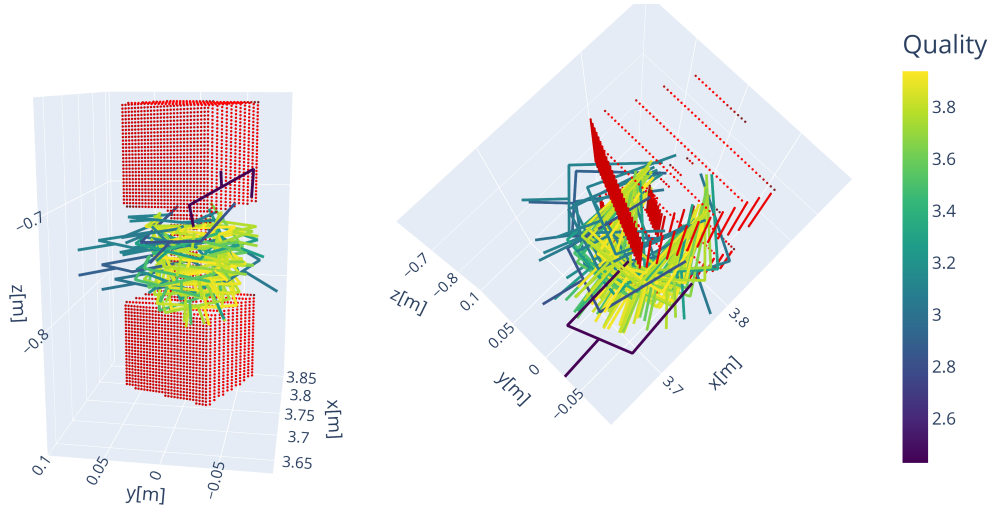


Figure 8.13: The 50 highest evaluated grasps from as many iterations on the barbell.

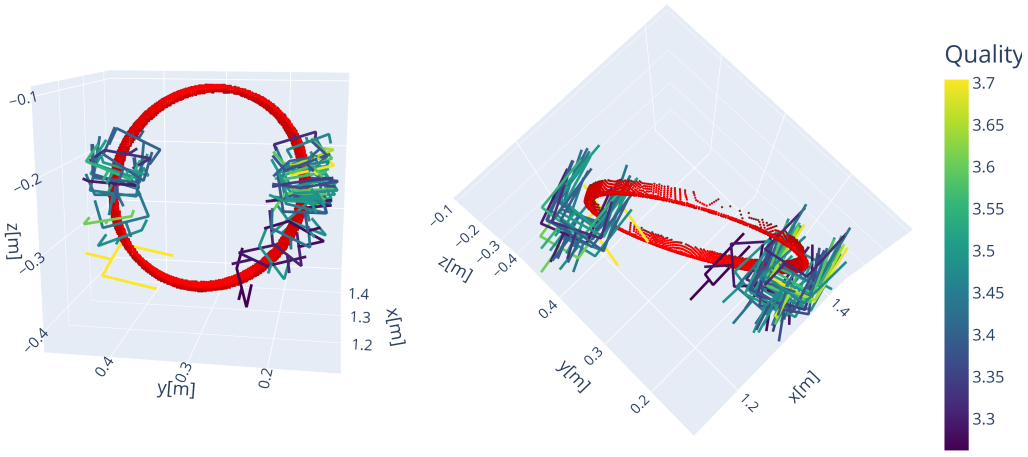


Figure 8.14: The 50 highest evaluated grasps from as many iterations on the circle.

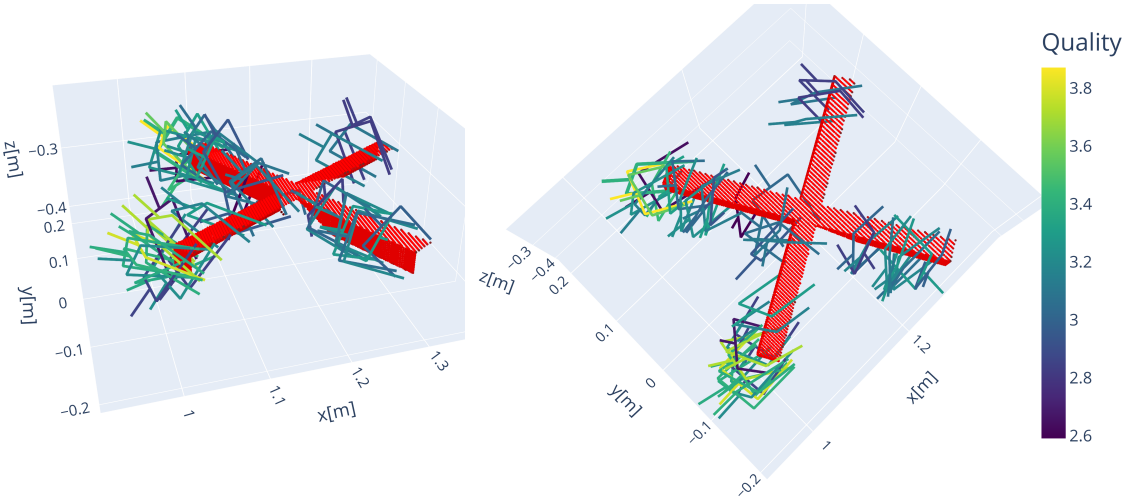


Figure 8.15: The 50 highest evaluated grasps from as many iterations on the cross.

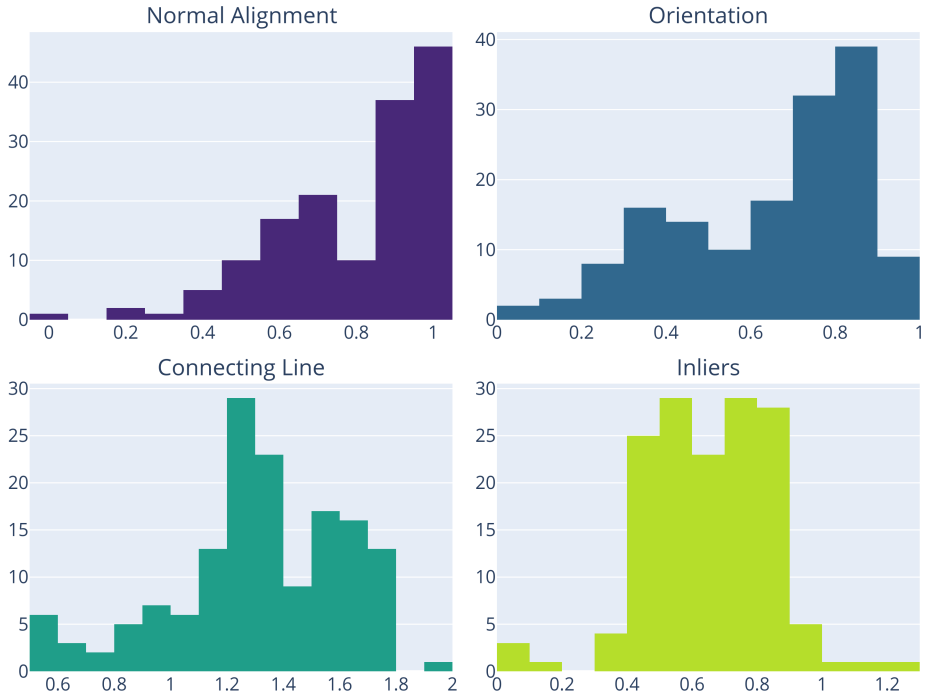


Figure 8.16: Histograms of scores for all grasps in Figures 8.13 to 8.15.

Chapter 9

Discussion

This chapter will discuss the performance of the grasp pose detection method as presented in the preceding chapter. Firstly, the details around sampling methods are discussed in Section 9.1. Thereafter, Section 9.2 brings up some strengths and shortcomings of the chosen quality metrics, before Section 9.3 sheds light on some future work that could further improve the method described in this thesis.

9.1 Sampling

The two proposed sampling methods have different properties proven to perform well in their intended situations. Uniform sampling generates poses that are evenly spread over the object, which proves to be useful in situations where the object has a graspable region away from the centroid. The weakness is that many samples are performed at the edges of the detected point cloud, which generates multiple poses such as the lower left one in Figure 8.12, where the object surface is misinterpreted as collision-free. The number of such samples could probably in most cases be reduced by implementing a surveying method of the scene before sampling, such that a more proper representation of the object is available.

Scaling the standard deviation when using normal sampling can be hard. As seen in Figure 8.6a, where the height of the object gives a high variance in the z direction. While the method still produces a high density of samples towards the centroid, much time is spent on samples that are easily identified as infeasible. Furthermore, objects that appear narrow to the camera do not always have a sufficiently high variance to fully cover the object in samples. A possible strategy that has not been tested could be to start out with a small standard deviation, then increase it repeatedly until adequate sampling results are found.

The biggest weakness of both sampling methods does not lie directly in the sampling, but rather in how collisions are checked. As collision checking takes orders of magnitude longer than every other step, this is the greatest limiting factor of the algorithm. However, as shown in Table 8.3, the sampling algorithms provide an average of barely 1% usable poses, which might indicate that improvements

could be made in the sampling strategy in order to provide better initial samples. In this way, fewer samples could be used in order to generate the same amount of usable poses. Nevertheless, with both sampling methods described in this thesis, the algorithm has been shown to provide suitable grasping poses for the three objects used in the simulation.

9.2 Evaluation

The curated collection of grasps in Section 8.4 showcases the properties of the different quality measures used. As mentioned, normal angle alignment compares the gripper normal with the point closest to the centroid of the closing region. For grasp candidates in orientations where the closest point poorly represents the majority of the surface in the closing region, this might become a poor indication of actual alignment. One such case could be for a corner or an edge, as the curvature of the region influences the estimated surface normal. On the other hand, the measure gives a fairly decent indication of how well the grasp aligns with the object, which was shown in [50] to be a measure commonly used with humans. This is also reflected by the histogram of the highest scoring grasps having normal scores heavily skewed towards 1, Figure 8.16.

Alignment of the connecting line is also highly tied to the accuracy of the estimated surface normals. Furthermore, in order to use this measure as efficiently as possible, the surface enclosed in the closing region should represent both contact planes accurately. Using the point cloud construction method discussed in this thesis the objects are only partially reconstructed in 3D, according to the viewpoint of the camera. Thus, information about two opposing surfaces is seldom available, limiting the ability of this quality measure. The quality measure nevertheless provides a decent indication of the desired property, and as seen in Figure 8.10 it too helps indicating how well a grasp is estimated to perform.

In Section 6.4 details about friction at the contact points are presented. The implemented metric measuring this alignment for the results shown in Chapter 8 completely disregards the friction coefficient, and only focuses on how well the connecting line aligns with the estimated surface normals at the contact points. Knowing the friction coefficient between the gripper and the object in the conditions where the object is placed could further improve the usefulness of this metric, as a more accurate representation of the friction cones could be achieved.

The orientation metric is a fairly straightforward way of making sure that orientations harder to achieve for the ROV are rated lower. However, this metric also requires some considerations of the object. For the cross seen in Figure 8.15, grasps at the ends have been evaluated better than those at higher roll and pitch. If the aim is to pick up the cross, this would most likely lead to an unstable grasp due to the wrench applied by gravity. On the contrary, if the cross is rigidly mounted

in the center and the goal is to turn in, like a valve, grasps at the ends can apply more torque to the center. This highlights some of the object-specific considerations that need to be applied for the orientation scoring function. In hindsight, the scoring function explained in Figure 6.3 should not periodically increase again, as this facilitates for poses where for instance a yaw of 180° is evaluated as very good when in reality there is no proper knowledge about the backside of the object. This problem has been avoided by using a sufficiently low standard deviation when sampling such that such poses are highly unlikely, but the scoring function should probably be revisited regardless.

Some discussion of the accuracy of the inlier measurement is given in Section 6.5. The empirical nature of the metric as implemented for this thesis makes it highly dependent on both the density and geometry of the point cloud. For objects such as the circle in Figure 8.14, grasps at a higher angle tend to be evaluated better, as they can fit more of the circle inside the grasping region. Given sufficient tuning and weighting of the inlier and normal alignment scores, however, this effect should be possible to minimize.

9.3 Future Work

In order for the simulation to be an accurate representation of the system in situ, the assumptions presented in Section 7.4 are of importance. It is worth pointing out a final time that the rough underwater conditions put the system presented in this thesis under pressure, as the presented grasping algorithms rely heavily on the accuracy of the point clouds produced. The validity of the assumptions was discussed a bit in Section 7.4, but further testing of the method outside of simulation will tell whether or not they are appropriately reasonable.

Furthermore, as collision checking is the part of the algorithm that takes the most time, there should be little to no loss in making the quality evaluation metrics more robust and advanced. By doing a proper reconstruction of the surface in the grasping region, more accurate measures connected to gripper geometry could be employed. The connecting line and friction cone calculations could be further improved with knowledge of friction coefficients and a more accurate representation of the gripper's finger movement. Not to mention that performing a survey of the area beforehand in order to get a better representation of the scene from multiple angles should be prioritized for higher accuracy and less uncertainty in the sampled grasps. An option is also to look into other methods of collision detection. It might be worth looking into a method based on the intersection of planes constructed by the gripper geometry and points in the closing region, such that only a few points need to be visited, rather than the whole point cloud.

The sampling algorithms proposed use random sampling to a large extent, in order to cover the object. However, implementing more prior information in

the sampling strategies could be explored. For instance, sampling orientations around a mean pointing in the direction of the object, rather than around a neutral orientation, might provide more viable poses. Further, introducing a method of choosing what sample method to use based on the object geometry could help avoid situations where many samples are infeasible, as in Figure 8.7c. In general, investigating more sampling methods in order to provide more feasible grasps could prove fruitful.

Chapter 10

Conclusion

This thesis has presented a full pipeline from stereo camera vision to candidate grasp pose generation for an autonomous underwater vehicle, fully based on geometrical considerations of the perceived scene. Stereo vision is introduced as a method of reconstructing the perceived scene as a 3D point cloud. To produce candidate grasp poses, two grasp pose sampling strategies have been presented. The first method uses a normal distribution with standard deviation based on the observed object's geometry, whilst the second method samples uniformly over the point cloud. The two sampling methods boast properties that are advantageous in different situations based on the observed object's geometry.

To evaluate the candidate grasp poses, this thesis implements quality measures based on the geometrical properties of the gripper and the observed point cloud. The suggested quality measures provide an intuitive way of measuring different aspects of a grasp's quality and allow for desired properties to be prioritized through weighting. Two metrics that evaluate the gripper's alignment with the object are suggested. A scoring function is introduced to give grasping poses with easy-to-achieve orientations better evaluations, such that the resulting grasps are more suitable for systems unable to individually move their gripper. Collision checking of a sampled gripper pose and the point cloud is a heavily limiting factor concerning the efficiency of the presented method, and further work should aim to improve this factor.

:wq

Bibliography

[1] G. N. Baturin, "Mineral resources of the ocean," *Lithology and Mineral Resources*, vol. 35, no. 5, pp. 399–424, Sep. 2000. DOI: 10.1007/bf02782727. [Online]. Available: https://doi.org/10.1007/bf02782727.

[2] F. FAO *et al.*, "The state of world fisheries and aquaculture," *Opportunities and challenges. Food and Agriculture Organization of the United Nations*, 2012.

[3] *Petronius Compliant Tower, All Areas - SkyscraperPage.com — skyscraperpage.com*, https://skyscraperpage.com/cities/?buildingID=23522, [Accessed 30-May-2023].

[4] E. Simetti, "Autonomous Underwater Intervention," *Current Robotics Reports 2020 1:3*, vol. 1, no. 3, pp. 117–122, Jun. 2020, ISSN: 2662-4087. DOI: 10.1007/S43154-020-00012-7. [Online]. Available: https://link.springer.com/article/10.1007/s43154-020-00012-7.

[5] R. Schettini and S. Corchs, "Underwater image processing: State of the art of restoration and image enhancement methods," *EURASIP journal on advances in signal processing*, vol. 2010, pp. 1–14, 2010.

[6] S. Bazeille, I. Quidu, and L. Jaulin, "Color-based underwater object recognition using water light attenuation," *Intelligent Service Robotics*, vol. 5, no. 2, pp. 109–118, Jan. 2012. DOI: 10.1007/s11370-012-0105-3. [Online]. Available: https://doi.org/10.1007/s11370-012-0105-3.

[7] D. L. Rizzini, F. Kallasi, F. Oleari, and S. Caselli, "Investigation of vision-based underwater object detection with multiple datasets," *International Journal of Advanced Robotic Systems*, vol. 12, no. 6, p. 77, Jan. 2015. DOI: 10.5772/60526. [Online]. Available: https://doi.org/10.5772/60526.

[8] J. Redmon, S. K. Divvala, R. B. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," *CoRR*, vol. abs/1506.02640, 2015. arXiv: 1506.02640. [Online]. Available: http://arxiv.org/abs/1506.02640.

[9] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "Ssd: Single shot multibox detector," in *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part I 14*, Springer, 2016, pp. 21–37.

[10] R. Mandal, R. M. Connolly, T. A. Schlacher, and B. Stantic, "Assessing fish abundance from underwater video using deep neural networks," in *2018 International Joint Conference on Neural Networks (IJCNN)*, 2018, pp. 1–6. DOI: 10.1109/IJCNN.2018.8489482.

[11] T. Katayama, T. Song, T. Shimamoto, and X. Jiang, "GAN-based color correction for underwater object detection," in *OCEANS 2019 MTS/IEEE SEATTLE*, IEEE, Oct. 2019. DOI: 10.23919/oceans40490.2019.8962561. [Online]. Available: https://doi.org/10.23919/oceans40490.2019.8962561.

[12] S. Xu, M. Zhang, W. Song, H. Mei, Q. He, and A. Liotta, "A systematic review and analysis of deep learning-based underwater object detection," *Neurocomputing*, vol. 527, pp. 204–232, Mar. 2023. DOI: 10.1016/j.neucom.2023.01.056. [Online]. Available: https://doi.org/10.1016/j.neucom.2023.01.056.

[13] R. O. Faria, F. Kucharczak, G. M. Freitas, A. C. Leite, F. Lizarralde, M. Galassi, and P. J. From, "A methodology for autonomous robotic manipulation of valves using visual sensing," *IFAC-PapersOnLine*, vol. 48, no. 6, pp. 221–228, 2015.

[14] N. Palomeras, A. Peñalver, M. Massot-Campos, P. Negre, J. Fernández, P. Ridao, P. Sanz, and G. Oliver-Codina, "I-AUV docking and panel intervention at sea," *Sensors*, vol. 16, no. 10, p. 1673, Oct. 2016. DOI: 10.3390/s16101673. [Online]. Available: https://doi.org/10.3390/s16101673.

[15] B. S. Zapata-Impata, P. Gil, J. Pomares, and F. Torres, "Fast geometry-based computation of grasping points on three-dimensional point clouds," *International Journal of Advanced Robotic Systems*, vol. 16, no. 1, Jan. 2019, ISSN: 17298814. DOI: 10.1177/1729881419831846. [Online]. Available: https://www.researchgate.net/publication/331358070_Fast_Geometry-based_Computation_of_Grasping_Points_on_Three-dimensional_Point_Clouds.

[16] A. ten Pas, M. Gualtieri, K. Saenko, and R. Platt, "Grasp Pose Detection in Point Clouds," *International Journal of Robotics Research*, vol. 36, no. 13-14, pp. 1455–1473, Jun. 2017, ISSN: 17413176. DOI: 10.48550/arxiv.1706.09911. [Online]. Available: https://arxiv.org/abs/1706.09911v1.

[17] T. I. Fossen, *Handbook of Marine Craft Hydrodynamics and Motion Control*. Wiley-Blackwell, 2021, vol. Second Edition, ISBN: 9781119575054.

[18] S. Dupré, "Inside the camera obscura: Kepler's experiment and theory of optical imagery," *Early Science and Medicine*, vol. 13, no. 3, pp. 219–244, 2008. DOI: https://doi.org/10.1163/157338208X285026. [Online]. Available: https://brill.com/view/journals/esm/13/3/article-p219_2.xml.

[19] R. I. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*, Second. Cambridge University Press, ISBN: 0521540518, 2004.

[20] C. B. Duane, "Close-range camera calibration," *Photogramm. Eng*, vol. 37, no. 8, pp. 855–866, 1971.

[21] Z. Zhang, "A flexible new technique for camera calibration," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, no. 11, pp. 1330–1334, 2000. DOI: 10.1109/34.888718.

[22] A. Agrawal, S. Ramalingam, Y. Taguchi, and V. Chari, "A theory of multi-layer flat refractive geometry," in *2012 IEEE Conference on Computer Vision and Pattern Recognition*, 2012, pp. 3346–3353. DOI: 10.1109/CVPR.2012.6248073.

[23] T. Treibitz, Y. Schechner, C. Kunz, and H. Singh, "Flat refractive geometry," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 34, no. 1, pp. 51–65, Jan. 2012. DOI: 10.1109/tpami.2011.105. [Online]. Available: https://doi.org/10.1109/tpami.2011.105.

[24] T. Łuczyński, M. Pfingsthorn, and A. Birk, "The Pinax-model for accurate and efficient refraction correction of underwater cameras in flat-pane housings," *Ocean Engineering*, vol. 133, pp. 9–22, 2017, ISSN: 00298018. DOI: 10.1016/j.oceaneng.2017.01.029.

[25] A. Sedlazeck, K. Koser, and R. Koch, "3d reconstruction based on underwater video from ROV kiel 6000 considering underwater imaging conditions," in *OCEANS 2009-EUROPE*, IEEE, May 2009. DOI: 10.1109/oceanse.2009.5278305. [Online]. Available: https://doi.org/10.1109/oceanse.2009.5278305.

[26] M. Johnson-Roberson, O. Pizarro, S. B. Williams, and I. Mahon, "Generation and visualization of large-scale three-dimensional reconstructions from underwater robotic surveys," *Journal of Field Robotics*, vol. 27, no. 1, pp. 21–51, Jan. 2010. DOI: 10.1002/rob.20324. [Online]. Available: https://doi.org/10.1002/rob.20324.

[27] T. Tuytelaars and K. Mikolajczyk, "Local invariant feature detectors: A survey," *Foundations and Trends® in Computer Graphics and Vision*, vol. 3, no. 3, pp. 177–280, 2007. DOI: 10.1561/0600000017. [Online]. Available: https://doi.org/10.1561/0600000017.

[28] A. Kirillov, E. Mintun, N. Ravi, H. Mao, C. Rolland, L. Gustafson, T. Xiao, S. Whitehead, A. C. Berg, W.-Y. Lo, P. Dollár, and R. Girshick, *Segment anything*, 2023. DOI: 10.48550/ARXIV.2304.02643. [Online]. Available: https://arxiv.org/abs/2304.02643.

[29] C. Harris, M. Stephens, *et al.*, "A combined corner and edge detector," in *Alvey vision conference*, Citeseer, vol. 15, 1988, pp. 10–5244.

[30] D. G. Lowe, "Object recognition from local scale-invariant features," in *Proceedings of the seventh IEEE international conference on computer vision*, Ieee, vol. 2, 1999, pp. 1150–1157.

[31] M. Brown, R. Szeliski, and S. Winder, "Multi-scale oriented patches," 2004.

[32] S. van der Walt, J. L. Schönberger, J. Nunez-Iglesias, F. Boulogne, J. D. Warner, N. Yager, E. Gouillart, and T. Yu, "Scikit-image: Image processing in python," *PeerJ*, vol. 2, e453, Jun. 2014. DOI: 10.7717/peerj.453. [Online]. Available: https://doi.org/10.7717/peerj.453.

[33] R. Szeliski, *Computer Vision Algorithms and Applications*. Springer London Ltd, 2010, ISBN: 9781848829343.

[34] L. Keselman, J. Iselin Woodfill, A. Grunnet-Jepsen, and A. Bhowmik, "Intel realsense stereoscopic depth cameras," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, Jul. 2017.

[35] H. C. Longuet-Higgins, "A computer algorithm for reconstructing a scene from two projections," *Nature*, vol. 293, no. 5828, pp. 133–135, 1981.

[36] R. I. Hartley and P. Sturm, "Triangulation," *Computer Vision and Image Understanding*, vol. 68, no. 2, pp. 146–157, Nov. 1997. DOI: 10.1006/cviu.1997.0547. [Online]. Available: https://doi.org/10.1006/cviu.1997.0547.

[37] Z. Zhang, "Microsoft kinect sensor and its effect," *IEEE MultiMedia*, vol. 19, no. 2, pp. 4–10, 2012. DOI: 10.1109/MMUL.2012.24.

[38] A. Reichinger, *Kinect Pattern Uncovered*, Apr. 2011. [Online]. Available: https://azttm.wordpress.com/2011/04/03/kinect-pattern-uncovered/ (visited on 07/09/2017).

[39] J. L. Bentley, "Multidimensional binary search trees used for associative searching," *Communications of the ACM*, vol. 18, no. 9, pp. 509–517, 1975.

[40] R. A. Brown, "Building a balanced k-d tree in $O(kn \log n)$ time," *Journal of Computer Graphics Techniques (JCGT)*, vol. 4, no. 1, pp. 50–68, Mar. 2015, ISSN: 2331-7418. [Online]. Available: http://jcgt.org/published/0004/01/03/.

[41] J. Viquerat, *Latex_recipes*, https://github.com/jviquerat/latex_recipes/, 2022.

[42] J. H. Friedman, J. L. Bentley, and R. A. Finkel, "An algorithm for finding best matches in logarithmic expected time," *ACM Transactions on Mathematical Software*, vol. 3, no. 3, pp. 209–226, Sep. 1977. DOI: 10.1145/355744.355745. [Online]. Available: https://doi.org/10.1145/355744.355745.

[43] R. B. Rusu, Z. C. Marton, N. Blodow, M. Dolha, and M. Beetz, "Towards 3d point cloud based object maps for household environments," *Robotics and Autonomous Systems*, vol. 56, no. 11, pp. 927–941, Nov. 2008. DOI: 10.1016/j.robot.2008.08.005. [Online]. Available: https://doi.org/10.1016/j.robot.2008.08.005.

[44] M. A. Fischler and R. C. Bolles, "Random sample consensus," *Communications of the ACM*, vol. 24, no. 6, pp. 381–395, Jun. 1981. DOI: 10.1145/358669.358692. [Online]. Available: https://doi.org/10.1145/358669.358692.

[45] P. Torr and A. Zisserman, "MLESAC: A new robust estimator with application to estimating image geometry," *Computer Vision and Image Understanding*, vol. 78, no. 1, pp. 138–156, Apr. 2000. DOI: 10.1006/cviu.1999.0832. [Online]. Available: https://doi.org/10.1006/cviu.1999.0832.

[46] J. Berkmann and T. Caelli, "Computation of surface geometry and segmentation using covariance techniques," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 16, pp. 1114–1116, 1994.

[47] Radu Bogdan Rusu, "Semantic 3D Object Maps for Everyday Manipulation in Human Living Environments," Ph.D. dissertation, Technische Universitaet Muenchen, Munich, Germany, Oct. 2009. [Online]. Available: https://mediatum.ub.tum.de/doc/800632/941254.pdf.

[48] X. Huang, G. Mei, J. Zhang, and R. Abbas, *A comprehensive survey on point cloud registration*, 2021. DOI: 10.48550/ARXIV.2103.02690. [Online]. Available: https://arxiv.org/abs/2103.02690.

[49] A. Nguyen and B. Le, "3d point cloud segmentation: A survey," in *2013 6th IEEE Conference on Robotics, Automation and Mechatronics (RAM)*, 2013, pp. 225–230. DOI: 10.1109/RAM.2013.6758588.

[50] R. Balasubramanian, L. Xu, P. D. Brook, J. R. Smith, and Y. Matsuoka, "Human-guided grasp measures improve grasp robustness on physical robot," in *2010 IEEE International Conference on Robotics and Automation*, IEEE, May 2010. DOI: 10.1109/robot.2010.5509855. [Online]. Available: https://doi.org/10.1109/robot.2010.5509855.

[51] R. M. Murray, Z. Li, S. S. Sastry, and S. S. Sastry, *A mathematical introduction to robotic manipulation*. CRC press, 1994.

[52] B. Robotics, *Bluerov2*, https://bluerobotics.com/store/rov/bluerov2/, [Accessed 31-May-2023], Apr. 2023. [Online]. Available: https://bluerobotics.com/store/rov/bluerov2/.

[53] R. B. Rusu and S. Cousins, "3D is here: Point Cloud Library (PCL)," in *IEEE International Conference on Robotics and Automation (ICRA)*, Shanghai, China, May 2011.

Appendix A

Image Operations

A.1 Linear Filtering

Linear filtering on images calculates an output image pixel $g(i,j)$ as a linear combination of its neighboring intensity values $f(i,j)$ and the weighted coefficients $h(k,l)$, known as the kernel.

$$g(i,j) = \sum_{k,l} f(i-k, j-l)h(k,l) = \sum_{k,l} f(k,l)h(i-k, j-l) \tag{A.1}$$

Which can also be expressed as $g(i,j) = (f * h)(i,j)$ where $*$ is the convolution operator.

A.2 Gaussian Blur

Gaussian blurring is achieved by convolving the image with a Gaussian kernel. A two-dimensional Gaussian kernel is given by

$$G(x,y) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{A.2}$$

where x and y represent the position relative to the center, and σ the standard deviation. A higher σ produces a more potent blurring effect. A Gaussian kernel of size 5×5 and with $\sigma = 1$ is shown in Equation (A.3), where numbers have been rounded to integers.

$$\begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 25 & 16 & 4 \\ 6 & 25 & 40 & 25 & 6 \\ 4 & 16 & 25 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix} \tag{A.3}$$

A.3 Image Derivatives

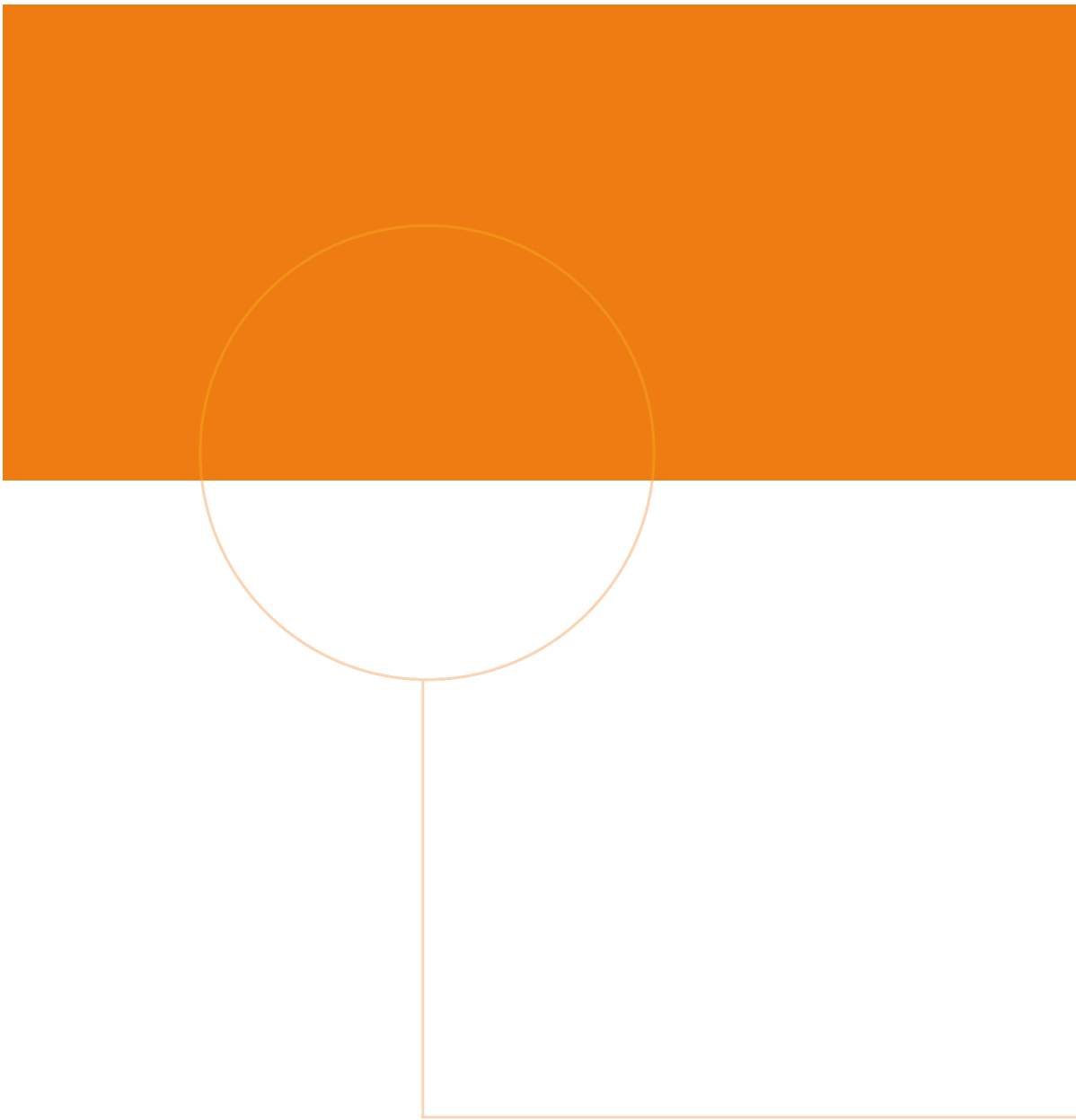
The partial derivative is defined as

$$\frac{\partial}{\partial x} f(x,y) = \lim_{\Delta \to 0} \frac{f(x+\Delta, y) - f(x,y)}{\Delta} \tag{A.4}$$

which in the discrete case approximates as

$$\frac{\partial}{\partial x} f(x,y) \approx \frac{f(x+\Delta x, y) - f(x,y)}{\Delta x} \tag{A.5}$$

In an image, Δx is 1, and the derivative is the difference between the current pixel value and the direct neighbor, i.e. $f(i+1,j) - f(i,j)$. Thus, image partial derivatives can be computed through convolution with the kernel $k = [-1, 1]$ for the x-direction and $k^\top$ in the y-direction.



NTNU

Norwegian University of
Science and Technology