MASTER THESIS

Real-Time Object Shape Detection using ROS, the KUKA LWR4+ and a Force/Torque Sensor

submitted by

Stephan Rau

University of Hamburg

Department of Computer Science

TAMS - Technical Aspects of Multimodal Systems

M.Sc. Computer Science

Matriculation-No: 6139152

Primary supervisor: Prof. Dr. Jianwei Zhang

Secondary supervisor: Dr. Norman Hendrich

Abstract

The ability of robots to handle complex tasks is continuously increasing. Sophisticated tasks often require sensors to enable robots to experience their surroundings to act in or react to changes in the environment. Sensors do not perform reliably in all circumstances which might result in wrong or insufficient information causing the inability of robot systems to reason or even lead to wrong decision making. Acting in an environment often requires to navigate in or to manipulate the environment, e.g., grasping objects or pushing them away. Naturally, vision systems are used to map the environment and detect objects to interact in it. However, vision systems may fail due to reasons as occlusion or poor lighting conditions. Tactile sensing systems, on the contrary, are not dependent on those conditions and constitute a better choice in many situations.
In this thesis, a concept for blind object surface exploration and object surface reconstruction using a tactile sensor has been developed and implemented. This way information about the environment of the robot can be gained independently of camera sensors. The aim is to supplement or even temporarily replace vision approaches. The resulting approach is based on force readings from a six-axis force/torque sensor to detect contacts between the robot and objects in the environment. The information on the contact points is utilized to locate and construct the shape of the objects.

Zusammenfassung

Die Fähigkeit von Robotern, komplexe Aufgaben zu bearbeiten, steigt stetig. Die Bewältigung anspruchsvoller Aufgaben erfordert oft die Wahrnehmung der Umgebung mittels Sensoren, um agieren zu können oder auf Änderungen zu reagieren. Bestimmte Bedingungen können dazu führen, dass Sensoren keine zuverlässige Werte liefern, was wiederum dazu führen kann, dass Roboter falsche oder ungenaue Informationen bekommen und daher nicht mehr in der Lage sind zu entscheiden, wie sie weiter agieren sollen oder sogar falsche Entscheidungen treffen. In einer Umgebung zu agieren bedeutet oft in ihr zu navigieren oder die Umgebung zu manipulieren wie beispielsweise beim Greifen oder Wegdrücken eines Gegenstands. Bildverarbeitungssysteme werden üblicherweise eingesetzt um eine Umgebung abzubilden oder Gegenstände zu erkennen. Allerdings können Bildverarbeitungssysteme aus unterschiedlichen Gründen wie Okklusion oder schlechter Lichtverhältnisse unbrauchbar werden. Taktile Systeme sind nicht von diesen Umständen abhängig und stellen oftmals eine bessere Wahl zur Objekterkennung dar.
In dieser Masterarbeit wurde ein Konzept für blinde Exploration von Objektoberflächen und Objektrekonstruktion mittels taktilem Sensor entwickelt und umgesetzt, um Informationen über die Umgebung eines Roboters unabhängig von Kamerasensoren zu gewinnen. Das Ziel des Konzepts ist die Ergänzung von Bildverarbeitungssystemes beziehungsweise die Möglichkeit einer temporären Nutzung des Konzepts anstelle ihrer um in unbekannten Umgebungen agieren zu können. Die resultierende Vorgehensweise basiert auf Messwerten eines sechs Achsen Kraft-/Drehmomentensensors zum Wahrnehmen von Kontakten zwischen Roboter und Objekten in der Umgebung. Die gewonnenen Informationen bezüglich der Kontaktpunkte werden zur Lokalisierung und Rekonstruktion der Oberfläche der Objekte verwendet.

Contents

List of Figures

1 Introduction

In this chapter first the topics of robotics and robots are introduced. More and more application areas are accessed by robots with increasing versatility in their capabilities to interact with their environment. The section on motivation discusses the need for blind shape detection alongside vision approaches followed by the description of the goals of the thesis. The section 1.5 on the outline of the thesis shows the structure of the thesis by reference to the following chapters.

1.1 Robotics and Robots

The International Organization for Standardization (ISO) defines a robot as "actuated mechanism programmable in two or more axes with a degree of autonomy, moving within its environment, to perform intended tasks" [12]. Robotics is defined as the science and practice of designing, manufacturing, and applying robots [12]. Phrased alternatively, robotics core is the development and control of robots as interdisciplinary task of the domains of computer science, electrical and mechanical engineering among others.
The usage of robots is manifold and generally takes place where the usage of robots is more economical than without, in dangerous environments, as service robots as well as in scientific research to test theories via application on real subjects. The usage of robots is manifold and generally takes place where the usage of robots is more economical than without, in dangerous environments, as service robots as well as in scientific research to test theories via the application on real subjects. Robots typically perform tasks which humans prefer not to do or where robots are more efficient. Tasks humans might not want to perform among others are repetitive tasks and dangerous tasks, for example, in manufacturing. Furthermore, they can be designed to execute one task or a category of tasks particularly efficient. Dangerous tasks can be consigned to robots to avoid causing harm to humans. Removing explosive devices, performing in radioactive environments [13] or executing manufacturing tasks in hostile environments represent such tasks. The purpose of robots to execute tasks in place of humans occurs in more and more types of environments. They act in healthcare services, entertainment, education tasks and other domains [14].
The ISO classifies robots in industrial and service robots, leaving out research driven robots. An industrial robot is "automatically controlled, reprogrammable, multipurpose manipulator, programmable in three or more axes, which can be either fixed in place or mobile for use in industrial automation applications" [12] and based on this definition a service robot is a "robot that performs useful tasks for humans or equipment excluding industrial automation applications" [12]. The International Federation of Robotics (IFR) provides statistics on the increase of estimated annual worldwide supply of industrial robots (figure 1.1). The numbers show large increases in supply and sale of robots in the last decade as well as forecasts increases in the

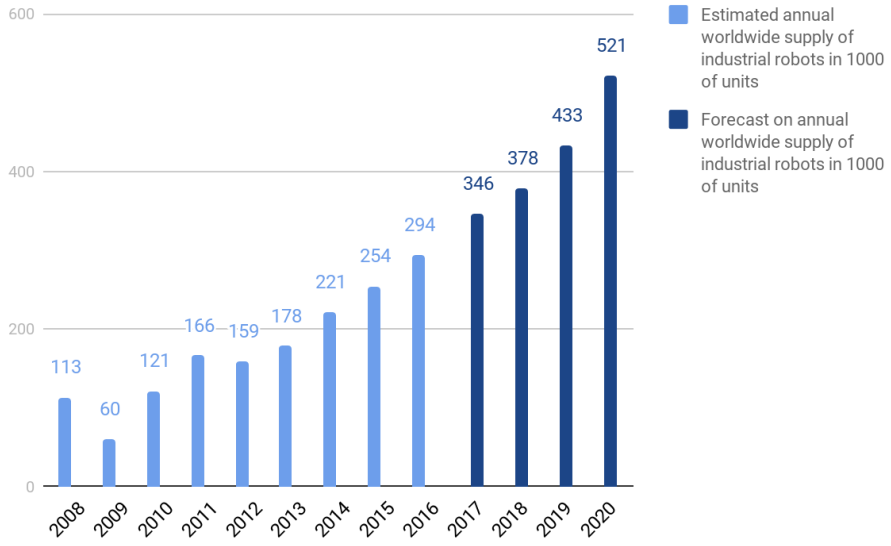


Figure 1.1: Estimated annual worldwide supply of industrial robots in 1000 of units [1]. The numbers of sold and supplied robots considerably increased over the years and remains increasing for the upcoming years.

demand for robots in the upcoming years. Furthermore, the IFR provides numbers on the operation stock of industrial robots stating 1.8 million units in 2016 with an average of 14% annual increase to 3.05 million units in 2020 [1].

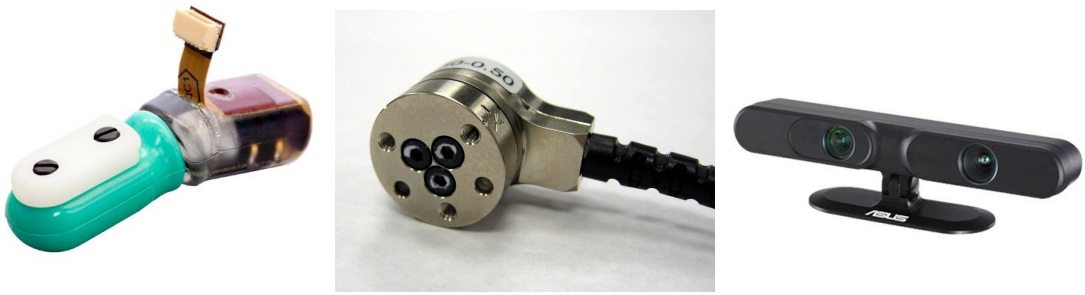
The statistics in figure 1.1 depict the increasing numbers of sold and supplied robots and indicate an increasing impact on society, e.g., by substituting for humans.

Independent of robot classification robotic sensing is mandatory for autonomy and sophisticated applications in robots. Robotic sensing permits robots to perceive their environment via tactile sensors, vision sensors, and others. The interpretation of sensor readings enables the development of robotic applications for non-static environments through the ability to react to changes in the environment. The development of more complex and autonomous robots and robotic applications is tied to advancements in sensor technology [15] as well as on other criteria as the design of the robots regarding the shape and kinematic specifications.

1.2 Motivation

An autonomous robot needs to be able to act in an environment to fulfill a purpose. While environment is understood as the surroundings regarding objects and the prevailing physical laws, acting in general means to move in an environment and to manipulate the environment. Regarding the manipulation of the environment, as can be the manipulation of a single object such as

a wooden block, the autonomous robot requires a manipulator and information about the object to be manipulated. A typical manipulation task is to grasp an object. Useful information for that purpose can be the position of the object concerning the robot, whether it is in motion, its weight and surface characteristics but foremost the shape and pose. That information enables the autonomous robot to reason about how to grasp the object, if possible. Considering motion in an environment a recurring task is to plan motions without collisions. Again information about position as well as shape and pose of the objects is crucial to ensure collision-free motion of the robot.
Different approaches exist to feed the robot with the necessary information to act autonomously to a certain degree in an environment. Three approaches shall be mentioned here.
First, the robot can be fed with a map corresponding to the environment if it is known beforehand as well as the position of the robot on the map. Likewise, information about inertia or motion of objects in the environment can be given to the robot beforehand. Continuing this approach by transferring information about the environment to the robot possibly leads to independence on sensing the environment with sensors affixed to the robot. Since all necessary information about the environment has to be known and represented beforehand, the robot does not act autonomously anymore but preprogrammed. Although industrial robots nowadays use sensors to act and react to environmental changes, they often act in known environments and have constantly recurring motion patterns. Hence they are not acting holistically autonomous.
Secondly and most widely used and investigated (cf. [16]) to get information about the environment, concepts using visual approaches as with 3D cameras or laser range finder are applied. They offer a rather unproblematic way of mapping the environment and gather information about the proximity of objects to the robot as well as taking shape and pose of objects.
Thirdly tactile approaches can be used to gain information about the environment by touching the existing objects. Similar to humans exploring an object by touch, robots can be equipped with sensors which detect the contact to an object. For this purpose force/torque sensors are commonly used and by exploring the object via force/torque sensing, the shape and pose of an object can be determined. Recent improvements on tactile sensors leading to more reliability regarding sensor readings and deployability on robots [15] led to an increase of interest in its use to explore environments and objects, respectively. Figure 1.2 shows several sensors to gather information about the environment. The BioTac tactile sensor modeled to mimic the physical and sensory capabilities of the human fingertip, the ATi six-axis force/torque sensor nano17e and the ASUS Xtion PRO depth camera are examples of sensors used to enhance the abilities of robotic systems. Let aside the first approach based on known environments; the visual and tactile approaches come with advantages and disadvantages. To constitute the tactile approach as reasonable and useful as well as complementary to the visual approach, the following argumentation describes the advantages and disadvantages of the visual approach and the benefit of researching effective ways of tactile approaches to gather information about the environment to be able to act in it autonomously. The shape of environments is generated more efficiently with visual approaches. More information can be gathered in less time than with tactile approaches due to the ability to not having to be in contact with objects in the environment but to sense from a distance. Additionally, to being faster in gaining information about the shape of the environment and not having to be in direct contact with objects as is the case with tactile approaches,



(a) (b) (c)

Figure 1.2: Sensors and sensor systems used at the TAMS research group of the University of Hamburg to sense a robotics environment. (a) BioTac tactile sensor [17], (b) ATI force/torque sensor nano17e [18], (c) ASUS Xtion PRO depth camera[19].

the likelihood for unwanted collisions is diminished. With visual approaches, a robot can react to possible collisions before the contact to the environment occurs and therefore constitutes a viable approach in Human-robot Interaction (HRI). Harm to robot and environment can be prevented. In contrast to tactile approaches gathering information from a distance leads to sparse or no information about the texture, inertia, and weight of the objects in the environment. In poor visibility conditions, vision approaches turn out to be inapplicable. Poor visibility conditions occur under various conditions and render sensor readings useless. Visual approaches use different techniques, e.g., stereo cameras or laser range finder. Hence, conditions leading to poor visibility are dependent on the used technique. Conditions for poor visibility can be poor illumination, direct sunlight, glare on objects caused by light sources, sensing under water, dust in smoky and foggy disaster environments and occlusion. Occlusion often occurs in manipulation tasks, where the end effector of a robot occludes the object to be manipulated.
In general, it can be stated that the advantages and disadvantages of visual approaches emerge mostly due to sensing from a distance and therefore not being in direct contact with the environment. Tactile sensing and perception are well-known in robotics [20, 21] and supplements or temporarily replaces vision approaches. In contrast to passive vision sensing, the level of control of active tactile sensing is typically more difficult as it changes the environment by manipulation such as moving an object while exploring it via tactile sensors. Additionally, in the case of gathering information about surface and position of an object, tactile sensors need to be in contact with it and probe or contour it following sophisticated strategies. It is an independent approach, complementary to vision and offers additional sensing capabilities as can be the normal direction of an object surface at the contact position. In contrast to vision, tactile approaches are less efficient in gathering data and gather far fewer data. However, Lederman and Klatzky and their colleagues [22, 23, 24] have shown that humans can recover shape and other object attributes reliably using touch alone. Therefore, sparse data might suffice to reconstruct objects.
To achieve robust reasoning in autonomous robots multi-modality as well as redundancy in sen-

sors considering their purpose is desirable. Sensor readings turn erroneous or useless in environmental conditions they are impractical for. Stereo cameras used to detect objects in sight, for example, produce useless sensor readings in darkness. Laser range finder or tactile sensors still produce reliable sensor readings in darkness to gather information about the environment concerning position of objects in relation to the robot, although occlusion prevents reliable laser range readings.
It can be stated that tactile approaches gathering information about the environment can temporarily replace visual approaches and supplement them. Multi-modal object representations allow more robust behavior for autonomous robots in manipulation tasks due to less liability on single sensor type readings. Approaches to exploring the surfaces of objects in the environment via tactile sensors are beneficial considering failure of vision approaches and more robustness through multi-modality.

1.3 Goal of the Thesis

The goal of the thesis is the development of a blind surface exploration approach based on force/-torque sensor readings to explore and reconstruct objects in an exploration area. Additionally, a hardware setup of robot arm manipulator, force/torque sensor, and end effector to apply the exploration and meet the requirements for the exploration approach is to be worked out.
Blind exploration tasks are defined as being able to extract specific object characteristics using tactile input and exploration strategies without vision sensors. Shape and location of objects describe such characteristics. As mentioned in section 1.2, manipulation is crucial for interaction between robot and environment. Before action can take place, the environment has to be modeled. The result of the approaches of the master thesis targets the modeling of the environment via reconstructing the shape of objects in three dimensions. The object of the thesis is to provide a supplemental and temporary replacing approach to model the environment of a robot to visual approaches.
The surface exploration contains the detection and exploration of objects to a certain degree to be able to reconstruct the object based on the gathered information during the exploration process.
The exploration approach requires real-time ability in motion planning and control as well as in collision detection.

1.4 Related Work

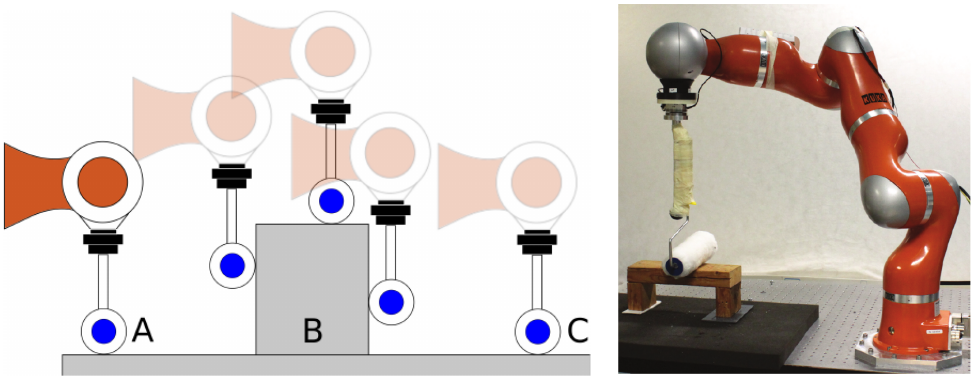
This section on related work summarizes selected recent work. Many research papers have been published in the last decades on the topic of interacting with or handle forces acting on a manipulator. The so-called force control problem [25] tackles the motion control problem where force/torque sensor readings are provided to manage an interaction of a robot manipulator with the environment as is the case in surface exploration. The history of the state of the art of the research findings towards the force control problem is found in [26] for the 1980s and later research findings in [27] and [28].

Hussein et al. [2] investigate the modeling of 3D models of objects through tactile exploration. A 6-DOF manipulator is equipped with a force/torque sensor at the wrist to measure forces acting upon a spherical end effector. The contact location and force normal at the contact point are used to determine the motion of the end effector along the object to be modeled. The paper shows a practical demonstration of object surface contouring, where a bottle is approached by the end effector of the manipulator and upon contact contours the bottle in circles (cf. figure 1.3). The orientation of the end effector thereby is static.

Figure 1.3: The experimental setup of Hussein et al to blindly perceive the shape of a bottle with force/torque sensor and a 6-DOF manipulator.

The work of Jamisola et al. [3] aim on building an information map of an unknown object or environment with discontinuities through fully autonomous haptic exploration. Discontinuities are defined as sharp turns and abrupt dips as can emerge at wall corners or cliffs. Compliant motion control enabled by force information gained with a force/torque sensor and its limitation is investigated. Two solutions are presented to overcome the difficulties in contouring discontinuities and simultaneously apply a predefined force normal to the object. The end effector constitutes a roller tool mounted on a KUKA LWR IV (cf. 1.4) and again the orientation of the end effector remains static while exploring the object.

Winkler and Suchỳ investigate surface contouring of unknown objects in [4, 29, 30] with force control approaches. Furthermore, they elaborate on tool orientation control. They state that it is necessary for some kinds of contour following tasks like polishing, deburring or grinding to keep the end effector in a constant orientation to the contact environment regarding the inclination angle to it. Although changes in orientations are considered, their approaches do not consider different objects or test beds to verify a general applicability of the tool orientation control. Figure 1.5 depicts the setup of Winkler and Suchỳ. Back et al. [5] developed a new end effector in shape of a finger to cope with the challenges in exploring surfaces. Similar to the approach in this thesis, an ATI nano17 has been installed as force/torque sensor. Different



(a) (b)

Figure 1.4: The figure on the left(a) depicts the motion outline of the end effector from A to C encountering the object at B. The figure shown in (b) pictures the setup of KUKA LWR IV and the roller tool in between the force/torque sensor and the object [3].

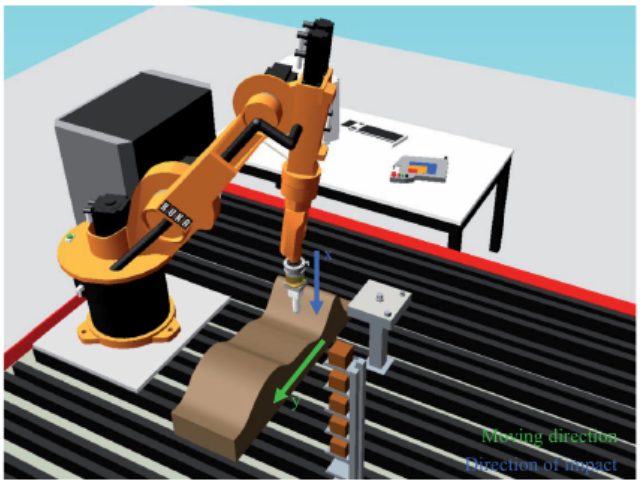


Figure 1.5: The setup of Winkler and Suchỳ consists of an industrial 6-DOF manipulator, a force/torque sensor and a probe contouring an Object in two dimensions [4].

shapes and materials contribute to the validity of the sensing finger. Additionally, to exploring surfaces through sliding along them, the pose of a known object is estimated based on their blind surface exploration approach. The range of motion, while not mounted on 6 or more DOF manipulators, remains low. The contact sensing finger is shown in 1.6. Using a dexterous hand, Bierbaum et al. [31] present a tactile exploration strategy for unknown objects. A five-fingered anthropomorphic hand is guided along a surface and builds a 3D object representation based on point clouds acquired by tactile sensor information. The dynamic potential field approach

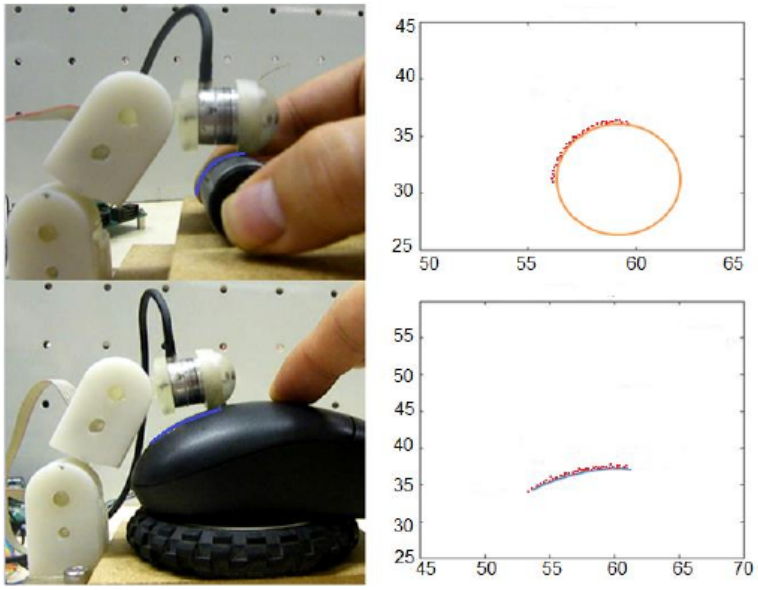


Figure 1.6: The contact sensing finger following object surfaces is shown in (a). The dots plotted in (b) represent the experienced contact points and trajectory of the finger [5].

Figure 1.7: The iCub performing haptic surface exploration on a bottle via sliding along a bottle is shown. The bottle is held by one hand while the fingers on the other hand slide along the object recording tactile data streams. [6]

known from the domain of mobile robot navigation is used in the proposed strategy for motion guidance of the fingers. Several objects are explored using this strategy.
In 2014 Sommer et al. [6] presented a haptic surface exploration approach to identify objects to gain information to grasp them. In contrary to the earlier introduced approaches, the object is held in one hand of an iCub humanoid robot, while the finger of the other hand slide over the object. Tekscan tactile sensors record the contact positions sensed for several objects. The contact positions are transferred into a point cloud which is then compared to point clouds of the known objects. Although this approach belongs to blind surface exploration, the object to be explored has to be in the hand of the iCub to start the exploration strategy. Figure 1.7 shows the iCub holding a bottle and exploring the surface of it via the fingers on the other hand.

The related work discussed in this section in comparison to the approach in this master thesis does not connect surface exploration respectively surface contouring and surface reconstruction. The emphasize lies either in contouring a surface or exploring a surface haptically in 2D or in exploring an object haptically and map to known objects. The approach of the master thesis is to explore an object, gain its surface information via force sensing and generate a 3D mesh polygon from the resulting contact points between end effector of the robot and the explored object. The mesh polygon could then be used to calculate grasping positions or generally as an object to be considered while acting in the environment.

1.5 Thesis Outline

The outline of the thesis is described as follows:

- Chapter 2 - Fundamentals:
  The fundamentals on kinematics, motion planning, interaction control and surface reconstruction (cf. sections 2.1, 2.3, 2.4, 2.6) among other topics necessary to plan and implement the goals fo the thesis are introduced.

- Chapter 3 - Experimental Setup:
  The hardware setup and used software libraries and frameworks are described.

- Chapter 4 - Concept:
  The elaborated concept on how to reach the set goals to pave the way for the chapter on Implementation is discussed.

- Chapter 5 - Implementation:
  Here, the implementation of the concepts is described for the object surface exploration (cf. section 5.3) and surface reconstruction (cf. section 5.4).

- Chapter 6 - Evaluation:
  The results of the conducted object surface explorations (cf. section 6.2.2) and surface reconstructions (cf. section 6.2.4) are shown and investigated here.

- Chapter 7 - Conclusion:
  The thesis is concluded (cf. section 7.1) and prospect for further work (cf. section 7.2) is given.

2 Fundamentals

The upcoming sections describe the fundamentals of surface exploration and surface reconstruction using a robotic arm. The fundamentals of surface exploration elaborate on kinematic topics such as orientation and position in space and joint kinematics. Additionally, the workspace of robotic arms is described as well as motion planning and interaction control. After illustrating the functional principle of force sensing regarding strain gauges and six-axis force/torque sensors, fundamentals on surface reconstruction from point clouds including surface reconstruction methods are given.

2.1 Kinematics

Robot kinematics is used to describe the characteristics of robotic systems concerning velocity, acceleration, and position of the links and joints of robots. The interaction of the elements of the kinematic chain of robots is elaborated by robot kinematics which constitutes the foundation for motion planning and controlling. The fundamental concepts are introduced in the following sections including the derivation of the pose of the last element in a kinematic chain using the concept of forward kinematics (cf. section 2.1.5).

2.1.1 Joint Space and Cartesian Space

The position and orientation of a manipulator most commonly are described in the joint or Cartesian space. The joint space is specified by the positions of the robot's joints. The set of n joint variables can be denoted as $n \times 1$ joint vector. The Cartesian space is derived from the joint space and measures position and orientation in Cartesian coordinate frames along orthogonal axes and according to the representations of subsection orientation and position.

2.1.2 Orientation and Position

The position and orientation of an object in an environment is a requirement of fundamental importance to robotics. Meaningful and rational motion planning is only possible when the position and orientation of the links and joints are known. The composition of position and orientation is called pose. A pose can be assigned to any point in space and is relative to a coordinate system. The position of a point can be denoted with a 3x1 position vector. A point P in a three-dimensional coordinate system A can be written as:

$$^{A}P = [p_x p_y p_z] \tag{2.1}$$

where x,y and z denote the three axes of frame A. To describe the orientation of an object, different representations exist.

Rotation Matrix To denote the orientation of an object, one can attach a frame i to it and describe it relative to the reference frame j. The orientation of i can be described by expressing the mutually orthogonal basis vectors $(x_i\ y_i\ z_i)$ in terms of the basis vectors $(x_j\ y_j\ z_j)$. The corresponding vector $({}^{j}x_i\ {}^{j}y_i\ {}^{j}z_i)$ can be written as 3x3 rotation matrix:

$$ {}^{j}R_i = \begin{pmatrix} x_i \cdot x_j & y_i \cdot x_j & z_i \cdot x_j \\ x_i \cdot y_j & y_i \cdot y_j & z_i \cdot y_j \\ x_i \cdot z_j & y_i \cdot z_j & z_i \cdot z_j \end{pmatrix} \tag{2.2} $$

The rotation of frame i about axis x_j through an angle θ is

$$ R_X(\theta) = \begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{2.3} $$

the same rotation about the y_j axis is

$$ R_Y(\theta) = \begin{pmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{pmatrix} \tag{2.4} $$

and the same rotation about the z_j axis is

$$ R_Z(\theta) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{pmatrix} \tag{2.5} $$

The combination of rotation matrices through matrix multiplication allow to express the orientation of a frame i relative to a frame k as

$$ {}^{k}R_i = {}^{k}R_j\ {}^{j}R_i \tag{2.6} $$

Euler Angles In contrast to the nine values used by the rotation matrix to denote the orientation, Euler angles only require three. The vector $(\alpha, \beta, \gamma)^T$ represents the orientation of a coordinate frame relative to another coordinate frame. Each angle denotes a rotation about an axis of a moving coordinate frame. As a result, the location of the axes of the successive rotations depends on the preceding rotations. 12 different possible orders of rotation about the axes exist, but all exhibit a singularity in the case of the first and last rotation axis being the same. Mathematical problems arise in calculating e.g. the angular velocity vectors due to the singularity [32]. This limits the usefulness of Euler angles to robotics.

Fixed Angles Euler angles denote the rotations along moving coordinate frames while fixed angles denote the orientation relative to fixed frames. The vector here is denoted as following $(\psi, \theta, \phi)^T$. 12 orders of rotation are possible and again the same singularities in comparison to Euler angles exist.

Angle-Axis The orientation of a coordinate frame i to another frame j can also be represented by a unit vector $\tilde{\omega}$ together with an angle θ. i is thereby rotated through the angle ϕ about an axis defined by $\tilde{\omega} = (\omega_x, \omega_y, \omega_z)^T$ relative to j. Typically the angle-axis representation is written as $\theta\tilde{\omega}$ or $(\theta\omega_x, \theta\omega_y, \theta\omega_z)^T$. Unfortunately, the rotation with -$\tilde{\omega}$ and -θ is equivalent to $\tilde{\omega}$ and θ.

Quaternion Similar to angle-axis the representation of quaternions is based on four parameters. Quaternions combine the advantages of rotation matrices and Euler angles. They are suited for computations and are short in notation.
The representation using Quaternions is popular in robotics. Quaternions do not suffer from singularities ([33] p. 51) as other representations like Euler angles and fixed angles do.

The quaternion can be written as the combination of a scalar and a vector

$$\dot{q} = s + v = s + v_1 i + v_2 j + v_3 k \tag{2.7}$$

where $s \in \mathbb{R}$ and $v \in \mathbb{R}^3$ are scalars and the orthogonal complex numbers i, j, k are defined such that

$$i^2 = j^2 = k^2 = ijk = -1 \tag{2.8}$$

holds. Quaternions of unit magnitude, also known as unit-quaternions, are used to represent rotations. For unit-quaternions holds

$$|\mathring{q}| = 1 \tag{2.9}$$

$$s^2 + v_1^2 + v_2^2 + v_3^2 = 1 \tag{2.10}$$

The unit-quaternion can be considered as rotation of θ about the unit vector $\tilde{n}$ which is related to the quaternion components by

$$s = \cos\theta/2, v = (\sin\theta/2)\tilde{n} \tag{2.11}$$

Savings in computation costs are achieved by compounding two quaternions instead of two orthonormal rotation matrices. The quaternion form demands 16 multiplications and 12 additions compared with 27 multiplications and 18 additions ([34] p. 36).

2.1.3 Joint Kinematics

The joint kinematics is determined by the contact surfaces between links and describes the freedom of motion between links. A single pair of connecting links constitutes a simple kinematic joint which can be classified to lower pair joints or higher pair joints. Contact on lower pair joints occurs on a surface while contact on higher pair joints occurs on points or along lines. Six forms for lower pair joints exist: revolute, prismatic, helical, cylindrical, spherical and planar joints. Revolute joints are common in modern robotics due to the possibility to easily actuate them via rotating motors. A typical use case for prismatic joints are 3 degrees of freedom applications, where coordinates in a 3D space have to be reached without changes in orientation of the end effector or the end of the robotic mechanism. Prismatic and revolute joints are shown in 2.1. Revolute joints (a) provide single-axis rotation function. Only one link of the kinematic joint is permitted to rotate and revolute joints have one degree of freedom. Prismatic joints (b) provide single-axis sliding of one link in the kinematic joint and have one degree of freedom.

2.1.4 Geometric Representation

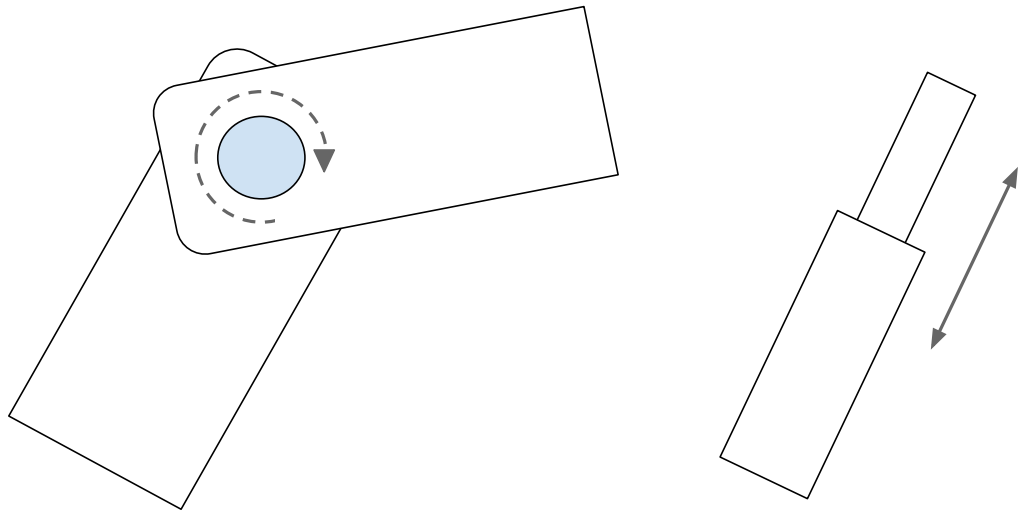
For consistency and computational efficiency, the Denavit-Hartenberg notation has been defined to attach frames to links to model the geometry of robotic mechanisms. The notation introduced by Denavit and Hartenberg in [35] has later been adapted in numeral ways for more intuitive presentation. Instead of using six parameters to locate a frame to another, the Denavit-Hartenberg notation only requires four. The convention is deployable on robot mechanisms with prismatic and revolute joints. Does a robot mechanism have other joint specifications, they have to be broken down into prismatic and revolute joints. The four parameters are composed of two link parameters namely a_i the link length, α_i the link twist and two joint parameters namely d_i the joint offset and θ_i the joint angle. The parameters corresponding to the KUKA LWR IV can be found at table 3.3. The notation with four parameters is achieved by placing the frame origins and axes in a manner that the x-axis of one frame is perpendicular to the z-axis of the following frame as well as intersects it. The procedure on how to achieve the right placement of frames in intermediate links in the chain is illustrated in figure 2.2 and has been pointed out in [7]:

1. The z-axis of frame i, Z_i, is coincident with the joint axis i.

2. The origin of frame i is located where the a_i perpendicular intersects the joint i axis.

3. X_i points along a_i in the direction from joint i to joint $i+1$. In the case of $a_i = 0$, X_i is normal to the plane of Z_i and Z_{i+1}.

4. Y_i is formed by the right-hand rule to complete the ith frame.



(a) Revolute joint (b) Prismatic joint

Figure 2.1: The figure represented on the left illustrates a revolute joint (a) and the figure shown in (b) represents a prismatic joint. The revolute joint rotates around an axis pointing towards the reader situated in the center of the blue cylinder connecting both links. The prismatic joint translates along the indicated arrows.

The Denavit-Hartenberg notation enables to locate frame i relative to frame i-1 by executing two rotations and two translations by multiplying transformation and rotation matrices as follows:

$$^{i-1}T_i = \mathbf{Rot}(x_{i-1},\alpha_i)\mathbf{Trans}(x_{i-1},a_1)\mathbf{Rot}(z_i,\theta_i)\mathbf{Trans}(z_i,d_i) \tag{2.12}$$

expanded to the equivalent and compact homogeneous transformation notation:

$$^{i-1}T_i = \begin{pmatrix} \cos(\theta_i) & \sin(\theta_i) & 0 & -a_i \\ -\sin(\theta_i)\cos(\alpha_i) & \cos(\theta_i)\cos(\alpha_i) & \sin(\alpha_i) & -d_i\sin(\alpha_i) \\ \sin(\alpha_i)\sin(\theta_i) & -\cos(\theta_i)\sin(\alpha_i) & \cos(\alpha_i) & -d_i\cos(\alpha_i) \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{2.13}$$

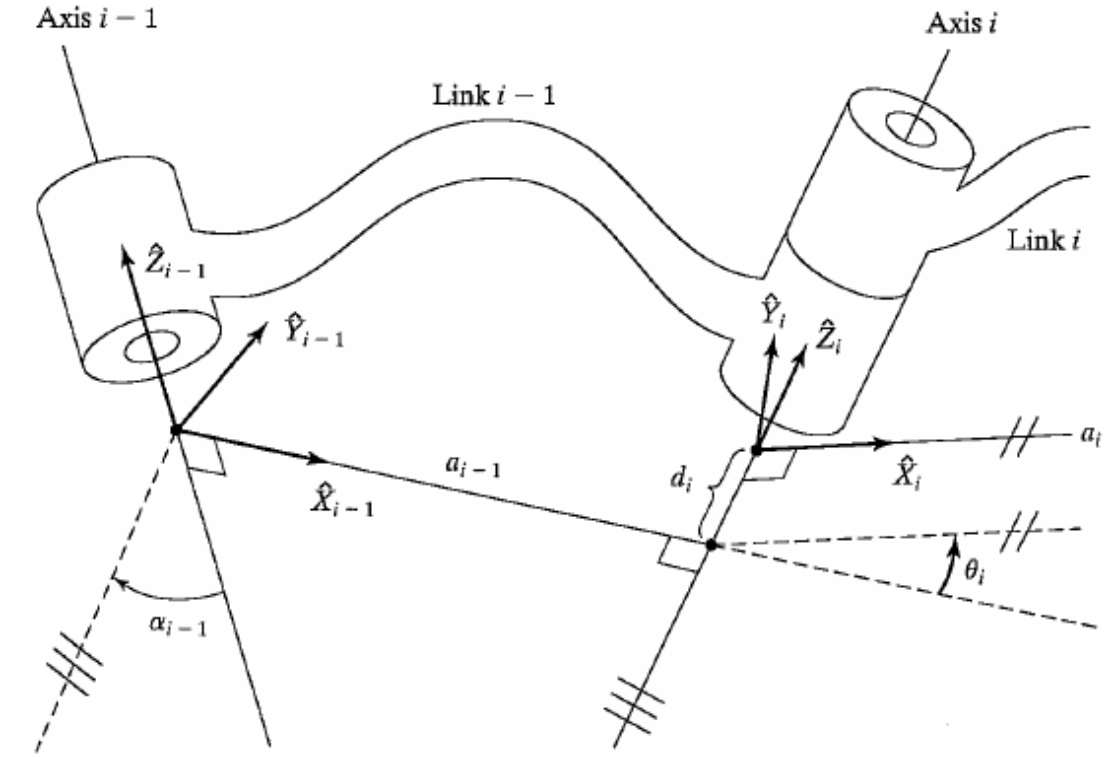


Figure 2.2: Illustration of link frame attachment according to the Denavit-Hartenberg (DH) notation ([7], p.68.) The placing of frame orientation and where to extract the DH-parameters is shown.

2.1.5 Forward Kinematics

The forward kinematics problem describes the need to calculate position and orientation of an end-effector of a serial-chain manipulator to its base. The geometric link parameters and joint values of the manipulator are thereby given. The forward kinematics problem can be solved by calculating the transformation from the base coordinate system to the end-effector coordinate system. For a robotic arm with serial chain from base to end-effector, the transformations from frame to adjacent frame can be concatenated. The transformation from base to end-effector frame can be denoted as

$$ {}^{0}T_6 = {}^{0}T_1 \, {}^{1}T_2 \, {}^{2}T_3 \, {}^{3}T_4 \, {}^{4}T_5 \, {}^{5}T_6 \tag{2.14} $$

where the superscript denotes the coordinate frame of the pose to be transformed and the subscript denotes the coordinate frame to be transformed to.

2.1.6 Inverse Kinematics

The inverse kinematics problem describes the need to calculate the required joint coordinates given a desired pose for the end-effector of a serial-chain manipulator. The equations to solve this problem are nonlinear and there might exist no solution or even multiple solutions [36]. Of course, the wanted end-effector pose has to lie in the manipulator's workspace. Solution strategies can be split into closed-form and numerical solutions. Closed-form solutions are in general faster than numerical solutions and readily identify all possible solutions ([32], p. 29). However, closed-form solutions are robot dependent and take advantage of geometric features of mechanisms in the robot. Closed-form solutions can be divided into algebraic and geometric methods where the algebraic method involve the identification of significant equations to bring the joint variables in a soluble form, whereas geometric methods involve the decomposition of the spatial geometry of the robotic arm into plane-geometry problems. The plane-geometry problems can be solved by using plane geometry. Does a closed-form solution exist, the problem of inverse kinematics can be solved by decomposition concerning inverse position kinematics and inverse orientation kinematics which enables us to specify a kinematics equation as such

$$ {}^{0}T_{6}\,{}^{6}T_{5}\,{}^{5}T_{4} \;=\; {}^{0}T_{1}\,{}^{1}T_{2}\,{}^{2}T_{3}\,{}^{3}T_{4}. \tag{2.15} $$

The robot independent numerical methods do not always allow to calculate all possible solutions. Stated in [32], p. 30., the categories symbolic elimination methods, continuation methods and interactive methods are the common numerical methods.

2.2 Workspace

The workspace of a robotic arm is the volume of all end-effector positions it can reach. Therefore the workspace is determined by the geometry of the robotic arm and the limits of the joints. The workspace can be divided into the reachable workspace and dexterous workspace [37]. The reachable workspace denotes the space the end-effector can reach. Within the dexterous workspace, the end-effector of the robotic arm can reach the wanted position with all orientations. Therefore the dexterous workspace is a subset of the reachable workspace. It has to be stated that a robotic arm with less than six degrees of freedom cannot span a dexterous workspace due to the inability of reaching a position with all orientations in 3D space.

2.3 Motion Planning

Is the configuration of a manipulator the specification of its joint values, motion planning can be defined as the determination of the configurations between a start- and a goal-configuration. All possible configurations of the manipulator are called the configuration space (C_{space}). The C_{space} is further divided into C_{free} and C_{obj}. C_{obj} denotes the subspace of configurations where collisions arise between the manipulator and the environment or with itself. C_{free}, on the other hand, denotes the configurations without collisions. A single configuration can be represented as a point in the C_{space} and therefore the plan of motion is determined by points in C_{free} representing

a path between start- and goal-configuration. To connect the points, a trajectory is determined by a motion planner specifying a time sequence of position, velocity, and acceleration of the joints based on path and manipulator constraints about dynamics. Point-to-point motion is conducted, when the initial and final point on a path is assigned while path motion is conducted when multiple points are assigned on the path.
Reference points of the trajectory are used as inputs for motion control systems determining the forces or torques to be conducted by the joint actuators.

2.4 Interaction Control

The execution of tasks often requires interaction control to handle manipulation tasks including contact between robot and objects. Simple motion control is failure-prone in manipulation tasks due to its dependency on sufficient certainty in knowledge about the robot's kinematic and dynamic as well as accuracy in motion execution. Furthermore the C_{space} has to be known permanently to be able to react to changes invalidating trajectories. Does the C_{space} unknowingly change resulting in invalid trajectories or inaccuracy in trajectory-following motion takes place, the risk of collision arises. The resulting deviation from the trajectory results in reaction of the motion control to reduce the deviation by adapting the forces or torques conducted by the joint actuators only leading to higher contact forces at the collision point.
Interaction control handles manipulation tasks through compliant behavior during the interaction. The Interaction control can be subdivided into passive and active interaction control.

2.4.1 Passive Interaction Control

In passive interaction control, the inherent compliance of a robot influences its end-effector trajectory reacting to forces acting upon it. The compliance can be achieved through the different parts of a robot such as the links, joints, the end-effector and position servo. Furthermore, robot arms with elastic joints and links are designed for interaction with humans to ensure safe interaction. A system with passive interaction control does not require force/torque sensors as it does not need to react to forces in changing the preplanned trajectory. Although passive interaction handles forces to a certain point, one cannot guarantee a safe and low force interaction due to the nonexistence of monitored force/torque values. A disadvantage of using passive interaction control lies in its lack of flexibility. Is an end-effector designed for a compliance task with a specific robot, another end-effector might have to be designed for another robot or another compliance task.

2.4.2 Active Interaction Control

In active interaction control, a control system designed for a specific purpose ensures the compliance. The measurement of contact forces and moments can be used to design those control systems by feeding the sensor values to the controller which then adapts the planned trajectory of the robot. In contrast to passive interaction control, it is more sophisticated. The sensor values have to be interpreted with respect to the current task or trajectory of the robot and lead to

adjusted motion. Naturally, it is more expensive due to the integration of more sensors in the robotic system. Often active interaction control uses passive compliance to a certain degree to ensure reasonable task execution speed and disturbance rejection capability [38]. This is because passive compliance does not need to interpret the force and moment values or calculate interaction motion and feedback to the active interaction control system always comes later than the motion inducing changes in the sensor values. Hence, for contact forces and moments, the active interaction control system without passive compliance cannot assure certain force and moment thresholds. This is especially the case when contacts arise between rigid bodies.

Two strategies for active interaction control can be distinguished. Indirect force control achieves force control through motion control while direct force control through the enclosure of a force feedback loop enabling the controlling of desired force and moment during motion. Indirect force control corresponds to either impedance or admittance control [39, 40]. The indirect force control is an impedance control if it reacts to motion deviation caused by interaction with the environment by generating forces while it is an admittance control if it reacts by deviating from the desired motion. Further, special cases of impedance and admittance control are stiffness control and compliance control [41]. Here the static relationship between end-effector pose deviation from the desired motion and contact force and moment is considered.

2.4.3 Interaction Tasks

In general, one can say that there is no need for explicit knowledge of the environment for indirect force control. Nevertheless, assumptions about the environment can help achieve satisfactory behavior in the interaction. Interaction tasks evoke complex contact situations between the environment and the contact point on the end-effector. To cope with contact situations in force control, tasks have to be specified. An intuitive way of describing a task can be done using task frames. Task frames are orthogonal reference frames where forces/torques and linear/angular velocities can be assigned to each axis. A task fame can thereby be specified at any origin. The so applied forces/torques or linear/angular velocities are called artificial constraints whereas natural constraints are imposed by the environment to the end-effector. In the case of rigid bodies, the natural constraints act complementary to the artificial constraints. In the case of blind surface contouring without desired application of forces or torques along the axes of the task frame, there still exist natural constraints. In the presence of a contact situation, forces and torques act upon the end-effector. The control strategy might use this information to change its preplanned trajectory with respect to the task frame to not make the contact forces rise too high but still move along the surface of the object.

2.5 Force Sensing

Force sensors can be divided into quantitative and qualitative sensors. While quantitative sensors measure the force and represent its value, qualitative sensors do not consistently represent force values but indicate a certain event, e.g., when a force threshold is reached. Strain gauges and load cells are examples of quantitative sensors, and a light switch is an example of a qualitative sensor as it reacts to a certain force by switching the switch. As stated in [42] different methods

Table 2.1: Characteristics of strain gauges with regard to used material, gauge factor, resistivity and thermal coefficient of resistance taken from Fraden and Suhir [43, 44]

Material	Gauge Factor, GF	Resistivity at $20°C$, $\mu\Omega \cdot$ cm	Thermal Coefficient of Resistance ($°C^{-1}$ x 10^6)	Remarks
Constantan (55% Cu, 45% Ni)	2.0	49	11	GF constant over wide range of strain; use below $360°C$
Platinum alloy (95% Pt, 5% Ir)	5	24	1250	Useful to $1000°C$
Silicon semiconductor	-100 to +150	10^9	90.000	High sensitivity

of sensing forces can be categorized. The occasionally used methods are 1 and 2 whereas method 3 is most commonly used:

1. The force is balanced against an electromagnetically developed force
2. The force is converted to a fluid pressure which is then measured
3. The strain produced by an unknown force in an elastic member is measured

The deformation of a physical body under force application (strain) can be used to measure the applied force by exploiting the piezoresistive effect. Resistive elastic sensors relate to strain by a function determining its electrical resistance. The ratio of change in resistance to strain, also known as gauge factor, can be denoted as:

$$GF = \frac{\Delta R/R_G}{\varepsilon}. \tag{2.16}$$

Where GF is the gauge factor, R_G the resistance of the undeformed gauge, ΔR the change in resistance caused by strain and ε the strain. The strain can further be defined as $\varepsilon = \Delta L/L_0$, L_0 being the original length of the strain gauge and the the absolute change in length ΔL. The gauge factor differs with respect to the material of the resistive elastic sensor. After Fraden and Suhir [43, 44], characteristics of some strain gauges are shown in table 2.1.

The effect of temperature on the resistance is not considered in the gauge factor formula despite semiconductor strain gauges being sensitive to changes of temperature. In practice changes in temperature exist and a simplified equation to calculate the resistance can be denoted as follows:

$$\frac{\Delta R}{R} = GF_\varepsilon + \alpha\theta. \tag{2.17}$$

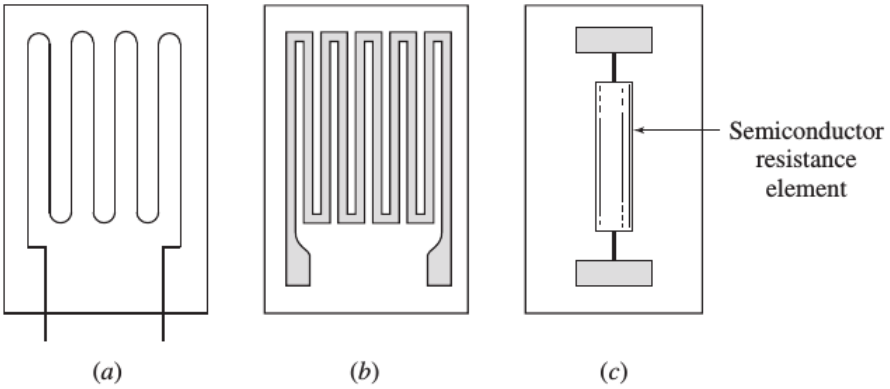


Figure 2.3: Common types of resistance strain gauges ([43], p. 507) from left to right: (a) bonded-wire gage, (b) foil gage and (c) the semiconductor gage

The temperature coefficient is denoted by α and θ denotes the change in temperature.

Figure 2.3 depicts the three common types of resistance strain gauges.

The combination of measuring elements as strain gauges enables to measure forces or torques along multiple axes. Robotic applications often require feedback of tools controlled in six axes. Thereby the six axes are divided into three translational and three rotational axes (cf. 2.4). A six-axis force/torque sensor (cf. 3.6) is able to measure the magnitude and direction of forces and torques in three dimensions. The information on force and torques acting on the tools are usually used to provide feedback during the execution of interaction tasks.

2.6 Surface Reconstruction from Point Clouds

The goal of surface reconstruction is to find an accurate continuous description of an object surface based on the available information gathered previously. In other words, the goal is to find a computer model best fitting the real object. A priori knowledge about the object and the quality of information gathered influence the outcome of the reconstruction and the applicability of reconstruction methods. In the case of blind surface exploration, a priori knowledge about the object is not available. The quality of information has different aspects. Concerning accuracy in spatial position of surface points tactile exploration might be precise although density in surface points is usually sparse.
In general, two problems arise in surface reconstruction from point clouds [45]:

- A surface is described continuously, and points of a point cloud are a discretization of the surface. Therefore the surface between the points has to be estimated.
- The constructed point cloud is affected by measurement errors.

In general, the transformation of point clouds to polygonal surfaces or polygonal meshes (see below) is based on four steps according to [46]:

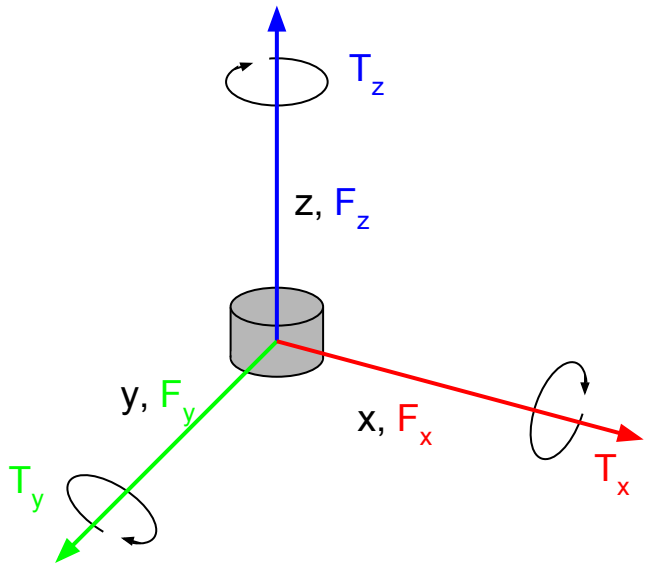


Figure 2.4: The force and torque directions are assigned to the axes of a 3D coordinate system. The forces are represented in the directions of the axes and the torques are represented as rotations on the axes. This configuration allows the modeling of the forces and torques acting in six axes on the gray cylinder.

- pre-processing: in this phase, erroneous data are eliminated alternatively, points are sampled to reduce the computation time;

- determination of the global topology of the object's surface: the neighborhood relations between adjacent parts of the surface have to be derived. This operation typically needs some global sorting step and the consideration of possible 'constraints' (e.g. break lines), mainly to preserve special features (like edges);

- generation of the polygonal surface: triangular (or tetrahedral) meshes are created satisfying certain quality requirements, e.g. limit on the meshes element size, no intersection of break lines, etc.;

- post-processing: when the model is created, editing operations are commonly applied to refine and perfect the polygonal surface

2.6.1 Polygon Meshes

Modeling surfaces can be done with polygonal modeling, a common approach in computer graphics respectively geometric modeling. Polygonal models are assembled by polygon meshes built up by vertices, edges and faces. A vertex is a point in space, an edge is the connection between two vertices and a face is a closed set of edges. In the case of the most common face in polygon meshes, the triangle, a face connects three vertices. Vertices are shared by faces so that contiguous faces build up a polygon. The connection between polygons with shared vertices is then called polygon meshes and define the shape of 3D models. The elements of a polygon mesh are shown in figure 2.5. Polygons can be classified based on their convexity or concavity.

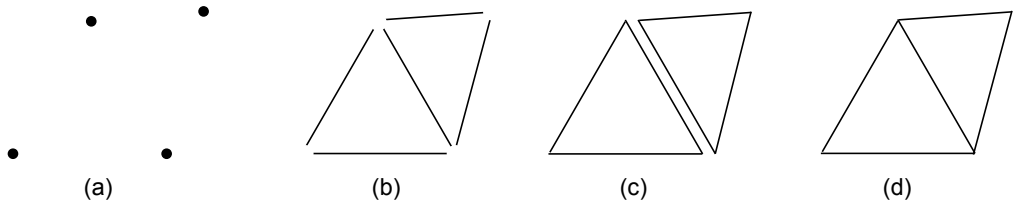


Figure 2.5: A simple polygon and its elements are shown. (b) illustrates the edges between the in (a) shown vertices. The faces built up by the edges are shown in (c). The connection between the faces sharing an edge results in a polygon model in (d).

2.6.2 Convex and Concave Polygons

A convex hull of a set X of points is the smallest convex set that contains X (illustrated in 2.6). A convex set is a region or space in within every pair of points can be connected by a straight line which is situated entirely in the region or space. Figure 2.7 pictures a convex (a) as well as a non-convex set (b).
Hence a convex polygon is defined as a polygon where all interior angles are less than or equal to 180° which is equivalent to the property of every point on line segments between two points of a polygon being inside or on the boundary of the polygon.
Concave polygons are non-convex polygons and therefore has at least one interior angle greater than 180°. A polygon with more than three sides can be concave (cf. 2.8) in contrary to a polygon with three sides. Naturally, the convex hull of a concave polygon contains points on the point connecting straight lines outside of the polygon. This characteristic renders the determination of the shape of polygons through the convex hull problematic by disregarding shape features of the polygon. The area occupied by a concave polygon can be determined by a concave hull, a shape minimizing the area of the polygon to a certain degree allowing interior angles of more than 180°.

2.6.3 Poisson Surface Reconstruction

Surface reconstruction can be done by utilizing Poisson's equation for the so-called Poisson Surface Reconstruction [47]. Using Kazhdan et al. paper on Poisson Surface Reconstruction

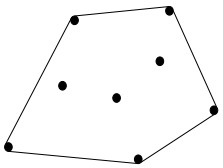


Figure 2.6: The smallest convex set of points shown here describes the convex hull of the set of points.

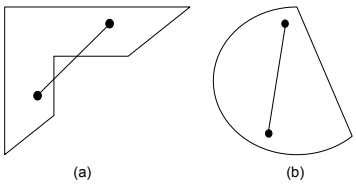


Figure 2.7: The plane illustrated in (a) depicts an area built by a non-convex (concave) set. The straight line connecting the points is partly outside the area. The plane illustrated in (b) shows an area built by a convex set showing a plane in which all potential sets of two points are connected by a straight line within the plane area.

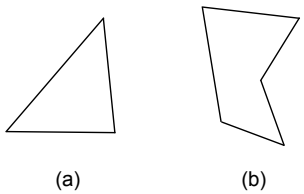


Figure 2.8: The polygon in (a) depicts a convex polygon with three sides. All polygons with three sides are convex. The polygon in (b) depicts a concave polygon (or non-convex polygon) with five sides.

and its illustration in 2.9, the description of the general idea of the approach follows. The vector field $\vec{V}$ is defined by oriented points sampled from a surface describing the normal to the surface pointing outwards. Based on the vector field, a 3D indicator function X is computed representing points inside the model with 1 and 0 outside of the model. The next and final step is the extraction of an appropriate isosurface to reconstruct the surface. The key to the indicator function is the characteristic that the gradient of the indicator function is a vector field that is zero almost everywhere but at points near the surface. At the surface, the values of the gradient's vector field are equal to the inward surface normal. Consequently, the oriented point samples can be interpreted as samples of the gradient of the model's indicator function. Determining the indicator function, therefore, is reduced to invert the gradient operator. Hence, find the gradient best approximating the vector field $\vec{V}$ of the scalar function X. This problem is a standard Poisson problem applying the divergence operator. Hence the computation of X with Laplacian's equation of the divergence of gradients can be denoted as follows:

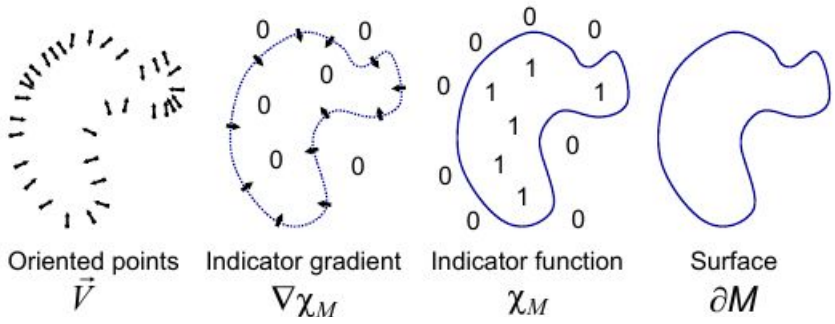


Figure 2.9: The illustration of Poisson reconstruction in 2D is shown. The vector field gained by oriented points sampled from a surface is described with $\vec{V}$. Based in this vector field, an indicator function X_M is determined representing points inside the model with 1 and outside with 0. At the surface, the gradient's vector field is equal to the inward surface normal. The determination of the indicator function can, therefore, be reduced to invert the gradient operator best approximating $\vec{V}$. This can be done via determination of a solution to the standard Poisson problem. Next the surface ∂_M can be reconstructed with the extraction of an isosurface.

$$\triangle_X \equiv \nabla \cdot \nabla_X = \nabla \cdot \vec{V}. \tag{2.18}$$

The Laplace operator is denoted with ∇. More in depth information on the approach can be gained in [47].

3 Experimental Setup

In this chapter, the experimental setup is described. Thereby the characteristics of the used hardware are elaborated as for example the calibration specifications of the used force/torque sensor and the properties of the 3D printed probe which is designed to be in contact with the to be explored objects. Furthermore, the used software libraries and frameworks are illustrated. The applied hardware and used software have to fulfill specific characteristics to be applicable for blind surface exploration. Precise motion execution and fast reaction times considering unforeseen events as the occurrence of contact between robot and environment are mandatory. The illustrated hardware setup in 3.1 has been chosen to fulfill those requirements.

3.1 KUKA LWR IV

KUKA and the German Aerospace Center (DLR) have developed the KUKA LWR IV as the outcome of a research collaboration. The KUKA LWR IV has been built to enable manufacturers and researchers to develop new robotic applications as well as to simplify the development of robotic applications. The KUKA LWR IV is equipped with position sensors on the motors and joint torque sensor at each joint which makes active compliance possible without the need for compliant end-effectors. Tables 3.2 and 3.1 show specifications of the robotic arm. Table 3.3 depicts the Denavit-Hartenberg Parameters of the LWR 4+ in its zero position (cf. figure 3.2).

New possibilities in robotic applications originated in the new capabilities of the LWR in contrast to industrial robots at the time. While industrial robotic arms have mainly been programmed to follow preprogrammed trajectories and automate processes, the ability to measure torques at the joints and react to forces and moments on-line led to new possibilities in research for robotic applications, e.g. autonomous interaction in unknown environments [48]. The LWR 4+'s characteristics suit needed abilities for reliable and robust human-robot interaction. It can react to unforeseen events due to the torque sensor measurements with compliant motion, has low weight and clamping of human body parts between links is less common [8]. Among other things the option to write control structures via the Fast Research Interface (FRI) (cf. 3.5) in the popular programming language C++ and furthermore the funding for robotics research in the world and especially in Europe made the KUKA LWR IV popular in robotic research laboratories. KUKA and DLR [48] state that the KUKA LWR IV has a servo-control rate of 3 kHz locally in the joints and 1 kHz overall. Together with its state control, the model of the robot, powerful drives and lightweight construction (cf. 3.1) enables active damping of vibrations for excellent motion performance concerning path accuracy and repeatability achieves programmable axis-specific and Cartesian compliance. Further as advantage of the compliance control the manual guidance of the robot is described.
The production of the KUKA LWR IV first started in 2008 for 60 robots and has at the time

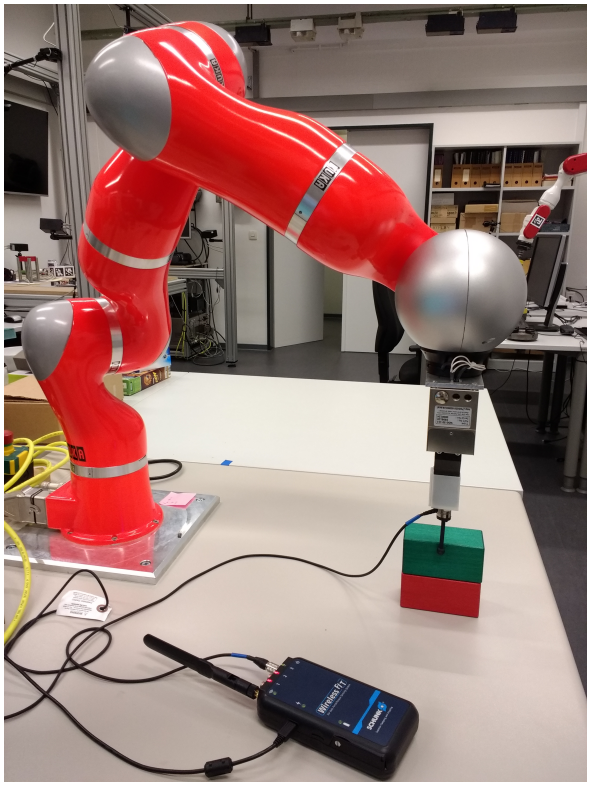


Figure 3.1: The hardware setup consisting of the KUKA LWR IV, the mounted WSG-50 gripper and attached probe with ATI six-axis force/torque sensor nano17e and the wireless force/torque device to send the force readings over wireless LAN is shown. Additionally, test objects for surface exploration are shown next to the probe in red and green. The setup is located in one of the research laboratories of the TAMS research group at the University of Hamburg.

Table 3.1: Axis data describing the KUKA LWR IV(cf. [49]) regarding payload, degrees of freedom among others are listed in the table.

Payload	7 kg
Degrees of Freedom	7
Number of axes	7
Repeatability (ISO 9283)	$+/-0.05mm$
Weight	$16kg$
Volume of working envelope	$1.84m^3$

not been released for use in production but has been sold to engineering departments and robot research laboratories. The later updated industrial version is sold under the name of KUKA LBR iiwa.

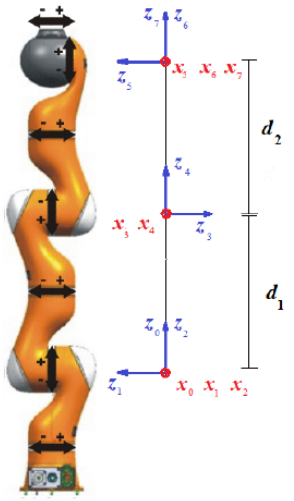


Figure 3.2: The frames of the KUKA LWR IV in zero position according to Denavit-Hartenberg notation are shown. The x-axes point toward the viewer [8].

Figure 3.3: The Denavit-Hartenberg parameters of the KUKA LWR IV in zero position are listed in the table.

Link	a_i	α_i	d_i	θ_i
1	0	$\pi/2$	0	q_1
2	0	$-\pi/2$	0	q_2
3	0	$-\pi/2$	$d_1 = 0.4m$	q_3
4	0	$\pi/2$	0	q_4
5	0	$\pi/2$	$d_2 = 0.39m$	q_5
6	0	$-\pi/2$	0	q_6
7	0	0	0	q_7

Table 3.2: The specifications of the KUKA LWR IV joints in terms of motion range, speed with rated payload and maximum torque are described (cf. [49])

Axis data	Motion Range	Speed with rated payload	Maximum torque
Axis A1 (Joint 1)	+/- 170°	110°/s	176 Nm
Axis A1 (Joint 2)	+/- 120°	110°/s	176 Nm
Axis A1 (Joint 3)	+/- 170°	128°/s	100 Nm
Axis A1 (Joint 4)	+/- 120°	128°/s	100 Nm
Axis A1 (Joint 5)	+/- 170°	204°/s	100 Nm
Axis A1 (Joint 6)	+/- 120°	184°/s	38 Nm
Axis A1 (Joint 7)	+/- 170°	184°/s	38 Nm

3.2 Schunk WSG-50

The Schunk WSG-50 [50] is a servo-electric two-finger parallel gripper with sensitive gripping force control and long stroke. Tactile sensing fingers from Weiss Robotics are installed on the WSG-50. The combination of the gripper and sensitive fingers represented in figure 3.4 will not be described any further as it only serves as a flange to comfortably attach the 3D printed end effector instrument and remains rigid at all times.

Figure 3.4: The Schunk WSG-50 two-finger parallel gripper is shown in this figure. The gripper is used to mount the probe on the KUKA LWR IV.

3.3 ATI Wireless Force/Torque Sensor System

The ATI Wireless Force/Torque Unit is able to control up to six ATI Multi-Axis Force/Torque transducers. Among others, the unit supports the ATI nano series. The unit provides an antenna and runs on an integrated rechargeable battery and is be charged via mini-USB connector. The IEEE 802.11 standard characterize the performance and range of the wireless device. ATI states that the typical range of the antenna in an office type environment is 30m ([51], p. 47). The battery life powers the device for one hour maximum with all six transducers at full measurement rate. The device additionally provides the possibility to store sensor data on a MicroSD card. In figure 3.5 the Wireless Force/Torque Unit (distributed by Schunk) is pictured. Further information about the specifications of the unit can be found in [51] on pages 10-13. To initially configure and install the wireless force/torque system on a computer, the wireless force/torque unit first has to be set up and configured to connect it with a transducer and display the sensor values. The installation procedure is found in the manual [51] on page 14 et seq.

The force/torque system comes with a Java demo application. In order to display the measured force and torque values, one has to set up a profile. Amongst other settings, the data packet rate, active transducer slots and force and torque units can be set. Then, after connecting to the IP corresponding to the wireless force/torque device, the application finally displays streaming data from the connected transducers.

The device can be accessed via command interface either by using a USB connection by way of a virtual serial port or wireless by Telnet communication on port 23 of the device. The interface allows to read and update system settings as well as to read the streaming data. Various settings can be updated. Among other commands it is possible to command whether to dump the UDP packets to console, to reset the device, to power control the connected transducers, to select the WLAN frequency band, to set the packet rate, to set a calibration for the transducers and to set a transducers bias. The possibilities are manifold. The example of setting the bias of a transducer demonstrates exemplary the use of the command interface:

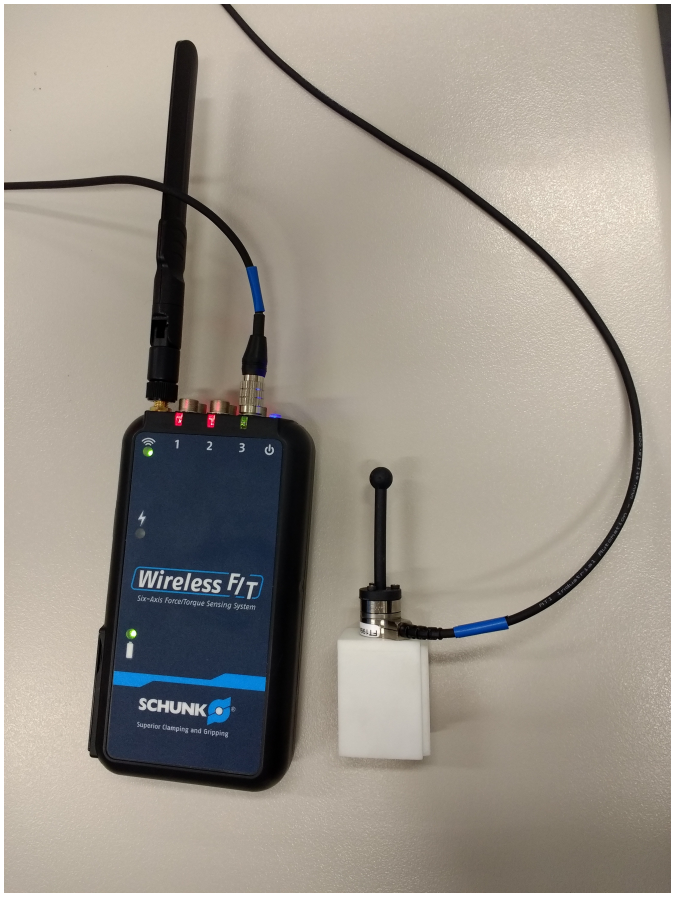


Figure 3.5: The pictured ATI/Schunk Wireless Force/Torque Unit is connected to the ATI nano17e transducer. The green LEDs show active status of the transducer slot 3, the device running in battery mode and active wireless LAN.

```
> BIAS 1 ON
```

The bias of transducer 1 is set at the current time. Once a bias point is set, the offset of the corresponding transducer is calculated according to the current load and forwarded to the wireless force/torque device. The offset and the reading of the transducer now adds to zero until force and/or torque differs to the bias point.
The ATI nano17e six-axis force-/torque transducer (cf. 3.6) is intended to be used with robotic and/or automated machines. The used nano17e SI-12-0.12 has physical specifications of 17 mm diameter, 14.5 mm height, 0.009 kg weight and therefore fits into restricted spaces of applications. An advantage of the small weight lies in the almost nonexistent influence on the payload of manipulators. The resolution of force and torque of the nano17 enables to sense a difference of 1/320 N in all three force axes as well as 1/64 Nm in the torque axes. Among other specifications (cf. 3.3), the resolution and sensing ranges enable a robot control system to react to subtle

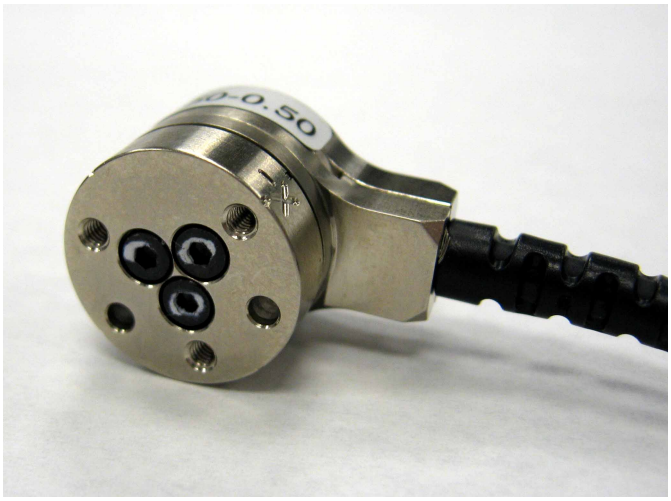


Figure 3.6: The ATI six-axis force/torque sensor nano17e is equipped with a stainless steel casing and provides holes for screws to mount the sensor on appropriate devices. More specifications on the sensor are listed in tables 3.3 and 3.4

Table 3.3: Nano17 Properties (cf. [51] p. 32)

Single-Axis Overload	(SI) Metric Units
Fxy	+/- 250 N
Fz	+/- 480 N
Txy	+/- 1.6 Nm
Tz	+/- 1.8 Nm
Physical Specifications	
Weight	0.00907 kg
Diameter	17 mm
Height	14.5 mm

Table 3.4: Nano17 SI-12-0.12 Calibration Specifications (cf. [51] p. 33)

Axes	Sensing Ranges	Resolution
Fxy	12 N	1/320 N
Fz	17 N	1/320 N
Txy	120 Nm	1/64 Nm
Tz	120 Nm	1/64 Nm

changes in force and torque values. The nano17 is typically used in dental research, robotic surgery, robotic-hand, and finger-force research [52].

3.4 3D printed Probe

In order to fulfill the requirements for an end effector in the surface exploration approach of the thesis, 3D printed objects (holder, sphere and stick combination) have been chosen pictured in 3.7. The fingers of the WSG-50 are plugged into the holder to flange the end effector onto it. The nano17e transducer is installed between holder and the stick and sphere combination.
The chapter on the concept of the implementation of the surface contouring and reconstruction

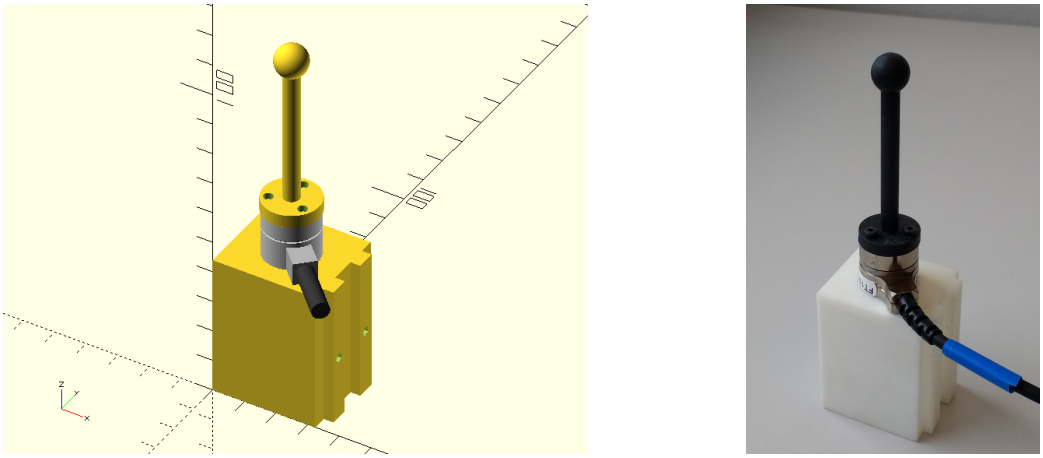


Figure 3.7: The sensing probe is a combination of the 3D printed objects and the ATI six-axis force/torque sensor nano17e. On the left figure the single models of the holder, stick and sphere as well as a mounting plate are shown. The dimensions of the model are specified in millimeters. The right figure shows the 3D printed and assembled probe with holder in white.

gives information on why a sphere has been chosen as last link respectively as link to be in touch with the to be explored surfaces.
The development of applications to explore and contour objects involves the risk of damaging the environment as well as the robot itself. Therefore the possibility to 3d print an end effector with compliance regarding the ability to bend to a certain degree and to break when to much force is acting upon it is a valuable asset. The holder stick and sphere combination is printed with a Formlabs Form 2 3D Printer which uses stereolithography (SLA) technology and artificial resin material.

3.5 Fast Research Interface - FRI

The Fast Research Interface Library [9] enables an easier use of the KUKA LWR IV functionality via C++. The FRI control system architecture is depicted in figure 3.8. The application programming interface permits the use of the robots functionality while the FRI Library is run on a remote computer. The communication between FRI and FRI Library runs over the User Datagram Protocol (UDP), and a cyclic time-frame of 1-100 ms can be set. The remote computer establishes a connection using Ethernet to the KRC (KUKA Robot Controller) via the FRI. Via this connection the KRC sends status information as joint sensor data, cartesian measured data, information about the robot state and other to the remote computer and expects in return bitfields for desired commands. Real-time access to the KRC is mandatory for robot control strategies coping with unforeseen sensor events to react instantaneously.

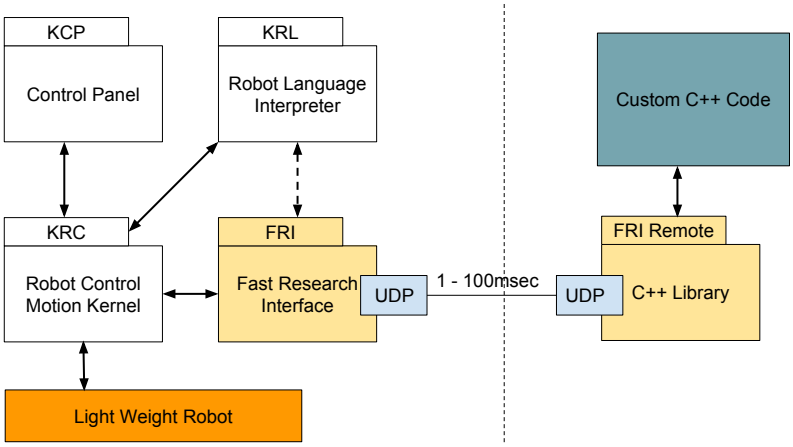


Figure 3.8: FRI control system architecture [9] showing the communication between applications executed on the remote computer and the FRI interface as well as the communication architecture between Robot Language Interpreter, Control Panel and Robot Control Motion Kernel interacting with the Light Weight Robot.

3.6 Reflexxes Motion Libraries - RML

The Reflexxes Motion Libraries (RML) is a motion planning library enabling trajectory calculation in under 1 millisecond [10] to be able to react to unforeseen sensor events. Instantaneously calculating an adjusted trajectory is crucial to control strategies operating in environments with natural and artificial force and torque constraints. Different RML Libraries exist. While Reflexxes Type I and II Motion Libraries are free to use, type III and IV are commercial. Type II is used in this work which enables to specify the target position and target velocity at the target position. As depicted in 3.9, the input parameters include the current state of motion, the kinematic motion constraints and the target state of motion. Within one control cycle, the new state of motion including position, velocity, and acceleration are calculated. Furthermore, the author of RML states that the new state of motion lies on the time-optimal trajectory to reach the desired target state of motion.

3.7 Flexible Collision Library - FCL

The Flexible Collision Library (FCL [53]) is a library for collision and proximity queries. It integrates several techniques for fast collision checking and proximity computation and is based on hierarchical representations. One can choose queries to include discrete collision detection, continuous collision detection, distance computation and others. FCL thereby supports collision checking and proximity calculations between basic shapes like boxes, cylinders and spheres as well as between triangle meshes/soups and point clouds. Unstructured meshes are represented as bounding volume hierarchy. The collision query returns a yes or no answer and optionally the

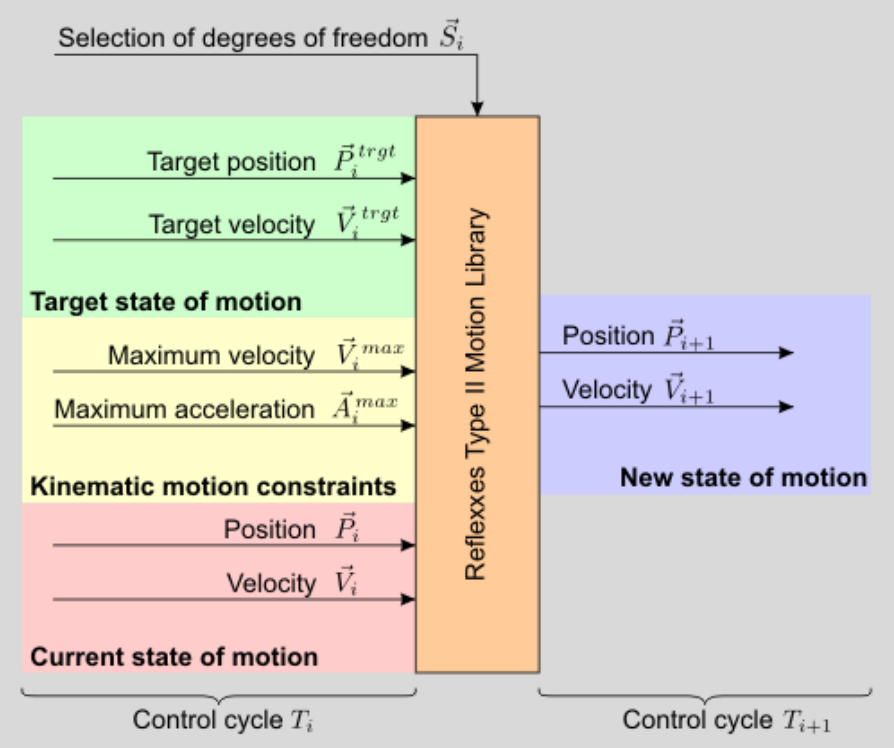


Figure 3.9: The interface of the Reflexxes Motion Libraries (RML) with its input and output values are shown in this figure. Based on the current state of motion and the kinematic motion constraints as well as the target state of motion, the goal motion states to arrive at the target state of motion are calculated within one control cycle. [10]

contact points and contact normal can be computed and returned. The distance query returns the smallest distance between two objects and optionally returns the two closest pair of points.
The FCL is being used in this work to calculate possible collisions for the next goal position of the end-effector. Although the queries for collision checking return reliable results in most of the cases, proximity queries are used in the thesis which return even more reliable results.

3.8 Point Cloud Library - PCL

The Point Cloud Library (PCL [54]) is a library with state-of-the-art algorithms for 3D perception regarding filtering, feature estimation, surface reconstruction, model fitting and others. PCL is free and a BSD licensed C++ library for 3D point cloud processing. Although PCL has been developed to be used with 3D sensing hardware such as the in robotics popular Microsoft Kinect, it, of course, is possible to generate point clouds from non-3D sensing hardware. In this thesis contact points sensed by a force/torque sensor are transferred to point cloud points to use the many functionalities PCL provides.

The basic processing pipeline of an interface in PCL can be divided into four parts:

- Create the processing Object (e.g. filter, segmentation, feature estimator)
- Use *setInputCloud* to pass an input point cloud to the processing module
- Set parameters
- Call compute (or filter, or segment) to get the output

3.9 OctoMap

The open-source framework OctoMap [55] enables the generation of volumetric 3D environment models through octrees. OctoMap uses octrees and probabilistic occupancy estimation to represent occupied, free and unknown areas. The authors of OctoMap state efficiency and probabilistic updates of occupied and free space as central property of the framework while keeping the memory consumption at a minimum [55]. The implementation of the framework is in the form of a self-contained C++ library.
Octrees to generate 3D environment models are hierarchical data structures for spatial subdivision in 3D. Each internal node in an octree has exactly eight children. Therefore three-dimensional space is recursively subdivided into eight octants and a node represents the space in a cubic volume. OctoMap calls nodes voxels for their characteristic of representing a volume. A predefined minimum voxel size defines the stop condition in the recursive subdivision of the space and determines the resolution of the octree.
In a basic form, octrees can model a Boolean property such as if a voxel is occupied. The tree structure enables pruning, e.g., in the case of occupancy of all the children of a parent node. Pruning can lead to a reduction of the number of nodes to be maintained in the tree and therefore time savings regarding operations to be performed on the tree.
OctoMap offers easy generation of 3D models via integration of point clouds such as from 3D laser scans or stereo cameras but also the insertion of single nodes in an existing octree structure. The structure of an octree is presented in 3.10.

3.10 Robot Operating System - ROS

The Robot Operating System [56] has been presented in 2009 at a workshop on open source software on the ICRA 2009. ROS has been developed to meet challenges many developers for robotic systems witness as having to deal with varying hardware. Some if those challenges are code reusability and handling big code bases which tend to emerge in robotic systems where subsystems as processing sensor reading, plan and control motion or reason under the premise of reaching a goal state are linked. ROS has been developed as part of the STAIR project [57] at Stanford University and the Personal Robots Program [58] at Willow Garage. ROS provides a multitude of tools, libraries and conventions which simplifies the development of robust applications. ROS can run on distributed computer systems and different operating systems although Ubuntu Linux is the most used operating system in the community. One of the strengths of ROS

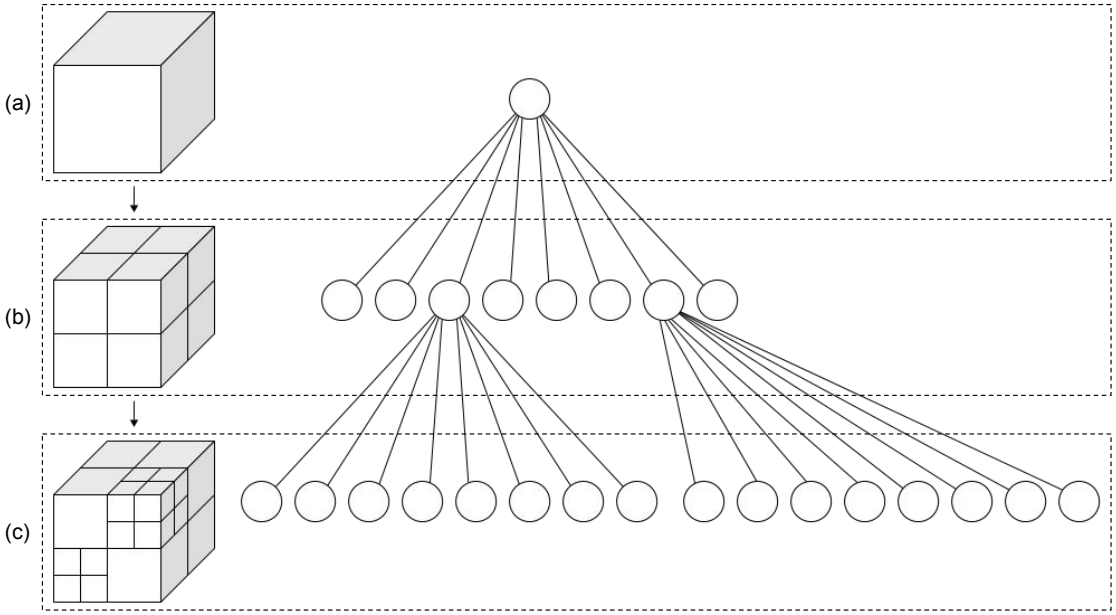


Figure 3.10: The structure of an octree representing volumes is shown. The origin node in (a) is the source node of the tree which is in (b) and (c) recursively subdivided. The origin volume is subdivided into eight volumes represented by eight nodes (b) which then are further subdivided to another eight volumes respectively nodes in (c). The minimum volume size defines the resolution of the octree. [11]

are the many available software modules for robot tasks which can easily be used and adapted. The goals of ROS are summarized by five idioms [56, p.1-3]:

- **Peer-to-peer**: A a connection between computer programs is established, the communication takes place directly between them without central routing service. Peer-to-peer targets scalability of a system under rising amounts of data to be processed.
- **Tools-based**: Individual tools d are small and together build a complex software system. The benefit of small tools is comparable to modularization. The task of modules respectively tools is clearer and improving them is easier when they are detached from logically independent parts of the software system. A canonical integrated development and runtime environment is not provided by ROS; separate programs provide tasks such as visualizing system interconnections, graphically plotting data streams or logging data.
- **Multi-lingual**: Software modules can be written in any programming languages as long as a corresponding client library exists. Client libraries exist for multiple languages as for C++, Python, LISP, Java, MATLAB, etc. Although many client libraries exist, C++ and Python are the most used languages.
- **Thin**: ROS encourages code development to occur in standalone libraries with no depen-

dencies on ROS to then wrap them to send and receive messages from other ROS modules. The developed modules allow an easy reuse outside of ROS and besides simplifies the creation of automated tests via standard continuous integration tools.

- **Free and Open-Source**: Commercial and non-commercial use is provided under the permissive BSD license of the core of ROS. Communication with interprocess communication (IPC) allows licensing of components of software systems build using ROS. Therefore, completely open source as well as partly open source development is enabled under the ROS license.

The primary goal of ROS is to support code reuse in robotics research and development. A distributed framework of processes which describe small programs can be grouped in packages and stacks, which can be shared and distributed. The design of ROS can be brought together in three levels of concepts [59]: The Filesystem level, the Computation Graph level and the Community level.

3.10.1 ROS Filesystem

ROS files are organized in a certain way which is described by the Filesystem concept level representing:

- **Packages** (cf. figure 3.11) constitute the atomic build and release item in ROS software. They represent single ROS software units containing runtime processes (nodes), ROS-dependent libraries, scripts, configuration and other files which have its use being stored in a package.
- **Metapackages** package multiple packages related to specific topics or functionality.
- **Package Manifests** (package.xml) hold metadata about a package as the likes of name, version, description and other information as dependencies.
- **Messages types** are message descriptions defining the data structures for messages sent in ROS.
- **Service types** define the data structures for request and response for services in ROS.

Figure 3.11 pictures the package *surface_contour* with a package typical structure of directories for source files, launch files, configuration files, message files and a CMake file as well as a package file in XML-format. The nodes in the source directory are started with roslaunch files from the launch directory. Roslaunch is a tool to easily start one ore multiple ROS nodes either locally or remotely via SSH. Additionally, parameters can be set via roslaunch files on the parameter service. The official build system of ROS is catkin which combines macros of CMake, an open source build system, and Python scripts. Although catkin's and CMake's workflow is similar, catkin adds additional functionality among others in terms of building multiple dependent projects at the same time. Each ROS package contains a package manifest file called package.xml and a CMakeLists.txt. The package.xml defines properties about the package such as the name, authors and dependencies on other catkin packages. The CMakeLists.txt contains

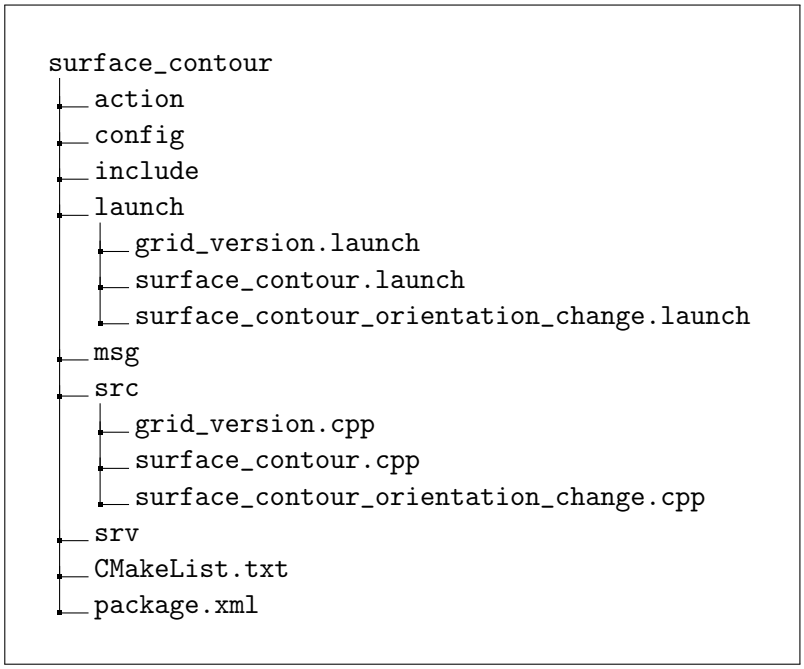


Figure 3.11: The typical structure of a ROS package is shown in this figure representing the package of *surface_contour* implemented in the thesis.

instructions to the CMake build system for building software packages.
Catkin packages can be built standalone, but the concept of workspaces [60] provides to build multiple and independent packages together at once. In general, the workspace is the place to modify, build and install catkin packages and the typical layout is shown in figure 3.12. The workspace contains up to four spaces with different purposes with respect to software development:

- **Source Space**: The source space contains one or multiple catkin packages and the source code.

- **Build Space**: CMake is invoked in the build space to build the catkin packages in the source space. The cache information as well as other intermediate files generated by the build process are kept here.

- **Devel Space**: Built targets are placed in the development space before being installed. The devel space has its benefit in being a testing and development environment.

- **Install Space**: Built targets can be installed in the install space although the install space does not have to be in the workspace.

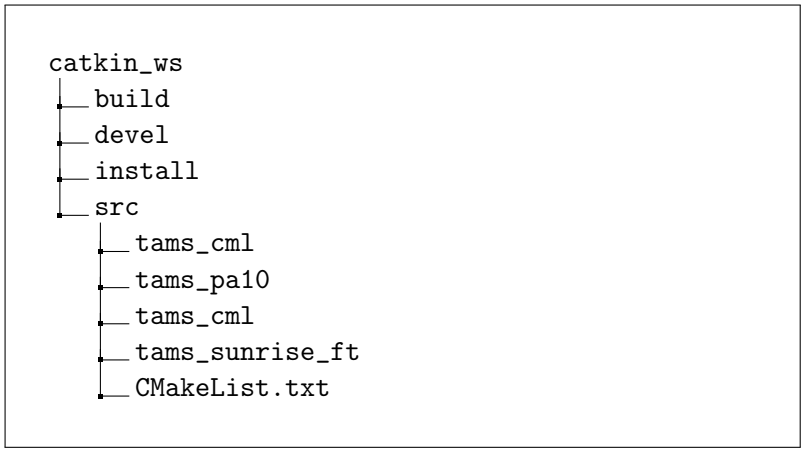


Figure 3.12: The typical structure of a ROS workspace is shown representing a catkin workspace, the corresponding directories of *build*, *devel*, *install* and *src*. ROS packages are found in the *src* directory.

3.10.2 ROS Computation Graph

The processes in ROS are called nodes and span a network of communication. This network is called the computation graph and consists of the following concepts:

- **Nodes** represent the units performing computation. ROS aims to build simple nodes rather than complex nodes which might be hard to debug. Therefore the development of multiple nodes can lead to complex programs and sophisticated functionality. The nodes usually are written using ROS client libraries such as roscpp and rospy, while roscpp is used with the programming language C++ and rospy with Python. By using the ROS communication methods, such as messages and topics, nodes can easily communicate with each other.

- The ROS **Master** enables the exchange of information between nodes by providing name registration and lookup to the rest of the nodes. Although it is required for exchange and lookup, the master is not responsible for the communication between nodes which bypasses a possible bottleneck in the case of scenarios with heavy communication needs between nodes.

- The **Parameter Server** serves as a storage for data in a central location. It can be accessed and modified from the nodes and is part of the master.

- **Messages** are sent between nodes to communicate. A message is simply a data structure containing a typed field over which corresponding data can be sent. Standard types are supported by ROS messages with types as the like of integer, floating point and boolean. Own message types can be build by combining standard types.

- **Topics** transport messages. They make use of a publisher/subscriber infrastructure. Does a node send a message via a topic, the node publishes it on the topic and is called publisher.

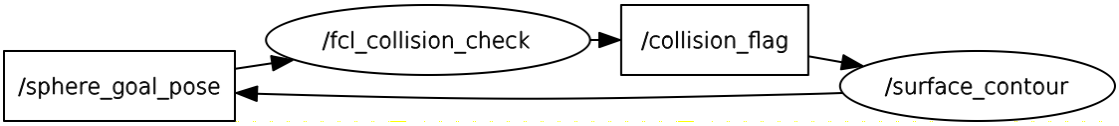


Figure 3.13: An example of an extraction of a more complex computational graph showing two nodes in rectangles and two topics in ellipses. The arrows show the data flow between nodes and topics. The */surface_contour* node publishes on the topic */sphere_goal_pose* and has a subscription on the topic */collision_flag*. The *fcl_collision_check* node publishes and subscribes vice versa.

Does a node subscribe to a topic it receives the corresponding messages sent over this topic and is called subscriber. Publishers and subscribers are not aware of each other's existence. Moreover, it is possible to have multiple publishers and subscribers on the same topic. Each topic has its unique name and is also defined by a message type.

- **Services** introduce a request/response communication interaction due to the need for another communication interaction concept than topics. Services can be compared to remote procedure calls and are defined by a pair of messages structures corresponding to request and response. A service is provided by a node which receives a request by another node, processes the request and sends the result message to the requesting node.

- **Bags** is a format for storing data in ROS. Sensor data, e.g., send over topics, can be stored in bags and replayed afterward. Bags enable to develop, test and debug ROS software without the need to have the robotic system available and running.

3.10.3 ROS Community

The ROS Community describes the resources where the exchange of software and knowledge takes place. The exchange of information with the community is crucial in open-source environments. Without platforms providing down- and upload of files or the possibility to ask and answer questions online, distributed software development hardly works, let alone dedicated software developers. The following resources describe the most important communities:

- **ROS Distributions** are versioned meta packages ready to be installed. It can be compared to Linux distribution and a ROS distribution maintain consistent versions across a set of software.

- **Repositories** most often maintained in *GitHub* provide open-source access to robot software components and are documented manifold on the ROS wiki.

- The **ROS wiki** is the main forum for documenting information about ROS. Tutorials, best practices, and information about packages and meta packages as well as error handling can be found here.

```
<link name="link_example">
  <inertial> ... </inertial>
  <visual> ... </visual>
  <collision>
    <origin xyz="0 0 0" rpy="0 0 0"/>
    <geometry>
      <cylinder radius="1" length="0.5"/>
    </geometry>
  </collision>
</link>
```

Figure 3.14: Example of an URDF link structure

3.10.4 Representing a Robot Model in ROS

The Universal Robot Description Format (URDF) and XML Macros (Xacro), both XML specifications, enable to model a robot in ROS. While URDF describes a robot model in terms of kinematic and dynamic properties, Xacro is used when large XML documents as with URDF occur to shorten XML files and make them more readable.

URDF

The files describing a robot model end with .urdf and base on an XML file format with a hierarchical structure. The URDF, furthermore to kinematic and dynamic description, enables a visual representation and collision model of the robot.
According to XML, URDF uses tags to build up the model of the robot. Commonly used tags are *links*, *joints* and the tag *robot* which encapsulates the entire XML file. The link elements describe the rigid bodies of the robot and have sub-tags describing inertial, visual and collision features. Figure 3.14 depicts an example of a link element from an URDF file. The collision model of the example link is a cylinder with a radius of 1 meter and length of 0.5 meters. Additional to describing a link by simple shape elements as the likes of a sphere, box, and cylinder, a 3D mesh can be imported to represent a robot link.

The joint elements represent the robot joints. Alongside the kinematics and dynamics of the joint, the limits on joint movement and velocity can be specified here. Different types of joints naming revolute, continuous, prismatic, fixed, floating and planar are supported. A URDF joint is located between two links and a typical syntax is as follows:

Inside the robot tag is defined the name of the robot and the links and joints constituting the robot model. A syntax example is given in figure 3.16

Xacro

The construction of robot models via URDF can become confusing with growing numbers in links and joints representing a robot model additional to concatenations of multiple URDF files

```
<joint name="joint_example" type="revolute">
  <origin xyz="0 0 1" rpy="0 0 3.1416"/>
  <parent link="link0"/>
  <child link="link1"/>
  <limit ... />
 </joint>
```

Figure 3.15: Example of an URDF joint structure

```
<robot name="robot_example">
  <link> ... </link>
  <joint>  ....  </joint>
  <link> ... </link>
  <joint>  ....  </joint>
</robot>
```

Figure 3.16: Example of an URDF robot structure.

to build up an environment in which a robot is situated. The URDF allows not only the description of robot models but also the description of the environment. In fact, it allows the description of whatever the URDF XML format syntax allows. Xacro enables modularity and re-usability of URDF files. It extends the functionality of URDF by introducing macros inside the robot description. In this way, robot models consisting of different dynamic and kinematic parts can be modularized. Additionally, basic concepts of programming such as variables and constants are introduced by Xacro which ease the process of building a robot model. An example of a Xacro file depicts the additional functionality:

Via roslaunch files, the xacro is converted to URDF and used as a parameter called 'robot_description'. Nodes are now able to access the robot model description to use it in their processes, e.g., extract joint limitations or visualize the robot model.

3.10.5 MoveIt!

MoveIt! [61] is widely used in robotics research providing a framework for motion planning, collision control, manipulation, navigation and kinematics under ROS. The MoveIt! core node provides the interface to calculate forward and inverse kinematics. The mandatory information about the kinematic is provided by the ROS parameter server. The parameter server provides the URDF, the SRDF (Semantic Robot Description Format) and configuration files. The SRDF and configuration files can comfortably be generated with the MoveIt! Setup Assistant and emerge from the URDF files. Although many possibilities in using MoveIt! exist for the application of the thesis; only the forward kinematics calculation is used due to manifold reasons. Crucial in the surface contouring task is the instantaneous trajectory change in case of a contact occurrence between end-effector and object. The standard trajectory planning algorithms of the OMPL (Open Motion Planning Library) used in MoveIt! do not fulfill that requirement.

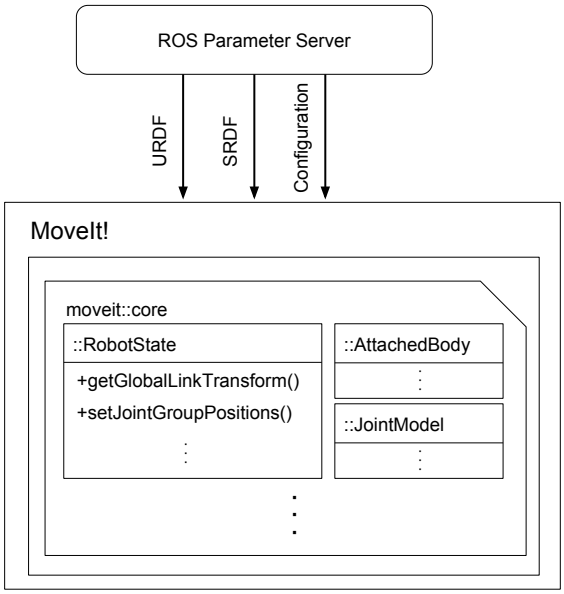


Figure 3.17: Overview on MoveIt! namespace moveit::core showing where to set joint values and get link transforms. The methods of *getGlobalLinkTransform()* and *setJointGroupPositions()* are located in the moveit::core::RobotState class and are used in the course of the surface exploration nodes to gather the current pose of the robots frames and to set joint positions for Cartesian goal positions of the probe.

To get the pose of a frame relative to a reference frame, *getGlobalLinkTransform* has to be called on a moveit::core::robot_state instance (cf. 3.17). Via *setJointGroupPositions* the joint values of the robot can be set on a *robot_state* instance. Usually, a robotic arms frames reach from the base of the robot to the end-effector. Setting the end-effector frame as link to be transformed to in *getGlobalLinkTransform* results in the Cartesian position of the end-effector link in relation to the base frame.

4 Concept

The chapter on concepts illustrates the approach on how to reach the set thesis goals. First, the decisions on the used hardware are shortly introduced with emphasis on the printed 3D probe. The concept of surface exploration respectively surface contouring is described as well as how to achieve surface reconstruction.
In blind surface exploration tasks based on single force sensors, knowledge about the environment is little or non-existent in comparison to vision-based approaches. Therefore and for simplicity, assumptions have been made including an exploration area in which the to be explored object is situated to not have to explore the whole workspace of the KUKA LWR IV.
Key to blind surface exploration is the strategy on how to reliably contour objects and in parallel avoid collisions with the environment. Concepts to achieve this behavior are described in the following as well as requirements for the concepts to succeed.

4.1 Hardware Setup

The hardware setup is shown in figures 4.1 and 3.1 has been chosen to meet the requirements for the approach described in this chapter to explore and contour objects in 3D to then reconstruct the shape of the object via surface reconstruction methods. More information on the hardware setup concerning the robot arm manipulator KUKA LWR IV and ATI nano17e is found in chapter 3.

3D printed probe The design of the tactile probe compound of sphere and stick is used to allow fairly comfortable surface contouring characteristics. The sphere constitutes the link and is expected to be in contact with the object surface during the contouring process. The stick constitutes the connection between sphere and force/torque sensor which itself is connected to the fingers of the WSG-50 gripper mounted on the KUKA LWR IV. Bulkiness in the end effector leads to higher chance of unwanted collisions between object and robot in blind surface exploration tasks. The force/torque sensor is installed between the 3D printed holder and the stick as part of the end effector. The proximity to the sphere and therefore proximity to the last link in the robot kinematic chain retains distortion of the measured force values emerging due to intrinsic compliance behavior of the robot.

The shape of a sphere comes with valuable characteristics for surface contouring. Does a sphere touch a plane surface of an object, only one contact point emerges and therefore eases the determination of the contact point to later being used in surface reconstruction. Furthermore the normal force vectors acting on a sphere always pass through its center and allow an easy determination of the contact point location.
The contact point P on the surface of the sphere can be determined as follows:

$$\vec{v} = [F_x, \ F_y, \ F_z] \tag{4.1}$$

$$|\vec{v}| = \sqrt{{F_x}^2 + \ {F_y}^2 + \ {F_z}^2} \tag{4.2}$$

$$P = [x, y, z] = r \left[\frac{v_1}{|\vec{v}|}, \ \frac{v_2}{|\vec{v}|}, \ \frac{v_3}{|\vec{v}|} \right] \tag{4.3}$$

P is the contact point on the sphere represented in the coordinate frame with origin at the center of the sphere. x, y and z are the coordinates of the contact point, r denotes the radius of the sphere. F_x,F_y,F_z are the elements of the force vector $\vec{v}$ and $|\vec{v}|$ represents the magnitude of the force vector.
The force normal represents the surface normal of the object and therefore the surface direction of the object in contact with a sphere is easily determined by the orthogonal vector to the force normal. In the case of three dimensions, the orthogonal vectors span a plane while in the case of two dimensions, orthogonal vectors directing in two opposite directions are obtained (cf. 4.2). Considering two dimensionalities, the tangent to the circle generated of the sphere touching it at the contact position is orthogonal to the force normal and can be placed at the contact point. Hence, the shape of the sphere allows easy determination of the contact point and based on the contact point location easy determination of the surface direction of the object.

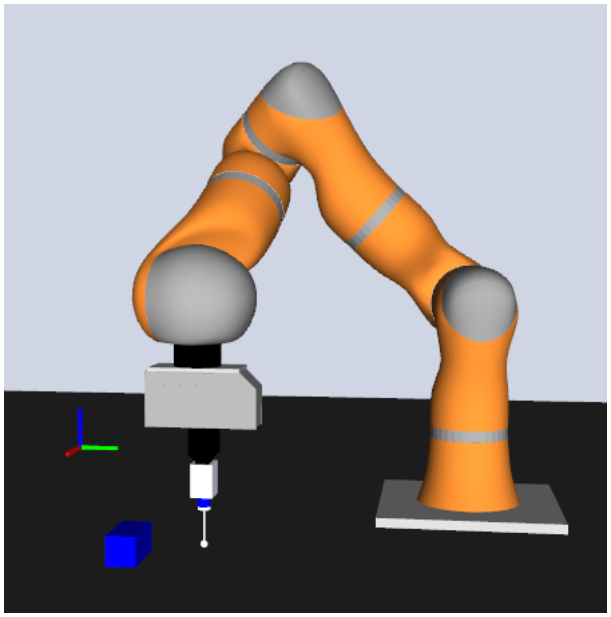


Figure 4.1: The hardware setup shown includes a KUKA LWR IV, a Schunk WSG-50, the 3D printed combination of holder, ATI six-axis force/torque sensor nano17e and probe.

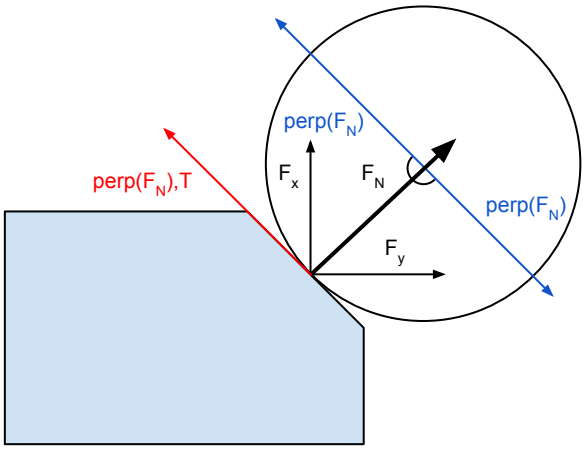


Figure 4.2: This figure illustrates exemplary an object exerting a force on a sphere and how to derive the surface direction of the object. The acting force between object and sphere passes through the sphere's center. The perpendicular vectors (perp(F_N)) of the force normal (F_N) point in the directions of the object surface direction. The tangent (T) of the contact point on the sphere is parallel to the perpendicular vectors of the force normal.

4.2 Surface Exploration/Contouring

Independent of the specific strategy for blind tactile exploration, the general process can be described simplified as a loop of measuring, analyzing, planning and motion. In contrast to vision approaches, the data to be analyzed is not generally available but has to be gathered via motion and contact to the environment.
Starting at the tactile sensor reading in 4.3, the readings are analyzed in the next step regarding on how to react to them motion-wise, e.g., set a new position goal for the robot. The motion trajectory then has to be planned, and finally, the execution of the motion is conducted. Motion execution, motion planning and sensor reading analysis are each dependent on the preceding process. The execution of tactile sensor reading is not dependent on the motion execution and is performed permanently, although the sensor reading outcomes are dependent on motion.

The general process of tactile exploration is to analyze based on the measurements of the tactile sensor, as can be a force/torque sensor, the current situation and subsequently plan the next motion and execute it. In motion, the sensor readings have to be permanently analyzed to be able to react instantaneously.
The approach for exploration of the thesis is divided into two parts. The strategy for the case of being in contact with the object differs from the strategy with no contact. Naturally, once a contact between end effector and object occurs, the strategy is to hold the contact while contouring the object. Is no contact being made yet, the strategy is to cover a grid where the object is supposed to be.

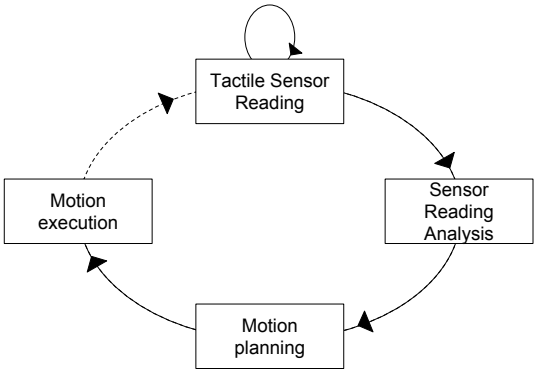


Figure 4.3: A loop describing the process of tactile sensing in blind surface exploration is shown and illustrate the sequential processes to analyze tactile sensor readings to plan motion and finally execute motion. The tactile sensor readings thereby are not dependent on the other processes concerning sequentiality, but its readings are dependent on the motion execution.

4.2.1 Motion Control

The motion control to explore the surface area and the surface of an object can, according to the interaction control section in the chapter fundamentals (cf. 2.4), be described as active interaction control. More specifically indirect force control by admittance control is applied together with point-to-point motion. In other words, if the force/torque sensor readings are evaluated to represent a contact to an object in the environment, the motion of the robotic arm is adjusted to not push into the object, but to contour the object once a contact has been made.
The determination of the trajectory of the robotic arm is based on point-to-point motion where a single point in Cartesian space describes the goal position of the tool center point. The TCP is the point in relation to which all robot positioning is defined as well as in our case constitutes the origin of the sphere coordinate system.
To perform admittance control via point-to-point motion, points describing goals resulting in motion not pushing into the environment objects respectively contouring the objects are to be found. The strategy on how to achieve this behavior if requirements (cf. 4.4) are met in a simple way is described in the following sections.

4.2.2 Assumptions and Simplifications

For the sake of simplification and applicability of the approach to explore and reconstruct an object, assumptions have to be defined. The assumptions concern the object and the setup to perform the exploration. The object is rigid and motionless throughout the exploration and furthermore convex to ensure the development of a general approach for a defined class of objects. A universal approach to ensure blind exploration and reconstruction of all kinds of object shapes with reasonable results is yet to be found, if possible. The exploration strategy is tightly coupled to the mechanics and kinematic of the robot's manipulator. The kinematic chain and joint

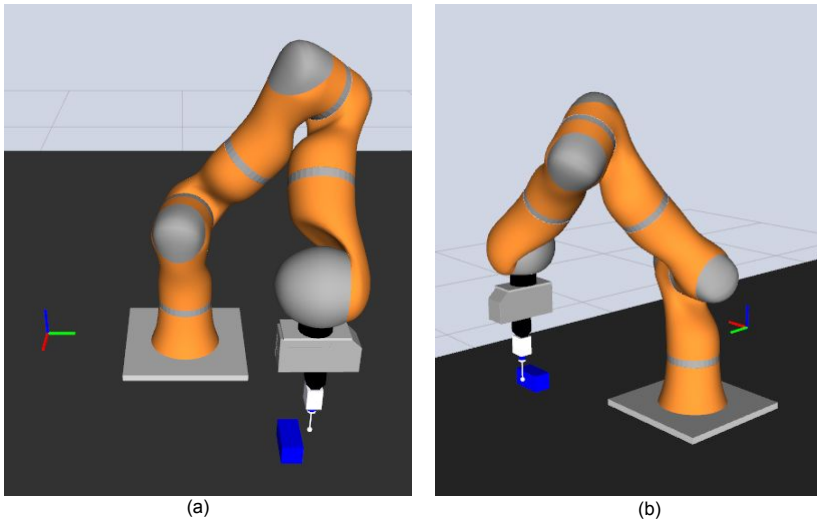


Figure 4.4: Figures (a) and (b) show the same joint configuration from different angles in which a straight line on the y-axis (green) in -y-direction is not possible. The joint configuration limits the ability to continuously follow the object surface beginning at an arbitrary position.

types, as well as joint limits, specify the reachable and dexterous workspace. The ability to reach positions in 3D in arbitrary end effector orientation is desirable in the case of exploration. Contouring an object without losing contact to the surface requires orientation change of the end effector and diminishes the likelihood of unwanted collisions between robot and object. Nevertheless, joint limits and kinematic chain based limitations in freedom of motion can result in the inability of an end effector to move in straight lines as in following an object contour (cf. 4.4). Of course, one might address this problem with appropriate motion planning, but has to be done before the motion takes place and in knowledge of the environment during the motion which is not known in blind surface exploration respectively contouring.

The figures of 4.4 represent a joint configuration of the KUKA LWR IV which does not allow further motion of the end effector in current orientation to contour the object. In order to reach the position goals needed to contour the object result in joint reconfigurations leading to high motion ranges including the loss of contact to the object and therefore the stoppage of surface contouring. High ranges of motion are accompanied by motion in the end effector not in convention with surface contouring and therefore unwanted. Additionally, the risk of collision with the environment increases due to limited knowledge about the surroundings.

Thus a setup has been chosen to preferably avoid this problem as can be seen in figure 4.1. In this joint configuration motion along the y-axis towards the object and surface contouring is ensured. To avoid the tedious and time-consuming blind search of the object to be explored in the workspace of the KUKA LWR IV, the objects to be explored are placed in the same position. Hence, the end effector motion, to begin with, can be considered given.

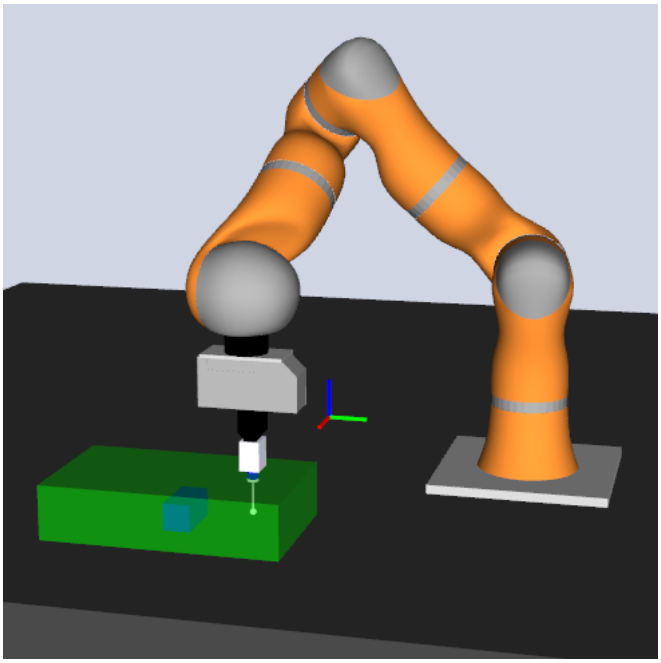


Figure 4.5: The green box in the hardware setup for the thesis represents the exploration area to find and contour the object. The coordinate frame in red, blue and green represents the static coordinate frame in which Cartesian position control goals are expressed

4.2.3 Exploration Area

Blind surface exploration is time expensive and therefore motion along an exploration area shown in figure 4.5 has been implemented. The TCP drives through the area following straight lines to cover the surface area in two dimensions. In the case of the pictured grid, the TCP is driven along the axes of y and x.

Knowing about the completion of object exploration with unknown shape is nontrivial. Therefore the position goals in the implemented exploration and contouring approaches are defined to generally alter no more than two values of the end effector position either in no contact or contact situation with respect to a static coordinate system. The idea is to simplify time efficiency in covering the exploration area via motion alignment on two axes and to develop an easily comprehensible exploration and contouring strategy assuring completion in the exploration of objects. Systematically covering an exploration area via position control in a static coordinate system is straightforward.

4.2.4 No Contact

In the case of no contact between probe and object in the exploration area, the end effector respectively the probe moves according to 4.6 being aligned to the axes of a static coordinate

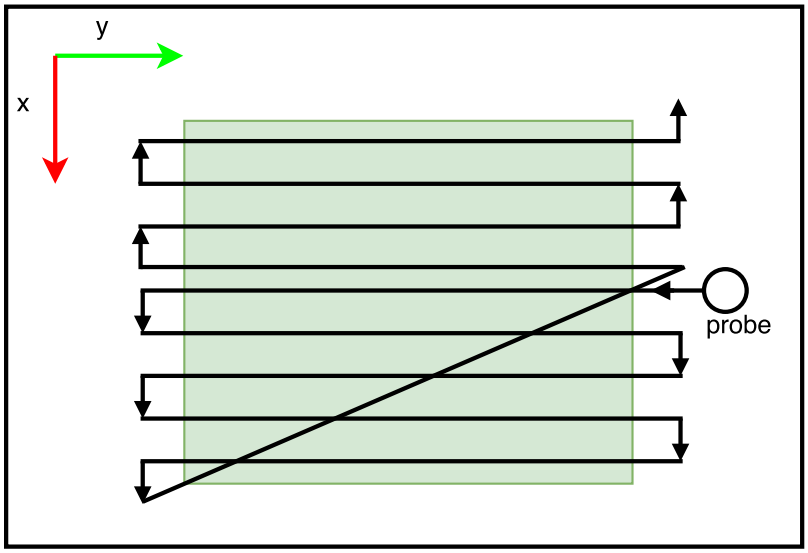


Figure 4.6: The strategy to explore the exploration area (green) is shown in top view. A combination of x-axis and y-axis motion at a time is conducted to cover the area. Once the end of the exploration area on the x-axis is reached, the probe is moved back to the point of the first contact. Then the rest of the area in plus x-axis direction is covered.

system. In the case of the exploration starting point in figure 4.6 the motion is conducted aligned to the y-axis and altered in the direction of the x-axis when the end of the exploration area has been reached in the direction of the y-axis. Next, the motion direction is in the opposite y-axis direction until again the end of the exploration area has been reached. The process is continued until the end of the area on the x-axis is reached. Depending on the starting point at an edge of the exploration area, the probe is now directed to the starting point to explore the remaining area now altering the direction in -x-axis direction ends the exploration area on the y-axis. Changes in distance x-axis motion allow more or less fine-grained coverage of the area.

4.2.5 Contact

What the concept of surface exploration provides for surface contouring is the probe in contact with an object. Although the implemented approaches in chapter 5 differ in rules on how to contour the object in specific cases, the general concept is similar. The sphere of the probe is to be in contact with the object during the process of contouring.
Is an object present in the exploration area collision detection has to be performed to neither harm the object nor the robot. The force/torque sensor is installed between the links of the 3D printed holder and the probe. The probe is mounted on the nano17e to transfer the forces and torques it is exposed to directly to the sensor. All robotic links except the probe have to be considered in collision detection as they are not represented in the nano17e's sensor readings.

Surface Contouring

The contouring strategy is composed of the following 3 phases:

1. The end effector motion is determined based on the force/torque sensor readings:
   The force/torque sensor indicates the force acting upon the sphere on the end effector. Based on this information, the contact point due to the characteristics of a sphere (cf. figure 4.2) can easily be determined as well as the expected surface direction being orthogonal to the force normal (cf. figure 4.8) at the contact point. Once the surface direction is determined, the next position goal can be set parallel to it.
   Figure 4.7 shows the robot in contact with the object with the sphere and is schematically pictured in figure 4.8. (1) Represents the the probe approaching the object in -y-direction. In (2) the force normal acting on the sphere on contact is shown to then determine the motion direction (3) to contour the object. Based on the characteristics of a sphere respectively circle described above, the object surface direction can be determined as motion direction.

2. The contact to the object has been lost:
   Is the contact to the object surface lost as can happen with surface discontinuities, the force normal sensed at the last contact point is taken and multiplied by -1 to point towards the opposite direction and used as motion direction for the end effector to re-approach the object.
   Figure 4.9 shows the loss of contact and the process of re-approaching the object on a surface discontinuation. The motion according to the force normal acting on the sphere leads to the loss of contact between sphere and object in the transition from 1 to 2. The last recorded force normal (LCF_N) is used to determine the direction of the end effector to re-initialize contact. A simple yet effective procedure with a sphere as a probe is to use the opposite direction of the LCF_N to specify the motion direction. Sub-figures 3 and 4 show the subsequent motion direction of the probe and the resulting force normal with corresponding perpendicular vector determining the direction for the next position goal.
   Can no contact be achieved through the re-approaching process for a certain distance, the object can no longer be contoured and the motion strategy to cover the grid as in no contact situation takes over.

3. Collision avoiding motion:
   In consequence of the shape of the manipulator, collisions can arise between object and robot. Is a collision distance between robot and object below a threshold, either a change in motion direction or a change in the end effector orientation is triggered. The change in motion direction takes advantage of the fact that the vectors orthogonal to the force normal span a plane, wherein each orthogonal vector constitutes a surface direction of the object. The change in motion direction aims to avoid the collision by not diminishing the collision distance.
   Figure 4.10 illustrates the occurrence of a collision threat between the end effector of the

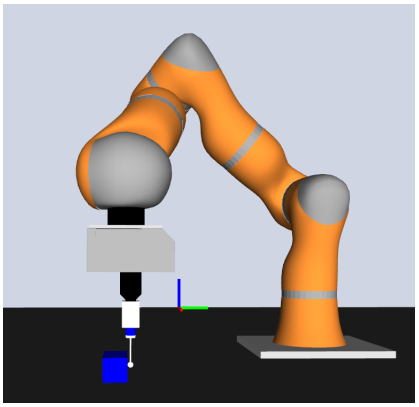


Figure 4.7: The probe is shown in contact to an object.

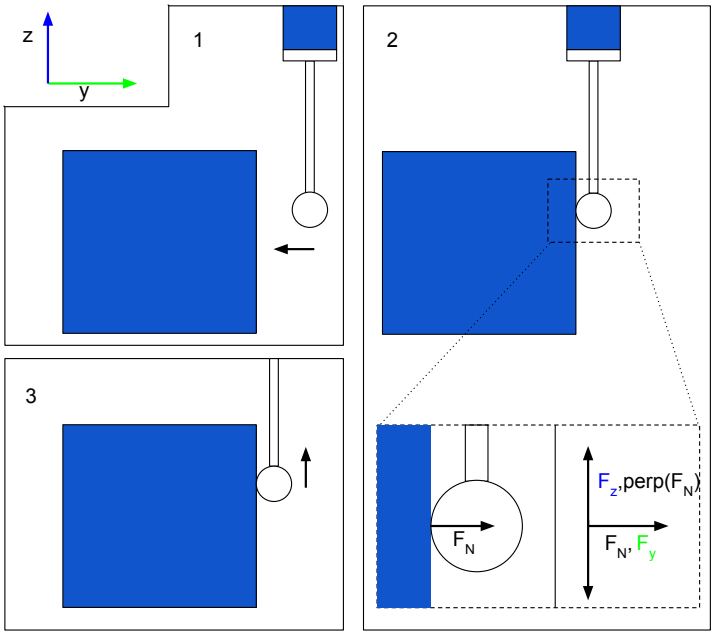


Figure 4.8: The figure shows the making of contact with the object (1), the force acting upon the sphere at contact (2) and the subsequent motion direction of the end effector in (3). The force normal (F_N) acting on the sphere is additionally shown in (2) and the resulting surface directions of the object respectively the perpendicular vectors ($perp(F_N)$) to the vector of the force normal (F_N).

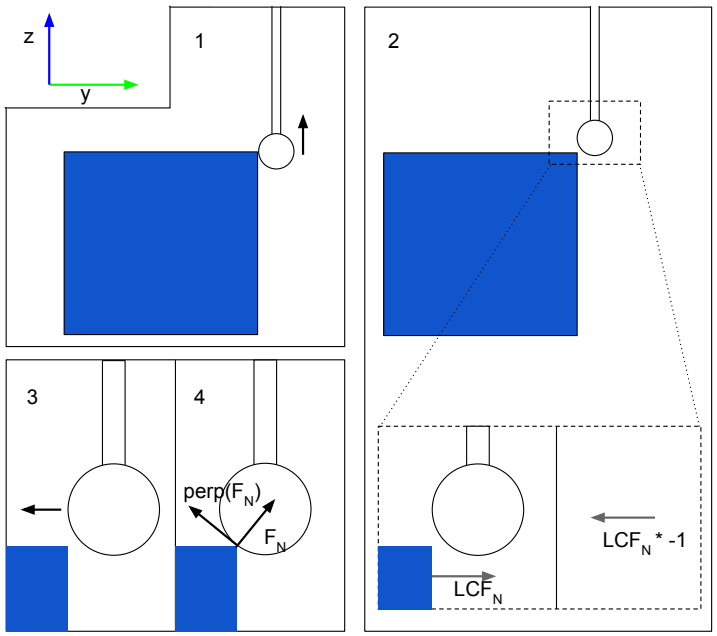


Figure 4.9: The reapproach process is visualized from 1-4. The sphere moves along the object surface in 1 and loses contact due to a discontinuity in 2. The last contact force normal (LCF_N) is multiplied by -1 to direct to the opposite and indicate the new motion direction. 3 and 4 show the following interaction between sphere and object.

robot and the object if the motion direction is retained in (a). One way of handling the situation is to change the motion direction in the opposite direction, hence moving parallel to the z-axis in +z direction (b). Alternatively, leaving the strategy of covering the grid as described above, the surface direction along the x-axis determined by the force normal acting on the sphere can be chosen to direct the end effector motion along the x-axis. As a result, the collision can be avoided. The change in orientation based on the position of the possible collision ensures to avoid the collision while maintaining the motion direction along the taken surface direction. Hence, the orientation change enables to contour more shape surface and is not entirely dependent on the impact of the shape of the end effector and manipulator on collision probability.
The figure 4.11 illustrates the change in end effector orientation to avoid collisions as well as to enable further contouring in the same motion direction.

Furthermore, other characteristics of the environment have to be taken into account in the used setup. The table on which the object is positioned has to be managed. For reasons of simplicity, the workspace of the KUKA LWR IV has been restricted to not get in contact with the table. Does the sphere reach a certain low height the motion strategy to cover the grid takes over.

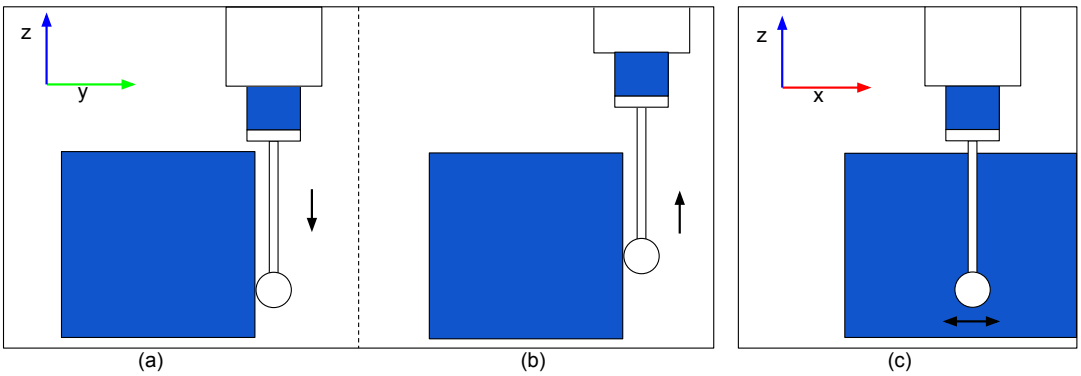


Figure 4.10: Strategies to avoid collision between robot and object are shown in the figures (b) and (c). The motion direction in (a) results in a collision if the motion in minus z-axis direction is continued. (b) and (c) show ways to avoid the collision. In (b) the motion direction is directed in the opposite direction on the z-axis. In (c) motion along the x-axis is performed to avoid collision respectively not to diminish the distance to the collision point.

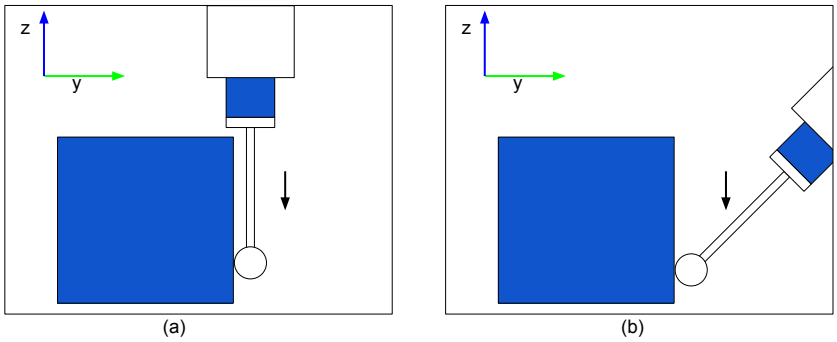


Figure 4.11: The strategy resulting in orientation change to avoid collision is shown. The collision to be expected in (a) can be avoided by changing the orientation of the end effector (b). Furthermore, motion along the object in z-axis can be continued.

Collision Detection

To perform collision detection, collision objects have to be modeled in an environment, and it has to be detected whether or not the collision objects intersect each other. A collision is detected in the case of an intersection. The Flexible Collision Library described in 3.7 is used to perform collision detection. Collision objects representing the links and shape of the robot are easily created and shown exemplary in figure 4.12.

On the other hand collision objects and contoured objects of the blind exploration process cannot be created as easily as their location and pose are entirely unknown. Via the OctoMap

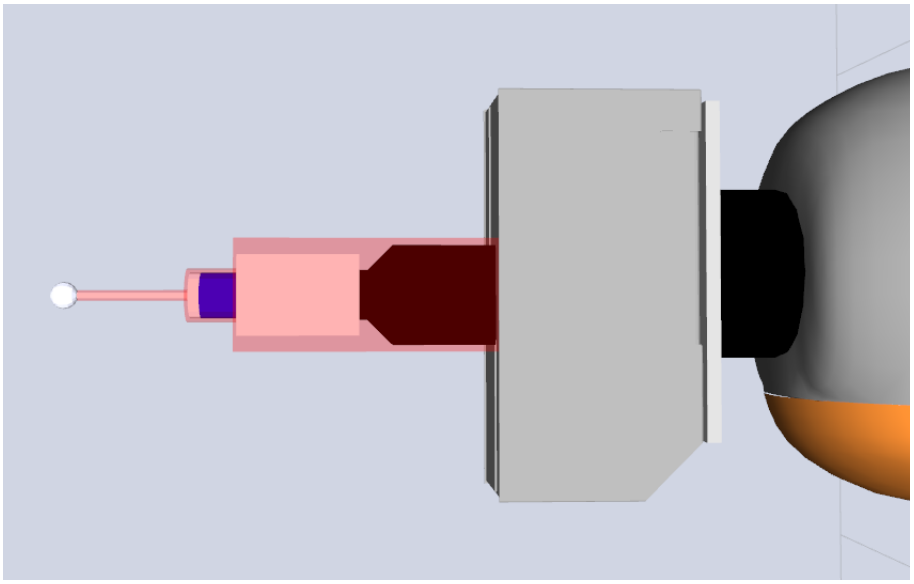


Figure 4.12: The finger of the WSG-50 gripper as well as the 3D printed holder, the ATI nano17e and the stick of the probe are surrounded by a collision shape shown in red. The sphere does not have a collision shape as it is supposed to be in contact with objects in the environment.

library (cf. section 3.9) a 3D occupancy grid is used to map the force readings interpreted as contact points to insert voxels in the grid and construct the collision object of the unknown object step by step (cf. figure 4.13). The single voxels can then be used to generate collision objects where each voxel has its corresponding collision object. Beginning with the first voxel-based collision object, collision detection is performed to avoid collisions during motion of the robot. To be able to avoid collisions, the minimal collision distance of collision pairs is observed to react in time if it drops under a particular threshold value.

4.3 Surface Reconstruction

The approach to reconstruct the shape of an object is based on building polygon meshes respectively build a convex or concave 3D hull of the contact point positions gathered through surface exploration. The approach makes use of some of the many surface reconstruction algorithms and methods emerged or used in computer vision such as moving least squares (MLS) and Poisson reconstruction.
Typically, data generated in computer vision is dense and the algorithms and methods perform best on those data sets and might fail on sparse data sets with respect to the generated shape in comparison to the real shape. Tactile surface exploration is time expensive and therefore one of its objectives is to perform time efficiently which leads to sparse data sets. In order to cope with this circumstance, upsampling can be used to generate more data points. Upsampling, on

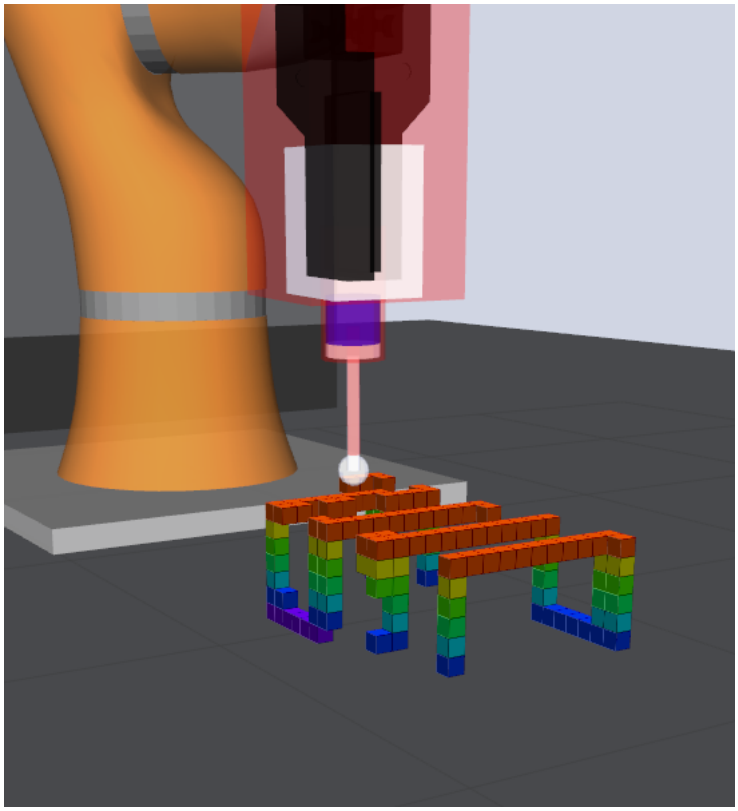


Figure 4.13: The occupied nodes of the octomap representing the contact points between the explored object and the sphere of the probe are shown with colored voxels. The color illustrates the height of the occupied nodes. The red voxels are located above the other colored voxels. The octomap has been generated during the exploration of a wooden block (cf. figure 6.1 (a)) and the shape of the object is already recognizable.

the other hand, approximates the positions of the generated data points and is prone to error due to the nature of approximation itself. The methods used in the thesis are Poisson surface reconstruction [62] and the computation of concave respectively convex hulls (cf. section 2.6). Both differ from each other in terms of tuning via parametrization and the requirements of the provided data set.

4.4 Requirements

Several requirements have to be met for the approaches of surface exploration and surface reconstruction as described in this chapter to work out. The requirements are denoted as follows for the two of surface exploration/contouring and surface reconstruction:

- **Surface Exploration/Contouring**:
  - Accurate and frequent force readings of the force/torque sensor
  - Accurate and frequent feedback on pose of the kinematic chain of the robot
  - Instantaneous determination of position goals based on force readings
  - Instantaneous motion execution of the robot arm based on commands sent from the position control
  - Dexterous workspace of the robot manipulator
  - Compliant but robust end effector
- **Surface Reconstruction**
  - Accurate contact positions and contact surface normals
  - Sufficiently dense data set with respect to the reconstruction method

The requirements for the reconstruction base on sound data sets to perform the corresponding algorithms to ensure valid results. The requirements for the exploration emphasize real-time ability regarding motion planning, motion control and collision checking. Additionally, the accuracy of force values and kinematic state of the robot is of upmost importance to ensure surface contouring and valid collision checking. Delays would result in unwanted behavior such as losing contact to the object or collisions between robot and environment. The end effector is needed to be compliant to a certain degree when in contact with an object or when the contact is initially made due to a characteristic of blind tactile exploration. In blind tactile object exploration, no collision check can be done if no knowledge about the position and shape of the object is known. Hence, a collision or contact is experienced when robot and object are in touch. In the case of both bodies being rigid with no compliance, the force acting on both either pushes one object away, shuts down the manipulator if a mechanism to avoid harm to its links and joints based on force magnitude acting upon them exists, or theoretically and immediately grows to a certain magnitude and can lead to a breakage of one of the bodies. The compliance in the end effector or kinematic chain of the robot permits reaction in time according to a moderate force and avoid damage to robot or environment.

5 Implementation

The implementation of the concepts earlier described is illustrated in this chapter. The implementation of the surface exploration and reconstruction is based on the concepts of chapter 4 and makes use of the ROS framework (cf. section 3.10) to reuse already existing nodes such as the `ros_fri` node for communication between the RML library and the FRI.
First, an overview of the collaborating modules is given in section 5.1 to illustrate the division of the implemented functionality. Then the integration of the implemented modules into the ROS framework is illustrated with an example of a URDF partially describing the probe visual and collision geometry. The core functionality and purpose of the implemented and used nodes represented in the ROS computation graph are presented. Section 5.3.3 illustrates the functionality of the implemented nodes for surface exploration and surface reconstruction. Particular emphasis is put on the functionality of the surface contour nodes. Three versions of surface contour nodes have been implemented, and the differences are exposed. The octoMaps generated by the contact points between probe and objects during surface contouring are transformed to point clouds to execute the surface reconstruction methods are shown in the evaluation chapter at 6.2.2.

5.1 Overview Collaborating/Functional Modules

An overview of the applied functionality for surface exploration and reconstruction is shown in figure 5.1. The data flow during execution is marked by arrows pointing from source of data to its destination. The surface exploration module processes information from the force/torque sensor, the collision detection module and the Moveit! kinematic service to determine the motion to be conducted by the probe to explore the surface area or contour an object. The new motion goal is forwarded to the ROS FRI controller module which transforms the motion goal to a trajectory. The trajectory path is finally sent to the Fast Research Interface of the KUKA LWR IV and executed. In order to determine the motion of the probe, the state of the KUKA LWR IV is shared with the ROS FRI controller in real-time. Hence the surface exploration module can operate on real-time position data of the robot's kinematic chain supplied by the MoveIt! kinematics service. The collision detection module detects collisions respectively distances between the robot and the environment based on the shape and motion of the robot and the gathered contact points during surface exploration representing objects.
The surface reconstruction is independent of the determination of motion to conduct surface exploration but operates on the information gathered by the force sensitive probe during the exploration.

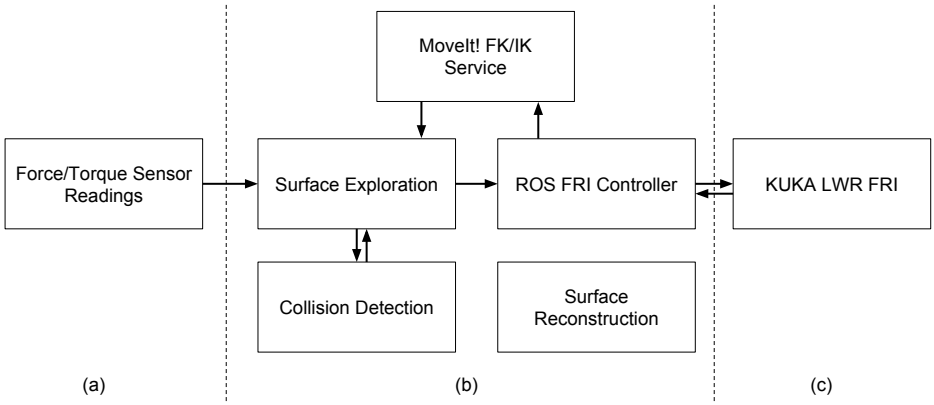


Figure 5.1: An overview of the used functionality modules in the thesis are shown. The rectangles represent the used and implemented functional modules to perform surface exploration and reconstruction. The arrows indicate the information flow between the modules. The dotted lines indicate the physical separation of the modules. The force/torque sensor readings are conducted on the ATI nano17e (a). The logic on surface exploration and contouring including the determination of motion goals and collision detection as well as the surface reconstruction are executed on a remote computer (b). Robot motion goals are commanded via the ROS FRI controller to the Fast Research Interface (FRI) architecture of the KUKA LWR IV (c) (cf. figure 3.8).

5.2 Integration in ROS

The catkin workspace in figure 5.2 includes the needed packages to run the surface exploration as well as surface reconstruction. Obviously the packages of `surface_exploration` and `surface_reconstruction` include the corresponding functionalities. The package of `tams_wireless_ft` includes the ROS node providing access to the force/torque sensor readings and the `ros_fri` package includes the ROS node which communicates with the KUKA LWR IV Fast Research Interface. The `tams_lwr_wsg50_description` and other packages contain configuration files declaring among others joint limits such as acceleration and velocity limits which have to be considered in motion planning but also the URDF and SRDF of the robot.
The division in different packages enables logical distribution of functionalities and easier maintainability and reusability.

The probe mounted on the WSG-50 gripper is designed to ease surface contouring with regard to following the surface of an object. Additionally, the design has been chosen to circumvent collisions between other parts of the robot and the environment (cf. figure 4.1). The URDF of the probe is attached to the already existing URDF of the robot. Exemplary figure 5.3 shows a segment of the XML-file describing the sphere of the probe, assigning a radius of 5 mm in visual and collision model. The color of the sphere is denoted to be silver.
The changes to the URDF of the robot lead to the need of adaptations in the KUKA controller

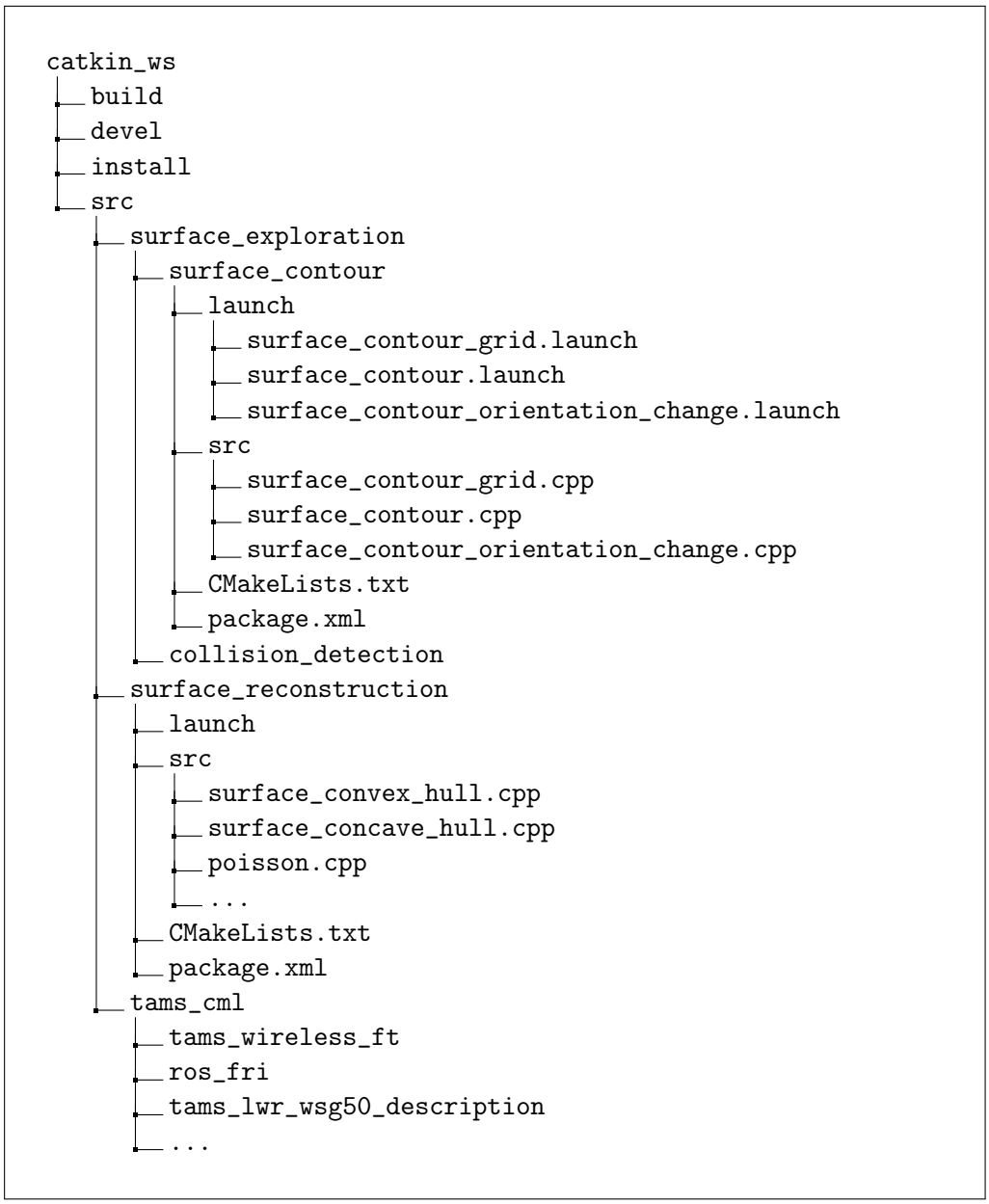


Figure 5.2: The catkin workspace used for the surface exploration and reconstruction is shown. Thereby a separation between the packages has been realized to modularize functionality to generate better maintainability and ease reusability.

```
<robot
...
   <link name="sphere">
    <inertial>
      <mass value="0.001"/>
      <origin xyz="0 0 0"/>
      <inertia ixx="0.1" ixy="0.0" ixz="0.0" iyy="0.1"
                           iyz="0.0" izz="0.1"/>
    </inertial>
    <visual>
      <material name="Silver" />
      <origin xyz="0 0 0" rpy="0 0 0" />
      <geometry>
        <sphere radius="0.005"/>
      </geometry>
    </visual>
    <collision>
      <origin xyz="0 0 0" rpy="0 0 0" />
      <geometry>
        <sphere radius="0.005"/>
      </geometry>
    </collision>
  </link>
...
</robot>
```

Figure 5.3: A segment of the URDF file of the probe is shown. The inertial, visual model and collision model of the sphere is described and the material color set to 'Silver'.

software. The TCP of the robot has initially been set at the fingers of the WSG-50 gripper and needed to be changed to the center of the sphere of the probe to guarantee motion goals being executed concerning the coordinate frame of the sphere.

The ROS computation graphs shown in the figures 5.4 and 5.5 illustrates the active nodes and topics at a certain point in time giving an overview of the data flow between the nodes.
Both figures show the nodes and topics being used during surface exploration although they differ. One figure shows the mandatory elements to execute surface exploration while the other figure shows supplementary nodes and topics which are used to visualize the procedure of exploration. Elements for visualization are, e.g., the `/tf` topic carrying the poses of the robot links or the collision object marker and `/BaseMarker` carrying the pose of the corresponding collision objects. The information on visualization are brought together in rviz, a 3D visualizer for the ROS framework.
The functionality of the nodes and the published respectively subscribed topics of the graphs are described as follows:

/wireless_ft The /wireless_ft node receives the force/torque sensor readings, converts the raw force data into Newtons and publishes it on the /wireless_ft/wrench_3 topic to the /surface_contour node. Additionally, the bias of the transducer is set to zero at the start of surface exploration to get meaningful force/torque values.

/surface_contour The /surface_contour node is the node determining the motion goals of the probe with regard to exploring the exploration area or contouring an object. The determination is based on several topics. Based on the /wireless_ft/wrench_3 information it can be determined if a contact to an object exists and how to find the surface direction of the object. The information of reaching a pose goal is communicated via /lwr/RMLFinalStateReached and the current joint state of the robot is subscribed to the /lwr/joint_states topic. Information about the existence of a collision threat is gained via the /collision_flag topic. The node publishes the pose goals of the probe via /lwr/jointPositionGoal and the wanted goal pose of the sphere on /sphere_goal_pose for the /fcl_collision_check node to determine to-be-collision situations. Additionaly, an octree structure describing the occurence of contacts is published on /octoMap.

/lwr The /lwr node represented in the graph denotes the ros_fri node implementing the robot controller communicating between KUKA LWR IV FRI and the /surface_contour node forwarding the robot joint states over /lwr/joint_states and receiving joint position goals over the topic /lwr/jointPositionGoal. Additionally the RML final state reached flag is communicated over /lwr/RMLFinalStateReached.

/fcl_collision_check The collision checking node determines if collision situation exists using the information of the octree of /octoMap, the goal pose of the sphere via /sphere _goal_pose and the current joint states of the robot via /lwr/joint_states. The information about possible collision are published on /collision_flag.

The graph in figure 5.5 is extended by elements used for visualization as follows:

/joint_state_merger The /joint_state_merger merges the wanted joints such as the likes of the KUKA LWR IV and the WSG-50 gripper and publishes them on topic /joint_states.

/robot_state_publisher The /robot_state_publisher receives the merged joint states and forwards them on the /tf and /tf_static to the /rviz node.

/fcl_collision_check The collision checker node publishes the poses of e.g. the force/-torque transducer via /CylinderMarker directly to /rviz.

/rviz Rviz visualizes the incoming information in 3D.

Overview on communication between nodes (selected topics for comprehensibility)

The roslaunch file shown in figure 5.6 shows to be loaded nodes and configuration files for surface exploration. In this example the joint_state_merger is not loaded. Hence, visualization in rviz will not show the motion of the robot nor the pose of the joint and link frames.

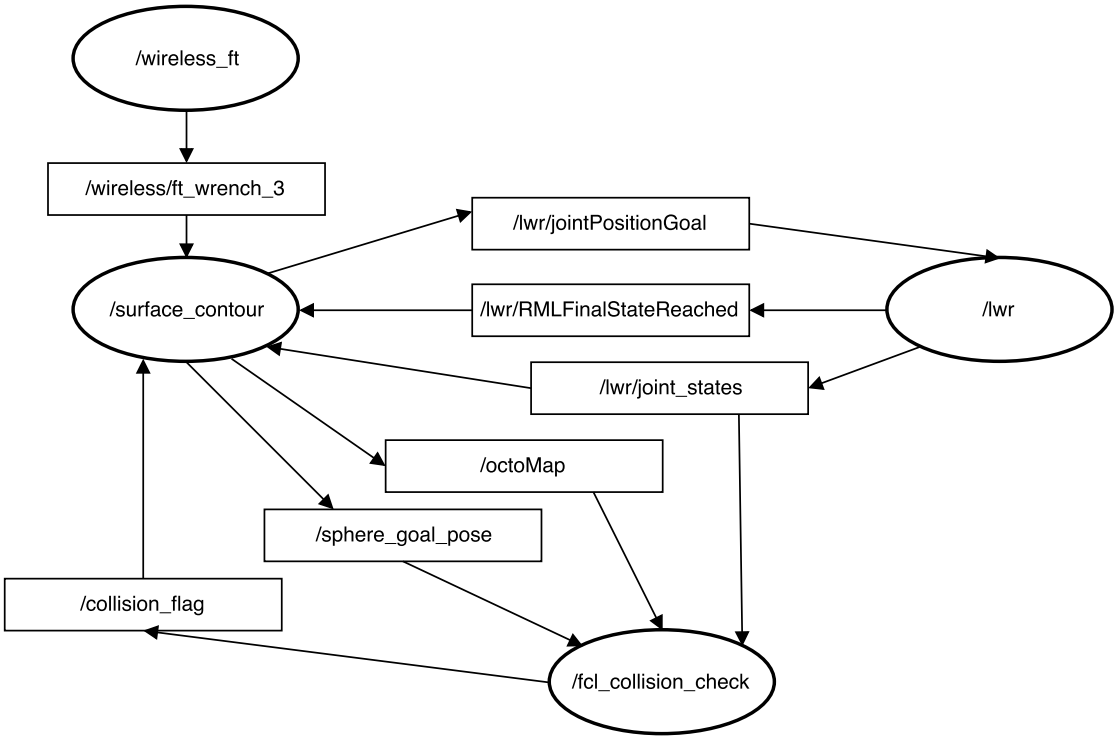


Figure 5.4: The ROS computation graph illustrating used topics and nodes is shown. The ellipses denote the nodes and the rectangles the topics. In contrast to the computation graph shown in 5.5, this computation graph shows only the mandatory elements used to conduct the surface exploration process without nodes and topics describing functionality towards visualization of the exploration process. The arrows between the elements show the communication via topics based on publisher/subscriber architecture.

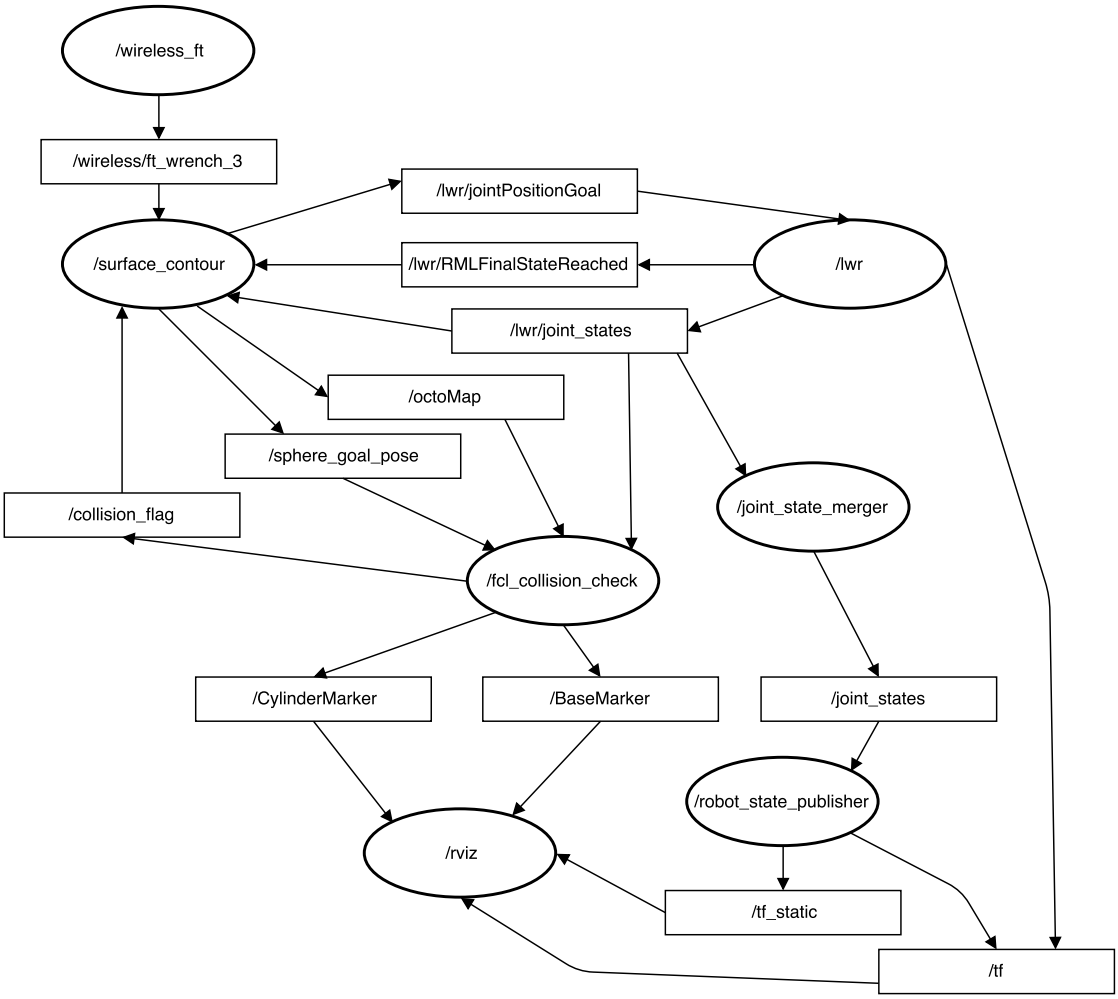


Figure 5.5: The ROS computation graph shows the used nodes and topics for surface exploration and the 3D visualization in rviz. The topics are illustrated in rectangles whereas the nodes are illustrated in ellipses. The arrows between the elements show the communication via topics based on publisher/subscriber architecture.

```
<?xml version="1.0"?>
<launch>
  <arg name="launch_joint_state_merger" default="false" />
  <arg name="launch_lwr" default="true" />    <!-- Kuka LWR with FRI -->
  <arg name="launch_rviz" default="true" />
...
<!-- upload the robot model: LWR, WSG-50 gripper, tables.. -->
  <param name="robot_description"
         command="$(find xacro)/xacro.py $(find tams_lwr)/launch/sphere_
    launch/lwr_with_wsg50_sphere.xacro" />
<!-- The semantic description that corresponds to the URDF -->
  <include file="$(find tams_lwr)/launch/sphere_launch/planning_context_
    sphere.launch">
     <arg name="load_robot_description" value="false"/>
  </include>
...
<!-- LWR arm controller node -->
  <group if="$(arg launch_lwr)">
    <include file="$(find tams_lwr)/launch/ros_fri.launch" />
  </group>
...
</launch>
```

Figure 5.6: Roslaunch files can be nested and contain commands to load other roslaunch files. The roslaunch file shown in this figure loads the launch files of `ros_fri.launch` and rviz. Additionally, the robot model containing the kinematic descriptions of among others the KUKA LWR IV, the WSG-50 gripper and probe. Once loaded to the parameter server, the kinematic description can be used to perform kinematic calculations on the models such as forward and inverse kinematics.

The arguments for the lwr (`ros_fri`) and rviz nodes are set to true. Hence they are loaded. Furthermore, the robot description, here the xacro of the environment including the robot and the semantic descriptions that correspond to the URDF files are loaded to be used for calculations on forward and backward kinematics.
Not included in the roslaunch file are the launch files for the nodes of `/wireless_ft`, `/fcl_collision_check` and `/surface_contour`. They are launched separately simplifying shut down of and restarting the nodes.

5.3 Surface Contouring

The implemented surface contouring nodes follow the concepts introduced in chapter 4 and put emphasis on real-time ability. First, the process of motion planning via RML and FRI is described. On this basis, pose goals for the probe generated by the surface exploration node can

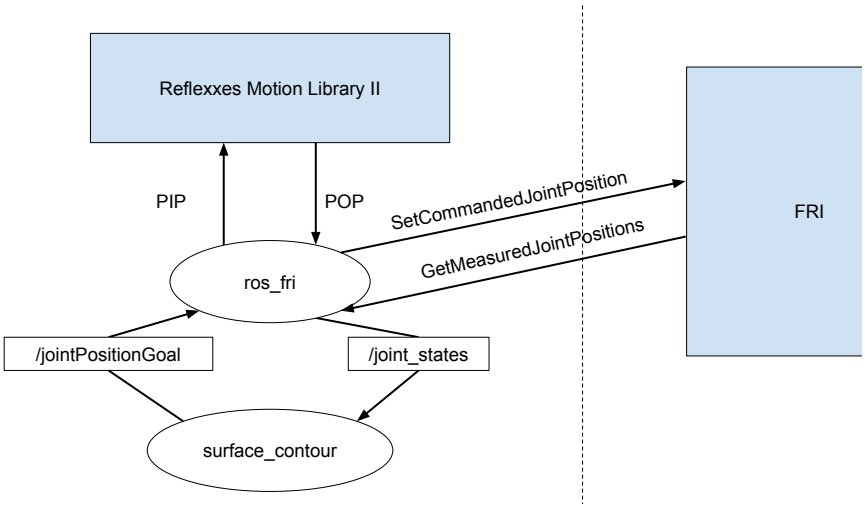


Figure 5.7: A communication diagram between the Libraries *RML*, *FRI* and the nodes `ros_fri`, `surface_contour` is shown. The `ros_fri` node is the connecting part between the other components transforming and transferring messages. Once a joint position goal has been sent to the `ros_fri` node, it is transformed to be send as position input parameters (PIP) as new target motion goal to the *RML*. The generated position output parameters denoting the next joint positions to be set on the FRI are transferred back to the `ros_fri` node and then sent to the FRI via the `SetCommandedJointPosition()` method. The `GetMeasuredJointPositions()` method of the FRI delivers the current joint positions which are then published on the `/joint_states` topic and subscribed by the `surface_contour` node to determine Cartesian pose goals for the probe based on current joint values.

be specified to explore the exploration area and contour objects. The general idea of the three implemented versions of surface exploration nodes is displayed in figure 5.8. The strategy on how to avoid collisions mostly distinguishes the surface exploration nodes. Therefore, emphasis is put on working out the differences in this aspect

5.3.1 Motion Planning via RML and FRI

The motion planning of the KUKA LWR IV is determined with the RML (cf. section 3.6) and the communicated over FRI (cf. section 3.5). The process of motion planning based on position goals send by the surface_contour node is illustrated in figure 5.7. Starting at the surface_contour node, the joint position goal to set the probe to a desired position is published on the `/jointPositionGoal` topic via an RML compatible data structure called `RML PositionInputParameters(PIP)` to the `ros_fri` node. Based on the current position and velocity, the limits for velocity and acceleration of the joints, the `RMLPositionOutputParameters (POP)` are determined by the RML. The parameters of the POP are then converted to an FRI compatible data structure and send via the `SetCommandedJointPositions()` method to the FRI.

Finally, the trajectory point determined by the RML is executed via the FRI on the KUKA LWR IV. The POP is fed into the PIP after each control cycle leading to a trajectory eventually reaching the desired position goal. Nevertheless, setting new position goals during non-finished trajectories is possible. The prevalent joint positions of the KUKA LWR IV are obtained via the `GetMeasuredJointPositions` method of the FRI library and published on the `/joint_states` topic.

5.3.2 Overview on Surface Contouring

Three different concepts for surface contouring nodes haven been implemented for the exploration task. The individual functionality and logic of the concepts overlap greatly and mostly differ in the approach of collision avoidance. Therefore a simplified UML activity diagram for the approaches illustrating the procedure of entering the grid following motion to start the surface exploration until the exploration is completed is shown in figure 5.8.
In the following description of the diagram, it has to be stated that there is a difference between *collision* and *contact*. Is the sphere of the probe in contact with an object of the environment, then a contact is prevalent. Is a link of the robot expected to be in contact with an object, it is called *collision*.
In the case of the exploration area not yet fully being covered, no contact and no collision is prevalent, then the grid following motion continues until the area is covered and the motion is stopped, or a contact or collision arises. In the case of an expected collision, a collision avoiding motion determined by a motion strategy with regard to the surface contouring version is conducted. Next, if a contact to the object has been made prior to the collision threat, the object surface is reapproached. Once a contact exists, the surface of the object is contoured until a collision is arising, the contact has been lost, or the surface exploration is terminated.

5.3.3 Functionality of Nodes

The functionality of the nodes earlier shown in the ROS computation graph is described in this section. The key concepts and methods of the nodes are presented.

`wireless_ft.cpp` The `wireless_ft` node is responsible for the gathering and transferring of meaningful force and torque values to the `surface_contour` node. Once the node is started, it first connects to the ATI Wireless F/T device via telnet. After establishing a connection, configurations as well as device parameters are set. Among others, active transducers and filter options are set. Additionally, at the start, the bias of the used transducer is set to zero at the starting position of the robot respectively the transducer. A UDP port is opened after the configuration has been finished and a command is sent to the Wireless F/T device to start streaming the sensor readings. The now incoming data is processed using the factory calibration matrices provided by ATI. The conversion from raw values to values specified in Newton are published into `geometry_msgs/WrenchStamped` messages representing a coordinate frame, timestamp and the force and torque values.

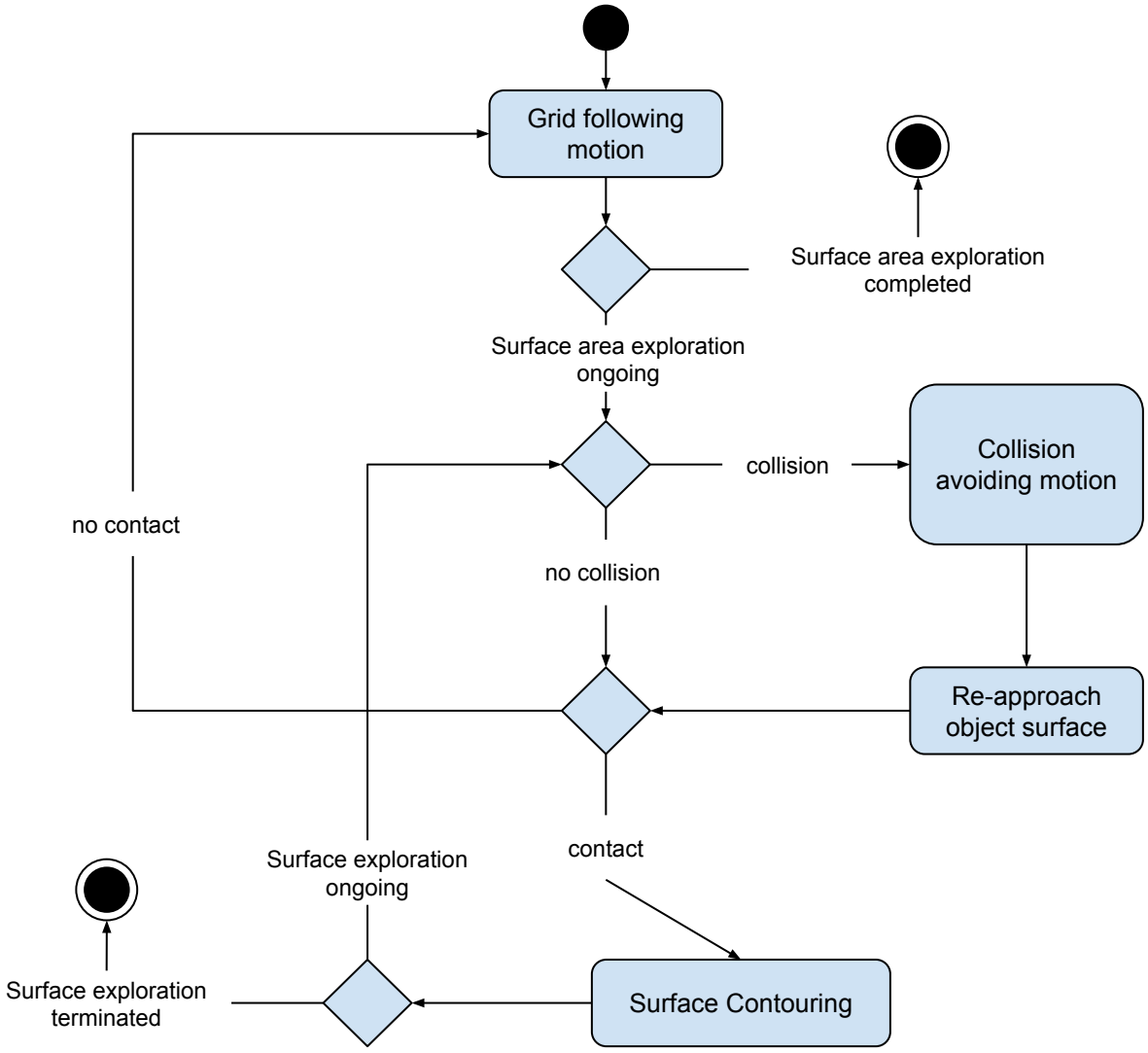


Figure 5.8: This simplified UML activity diagram describes the core ideas of the implemented surface exploration nodes. Starting with a grid following motion to cover the exploration area, the motion is aborted in the case of a collision threat or the case of a contact to the probe. In the case of a collision threat, a collision avoiding motion according to the specific surface exploration node is conducted. In the case of a contact situation between the sphere of the probe and the object, a surface contouring motion is started and continued until the object surface is lost or the exploration is terminated due to the object being outside the exploration area. Is the contact to the surface lost, a reapproach process is executed to reenter in contact with the object. Does the re-approach process fail, the motion strategy of exploration area covering via grid motion is executed. Once the exploration area has been covered, the surface exploration is terminated.

`fcl_collision_detection.cpp` The collision detection node heavily uses the FCL (cf. section 3.7). The node aims to check for collisions between objects. For reasons of real-time ability of collision detection, the objects taken into account for collision checking are represented only by the links near to the sphere of the probe. Additionally considered for collision checking are the occupied voxels of the octoMap being send from the `surface_contour` node via the topic `/octoMap`. Thereby the occupied voxels are transferred to collision objects represented by a cube. The size of the cubes can be adapted by changing the octree size of the octoMap. Furthermore, to check for collisions between objects, the collision objects represented by the robot links can be visualized in rviz via markers. The markers are published via topics to the rviz node.
The `fcl::distance()` method approximates the distance between single collision objects or arrays of collision objects. The minimal distance between the objects is compared to a threshold value. Is the threshold value undershot by the determined minimal collision distance, a `collision_flag` is set to true and published via the topic `collision_flag`.
Knowledge about the kinematic state of the robot is of great importance when collision checks are executed, and therefore with every ROS loop calling the `collision_check()` method of the node, the information about the kinematic state of the robot is updated via the forward kinematics functionality of MoveIt! (cf. section 3.10.5). Additionally, to publishing the `collision_flag`, further information is published on the same topic. As the `fcl::distance()` function determines a collision pair representing the two points on collision objects with the minimal distance to each other, the information on those points is used to determine the relative position of the probe to the collision point generated by an occupied octoMap voxel. This information can then be used to reorient the probe to avoid a collision. Information on the real-time ability of the collision detection can be gained in 6.1.

`ros_fri.cpp` The `ros_fri` node is the connecting link (cf. section 5.3.1) between the RML, FRI and nodes using those libraries to control the KUKA LWR IV such as the surface_contour node. Furthermore, the node reads and enforces robot joint limits, velocity, acceleration and effort limits.
The functionality of the FRI library is provided over the `ros_fri` node including information about the state of the robot including joint positions and drive temperatures as well as the ability to set motion goals to the FRI of the KUKA LWR IV. The communication between the node and other ROS nodes is realized over topics. The topic of `/lwr/RMLFinalStateReached` has been additionally implemented to communicate that a final motion goal position has been reached.

`surface_contour.cpp`, `surface_contour_orientation_change.cpp`,`grid_version.cpp` The three versions of surface exploration nodes as can be understood from section 5.3.2 share most of their core functionality except for the approach to avoid collisions and the subsequent motion strategy. Therefore the following description of the functionality with the aid of the implemented core methods is valid for all the surface exploration nodes and a distinction is made for the collision avoidance.
It is to be understood (cf. sections 4.2.4, 4.2.5) that the motion of the probe is always conducted along one axis of the world coordinate frame (WCF) at a time in the case of no contact to an object and along two axes of the world coordinate frame in the case of contact to an object to

follow its surface. Furthermore, if in the following descriptions a position goal is determined or published, the forward and inverse kinematic services of MoveIt! (cf. section 3.10.5) are used to transfer Cartesian coordinate frame points to joint position values of the KUKA LWR IV.

`setMotionState()` Three motion states namely`X_Y`, `Z_Y` and `TO_FIRST_CONTACT` exist. If the `X_Y` motion state is active, motion is conducted on the x-axis of the WCF and in contact situation additionally, motion on the y-axis is executed if necessary or wanted for surface contouring. In the case of `Z_Y` motion is conducted on the y-axis of the WCF and in contact situation motion along the z-axis can additionally be executed. In the case of `TO_FIRST_CONTACT` the motion is not limited to one or two axes, but motion is directly conducted towards the first contact point experienced during the process of surface exploration.
Via the method `setMotionState()`, the desired motion state can be selected.

`estimatedExternalTCPWrenchCallback()` The callback on `estimatedExternalTCP -WrenchCallback()` processes the wrench published via the `/wireless_ft/wrench_3` topic (cf. figure 5.4) and holds ready the current values to be used in the node.

`forceToWCFRotation()` The `forceToWCFTransform()` method rotates vectors of the force/-torque sensor frame to the world coordinate frame and therefore eases the dealing with forces expressed in the sensor frame such as the wrench published over the `/wireless_ft/wrench_3` . Since the frame of the force/torque sensor has the same orientation as the frame of the sphere the rotation is also used in other parts of the node e.g. in the `buildOctomap()` method to rotate contact points expressed in the sphere frame to be expressed relative to the WCF.

`pointOnSphere()` In the case of a contact between the sphere and an object, the `pointOn Sphere()` method determines the position of the contact point expressed in the sphere frame. The calculation on how to determine the contact point on the sphere surface is described in equations of 4.1, 4.2 and 4.3.
The point on the sphere surface is forwarded to the `buildOctomap()` method.

`buildOctomap()` The `buildOctomap()` method takes the contact points determined by the `pointOnSphere()` method and fills them into an octoMap after transforming them to the WCF. Hence every contact point is entered in an octoMap which is published on the `/octoMap` topic. The so built up octoMap represents the points in space on where an object has been touched. At the end of the surface exploration, the final octoMap is written to a binary file to be later processed by surface reconstruction algorithms. An octoMap representing the contact points experienced in the exploration of the polystyrene object shown in figure 6.1 (f) is displayed in figure 5.9.

`generateSurfaceTangent()` This method is called when the probe is in contact with an object and generates the surface tangent using the current force values. The procedure of how to determine the object surface tangent can be understood in figure 4.2. Naturally, with regard to two dimensions, a tangent can be determined in two directions. Dependent on

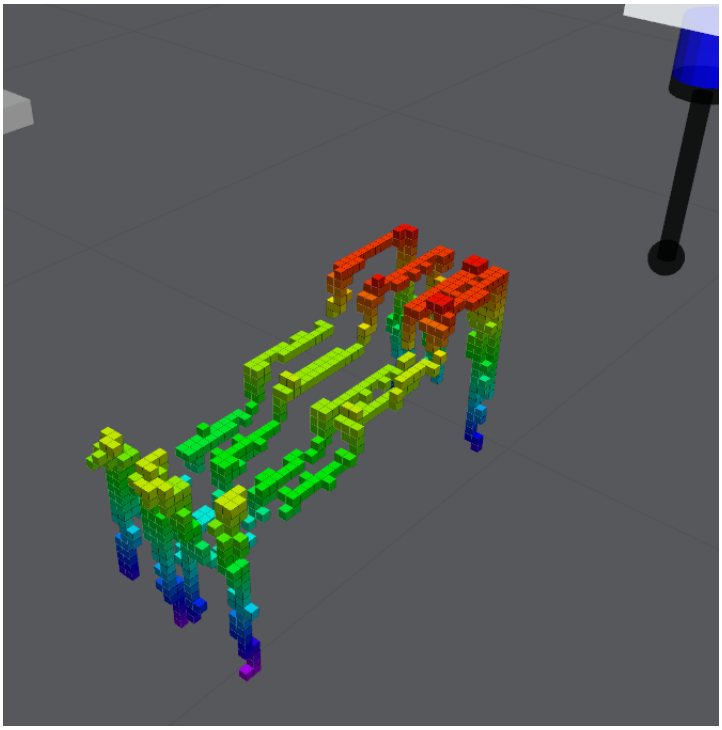


Figure 5.9: The octoMap represents the contact points experienced in the surface exploration of the polystyrene object shown in 6.1 (f). The `grid_version` surface contour node has been used to explore the object and the side length of the cubes representing the occupied nodes have a side length of 2 mm. The height of the nodes is color encoded. The purple cubes denote the lowest and the red nodes the highest height.

the motion direction of the superordinate grid to cover the exploration area combined with the motion state, the wanted direction of the tangent is generated.

`collisionFlagCallback()` The data sent over the /collision_flag topic is processed by the `collisionFlagCallback()` method. Depending on the collision avoidance strategy of the nodes, a new motion state is set, surface contouring aborted or an orientation change of the probe is initiated.

`reapproachObject()` The `reapproachObject()` method aims at moving the sphere back to the object if the contact has been lost. Figure 4.9 in the concept chapter shows how to reapproach the object by moving in the opposite direction of the last sensed contact force. Is no contact reinitiated after a certain motion distance, the reapproach motion is aborted and motion is conducted according to the exploration area grid covering strategy.

`exploration()` The `exploration()` method is called in a ROS loop as long as the exploration is not finished checking whether a certain threshold force is acting on the force/torque

sensor. Once the threshold value is overstepped, either the `contourTracking()` method is called, or if the force value is overlarge, immediately a position goal in the direction of the force acting on the sphere is published to release the forces acting on the probe respectively the object. In the case of no contact, position goals to cover the exploration area according to the motion state are published and received by the `ros_fri` node.

`contourTracking()` The `contourTracking()` method called when a force value threshold is overstepped, gathers information on the object surface tangent direction via the `generate SurfaceTangent()` method and publishes the according joint position goal. Additionally, the `pointOnSphere()` method is called which then calls `buildOctomap()` as the execution of `contourTracking()` denotes the contact to an object.

`publishPositionGoal()` The purpose of the `publishPositionGoal()` method is to publish the determined joint position goals on the `/lwr/jointPositionGoal` topic in the expectation to lead to the desired motion of the KUKA LWR IV.

Collision avoidance strategies of the surface_exploration nodes The strategies of the nodes concerning collision avoidance are illustrated in the figures 5.10, 5.11, 5.12 and the descriptions can be obtained from the corresponding captions. The collision avoidance strategy of the `grid_version` node assumes that a collision threat only arises in surface contour motion in consequence of driving against the object with the force/torque collision model. This is usually the case when the object is longer on the z-axis than the stick and sphere of the probe combined (cf. figure 4.10 (a)). Keep in mind that the orientation of the probe does not alter in the nodes of `grid_version` and `surface_contour`. The changes in orientation of the `surface_contour _orientation_change` node selects between two quaternions describing the goal orientation of the probe to reach in order to be able to continue the contour motion along the object. The quaternions are chosen to make the probe point towards the object to gain space between the collision objects of the force/torque sensor, the holder, and the object.

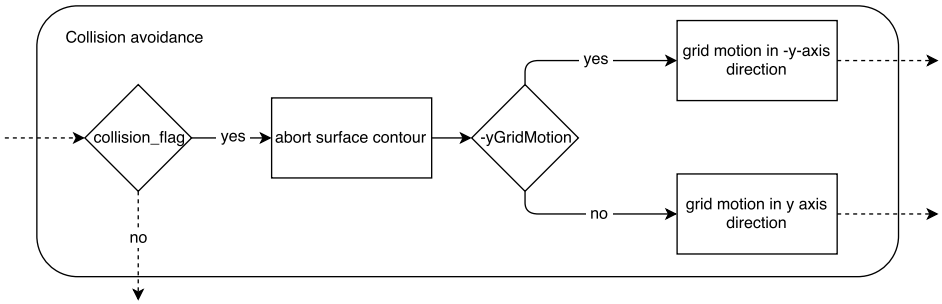


Figure 5.10: The collision avoidance strategy of the `grid_version` node is to abort a surface contour motion in the case of a collision threat and to return to the grid covering motion strategy. The superordinate grid motion direction to cover the exploration area is maintained.

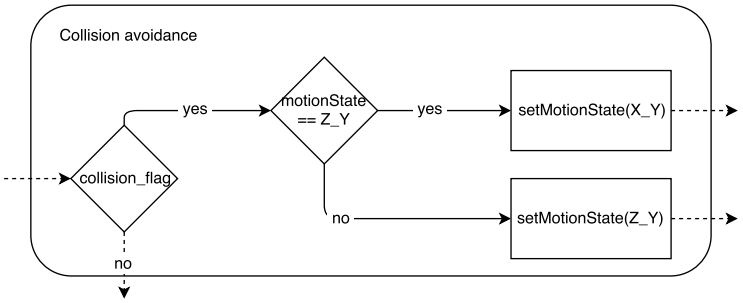


Figure 5.11: The collision avoidance strategy of the `surface_contour` node changes in the case of a collision threat the motion state from Z_Y to `X_Y` or vice versa but the object is continuously contoured. The change on the motion state causes motion on other axes to circumvent the collision. Additionally an object is contoured not only along two axes but on three if a collision threat occurs. This can result in a more detailed picture of the object expressed in the octoMap.

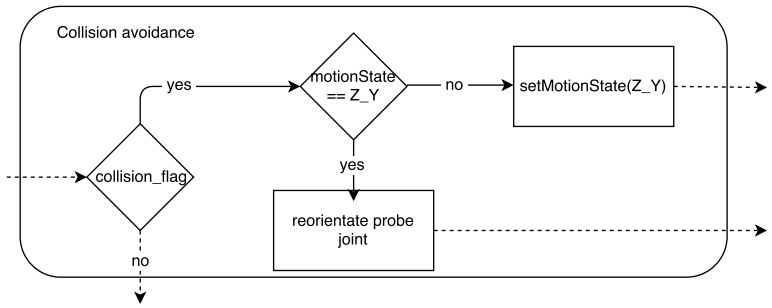


Figure 5.12: The collision avoidance strategy of the `surface_contour_orientation_change` differs based on the motion state. Is the motion state not `Z_Y` in the case of a collision threat, the motion state is set to `Z_Y` and similar to the strategy in the `surface_contour` node, the collision threat is attempted to be taken off via motion on other axes. On the other hand, if the motion state is `Z_Y`, the orientation of the probe is changed to cope with the possible collision (cf. figure 4.11).

5.4 Surface Reconstruction

The surface reconstruction aims to reconstruct the shape of the explored object via building a polygon mesh. The general pipeline to transform point clouds to polygon meshes is similar for all three approaches and are mentioned in the chapter on the fundamentals in the section on surface reconstruction 2.6 and the chapter of the used setup in section 3.8 regarding the Point Cloud Library (PCL).
The octoMap library provides a tool to comfortably convert octoMaps to point clouds called `octree2pointcloud`.

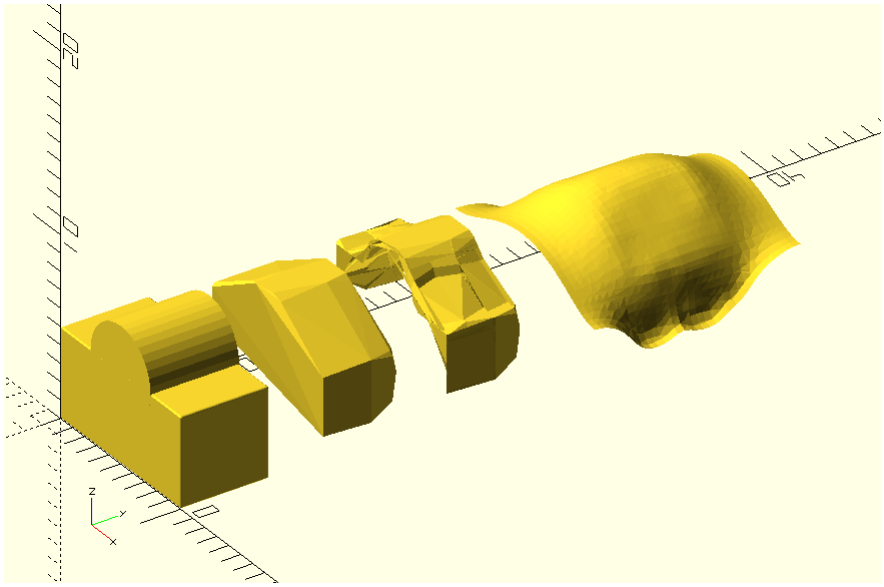


Figure 5.13: The reconstructed shapes via the three reconstruction approaches based on the same object and point cloud are shown. From left to right, the first object is a model describing the shape of the object. The second is the reconstructed surface of the convex hull approach followed by the concave hull approach. The right most surface is the result of the Poisson surface reconstruction.

The convex hull as well as the concave hull of the generated point clouds corresponding to the explored objects are determined to reconstruct their shape. PCL provides methods to do so. Furthermore, another approach called Poisson surface reconstruction (cf. section 2.6.3) is executed. The following descriptions show the pipeline of the programs executing the reconstruction and parameter for the approaches. The figure 5.13 shows examples of reconstructed shapes via the three reconstruction approaches based on the same object and initial point cloud.

`surface_concave_hull.cpp` The `surface_concave_hull` node takes as input a point cloud file in `.pcd` format and outputs a polygon mesh in `.stl` format. The input point cloud is set as input cloud for a `pcl::ConcaveHull` object. The concave hull point cloud type has among others the parameters of dimensionality and alpha. The dimension is set to 3 to generate a 3D concave hull and the alpha value is set to 0.015 describing the maximum size of the resultant hull segments. Next, the reconstructing method is called, and the resulting polygon mesh is saved to the output file.

`surface_convex_hull.cpp` The `surface_convex_hull` node takes again as input a point cloud in .pcd format and outputs a polygon mesh in `.stl` format. The input cloud is loaded into a `pcl::ConvexHull` object. Likewise to the `surface_convex_hull` approach, the dimension is set to 3. Then the reconstruct method is called, and the resulting polygon mesh is saved to the declared output file.

`poisson.cpp` The Poisson surface reconstruction node takes as input a point cloud file in `.pcd` format and outputs a polygon mesh in `.stl` format. As the approach of Poisson surface reconstruction needs surface normals associated with the points of the point clouds to be applicable and to improve the result of the approach the input cloud is processed. First, the input cloud is set as input cloud to a `MovingLeastSquares` object of PCL and a method to upsample the point cloud as well as computing the normals is executed. Then the resulting point cloud is processed by a `NormalEstimationOMP` object of PCL to determine curvature information of each 3D point. The resulting point clouds generated by the `MovingLeastSquares` object and the `NormalEstimationOMP` object is then concatenated and processed by a PCL Poisson object to finally reconstruct the polygon mesh. Then the polygon mesh is saved in the output file.

6 Evaluation

The evaluation of the implemented concepts on surface exploration and reconstruction discusses performance characteristics of the implemented and used ROS nodes as real-time ability is crucial for the execution of the approaches. In the section on object descriptions (cf. 6.2.1) the explored objects are presented which mostly represent simple polyhedra. The point clouds corresponding to the objects and version of surface contour node are shown in 3D and discussed in section 6.2.2. The shape of the cylinder and block object have been modeled in 3D and plotted together with the corresponding point clouds to better visualize the accuracy of the location of the points on the object surfaces.
The results of the surface reconstruction methods are shown and discussed in section **??**. Thereby the results of the approaches of concave and convex hull generation are compared. The Poisson surface reconstruction approach, in contrast, performs worse in the accuracy of reconstructed surface.

6.1 Real-Time Ability of Position Goal Determination and Collision Detection

The ability to react instantaneously to unforeseen events such as contact to objects in blind exploration or blind motion is of utmost importance and has been elaborated in the *Introduction* chapter of the thesis (cf. 1). The process to determine new position goals via the nodes of the surface exploration needs to be real-time, as well as the parallel running collision detection. Collision detection remains a computationally hard problem for motion planning ([32] p. 139) in robotics and therefore the duration of the applied FCL method is elaborated. The nodes of `collision_detection`, `ros_fri` and the three versions for surface exploration are permanently running in loops upon start-up permitting high frequencies. Tables 6.1 and 6.2 show the duration of one loop iteration. Thereby the duration between the surface exploration loops (`grid_version`, `surface_contour`, `surface_contour_orientation_change`) do not distinguish in their loop times and are subsumed under the term of `surface_exploration` in the table.
For the measurement of the loop iterations and the duration of the collision checks, `ros::Time::now()` has been used for the generation of time stamps at the beginning and end of a loop iteration. The subtraction between the two points in time yield the duration of the processes.

The flexible collision library (FCL) provides different possibilities to store collision objects in data structures and call collision detection methods on them. The usage of simple collision shapes (e.g., boxes, cylinders) in comparison to detailed meshes and the application of `BroadPhaseCollisionManager` objects to avoid $O(n^2)$ complexity [53] in collision checking

Table 6.1: The duration of one loop iteration is shown for the different nodes in the ROS computation graph expect the collision checking node, which is in table 6.2.

Node	Measured loop iteration duration in ms
wireless_ft	0.2 - 7
ros_fri	10
surface_exploration	8.4

Table 6.2: The duration of the collision checking method is shown in this table. The volume of the nodes with 8 mm^3 corresponds to a cube with side length of 2 mm and the 125 mm^3 corresponds to a cube with side length of 5 mm.

Number of Occupied Octree nodes	Duration in ms on Nodes with Volume of $8mm^3$	Duration in ms on Nodes with Volume of $125mm^3$
10	0.3 - 0.5	0.5 - 1
100	0.4 - 1	0.8 - 1.3
1000	0.9 - 1.6	1.3 - 3
10000	8 - 13	9 - 14

has been considered to be the least time expensive approach. Applied to collision checking in the case of surface exploration, two collision managers hold the collision objects represented by the occupied nodes of the generated octoMap (cf. figure 4.13). Moreover, the collision objects representing the shapes of the robot links in the neighborhood of the probe (cf. figure 4.12) are hold by the collision managers. Only those links have been considered in collision checking to ensure real-time ability. More and bigger respectively more detailed bodies considered for collision checking result in longer execution times. The collision check calculation defines the duration of the `fcl_collision_check` node loop.

The number of occupied octree nodes generated during surface explorations has not exceeded 1000 nodes and therefore should not have been a limiting factor to the real-time ability for the determination of motion goals. It can be stated that the durations of the used nodes do not exceed 10 ms and should provide real-time ability. The evaluation of further potentially limiting factors as limits in kinematics and dynamics as well as the communication delays between the functional modules describing the hardware setup is outside of the scope of this thesis.

6.2 Accuracy in Reconstructed Object Surfaces

6.2.1 Object Descriptions

The objects utilized for the exploration and reconstruction process are simple wooden polyhedra with smooth faces, rounded edges, and symmetries. Convex, as well as concave polyhedra, have been taken into account. Additional to the wooden polyhedra, slightly more complex objects made of polystyrene have been explored.
The figures in 6.1 show photos of the explored and reconstructed objects. The block has a width

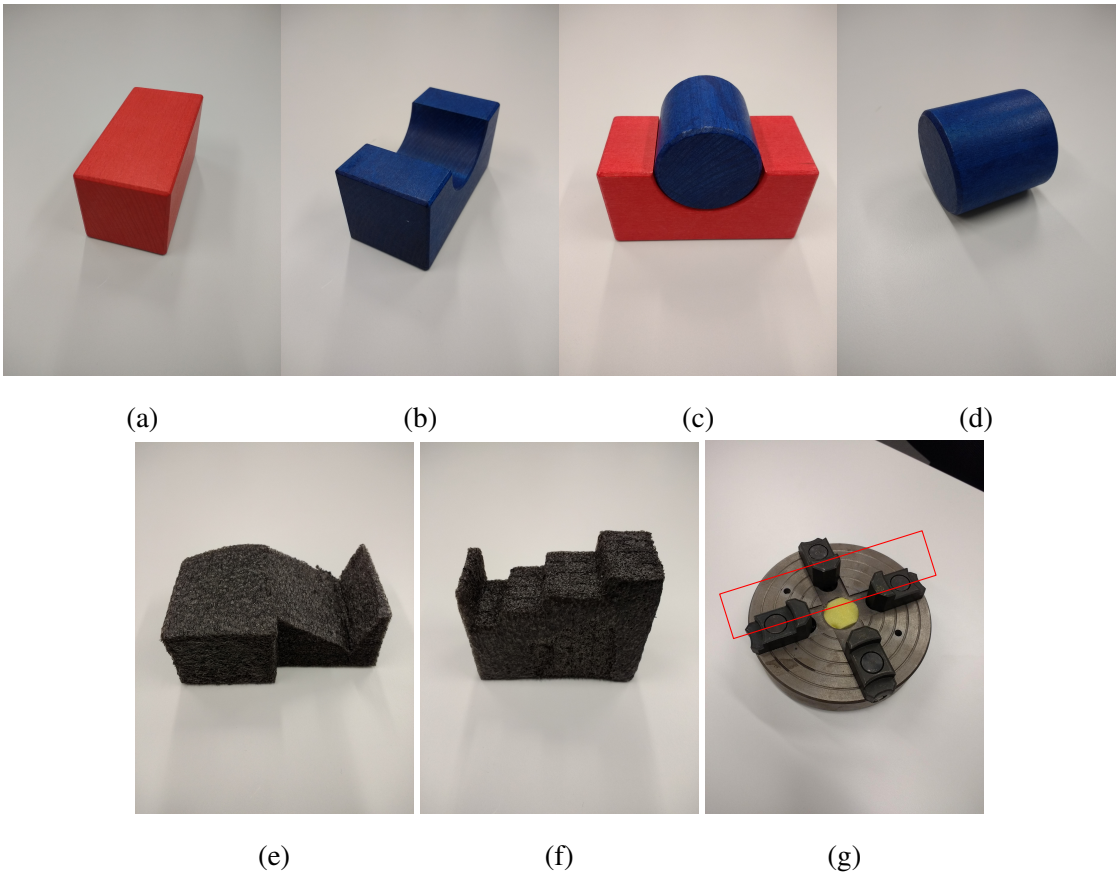


Figure 6.1: The different objects used for surface exploration and reconstruction are represented. The figures from (a)-(d) are wooden while in (e) and (f) polystyrene objects are shown. A simple block is shown in (a) and a block with bulge is shown in (b). (d) shows a cylinder which is put in the bridge shown in (c). Two more complex objects are shown in (e) and (f). The selected area of the jaw chuck shown in (g) describes its explored area.

height and depth of 5 cm and a width of 10 cm. The cylinders radius is 2,4 cm and has a height of 5 cm. The bulge on the block in (b) and (c) has the volume of half the volume of the cylinder in (d).

6.2.2 Generated Point Clouds

OctoMaps have been recorded during the process of surface contouring. The contact positions on the sphere of the probe in contact with an object have been taken to locate the nodes of the octoMap. Then they have been transformed to point clouds via the `octree2pointcloud` tool. The figures 6.2-6.20 represent the generated point clouds based on the three surface contour nodes and are shown on the upcoming pages. The block and cylinder objects are modelled ac-

cording to their metrics and plotted together with the point clouds to visualize the conducted contouring motion better. The figures show the modelled objects with transparent as well as non-transparent faces to indicate points located on object boundaries or within the boundaries and behind the object model. Points only partially shown are located on the boundaries of the object models.
All objects have been explored with an octomap node size of 2 mm and 1,5 cm x-axis motion distance at exploration area boundaries (cf. figure 4.5) for the surface exploration nodes of `surface_contour` and `surface_contour_orientation_change` and with 1 cm motion distance for the `grid_version` node.
The generated point clouds of the `surface_contour_orientation_change` resemble the point clouds generated by the grid_version, thus also the reconstructed surfaces. The difference in both approaches is the collision avoidance strategy. While surface contouring is aborted in the case of a collision threat with the `grid_version` node, the orientation change to handle the situation allows the continuation of the surface contouring (cf. figure 4.11). Exemplary figure 6.4 shows the generated point cloud corresponding to the wooden block by the `surface_contour_orientation_change` node. One block has been stacked on top of another block to illustrate the benefit of the orientation change.
Merging the resulting point clouds of multiple surface exploration processes on the same object results in more points describing the shape. Figure 6.17 shows exemplary four merged point clouds encoded in different colors.

6.2.3 Comparison between Surface Reconstruction Objects

The generated point clouds have been used to reconstruct the surfaces of the explored objects. The performance of the approaches differs strongly. The approach of Poisson surface reconstruction appears to be unqualified for the prevalent conditions of sparse data in comparison to point clouds generated by cameras. The result of a Poisson surface reconstruction is shown in figure 5.13 representing the surface on the right. As the results of the approach for the other objects turned out to look similar to this reconstruction and do not represent much resemblance to the objects, they are not shown in the following object surface reconstruction figures.

6.2.4 Surface Reconstruction Objects of Convex and Convave Hull Reconstruction

The reconstructed surfaces naturally differ to the metrics of the objects due to the superordinate grid following strategy to cover the exploration area. The x-axis motion of the boundaries can lead to failure to observe the objects ends with regard to the x-axis. Additionally, the objects are located on a table and can therefore not be contoured at the underside.
Evaluating the surface reconstruction approaches of the convex hull and concave hull generation, no best solution is to be found. The accuracy of the reconstructed surfaces depends on the explored objects. The simple shapes of the block (6.21, 6.22) and cylinder (6.28, 6.29, 6.30, 6.31) are best reconstructed via the convex hull approach as they themselves are defined by a convex hull. The other objects, the bridge, combination of block and cylinder, the polystyrene

objects as well as the object slice tend to be better reconstructed with the concave hull approach. The concave hull approach allows in contrast to the convex hull approach to represent the shape of non-convex shaped objects. The result of convex hull surface reconstruction neglects shape characteristics specifically shown in figures representing the bridge object reconstruction (cf. figures 6.23, 6.25) and the polystyrene object shapes shown in 6.32 and 6.33. Multiple surface explorations have been completed to get different point clouds describing the bridge (6.17). Merging the points to one point cloud results in less space between the single points and more coverage of the shape surface. This allows more precise and robust surface reconstruction due to denser data and reduction on the dependency of single, possibly faulty, point clouds. However, even small changes in the location of the explored object result in shifted point clouds. The surface reconstruction shape based on merged point clouds is shown in figure 6.24.

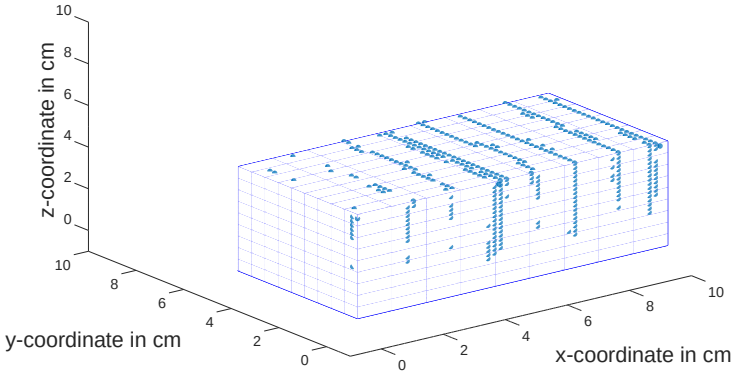


Figure 6.2: The point cloud corresponding to the surface exploration via `grid_version` surface contour node performed on the block object (cf. 6.1 (a)) is shown with a non-transparent model.

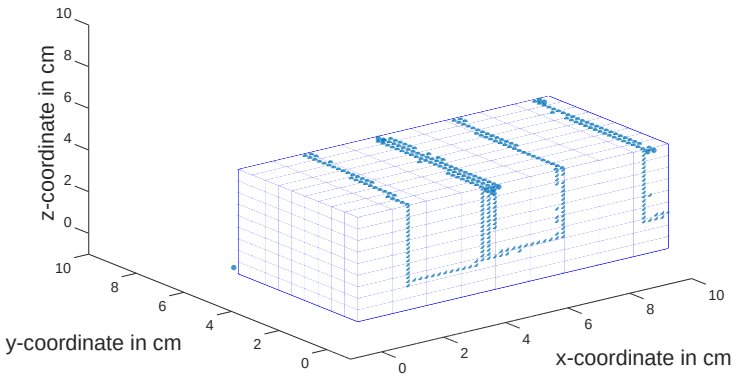


Figure 6.3: The point cloud corresponding to the surface exploration via `surface_contour` node performed on the block object (cf. 6.1 (a)) is shown with a non-transparent model.

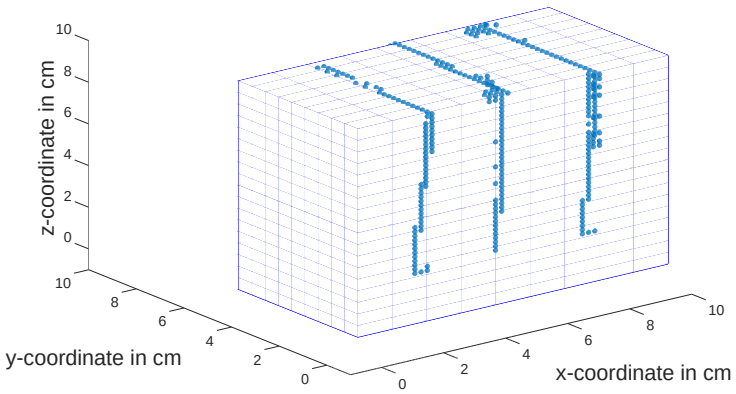


Figure 6.4: The point cloud corresponding to the surface exploration via `surface_contour_orientation_change` node performed on two block object stacked on top of each other (cf. 6.1 (a)) is shown with a non-transparent model. The points of the point cloud do not reach the bottom of the stacked blocks as the exploration area has been set to be situated above the table shown in figure 3.1.

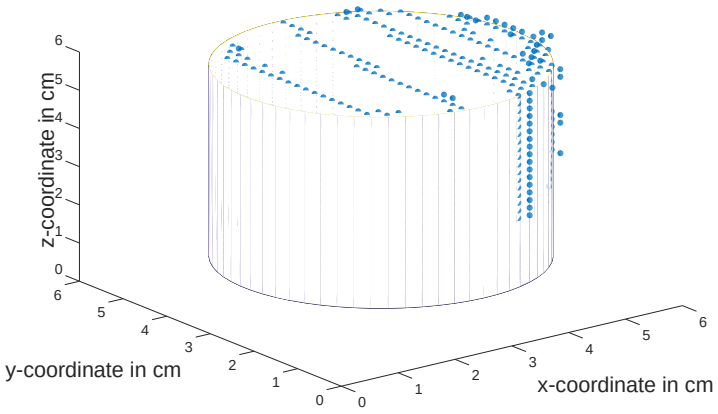


Figure 6.5: The point cloud corresponding to the surface exploration via `grid_version` surface contour node performed on the vertical cylinder object (cf. 6.1 (d)) is shown with a non-transparent model. Some measured points lie slightly inside the cylinder and are not visible (cf. 6.6)

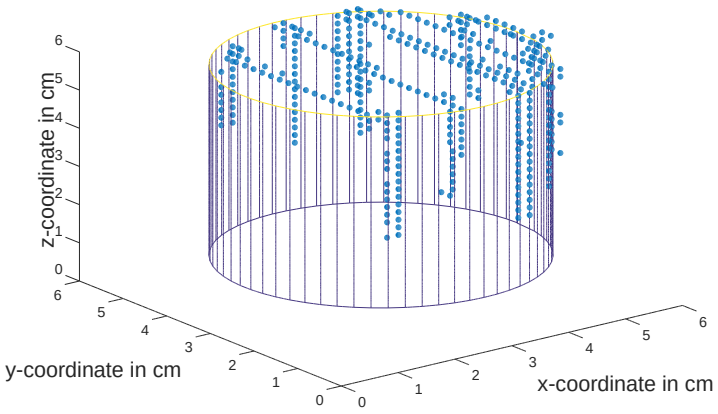


Figure 6.6: The point cloud corresponding to the surface exploration via `grid_version` surface contour node performed on the vertical cylinder (cf. 6.1 (d)) object is shown with a transparent model.

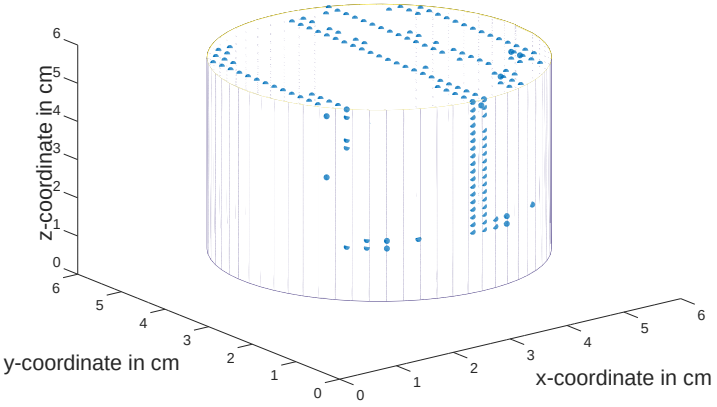


Figure 6.7: The point cloud corresponding to the surface exploration via `surface_contour` node performed on the vertical cylinder object (cf. 6.1 (d)) is shown with a non-transparent model. Some measured points lie slightly inside the cylinder and are not visible (cf. 6.8)

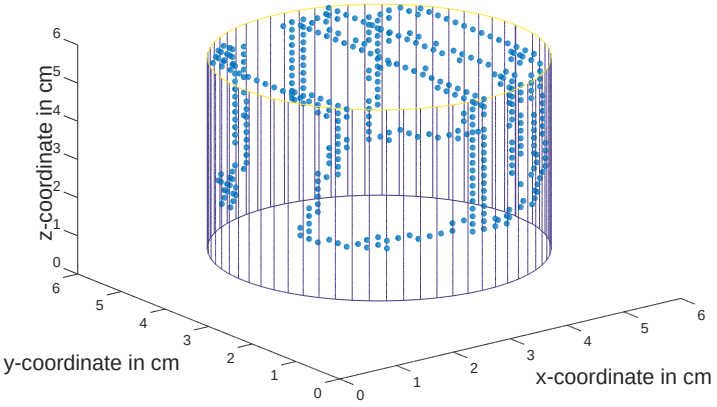


Figure 6.8: The point cloud corresponding to the surface exploration via `surface_contour` node performed on the vertical cylinder object (cf. 6.1 (d)) is shown with a transparent model.

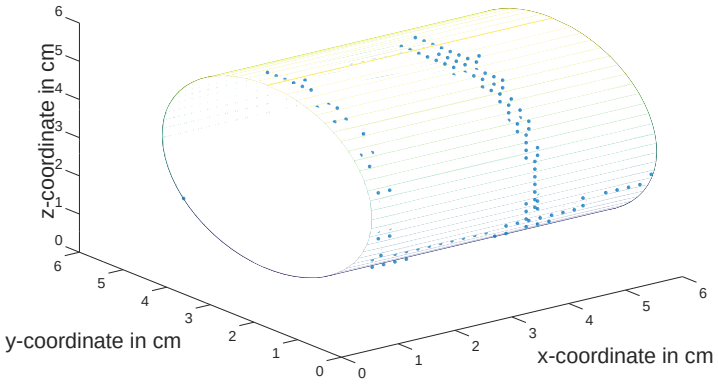


Figure 6.9: The point cloud corresponding to the surface exploration via `surface_contour` node performed on the horizontal cylinder object (cf. 6.1 (d)) is shown with a non-transparent model. Some measured points lie slightly inside the cylinder and are not visible (cf. 6.10)

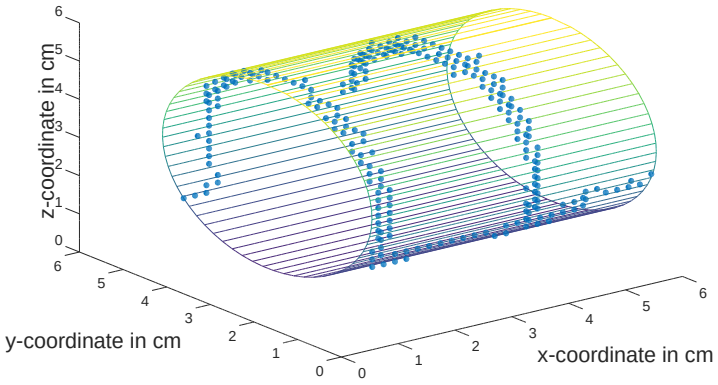


Figure 6.10: The point cloud corresponding to the surface exploration via `surface_contour` node performed on the horizontal cylinder object (cf. 6.1 (d)) is shown with a transparent model.

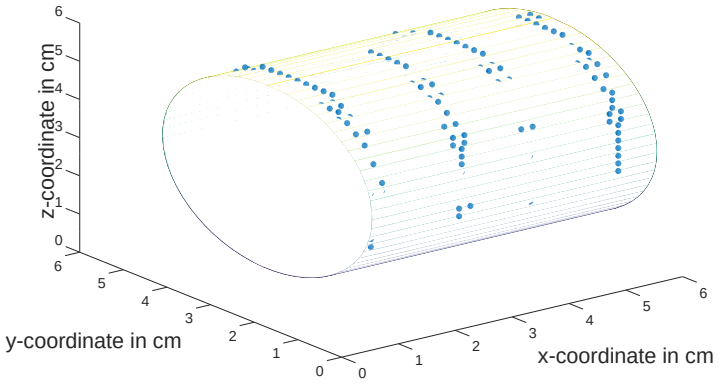


Figure 6.11: The point cloud corresponding to the surface exploration via T surface contour node performed on the horizontal cylinder object (cf. 6.1 (d)) is shown with a non-transparent model. Some measured points lie slightly inside the cylinder and are not visible (cf. 6.12)

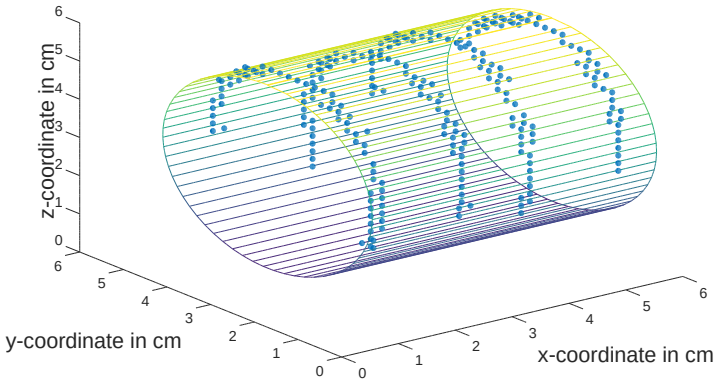


Figure 6.12: The point cloud corresponding to the surface exploration via `grid_version` surface contour node performed on the horizontal cylinder (cf. 6.1 (d)) object is shown with a transparent model.

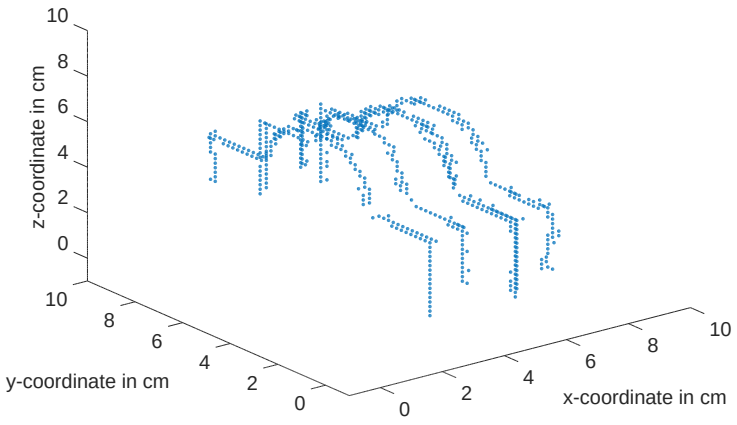


Figure 6.13: The point cloud corresponding to the surface exploration via `grid_version` surface contour node performed on the block and cylinder combination object (cf. 6.1 (c)) is shown.

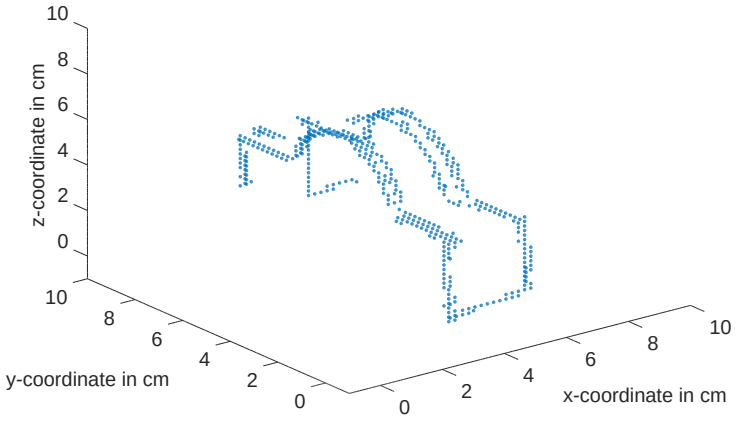


Figure 6.14: The point cloud corresponding to the surface exploration via `surface_contour` node performed on the block and cylinder combination object (cf. 6.1 (c)) is shown.

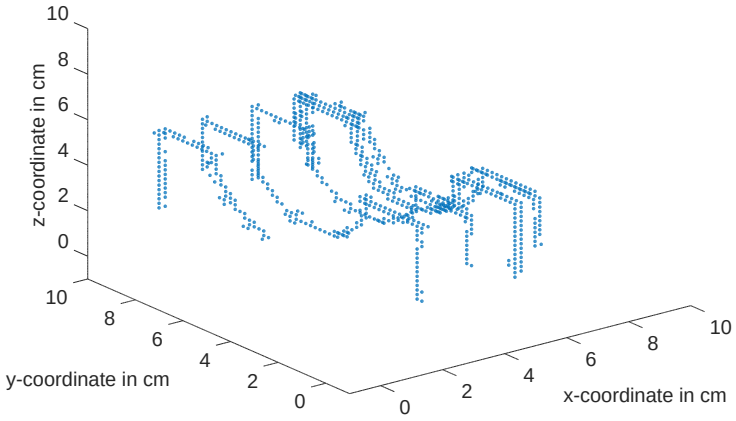


Figure 6.15: The point cloud corresponding to the surface exploration via `grid_version` surface contour node performed on the bridge object (cf. 6.1 (b)) is shown.

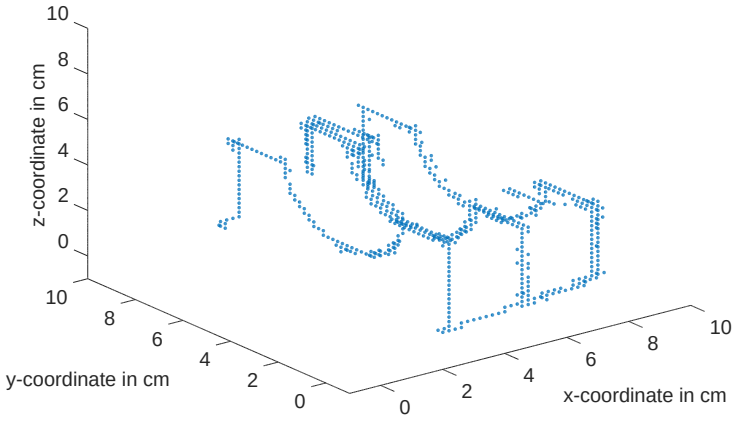


Figure 6.16: The point cloud corresponding to the surface exploration via `surface_contour` node performed on the bridge object (cf. 6.1 (b)) is shown.

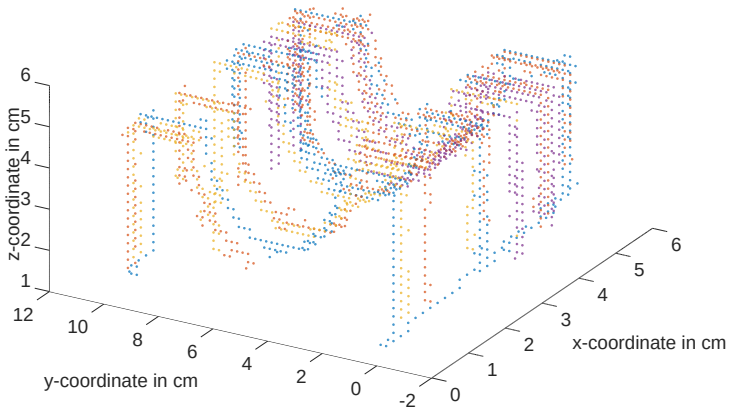


Figure 6.17: The merged point clouds corresponding to the surface exploration via `surface_contour` and `grid_version` node performed on the bridge object (cf. 6.1 (b)) is shown. The blue points correspond to the exploration via surface_contour node and the red, orange and purple point clouds to the `grid_version`.

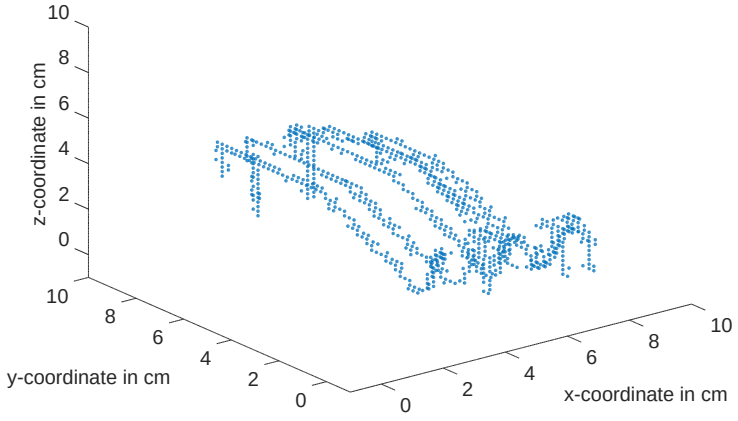


Figure 6.18: The point cloud corresponding to the surface exploration via `grid_version` surface contour node performed on the polystyrene object (cf. 6.1 (e)) is shown.

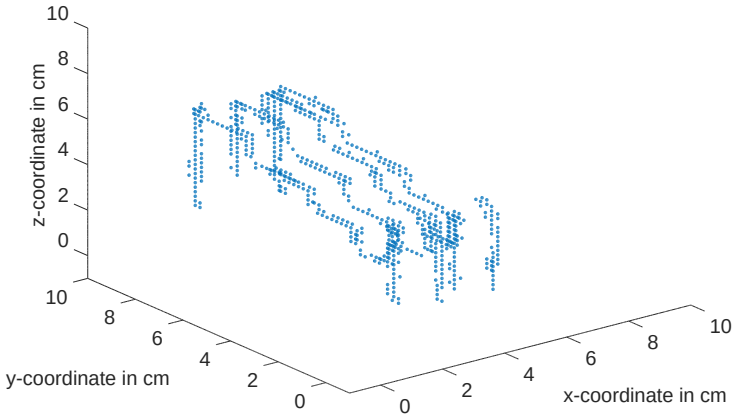


Figure 6.19: The point cloud corresponding to the surface exploration via `grid_version` surface contour node performed on the polystyrene object (cf. 6.1 (f)) is shown.

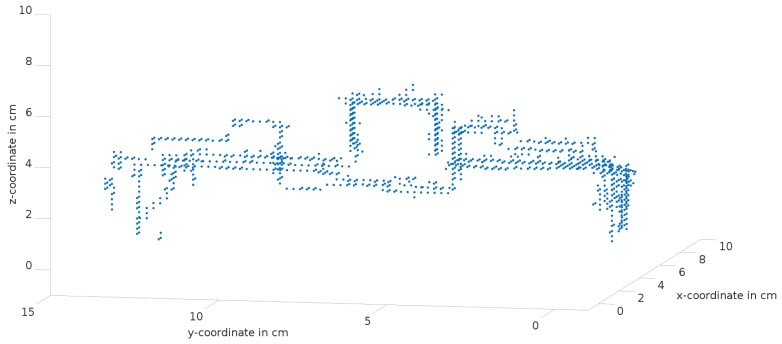


Figure 6.20: The point cloud corresponding to the surface exploration via `grid_version` surface contour node performed on the jaw chuck (cf. 6.1 (f)) is shown.

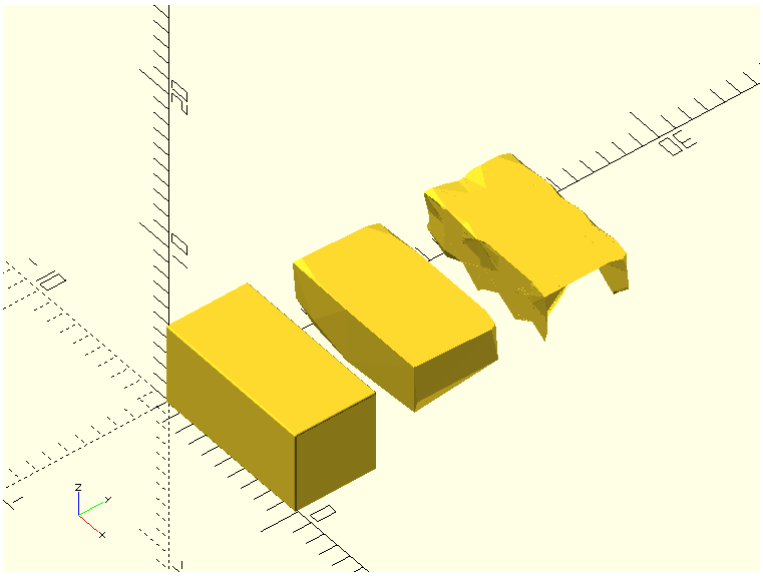


Figure 6.21: The reconstructed object shapes of the wooden block modeled on the left are shown. The object in the middle is the reconstructed object shape of the convex hull approach and on the right is the result of the concave hull approach. The reconstruction is based on the point cloud generated by the `grid_version` node.

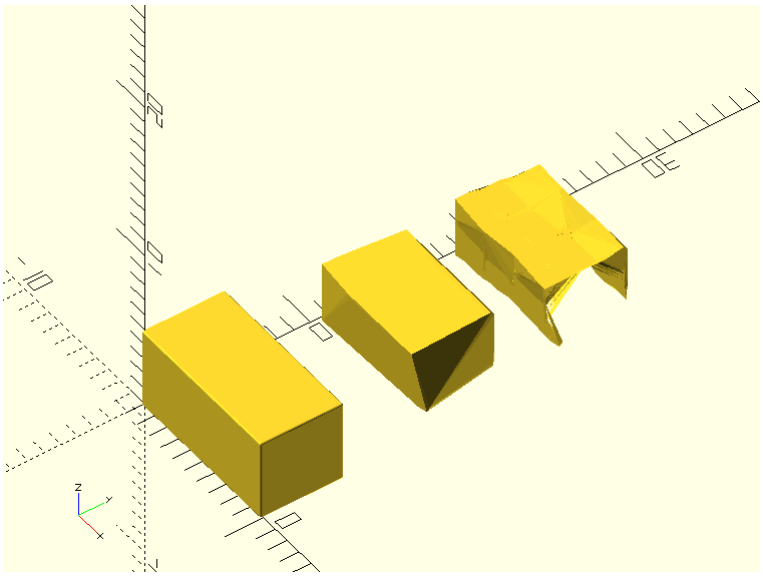


Figure 6.22: The reconstructed object shapes of the wooden block modeled on the left are shown. The object in the middle is the reconstructed object shape of the convex hull approach and on the right is the result of the concave hull approach. The reconstruction is based on the point cloud generated by the `surface_contour` node.

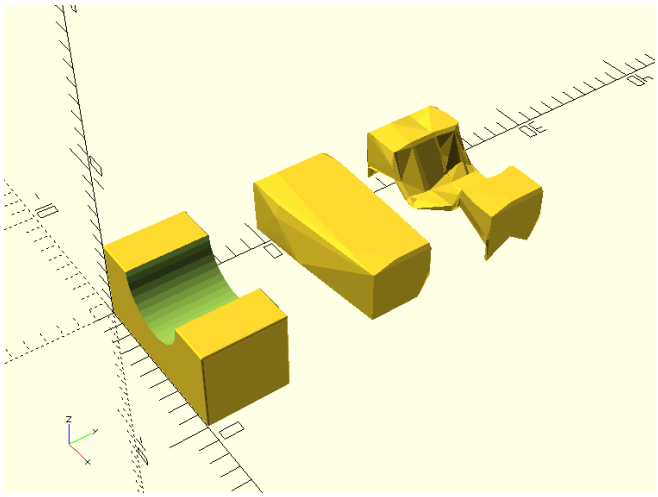


Figure 6.23: The reconstructed object shapes of the bridge object modeled on the left are shown. The object in the middle is the reconstructed object shape of the convex hull approach and on the right is the result of the concave hull approach. The reconstruction is based on the point cloud generated by the `grid_version` node.

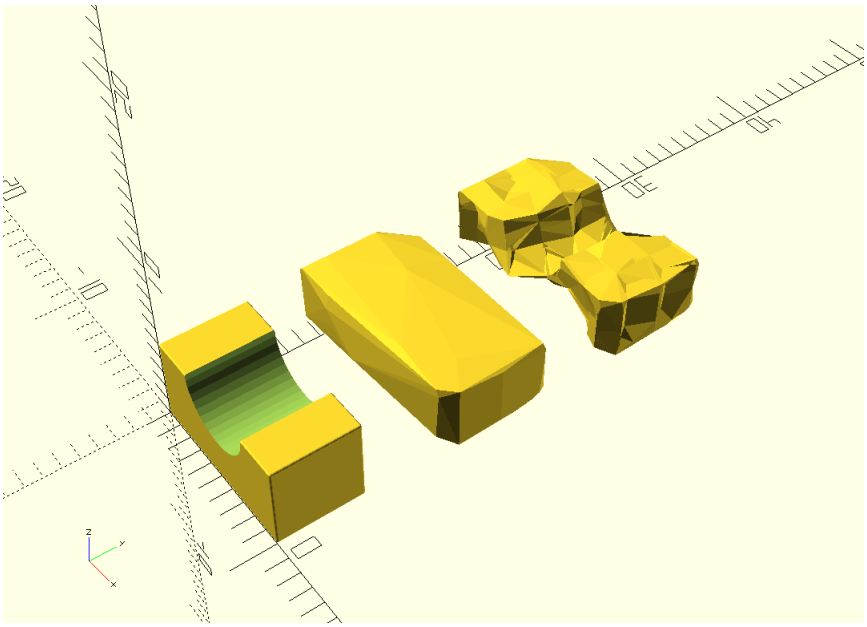


Figure 6.24: The reconstructed object shapes of the bridge object modeled on the left are shown. The object in the middle is the reconstructed object shape of the convex hull approach and on the right is the result of the concave hull approach. The reconstruction is based on the point clouds generated by the `grid_version` and `surface_contour` nodes shown in figure 6.17.

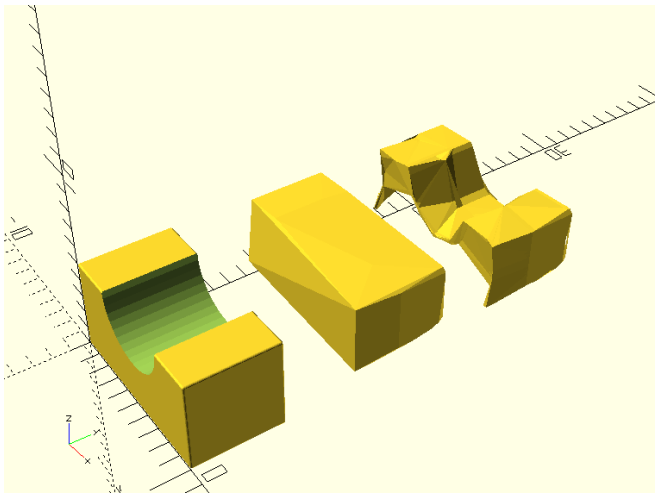


Figure 6.25: The reconstructed object shapes of the bridge object modeled on the left are shown. The object in the middle is the reconstructed object shape of the convex hull approach and on the right is the result of the concave hull approach. The reconstruction is based on the point cloud generated by the `surface_contour` node.

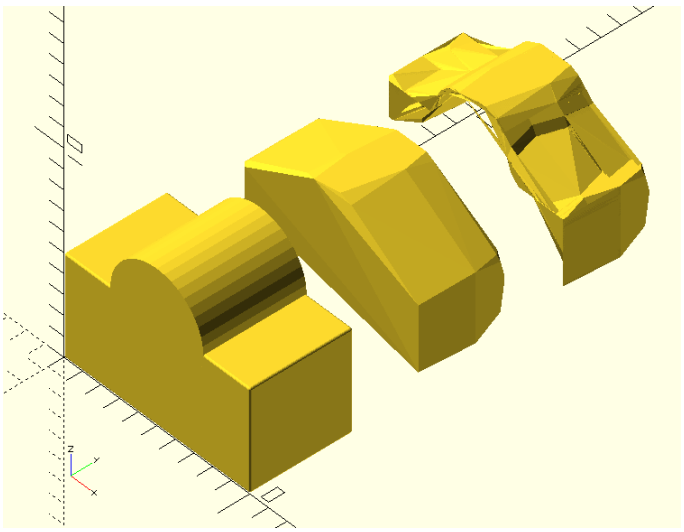


Figure 6.26: The reconstructed object shapes of the block/cylinder combination modeled on the left are shown. The object in the middle is the reconstructed object shape of the convex hull approach and on the right is the result of the concave hull approach. The reconstruction is based on the point cloud generated by the `grid_version` node.

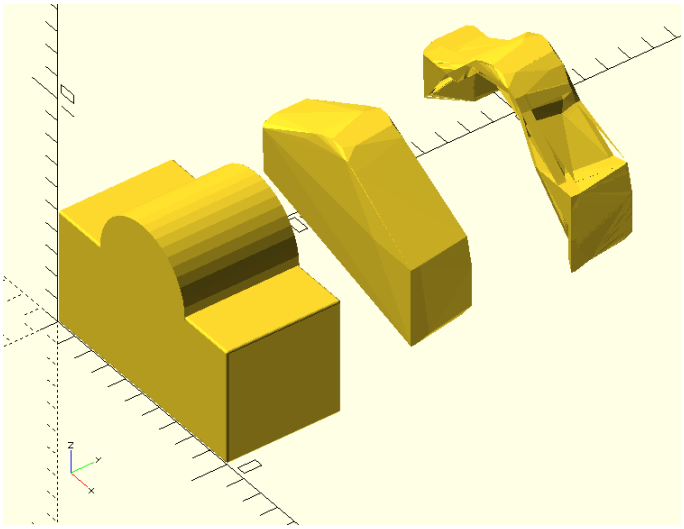


Figure 6.27: The reconstructed object shapes of the block/cylinder combination modeled on the left are shown. The object in the middle is the reconstructed object shape of the convex hull approach and on the right is the result of the concave hull approach. The reconstruction is based on the point cloud generated by the `surface_contour` node.

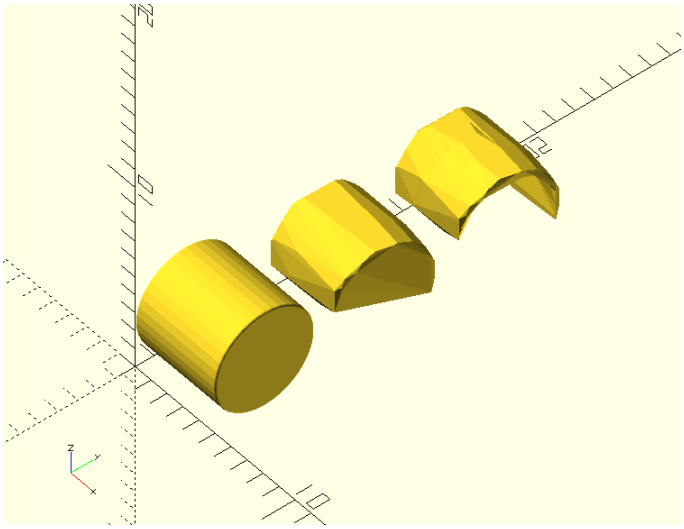


Figure 6.28: The reconstructed object shapes of the cylinder object (horizontal) modeled on the left are shown. The object in the middle is the reconstructed object shape of the convex hull approach and on the right is the result of the concave hull approach. The reconstruction is based on the point cloud generated by the `grid_version` node.

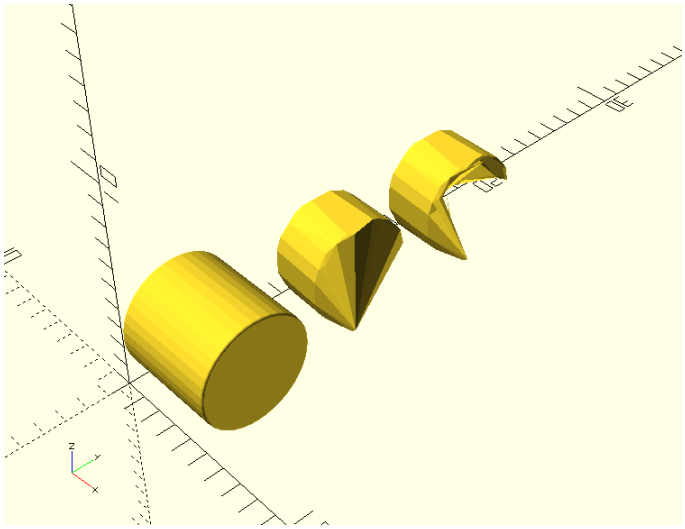


Figure 6.29: The reconstructed object shapes of the cylinder object (horizontal) modeled on the left are shown. The object in the middle is the reconstructed object shape of the convex hull approach and on the right is the result of the concave hull approach. The reconstruction is based on the point cloud generated by the `surface_contour` node.

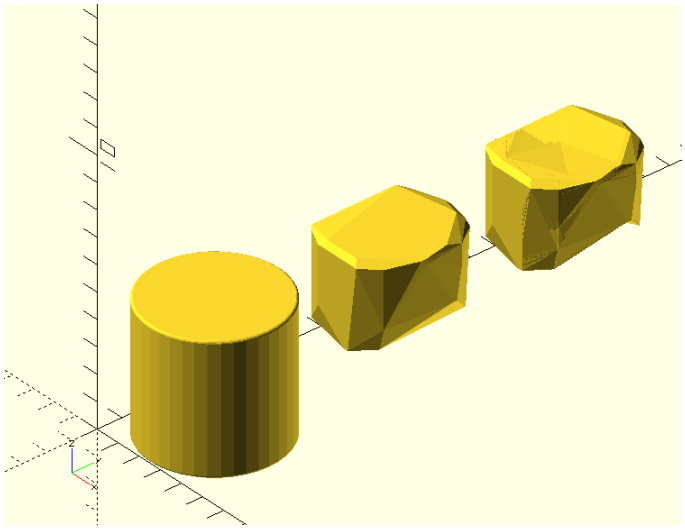


Figure 6.30: The reconstructed object shapes of the cylinder object (vertical) modeled on the left are shown. The object in the middle is the reconstructed object shape of the convex hull approach and on the right is the result of the concave hull approach. The reconstruction is based on the point cloud generated by the `grid_version` node.

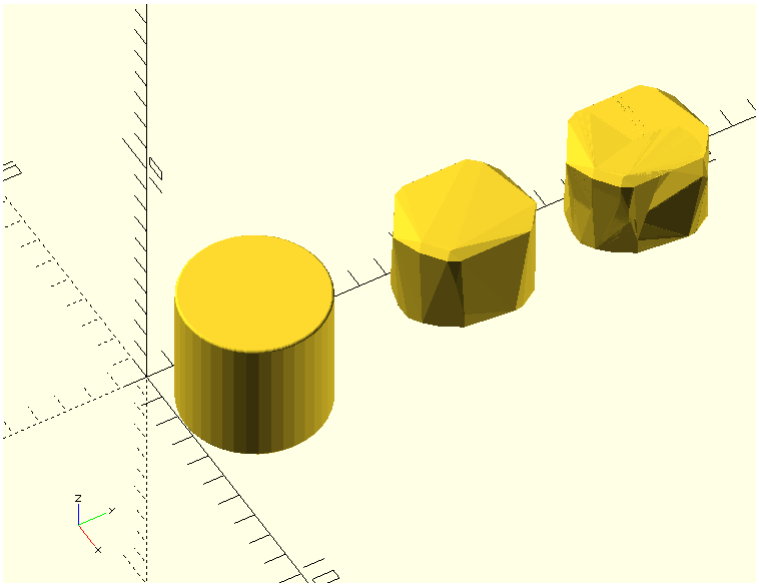


Figure 6.31: The reconstructed object shapes of the cylinder object (vertical) modeled on the left are shown. The object in the middle is the reconstructed object shape of the convex hull approach and on the right is the result of the concave hull approach. The reconstruction is based on the point cloud generated by the `surface_contour` node.

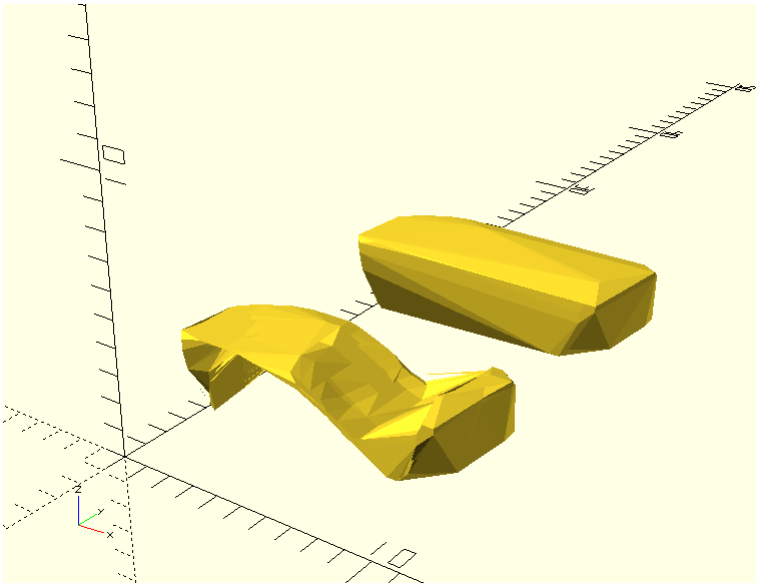


Figure 6.32: The reconstructed object shapes of the polystyrene object 1 are shown. The object on the left is the reconstructed object shape of the concave hull approach and on the right is the result of the convex hull approach. The reconstruction is based on the point cloud generated by the `grid_version` node.

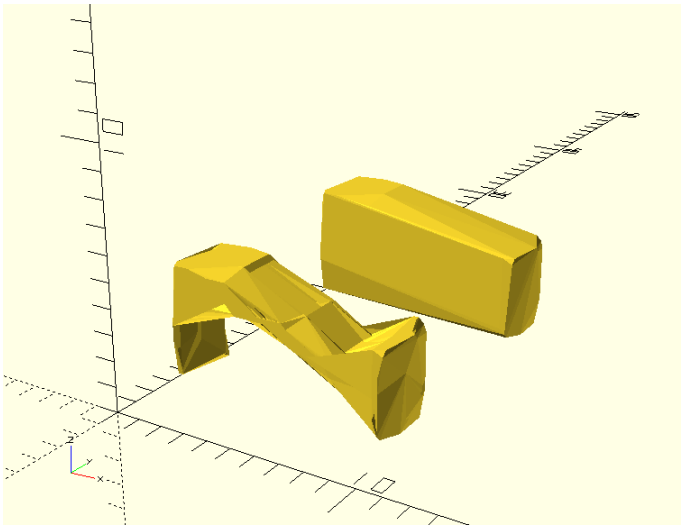


Figure 6.33: The reconstructed object shapes of the polystyrene object 2 are shown. The object on the left is the reconstructed object shape of the concave hull approach and on the right is the result of the convex hull approach. The reconstruction is based on the point cloud generated by the `grid_version` node.

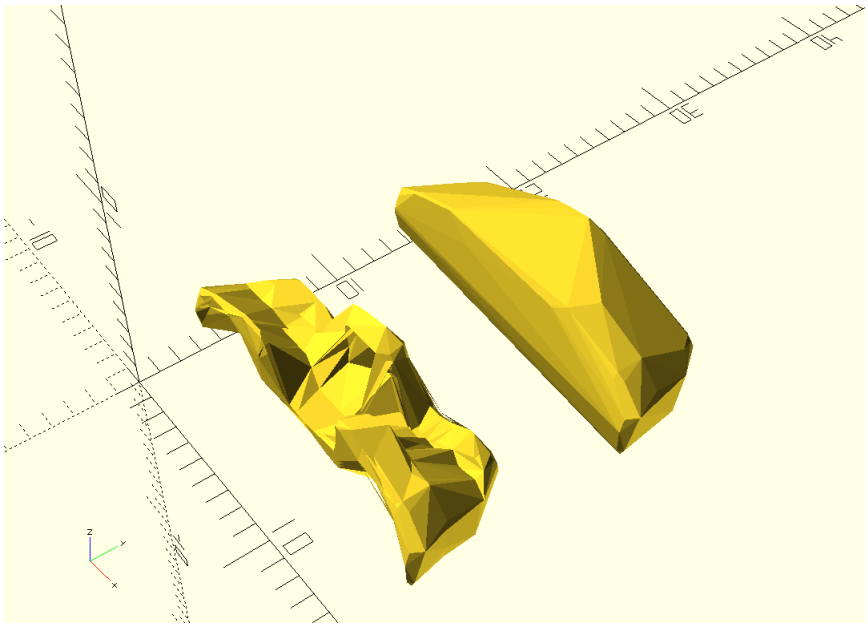


Figure 6.34: The reconstructed object shapes of the the object shown in figure 6.1(f) modeled on the left are shown. The object in the middle is the reconstructed object shape of the convex hull approach and on the right is the result of the concave hull approach. The reconstruction is based on the point cloud generated by the `grid_version` node.

7 Conclusion

This chapter provides a conclusion on the thesis in section 7.1 and a perspective for further work in respect of potential for improvements.

7.1 Conclusion of the Thesis

Approaches for object surface exploration and surface reconstruction to determine the shape of objects in the environment of a robotic arm have been conceptualized and implemented. The software components used for surface exploration have been implemented with the Robot Operating System (ROS). A tactile based approach for surface exploration has been chosen and the modeling of polygon meshes to reconstruct the shape of the objects.
The realized concept of surface exploration utilizes a six-axis force/torque sensor to detect contacts between the robot and the environment. The sensor is integrated into the end-effector of the robotic arm representing a force sensing probe. The measured forces acting on the probe are analyzed to determine the surface directions of the touched objects to generate motion goals for object surface contouring. Indirect force control by admittance control is applied to achieve interaction control. The strategy to explore object surfaces includes collision checking to avoid unwanted collisions between links of the robots kinematic chain other than the end-effector. Therefore, the locations of the contact points are used to place small collision objects which then are collision checked against the collision models of the robot. Three different approaches to avoid collisions but maintaining surface exploration have been implemented.
The implemented object surface reconstruction uses polygon meshes to represent the reconstructed shape of the objects. To generate the polygon meshes, first, the contact points of the surface exploration describing the surface of the objects are transformed into point clouds and then processed. The concave hull and convex hull of the point clouds are generated, and the Poisson surface reconstruction method is utilized to obtain polygon meshes.
The surface exploration is conducted on a KUKA LWR IV, while the used sensor is an ATi nano17e six-axis force/torque sensor and the end-effector probe is 3D printed. The explored objects mostly represent simple polyhedra.

7.2 Further work

Further work can be conducted towards the performance of the implemented approaches regarding time consumption in surface exploration and accuracy of the reconstructed object models. Although the duration time of surface exploration has been kept in mind while working on the thesis, the duration has not explicitly been investigated. Further work regarding the accuracy of the reconstructed object shapes is necessary to make statements about favorable parameters for

surface exploration and surface contouring paths. Plenty algorithms for surface reconstruction from point clouds exist. Investigation towards the utilization of other algorithms and tuning of the parameters of the used methods might result in more accurate polygon meshes.

Bibliography

[1] International Federation of Robotics, "Executive Summary World Robotics 2017 Industrial Robots.", https://ifr.org/downloads/press/Executive_Summary_WR_Service_Robots_2017_1.pdf, Online; accessed 30-October-2017.

[2] H. Al Hussein, T. Caldeira, D. Gan, J. Dias, and L. Seneviratne, "Object shape perception in blind robot grasping using a wrist force/torque sensor," in *Electronics, Circuits, and Systems (ICECS), 2013 IEEE 20th International Conference on*. IEEE, 2013, pp. 193–196.

[3] R. S. Jamisola, P. Kormushev, A. Bicchi, and D. G. Caldwell, "Haptic exploration of unknown surfaces with discontinuities," in *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*. IEEE, 2014, pp. 1255–1260.

[4] A. Winkler and J. Suchy, "Force controlled contour following by an industrial robot on unknown objects with tool orientation control," in *ISR/Robotik 2014; 41st International Symposium on Robotics; Proceedings of*. VDE, 2014, pp. 1–6.

[5] J. Back, J. Bimbo, Y. Noh, L. Seneviratne, K. Althoefer, and H. Liu, "Control a contact sensing finger for surface haptic exploration," in *Robotics and Automation (ICRA), 2014 IEEE International Conference on*. IEEE, 2014, pp. 2736–2741.

[6] N. Sommer, M. Li, and A. Billard, "Bimanual compliant tactile exploration for grasping unknown objects," in *Robotics and Automation (ICRA), 2014 IEEE International Conference on*. IEEE, 2014, pp. 6400–6407.

[7] J. J. Craig, *Introduction to robotics: mechanics and control*. Pearson Prentice Hall Upper Saddle River, 2005, vol. 3.

[8] C. Gaz, F. Flacco, and A. De Luca, "Identifying the dynamic model used by the KUKA LWR: A reverse engineering approach," in *Robotics and Automation (ICRA), 2014 IEEE International Conference on*. IEEE, 2014, pp. 1386–1392.

[9] G. Schreiber, A. Stemmer, and R. Bischoff, "The fast research interface for the KUKA LWR robot," in *IEEE Workshop on Innovative Robot Control Architectures for Demanding (Research) Applications How to Modify and Enhance Commercial Controllers (ICRA 2010)*, 2010, pp. 15–21.

[10] T. Kröger, "Opening the door to new sensor-based robot applications—The Reflexxes Motion libraries," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*. IEEE, 2011, pp. 1–4.

[11] Wikipedia, "Schemazeichnung eines Octrees, einer Datenstruktur der Informatik.", https://commons.wikimedia.org/wiki/File:Octree2.png, Online; accessed 09-November-2017.

[12] International Organization for Standardization (ISO), "ISO 8373 Robots and robotic devices.", https://www.iso.org/obp/ui/#iso:std:iso:8373, Online; accessed 29-October-2017.

[13] M. Yamaguchi, "Swimming robot probes Fukushima reactor to find melted fuel.", https://phys.org/news/2017-07-robot-probes-fukushima-reactor-fuel.html, Online; accessed 29-October-2017.

[14] J. Dietsch, "People meeting robots in the workplace [industrial activities]," *IEEE Robotics & Automation Magazine*, vol. 17, no. 2, pp. 15–16, 2010.

[15] M. I. Tiwana, S. J. Redmond, and N. H. Lovell, "A review of tactile sensing technologies with applications in biomedical engineering," *Sensors and Actuators A: physical*, vol. 179, pp. 17–31, 2012.

[16] S. Chen, "Kalman filter for robot vision: a survey," *IEEE Transactions on Industrial Electronics*, vol. 59, no. 11, pp. 4409–4420, 2012.

[17] SynTouch, "Syntouch Sensor Technology.", https://www.syntouchinc.com/sensor-technology/, Online; accessed 30-October-2017.

[18] ATI, "ATI nano17e.", http://www.ati-ia.com/products/ft/ft_models.aspx?id=Nano17global/products/ew6QHQeiYGMMAn69/P_setting_fff_1_90_end_500.png, Online; accessed 30-October-2017.

[19] ASUS, "ASUS Xtion PRO.", https://www.asus.com/media/global/products/ew6QHQeiYGMMAn69/P_setting_fff_1_90_end_500.png, Online; accessed 30-October-2017.

[20] P. K. Allen and K. S. Roberts, "Haptic object recognition using a multi-fingered dextrous hand," in *Robotics and Automation, 1989. Proceedings., 1989 IEEE International Conference on*. IEEE, 1989, pp. 342–347.

[21] R. Bajcsy, D. Brown, J. Wolfeld, and D. Peters, "What can we learn from one finger experiments?" in *International Symposium on Robotics Research*, 1984, pp. 509–527.

[22] R. L. Klatzky, S. J. Lederman, and V. A. Metzger, "Identifying objects by touch: An "expert system"," *Attention, Perception, & Psychophysics*, vol. 37, no. 4, pp. 299–302, 1985.

[23] S. J. Lederman and R. L. Klatzky, "Hand movements: A window into haptic object recognition," *Cognitive psychology*, vol. 19, no. 3, pp. 342–368, 1987.

[24] S. J. Lederman *et al.*, "The physiology and psychophysics of touch," in *Sensors and sensory systems for advanced robots*. Springer, 1988, pp. 71–91.

[25] B. Siciliano and L. Villani, *Robot force control*. Springer Science & Business Media, 2012, vol. 540.

[26] D. E. Whitney, "Historical perspective and state of the art in robot force control," *The International Journal of Robotics Research*, vol. 6, no. 1, pp. 3–14, 1987.

[27] M. Vukobratović and Y. Nakamura, "Force and contact control in robotic systems," in *Tutorial at the IEEE conference on robotics and automation, Atlanta, GA*, 1993.

[28] J. De Schutter, H. Bruyninckx, W.-H. Zhu, and M. W. Spong, "Force control: a bird's eye view," in *Control Problems in Robotics and Automation*. Springer, 1998, pp. 1–17.

[29] A. Winkler and J. Suchỳ, "Explicit and implicit force control of an industrial manipulator—an experimental summary," in *Methods and Models in Automation and Robotics (MMAR), 2016 21st International Conference on*. IEEE, 2016, pp. 19–24.

[30] A. Winkler and J. Suchy, "Position feedback in force control of industrial manipulators-an experimental comparison with basic algorithms," in *Robotic and Sensors Environments (ROSE), 2012 IEEE International Symposium on*. IEEE, 2012, pp. 31–36.

[31] A. Bierbaum, M. Rambow, T. Asfour, and R. Dillmann, "A potential field approach to dexterous tactile exploration of unknown objects," in *Humanoid Robots, 2008. Humanoids 2008. 8th IEEE-RAS International Conference on*. IEEE, 2008, pp. 360–366.

[32] B. Siciliano and O. Khatib, *Springer handbook of robotics*. Springer, 2016.

[33] J. M. McCarthy, *Introduction to theoretical kinematics*. MIT press, 1990.

[34] P. Corke, *Robotics, vision and control: fundamental algorithms in MATLAB*. Springer, 2011, vol. 73.

[35] J. Denavit, "A kinematic notation for lower-pair mechanisms based on matrices," *ASME J. Appl. Mech.*, pp. 215–221, 1955.

[36] J. Duffy, *Analysis of mechanisms and robot manipulators*. Edward Arnold London, 1980.

[37] K. Waldron and A. Kumar, "The dextrous workspace," *ASME Paper*, no. 80, 1980.

[38] J. De Schutter and H. Van Brussel, "Compliant robot motion i. a formalism for specifying compliant motion tasks," *The International Journal of Robotics Research*, vol. 7, no. 4, pp. 3–17, 1988.

[39] N. Hogan, "Impedance control: An approach to manipulation," in *American Control Conference, 1984*. IEEE, 1984, pp. 304–313.

[40] H. Kazerooni, T. B. Sheridan, and P. K. Houpt, "Robust compliant motion for manipulators, parts i-ii," *IEEE Transaction of Robotics and Automation*, 1986.

[41] J. K. Salisbury, "Active stiffness control of a manipulator in cartesian coordinates," in *Decision and Control including the Symposium on Adaptive Processes, 1980 19th IEEE Conference on*, vol. 19. IEEE, 1980, pp. 95–100.

[42] E. O. Doebelin and D. N. Manik, *Measurement systems: application and design*. McGraw-Hill, 2007.

[43] J. Fraden, *Handbook of modern sensors: physics, designs, and applications*. Springer Science & Business Media, 2004.

[44] E. Suhir, "How to make a device into a product: Accelerated life testing it's role attributes challenges pitfalls and interaction with qualification testing," *Micro-and Opto-Electronic Materials and Structures: Physics, Mechanics, Design, Packaging, Reliability", Springer*, 2007.

[45] F. Bellocchio, N. A. Borghese, S. Ferrari, and V. Piuri, *3D surface reconstruction: multi-scale hierarchical approaches*. Springer Science & Business Media, 2012.

[46] R. Fabio, "From point cloud to surface: the modeling and visualisation problem–international archives of the photogrammetry, remote sensing and spatial information sciences," vol XXXIV-5/W10-2001, Tech. Rep., 2001.

[47] M. Kazhdan, M. Bolitho, and H. Hoppe, "Poisson Surface Reconstruction," in *Proceedings of the Fourth Eurographics Symposium on Geometry Processing*, ser. SGP '06. Aire-la-Ville, Switzerland, Switzerland: Eurographics Association, 2006, pp. 61–70. [Online]. Available: http://dl.acm.org/citation.cfm?id=1281957.1281965

[48] R. Bischoff, J. Kurth, G. Schreiber, R. Koeppe, A. Albu-Schäffer, A. Beyer, O. Eiberger, S. Haddadin, A. Stemmer, G. Grunwald *et al.*, "The KUKA-DLR Lightweight Robot arm-a new reference platform for robotics research and manufacturing," in *Robotics (ISR), 2010 41st international symposium on and 2010 6th German conference on robotics (ROBOTIK)*. VDE, 2010, pp. 1–8.

[49] KUKA, "KUKA LWR.", https://www.kukakore.com/wp-content/uploads/2012/07/KUKA_LBR4plus_ENLISCH.pdf, Online; accessed 05-November-2017.

[50] Schunk, "Schunk WSG.", https://schunk.com/gb_en/gripping-systems/series/wsg/, Online; accessed 31-October-2017.

[51] ATI, "ATI Network Force/Torque Sensor System.", http://www.ati-ia.com/app_content/documents/9610-05-1031.pdf, Online; accessed 31-October-2017.

[52] ATI, "F/T Sensor: Nano17.", http://www.ati-ia.com/products/ft/ft_models.aspx?id=Nano17, Online; accessed 31-October-2017.

[53] J. Pan, S. Chitta, and D. Manocha, "FCL: A general purpose library for collision and proximity queries," in *Robotics and Automation (ICRA), 2012 IEEE International Conference on*. IEEE, 2012, pp. 3859–3866.

[54] R. B. Rusu and S. Cousins, "3D is here: Point Cloud Library (PCL)," in *Robotics and automation (ICRA), 2011 IEEE International Conference on*. IEEE, 2011, pp. 1–4.

[55] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard, "OctoMap: An efficient probabilistic 3D mapping framework based on octrees," *Autonomous Robots*, 2013. [Online]. Available: http://octomap.github.com

[56] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "ROS: an open-source Robot Operating System," in *ICRA workshop on open source software*, vol. 3, no. 3.2. Kobe, 2009, p. 5.

[57] M. Quigley, E. Berger, A. Y. Ng *et al.*, "Stair: Hardware and software architecture," in *AAAI 2007 Robotics Workshop, Vancouver, BC*, 2007, pp. 31–37.

[58] K. A. Wyrobek, E. H. Berger, H. M. Van der Loos, and J. K. Salisbury, "Towards a personal robotics development platform: Rationale and design of an intrinsically safe personal robot," in *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*. IEEE, 2008, pp. 2165–2170.

[59] ROS wiki, "ROS Concepts.", http://wiki.ros.org/ROS/Concepts, Online; accessed 31-October-2017.

[60] ROS wiki, "ROS Workspaces.", http://wiki.ros.org/catkin/workspaces, Online; accessed 31-October-2017.

[61] S. Chitta, I. Sucan, and S. Cousins, "Moveit![ROS topics]," *IEEE Robotics & Automation Magazine*, vol. 19, no. 1, pp. 18–19, 2012.

[62] M. Kazhdan, M. Bolitho, and H. Hoppe, "Poisson Surface Reconstruction," in *Symposium on Geometry Processing*, A. Sheffer and K. Polthier, Eds. The Eurographics Association, 2006.

Eidesstattliche Erklärung

Hiermit versichere ich an Eides statt, dass ich die vorliegende Arbeit im Masterstudiengang Informatik selbstständig verfasst und keine anderen als die angegebenen Hilfsmittel – insbesondere keine im Quellenverzeichnis nicht benannten Internet-Quellen – benutzt habe. Alle Stellen, die wörtlich oder sinngemäß aus Veröffentlichungen entnommen wurden, sind als solche kenntlich gemacht. Ich versichere weiterhin, dass ich die Arbeit vorher nicht in einem anderen Prüfungsverfahren eingereicht habe und die eingereichte schriftliche Fassung der auf dem elektronischen Speichermedium entspricht.

Hamburg, den 28.11.2017 Vorname Nachname

Veröffentlichung

Ich stimme der Einstellung der Arbeit in die Bibliothek des Fachbereichs Informatik zu.

Hamburg, den 28.11.2017 Vorname Nachname