Professorship of Lunar and Planetary Exploration
TUM School of Engineering and Design
Technical University of Munich

Graspable Target Detection for Hexapod Robots

LPE-MA-2023-03

Jitao Zheng

Thesis for the Attainment of the Degree

Master of Science

at TUM School of Engineering and Design of the Technical University of Munich.

Examiner:	Prof. Dr. Philipp Reiss Professorship of Lunar and Planetary Exploration
Supervised by:	Christian Gscheidle, M.Sc., LPE, TUM Dr. Mickaël Laîné, SRL, Tohoku University Prof. Dr. Kazuya Yoshida, SRL, Tohoku University
Submitted by:	Jitao Zheng Matriculation Number: 03735860
Submitted:	Munich, 31.10.2023

Acknowledgements

I would like to acknowledge and give my warmest thanks to my Tohoku University supervisors Professor Kazuya Yoshida and Assistant Professor Mickaël Laîné for integrating me into the great teams of SRL and AMC and making this amazing project possible. Their guidance and advice carried me through all the stages of my project. Thank you for all the enjoyable moments throughout my stay in Sendai. I would also like to express my sincere gratitude to my TUM supervisor Christian Gscheidle, for his brilliant comments and guidance during the writing process and for organizing frequent meetings that provided me with valuable suggestions.

I would also like to thank Assistant Professor Kentaro Uno and my fellow students in the Limb Robotics Team, who laid the foundations for this research and were always there to help me with crucial questions.

I would also like to give special thanks to my friends and colleagues Diane Vincent and Elian Neppel for their great support during the most difficult phase of programming. Their advice greatly enhanced my comprehension of robotics and programming improved decisively the quality of my work.

To my colleagues and fellow students at the Space Robotics Lab, and my supportive family, I would like to express my deepest appreciation. Your unwavering support and collaboration have been invaluable. Finally, I would also like to extend my thanks to the Mühlfenzl-Stiftung for their financial support. Their assistance in funding my research in Sendai have been instrumental in bringing this work to fruition.

Zusammenfassung

Mit dem Aufstieg kommertieller Raumfahrt und den jüngsten Erfolgen bei Erkundungsmissionen auf Asteroiden kam in den letzten Jahren vermehrt die Frage nach der Entwicklung eines Erkundungsroboters auf, der speziell für Fortbewegung in der Mikrogravitation und auf rauem Terrain optimiert ist. Mehrbeinige autonome Roboter mit mehrgliedrige Spike-Greifern haben hierbei einen entscheidenden Vorteil gegenüber klassischen, beräderten Rovern. Die immer leichter werdenden und präziseren Sensoren für die Tiefenerfassung auf dem Markt können zudem das Potenzial eines solchen Rovers erheblich steigern um geeignete Punkte im Terrain in Echtzeit für die Fortbewegung und Fixierung zu detektieren.

Die vorliegende Arbeit beschäftigt sich mit der Entwicklung und Implementierung eines Algorithmus zur Erkennung greifbarer Ziele für den Hexapod-Rover SCAR-E für terrestrischen Anwendungen sowie für planetare Exploration. Der Prototyp des Rovers wurde vom *Space Robotics Laboratory (SRL)* an der Tohoku University in Zusammenarbeit mit der *Asteroid Mining Corporation Ltd. (AMC)* entwickelt und verwendet ein visuelles Sensorsystem, das aus sechs Tiefenkameras besteht, um seine Umgebung wahrzunehmen. Der erste Entwicklungsschritt ist die Erstellung einer hochauflösenden virtuellen 3D-Umgebung für Simulationen. Mit Hilfe der Kinect v2 RGB-D Kamera wird in Verbindung mit der Real-Time Appearance-Based Mapping (RTAB-Map) Technologie eine detaillierte 3D-Karte eines repräsentativen felsigen Geländes erstellt, die durch Optimierungs- und Post-Processing-Algorithmen verfeinert wird.

Anschließend werden die gewonnenen Geländedaten verwendet, um einen Algorithmus zur Erkennung von greifbaren Zielen in der Umgebung des Roboters zu entwickeln und zu testen. Mit Hilfe von Tiefenbildern konzipiert der Algorithmus eine Rundum-Punktwolke für das Terrain in der Umgebung des Rovers. Im nächsten Schritt wird das Bezugssystem der Punktwolke nach dem Normalenvektor ihrer Regressionsebene ausgerichtet. Im Schatten liegende, fehlende Anteile werden in einer Delaunay-Triangulation ergänzt und die Geländedaten in ein diskretes Voxelgitter umgewandelt. Auf der Grundlage der Geometrie des Greifers wird eine dreidimensionale Maske angefertigt, die das Ausmaß des Greifumfangs des Greifers repräsentiert und dazu dient, jeden Punkt des voxelisierten Geländes auf seine Wahrscheinlichkeit hin, ein greifbares Ziel zu sein, zu bewerten. Parallel dazu werden im Gelände alle konvexen Bereiche mithilfe einer Krümmungsanalyse ermittelt und die Schnittmenge der beiden Methoden gebildet. Der Algorithmus nutzt das für SCAR-E angepasste *ClimbLab*-Konzept und ist für die Echtzeitausführung innerhalb eines ROS/ROS2-Frameworks konzipiert.

Simulationsergebnisse in verschiedenen Szenarien zeigen die Wirksamkeit des Algorithmus bei der Erkennung greifbarer Oberflächen unter Berücksichtigung der Geländekomplexität. Die Ergebnisse bestätigen die Eignung des Algorithmus an den mehrgliedrigen Greifer und den Multi-Sensor des optischen Aufbaus des SCAR-E und legen eine hohe Zuverlässigkeit der Voraussagen bezüglich der Greifbarkeit im felsigen Gelände nahe. Zukünftige Verbesserungen zielen darauf ab, die Geschwindigkeit zu optimieren, die Integration der Greifer-Funktionalität zu verfeinern und die Erkennung von steilem Gelände zu verbessern.

Abstract

With the rise of commercial spaceflight and the success of recent asteroid exploration missions, the need for an exploration robot designed to operate in microgravity and rugged terrain has emerged in recent years. Multi-legged autonomous robots with multi-section spike grippers have a decisive advantage over classic, wheeled rovers in terms of mobility in microgravity. In addition, the increasingly lighter and more precise depth imaging sensors on the market have the potential to significantly increase the rover's ability to detect graspable targets in real time.

The study deals with the development and implementation of an algorithm for the detection of graspable targets for the fixation and locomotion of the hexapod rover SCAR-E for terrestrial applications and for planetary exploration. The prototype rover, developed by the *Space Robotics Laboratory (SRL)* at Tohoku University in collaboration with *Asteroid Mining Corporation Ltd. (AMC)*, utilizes a visual sensor system consisting of six depth cameras to perceive its surroundings. The first development step is to create a high-resolution virtual 3D environment for simulations. By using the Kinect v2 RGB-D camera in conjunction with Real-Time Appearance-Based Mapping (RTAB-Map) technology, a detailed 3D map of a representative rocky terrain is created, which is refined using optimization and post-processing algorithms .

Subsequently, the terrain data obtained is utilized to test the algorithm for detecting graspable targets in the robot's vicinity. Using depth images, the algorithm creates a panoramic point cloud for the terrain surrounding the rover. In the next step, the reference frame of the point cloud is aligned according to the normal of its regression plane, missing parts are added in a Delaunay interpolation and the terrain data is converted into a discrete voxel grid. Based on the geometry of the gripper, a three-dimensional mask is created which represents the gripping extent of the gripper and is used to evaluate each point of the voxelized terrain for its probability to be a graspable target. In parallel, convex areas in the terrain are determined using a curvature analysis and the intersection of the two methods is formed. The algorithm is inspired by the SRL-*ClimbLab* concept, adapted for SCAR-E and is designed for real-time execution within a ROS/ROS2 framework.

The test results in various scenarios demonstrate the effectiveness of the algorithm in detecting surfaces that can be grasped while taking into account the terrain complexity. The findings confirm the adaptability of the algorithm to the panoramic perception and gripper requirements of SCAR-E and suggest a high reliability and accuracy of the predictions regarding accessibility in rocky terrain. Future improvements aim to optimize speed, enhance the integration of gripper's complete functionality, and improve the detection of steep terrain.

Contents

List of Figures

List of Tables

List of Symbols

Symbol	Unit	Description
$\mathbf{T} \in \mathbb{R}^{3\times4}$		Camera extrinsic matrix
$\mathbf{K} \in \mathbb{R}^{3\times3}$		Camera intrinsic matrix
$\mathbf{R} \in \mathbb{R}^{3\times3}$		Rotation matrix
$\mathbf{t} \in \mathbb{R}^{3}$		Translation matrix
$\Sigma_T = (x_T, y_T, z_T)$		Terrain based reference frame
$\Sigma_C = (x_C, y_C, z_C)$		Camera based reference frame
$\Sigma_P = (x_P, y_P)$		Pixel coordinate system
$\Sigma_R = (x_R, y_R, z_R)$		Robot based reference frame
u, v	[px]	Discrete pixel coordinates
f	[m]	Focal length
f_u, f_v	[px]	Focal length in pixel
c_x, c_y	[m]	Horizontal and vertical offset
c_u, c_v	[px]	Horizontal and vertical offset in pixel
ρ_u, ρ_v	[m/px]	Meter per pixel ratios
h, w	[px]	Camera resolution
α_u, α_v	[deg]	Horizontal and vertical angle of view
d	[px]	Disparity between two cameras
D	[m]	Depth
B	[px]	Baseline (Distance between two cameras)
$\mathbf{r}_{CP} \in \mathbb{R}^{3}$		Vector from camera to point
$\mathbf{r}_{CT} \in \mathbb{R}^{3}$		Vector from camera based frame to terrain based frame
$\mathbf{X} \in \mathbb{R}^{3\times N}$		Deviation matrix
$\mathbf{n}_T \in \mathbb{R}^{3}$		Normal vector of regression plane
$\mathbf{T}_{CT} \in \mathbb{R}^{3\times3}$		Transformation matrix from camera to terrain based frame
N		Number of points
$\mathbf{1} \in \mathbb{R}^{N}$		Vector of ones
$\mathbf{P}_R \in \mathbb{R}^{3\times N}$		Entirety of points in robot based frame
$\mathbf{P}_C \in \mathbb{R}^{3\times N}$		Entirety of points in camera based frame
$\mathbf{P}_T \in \mathbb{R}^{3\times N}$		Entirety of points in terrain based frame
$\mathbf{x}_T, \mathbf{y}_T, \mathbf{z}_T \in \mathbb{R}^{3}$		x-, y- and z-directional unit vectors of terrain based frame

Symbol	Unit	Description
α	[deg]	Rotation around x
β	[deg]	Rotation around y
γ	[deg]	Rotation around z
GS	[m]	Grid size
w_x, w_y	[m]	Side length of voxel array
cs	[m]	Cube size
R_{voxel}	[1/mm]	Voxel ratio
$\varnothing_P$	[mm]	Palm diameter
$\varnothing_{FJ}$	[mm]	Palm diameter of finger joints
L_F	[mm]	Finger length
L_S	[mm]	Spine length
D_s	[mm]	Spine length
θ_o	[deg]	Gripper opening angle
θ_c	[deg]	Gripper closing angle
R_{os}	[mm]	Opening spine radius
D_{os}	[mm]	Opeining spine depth
H_c	[mm]	Closing height
L_{GM}		Gripper mask side length
R_T		Top solid radius
H_{cl}		Gripper mask top clearance
R_B		Gripper mask bottom void space
R_g		Radius of grippable area
$R_{u,i}$		Inner unreachable radius
DC		Distance from the center
$\mathbf{P}_{subset} \in \mathbb{R}^{3\times N}$		Entirety of points in the Subset
$\mathbf{P}_{GM} \in \mathbb{R}^{3\times N}$		Entirety of points in the Gripper Mask
G		Graspability score
G_P		Penalized Graspability
k_1, k_2		Principal curvatures
K		Gaussian curvature
H_{add}		Additional Void Gripper Mask Layers

Acronyms

3-DOF three degrees of freedom

AMC Asteroid Mining Corporation Ltd.

AUC Area Under the Curve

FN False Negatives

FNR False Negative Rate

FP False Positives

FPR False Positive Rate

GSV Gripper Solid Voxels

ICP Iterative Closest Point

IMU Inertial Measurement Unit

IR infrared

MASCOT Mobile Asteroid Surface Scout

OpenCL Open Computing Language

OpenCV Open Source Computer Vision Library

OpenGL Open Graphics Library

PCD Point Cloud Data

PCL Point Cloud Library

PGM platinum-group metals

PLY Polygon File Format

PSR Permanent Shadowed Regions

RGB-D Color-depth camera

RMSE Root Mean Square Error

RNA ribonucleic acid

ROC Receiver Operating Characteristic

ROS Robot Operating System

ROS2 Robot Operating System 2

RTAB-Map Real-Time Appearance-Based Mapping

SBA Sparse Bundle Adjustment

SCAR-E Space Capable Asteroid Robotic Explorers

SLAM Simultaneous Localization and Mapping

SRL Space Robotics Laboratory

SRL-GTD SRL Graspable Target Detection Algorithm

SSV Subset Solid Voxels

TN True Negatives

TNR True Negative Rate

ToF Time-of-Flight technique

TP True Positives

TPR True Positive Rate

TSV Threshold of Solid Voxels

VTA Voxelized Terrain Array

1 Introduction

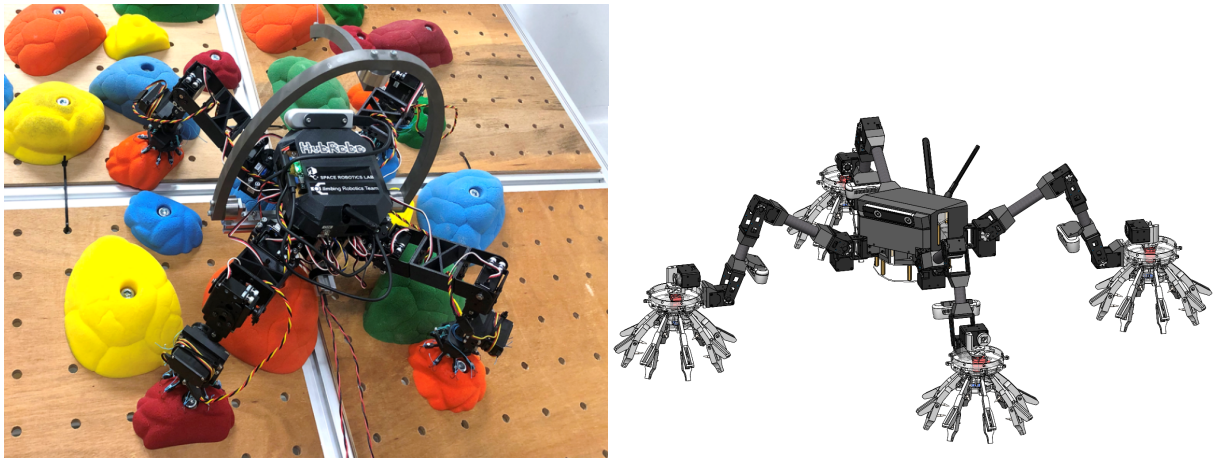
The emergence of commercial spaceflight and the sustained enthusiasm in asteroid exploration have paved new ways and opportunities for the advancement of space resource utilization, and have also brought a new impetus to the development of robotic exploration rovers. The Hayabusa 2 mission, featuring its pioneering hopping rover MINERVA-II [3], offered unprecedented insights into the asteroid 162173 Ryugu [4], despite mobility limitations arising from its uncontrolled motion system [5]. This success has sparked interest in the viability of near-Earth asteroid resources and laid the groundwork for new potential markets in the future. However, the in-situ exploitation of planetary resources can in many cases only take place in difficult-to-access terrains, such as in Permanent Shadowed Regions (PSR) on the Moon or in heavily rugged terrain on asteroids. Thus, in addition to the classic design of wheeled exploration rover, the development of non-wheeled, multi-limbed approaches appears particularly promising.

Multi-limbed robots would not only be able to navigate complex and rocky terrain, but they would also be capable of performing missions in cavities and overhangs. However, designing such a rover poses not only major challenges to the gripping arms, but also to the pathfinding system. The design of the gripper only allows the rover to find stable footing on certain types of terrain, such as rocks or icy ground. Thus, each step of the rover must be carefully chosen before it is set. The visual sensor of a multi-limbed robot must include real-time recognition of the properties of the surface and the detection of graspable target. By adapting existing technologies from Simultaneous Localization and Mapping (SLAM) technologies and geometry-based salient shape recognition algorithm, it may be possible to obtain accurate information about the terrain in the vicinity of the rover.

The Space Robotics Laboratory (SRL) of Tohoku University, led by Professor Kazuya Yoshida, is dedicated to the research and development of robotic systems for space science and exploration missions. Several teams are currently working on rover projects at the SRL with the aim of developing and optimizing robotic explorers. These include the Space Capable Asteroid Robotic Explorers (SCAR-E) rover project, a cooperation between the SRL and the British company Asteroid Mining Corporation Ltd. (AMC). As of July 2023, the mechanical design of the rover is finalized, and a prototype EM1 has been assembled. The integration of sensors and the mapping and pathfinding system are currently under development.

1.1 Limbed Robots of SRL

The Limb Robotics Team and the Asteroid Mining Team are two experienced teams in SRL dedicated in the development of climbing capable robots. The Limb Robotics Team specializes, among other topics, in optimizing the technology around the four-legged climbing robot HubRobo and developing solutions for challenging pathfinding in complex climbing terrain. The Asteroid Mining Team on the other hand works on



(a) With spine-type passive grippers (b) With new re-dimensioned grippers

Fig. 1–1: HubRobo is a quadrupedal climbing robot prototype with a full mass of 3 kg[6].

approaches for exploration in unknown planetary environments and for terrestrial applications, including rescue missions and the commercial exploitation of resources.

1.1.1 HubRobo

The quadruped robot HubRobo with a symmetric insect-type configuration (as shown in Fig. 1–1a) has three degrees of freedom (3-DOF) actuated joints in each limb, a passive compliant spine gripper at each foot, and an actuator to open/close the gripper. The spine-type grippers have been developed considering a natural rocky surface, where the roughness of the object to be grasped is high [6]. Following the initial idea, several optimizations were subsequently made to the grippers, including the integration of a multi-axis force/torque sensor into the palm and an increasing of the dimensions and opening/closing range of the grippers, as shown in Fig. 1–1b.

HubRobo is equipped with an Inertial Measurement Unit (IMU) to estimate the base linear and angular displacement and a depth camera used for 3D terrain sensing. Both sensors are utilized for SLAM-purposes. The gait planner ClimbLab [7] includes a grasping location selection algorithm based on salient shape detection, along with a non-periodic swing limb selection which is able to select a limb that has the most graspable options in the direction of the target location. This study will utilize and adapt ClimbLab's salient shape detection algorithm as part of SCAR-E's proposed gait planning system.

1.1.2 SCAR-E

The SCAR-E rover, a hexapod robot designed for asteroid mining, planetary exploration and terrestrial missions, has 3-DOF gear actuated joints in each modular assembled robot leg. Fig. 1–2 shows a render of the robot's design. SCAR-E employs a soft gripping solution with segmented spined finger joints to conform to uneven shapes and apply even gripping force. Based on the action of a ball screw and the pulling of tether,

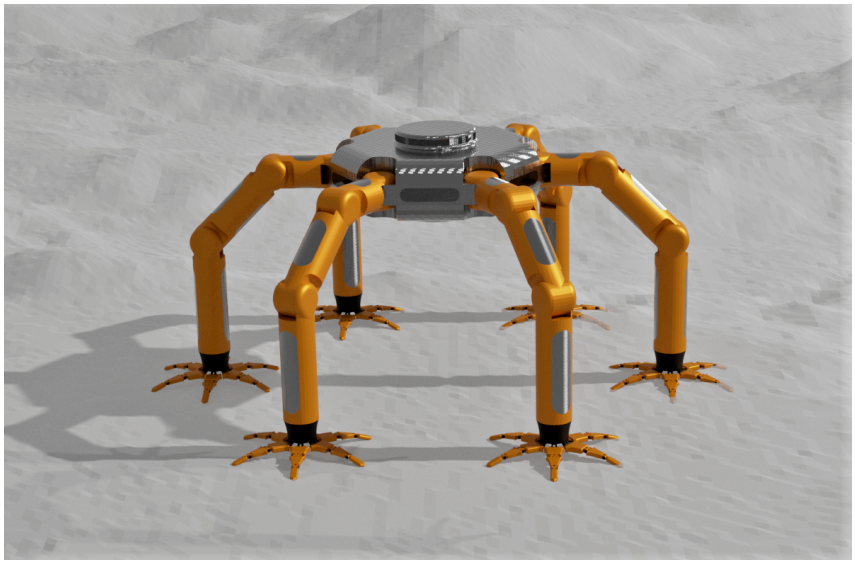
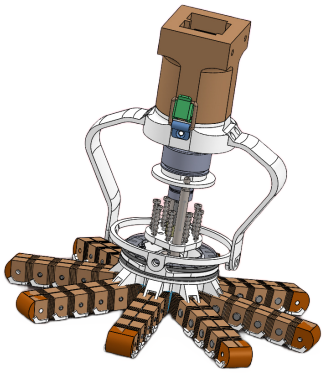
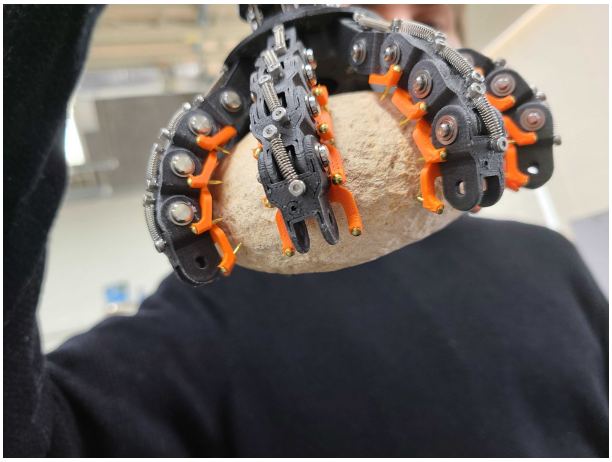


Fig. 1–2: Render of the SCAR-E robot.



(a) Proposed gripper design.



(b) Prototype of SCAR-E's gripper.

Fig. 1–3: Soft gripping solution of SCAR-E [8].

soft gripping fingers are actuated to match different target shapes, and pressure is then applied onto the surface of the target to secure the gripping action [8]. The design of the gripper is shown in Fig. 1–3a and the assembled prototype made for testing in Fig. 1–3b.

An unique feature of SCAR-E is the conscious omission of a defined front, back, top or bottom. This circumstance gives the robot greater flexibility with regard to, for example, movement in narrow cavities. The visual system consisting of six Color-depth camera (RGB-D) vision sensors provides a 360 degree environmental perception, which can be further supplemented with lidars on the top and the bottom of the main body.

An agile motion-planning has been developed for motion through a map of potential footholds, achieved through efficient inverse kinematics and neural network approximation [9]. Furthermore, an A* algorithm was utilized for an adaptive-gait navigation

in rough terrains with minimal footholds, which enables a motion acting in under one minute [9].

1.2 Project Outline

The proposed SRL Graspable Target Detection Algorithm (SRL-GTD) analyzes depth images obtained from six Intel RealSense RGB-D cameras to identify suitable gripping locations and accessible footholds in the robot's immediate vicinity and mid-range.

Within the scope of this work, the objective is to determine whether the proposed algorithm is suitable for detecting correct graspable points in a rocky terrain and whether the requirements placed on the rover SCAR-E in terms of the geometry of the gripper, the constellation of sensors and the range of applications can be met. In addition, the scope and limitations of the algorithm will be established through tests in terrains of varying complexity.

As SCAR-E is still under development, the algorithm is tested by simulations. For this purpose, a replica of a representative rocky terrain test side is used as the simulation environment, which is created using SLAM and graph optimisation methods. An adaptation and extension of the ClimbLab concept is carried out with the aim of detecting salient shapes suitable for grasping (graspable targets) based on the gripper's geometry in real time within a Robot Operating System (ROS) / ROS2 framework using C++. The prospective algorithm SRL-GTD should comprise several key steps, including perceiving the representative simulation environment, processing depth images into panoramic point clouds, base transformation for terrain alignment, implementing Delaunay Triangulation for accurate occluded area compensation, and voxelization. Essentially, the algorithm incorporates enhanced novel criteria to identify positions within the terrain matrix suitable for gripping.

In order to evaluate the results, the study develops metrics to compare the results of a field test with the predictions made by the algorithm. Parts of the study have been conducted in Sendai at Tohoku University, as the SRL teams have extensive expertise in the fields of limb robotics and terramechanics.

2 State of the Art

This chapter provides a kick-off by first introducing the surface characteristics of asteroids, which is relevant to the design of the testing environment, followed by the fundamentals of depth imaging and point clouds generation. Finally, The vision sensor technology intended for the SCAR-E rover, as well as the proposed algorithm for terrain mapping and graspable target detection is presented.

2.1 Surface Characteristics of Asteroids

To develop a system for in-situ exploration of asteroids, it is important to understand the surface characteristics of asteroids rock composition. Therefore, the classification of asteroids and the potential significance for asteroid mining is discussed briefly in the following sections.

2.1.1 Asteroid Classification

Asteroids exhibit diverse surface features that vary in size and shape. They can be broadly classified into different categories based on their spectra, albedo, and surface texture. Common classifications include S-type (stony), C-type (carbonaceous) and M-type (metallic), among others[10].

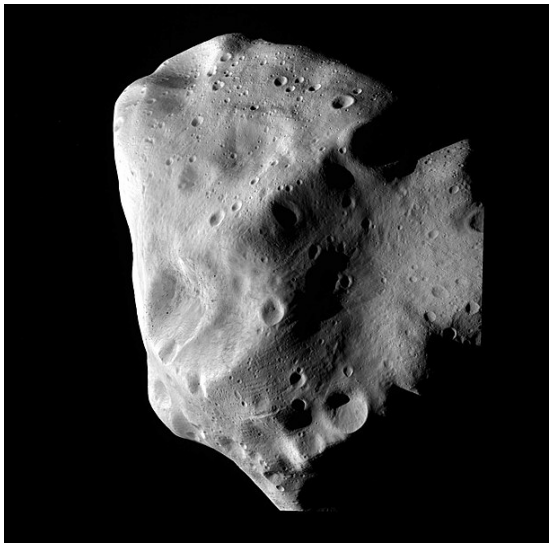


Fig. 2–1: M-type asteroid 21 Lutetia [1].

Fig. 2–2: S-type asteroid 433 Eros[2].

The surface texture of asteroids is often influenced by impact cratering, regolith formation and space weathering. The compositional variations of asteroids are linked to their formation distances from the Sun [11]. Observations from spacecraft missions and ground-based telescopes have revealed a diverse range of features, such as impact craters, boulders, ridges, and grooves. Spectral analysis of asteroids provided valuable information about the mineral composition of the rocks present. For

Tab. 2–1: Main asteroid compositional types and their number of objects larger than 30 km in the main asteroid belt (2.3 AU - 3.2 AU) [15, 16]. Other spectral types such as V-, D- or E-type asteroids make up about 26 % of objects in the asteroid belt.

Type	Visual geometric albedo	Spectral reflectivity (0.3 µm to 1.1 µm)	No.	Percentage
C	<0.07	Neutral, slight absorption blueward of 0.4 µm	505	49 %
S	0.07 - 0.23	Reddened, typically an absorption band ∼0.9 µm to 1.0 µm	175	17 %
M	0.07 - 0.23	Featureless, sloping up into red	85	8 %

instance, C-type, the most common asteroid type in our solar system, are often rich in organic compounds. They probably consist of clay and silicate rocks, and are dark in appearance [12]. M-type asteroids are characterized by high metal content. Some of them underwent high temperatures post-formation, causing partial melting, wherein iron sank to their centers, while basaltic (volcanic) lava was forced to the surface [13]. S-type asteroids on the other hand are moderately bright and consist mainly of iron- and magnesium-silicates. Due to their volatile-poor composition, S-type asteroids have relatively high density compared to C-type asteroids [14].

The asteroid classification system usually categorizes asteroids according to their spectra, an overview of which is given in [11]. The criterions and the abundance of the three main types of asteroids in the main belt are shown in tab. 2–1.

2.1.2 Significance for Exploration

The exploration of asteroid resources holds commercial significance that have captured the attention of space agencies and private companies. With about 80 objects larger than 30 km, M-type asteroids are the rarest of the major asteroid types in the Main Asteroid Belt. In particular, the presence of higher density compositions such as iron-nickel are detected by radar observations and can make up well over 90 % of the total mass[15]. Precious metals such as platinum-group metals (PGM) are believed to be in abundance on M-type asteroids. Although PGM contents in asteroids likely do not reach as high as previously thought, as the data that come along with these claims are decades old, incomplete, and of questionable quality [17]. Nevertheless, most iron meteorites has higher combined PGM concentrations than almost any terrestrial deposit [17].

C-type asteroids on the other hand are rich in volatile compounds and are believed to contain significant amounts of water in the form of hydrated minerals and water ice [18]. C-type asteroids may also contain organic materials, which can potentially serve as the basis for sustaining human life in space habitats. For instance, samples from the near-Earth C-type asteroid 162173 Ryugu (Fig. 2–3a) collected by the Hayabusa2 mission showed the presence of organic compounds, such as uracil (one of the four components in ribonucleic acid (RNA)) and Vitamin B3 [19]. Furthermore, while car-

(a) 162173 Ryugu during approach [21].

(b) Surface taken by the MASCOT lander [22].

Fig. 2–3: Images of the C-type asteroid 162173 Ryugu take by the Hayabusa 2 mission 2018.

bonaceous asteroids are not typically as rich in metals as metallic asteroids, some may still contain valuable metals such as platinum [20].

The surface of C-type asteroids typically exhibits distinctive features that are shaped by various geological processes and are significantly more rugged than their M- or S-type counterparts. The surface of C-type asteroids is most likely covered by a thin layer of regolith, which is composed of fine dust, small rocks, and debris [23]. Furthermore, C-type asteroids may exhibit grooves and ridges in large scale, with boulders and rocky outcrops scattered across the surface. The photographs taken by the Hayabusa 2 lander Mobile Asteroid Surface Scout (MASCOT) show mainly dark, decimeter to meter sized angular boulders with a more irregular and crumbly surface, but sometimes also smooth boulders [24], as shown in Fig. 2–3b. The abundance of regolith is however not as large as expected [24]. The design of SCAR-E is particularly well suited to this type of terrain, with the spiked gripper arms designed to cling to particularly rough surfaces. At the same time, these circumstances also place greater demands on the pathfinding system, as the graspable points and the most efficient route to the destination must be carefully chosen so that the rover's limbs do not become entangled along the way.

2.2 Depth Imaging

Depth imaging using stereo cameras involves capturing scenes from two or more viewpoints to extract the 3D structure of objects. Stereo vision relies on the principle of triangulation, where the depth of a point in the scene can be estimated based on the disparity between its projections on the two camera sensors.

2.2.1 Camera Matrix

This section will use several coordinate frames defined as followed:

- Terrain based coordinate system $\Sigma_T = (x_T, y_T, z_T)$, which is fixed to the ground.
- Camera based coordinate system $\Sigma_C = (x_C, y_C, z_C)$, placed on the camera optical center.
- Pixel coordinate system $\Sigma_P = (x_P, y_P)$, being the coordinates on the 2D image plane of the camera.

The camera extrinsic matrix $\mathbf{T}$ in Eq. 2–1 defines a transformation matrix from the terrain coordinate system Σ_T to the camera coordinate system Σ_C. It can be decomposed into a rotation $\mathbf{R}$ and a translation $\mathbf{t}$.

$$\mathbf{T} = \begin{bmatrix} \mathbf{R} & | & \mathbf{t} \end{bmatrix} = \begin{bmatrix} r_{1,1} & r_{1,2} & r_{1,3} & t_1 \\ r_{2,1} & r_{2,2} & r_{2,3} & t_2 \\ r_{3,1} & r_{3,2} & r_{3,3} & t_3 \end{bmatrix}. \tag{2–1}$$

The elements of Eq. 2–1 are derived from the physical location or orientation of the camera. Eq. 2–2 depicts the transformation of a point (X_T, Y_T, Z_T) from the terrain reference frame to the camera reference frame.

$$\begin{bmatrix} X_C \\ Y_C \\ Z_C \\ 1 \end{bmatrix} = \begin{bmatrix} \mathbf{R}_{3\times3} & \mathbf{t}_{3\times1} \\ \mathbf{0}_{1\times3} & 1_{1\times1} \end{bmatrix} \begin{bmatrix} X_T \\ Y_T \\ Z_T \\ 1 \end{bmatrix} \tag{2–2}$$

The operation of rotation and translation of terrain coordinates can be represented as a 4×4 matrix.

The intrinsic matrix is defined in Eq. 2–3. $\mathbf{K}$ denotes a transformation matrix that converts points from the 3D camera coordinate system Σ_C to a 2D homogeneous pixel coordinate system Σ_P.

$$\mathbf{K} = \begin{bmatrix} f & 0 & c_x \\ 0 & f & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{2–3}$$

f is the focal length, c_x, c_y the horizontal and vertical offset. The intrinsic matrix depends only on camera characteristics and requires inner values for the camera such as focal length or optical center. The perspective projection of an image can be modeled by the ideal pinhole camera.

As shown in Fig. 2–4, in the pinhole camera model, a point in space with coordinates $P = (X_c, Y_c, Z_c)$ is mapped to the point on the image plane where a line joining the point P to the centre of projection meeting the image plane [25].

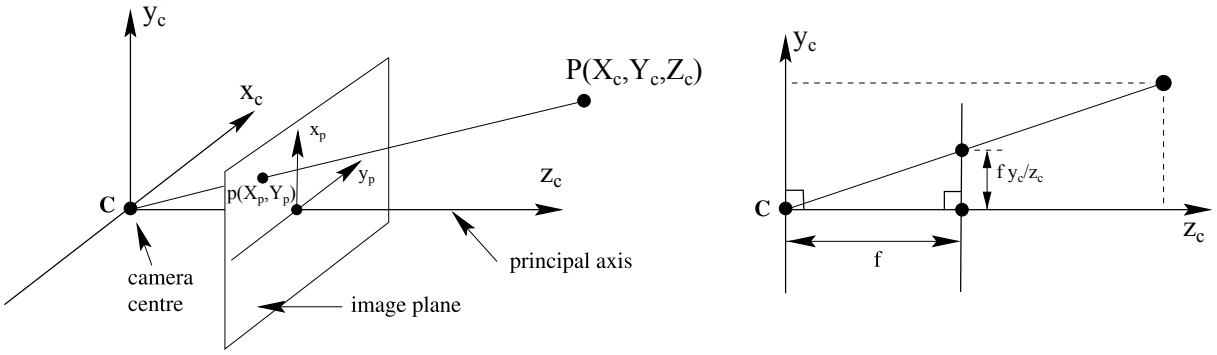


Fig. 2–4: Pinhole camera geometry. C is the camera center and p the principal point. The camera centre is here placed at the coordinate origin [25].

By projecting the x_C and y_C coordinates of the points onto the 2D plane, the 2D plane is at focal length f distance away from the camera. Eq. 2–4 shows the projection x_P, y_P on the image plane, which can be found by the law of similar triangles.

$$\begin{aligned} x_P &= f\frac{x_C}{z_C} \\ y_P &= f\frac{y_C}{z_C} \end{aligned} \tag{2–4}$$

In digital imaging, the pixel coordinates (u, v) represents the integer values by discretizing the points in the image coordinate system, defined in Eq. 2–5. Pixel coordinates of an image are discrete values within a range that can be achieved by dividing the image coordinates by pixel width and height. Furthermore, the pixel coordinates system has the origin at the left-top corner, hence the camera offsets c_x, c_y are also required alongside the discretization.

$$u = \frac{1}{\rho_u} f \frac{x_c}{z_c} + c_x \frac{1}{\rho_u}$$
$$v = \frac{1}{\rho_v} f \frac{y_c}{z_c} + c_y \frac{1}{\rho_v} \tag{2–5}$$

Whereof ρ_u and ρ_v are the meter per pixel ratio in the respective image dimension. Thereof, we can merge the focal length and offset in discrete pixels as stated in Eq. 2–6.

$$f_u = \frac{f}{\rho_u}$$
$$f_v = \frac{f}{\rho_v}$$
$$c_u = \frac{c_x}{\rho_u} \tag{2–6}$$
$$c_v = \frac{c_y}{\rho_v}$$

Consequently the discrete intrinsics matrix K_{uv} can be represented with Eq. 2–7.

$$\mathbf{K}_{uv} = \begin{bmatrix} f_u & 0 & c_u \\ 0 & f_u & c_v \\ 0 & 0 & 1 \end{bmatrix} \tag{2–7}$$

In case of pinhole camera model, if the resolution of the camera is known, i.e. the height h and width w of the resulting image in pixels, and the horizontal and vertical angles of view α_u, α_v are also known, the focal length can be easily specified as Eq. 2–8 states.

$$f_u = \frac{w}{2\tan(\alpha_u/2)}$$
$$f_v = \frac{h}{2\tan(\alpha_v/2)} \tag{2–8}$$

In real stereo vision systems, the location and optical parameters of each separate camera must be calibrated to remove the distortions in the image so that triangulation methods can be used to determine the correspondence between pixels in each image.

2.2.2 Depth Calculation

Depth can be calculated using triangulation. Given corresponding points $P_1 = (u_1, v_1)$ in the left image and $P_2 = (u_2, v_2)$ in the right image, the disparity d can be calculated as shown in Eq. 2–9.

$$d = u_1 - u_2 \tag{2–9}$$

The depth D can be calculated using the equation in Eq. 2–10

$$D = \frac{B \cdot f}{d}, \tag{2–10}$$

where B is the baseline, the distance between the two cameras.

2.2.3 Point Cloud Generation

Once the depth D is calculated, we can generate a 3D point cloud by converting the image coordinates to terrain coordinates Σ_T:

$$\begin{bmatrix} x_T \\ y_T \\ z_T \end{bmatrix} = \mathbf{K}_{uv}^{-1} \cdot \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \cdot D \tag{2–11}$$

$$z_T = D$$

$$x_T = \frac{(u - c_u) \cdot D}{f_u} \tag{2–12}$$

$$y_T = \frac{(v - c_v) \cdot D}{f_v}.$$

This process is repeated for each pixel in the depth image to create a complete point cloud representation of the scene.

2.3 Depth Cameras

Depth cameras are vision systems that manipulate the properties of their environment, primarily visually, to capture 3D scene data from their field of view [26]. A common way to generate depth images with an RGBD camera is the active stereoscopic camera method, of which the Intel RealSense Depth Module D430 or the Kinect for Xbox One are the prominent representatives. More sophisticated systems, such as the Kinect v2, use the Time-of-Flight technique (ToF) technique, in which the return time of the light pulse after reflection from an object in the scene provides depth information of the scene of interest [27].

2.3.1 Active Stereoscopy

Active stereo vision is a method of stereo vision that actively uses a light source, such as a laser or structured light, to simplify the process of matching stereo images. The scene is textured by projecting infrared light or another light source on it, reducing the need for an external light source. Thus, this approach is particularly useful in areas with poor lighting or lack of texture.

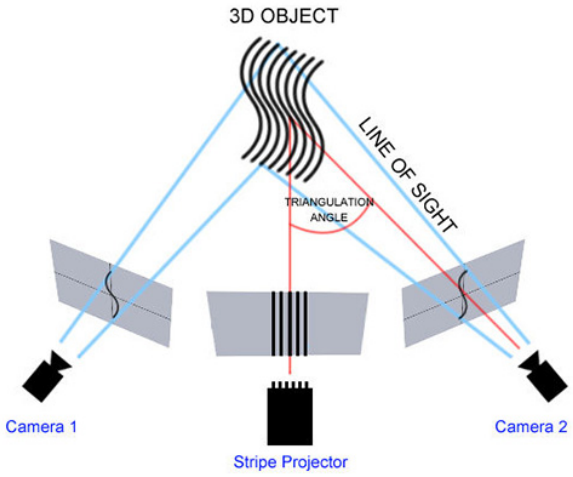


Fig. 2–5: Structured light 3D scanning in a stereo camera system.

Fig. 2–5 shows a schematic of this approach. The infrared (IR) projector emits a series of structured light patterns onto the scene. These patterns are modulated in a way that allows the depth camera to infer the displacement of each pixel in the scene from one frame to the next. Next, the IR camera within the depth module captures the reflected IR patterns from the scene. The shifts in the patterns provide information about the disparities between corresponding points in the stereo image pair. The depth processor analyzes the disparities between the projected and observed IR patterns, using triangulation principles to calculate the distances between the camera and various points in the scene. These distances are converted into depth values and represented in a depth map.

However, there are drawbacks to this approach. The effectiveness of active stereo is reduced in direct sunlight and in places where there is a lot of the same external light source.

2.3.2 RGB-D Camera

The lightweight Intel RealSense D435i depth camera was used to test the SCAR-E rover prototype. It is designed for a variety of applications including robotics, virtual reality and 3D scanning. The main components of the camera are shown in Fig. 2–6:

- **Infrared projector:** Projects non-visible static IR pattern to improve depth accuracy in scenes with low texture

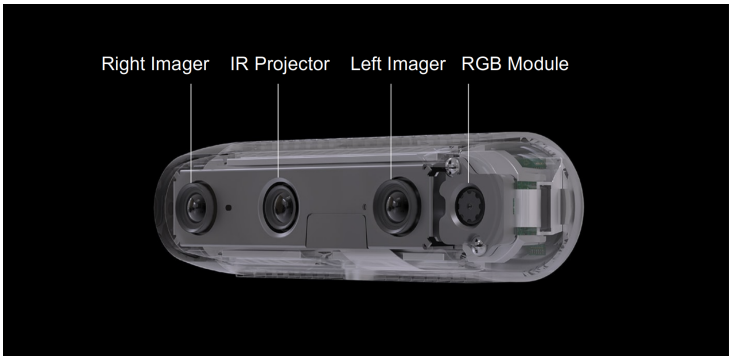


Fig. 2–6: Main components of the Intel RealSense D435i RGB-D camera [28].

- **Stereo cameras:** The D435i features two synchronized global shutter cameras that capture the visual data of the scene. These cameras provide the input for depth calculations using stereo vision techniques.
- **Inertial Measurement Unit (IMU):** The IMU consists of sensors (such as accelerometers and gyroscopes) that provide information about the camera's motion and orientation. This data can be used to enhance the accuracy of motion tracking, especially in dynamic environments.
- **RGB Sensors:** The D435i is equipped with RGB sensors that capture color information of the scene. These sensors work alongside the depth data to provide a complete visual representation.[28]

2.4 Real-Time Appearance Based Mapping

Accurate terrain mapping is a crucial technology for autonomous field robots, especially for climbing robots navigating in challenging terrain. SCAR-E employs six RGB-D sensors placed on its main body to scan the terrain and generate a panoramic view of the surroundings and utilizes Real-Time Appearance-Based Mapping (Real-Time Appearance-Based Mapping (RTAB-Map)) technique for simultaneous localization and mapping (SLAM) and for the generation of a representative simulation environment for testing the graspable target detection algorithm.

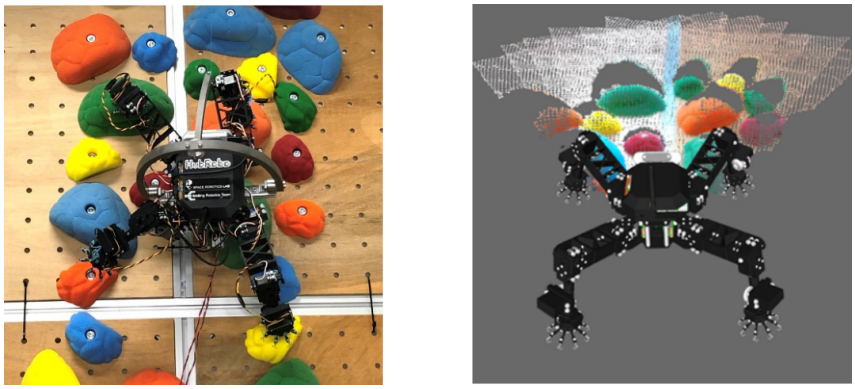


Fig. 2–7: Terrain sensing and mapping by the limbed climbing robot HubRobo [6].

2.4.1 Development

SLAM using real-time appearance-based loop closure detection was gradually developed by M. Labbé and F. Michaud., who introduced a memory management strategy to efficiently handle large-scale maps [29], enhancing the loop closure detection process. The proposed approach optimizes memory usage, ensuring the system's real-time performance. Subsequently, a technique for online appearance-based loop closure detection in SLAM applications was presented in [30]. It emphasizes scalability and long-term operation by enabling loop closure detection in real-time. The scope of loop closure detection was extended in [31] to multi-session scenarios, facilitating large-scale SLAM across time and spatially divided scanning sessions.

Experimental validation of the multi-session SLAM framework involves an autonomous robot with laser rangefinder and RGB-D camera navigating successfully 10.5 km across 11 sessions in an indoor setting, encountering over one hundred people [32]. Finally, RTAB-Map as an open-source software with full ROS integration was introduced 2019 in [33], and a technique to enhance the system's ability to robustly re-localize even under varying lighting conditions by leveraging multi-session information was presented in [34].

2.4.2 Operation

RTAB-Map takes in data from sensors like RGB-D cameras or Lidar, which provide information about the robot's surroundings, including visual images and depth measurements. The algorithm then extracts and saves features from the sensor data, which could include keypoints in images or distinctive points in point clouds. These features help the system to identify unique points in the environment.

Loop closure detection is the identification of previously explored areas when the robot revisits them. The detection of loop closures allows the system to correct accumulated errors in its position estimate in the process of recognition and improve the overall accuracy of the map. In addition to loop closure, RTAB-Map can recognize places it has been to even when the environment has changed slightly.

During the scanning procedure, RTAB-Map constructs a graph that represents the

robot's trajectory and the relationships between different places it has visited. This graph is then optimized using integrated robust graph optimization approaches such as TORO [35], g2o [36] or GTSAM [37] to improve the accuracy of the robot's estimated trajectory and the map. Using the optimized trajectory and loop closure information, RTAB-Map builds a detailed map of the encountered environment.[33] The operation of RTAB-Map is shown in Fig. 2–7 with HubRobo as an example.

2.4.3 Post Processing

RTAB-Map includes several graph refinement post processing measures such as Sparse Bundle Adjustment (SBA) and Iterative Closest Point (ICP) to improve the result. SBA works by iteratively optimizing the positions and orientations of the cameras along with the 3D coordinates of the observed points in the scene. It minimizes the discrepancies between the observed image points and their corresponding projections in the 3D space. These discrepancies, also known as reprojection errors, are the differences between the actual positions of the image points and their predicted positions based on the camera poses and scene structure. The term "sparse" refers to the fact that it operates on a subset of the available data, in contrast to "dense" methods that work with every pixel in the images.[38, 39]

The ICP algorithm, first introduced by Chen and Medioni [40], and Besl and McKay[41], is used to align two 3D point clouds by iteratively refining a transformation that minimizes the error between corresponding points. It operates with a fixed reference point cloud and transforms the source point cloud to best match the reference. The transformation is adjusted through iterations, aiming to minimize the sum of squared differences between the coordinates of matched points. In RTAB-Map, due to the overlapping point clouds of each frame, ICP can be used to refine misaligned neighboring links [33]. The algorithm's steps are described in [42].

2.5 Graspable Target Detection

Methods for the detection of salient shapes are commonly used in robotics for tasks like object recognition in robot hand manipulation. In order to enhance the efficiency of a spine-type gripper designed for safe climbing movement, it is crucial to identify viable shapes that can be securely grasped from the surveyed terrain. These graspable shapes can serve as distinct options for algorithms controlling both locomotion and gripper actions.This chapter introduces a technique to extract these information from sensor data collected by the robot.

2.5.1 Previous Works

Point cloud based grasping point detection and collision prevention has been used in industry for over ten years, for instance in garbage collection robots [43]. Grasp planning by topological object segmentation and categorization using Reeb graphs has been discussed in [44]. For robot grasping in partly occluded areas, a prediction method was developed using Gaussian mixture model in [45]. In recent years there has

been a move towards relying more on neural networks for the detection of graspable targets [46, 47, 48, 49].

The main source for the implementation of the salient shape detection algorithm for application on SCAR-E robots comes from ClimbLab [50], which introduces the MATLAB simulation platform, the associated GitHub repository, and published papers for the IEEE about non-periodic gait planning [7] and the robot HubRobo[6]. Furthermore, internal SRL presentations and other SRL documentations serve as a reference for understanding and implementation.

While RGB image-based techniques have been successful utilized for salient shape detection[51, 52], the effectiveness of color data is limited in natural rocky terrains. A previous study focused on detecting salient features in 3D meshes [53]. In this context, the challenge is not only identifying salient features but also ensuring they are suitable in size and shape for gripper manipulation, especially for a spine-type gripper like the one in HubRobo [54] or in SCAR-E [8]. The preferred shape is convex, rather than a flat or concave shape [7].

The process of the algorithm used is detailed in Dr. Uno's dissertation [55] and is summarized here for a deeper understanding.

2.5.2 Regression Plane Transformation

The original RGB-D sensor's raw point cloud is defined in relation to the camera imager's frame. To ensure the terrain map's independence from the camera's relative pose, the point cloud's reference coordinate system is transformed from a camera-oriented frame to a terrain-oriented frame.

This is accomplished by computing a regression plane using the least-squares method from the point cloud data. The resulting coordinate system on the regression plane aligns the z-axis with the plane's normal direction.

The aim is to compute the centroid of N points on a plane that minimizes the squared distance $\mathbf{r}_{CP}$,

$$\mathbf{r}_{CP} = \begin{bmatrix} x_{CP} & y_{CP} & z_{CP} \end{bmatrix}^T \tag{2–13}$$

from the camera to the points. This plane is associated with the camera frame Σ_C, and CP subscripts denote the reference from the *camera* to the *point*. The centroid becomes the origin of the regression plane based on local terrain information Σ_T, where T represents terrain. The vector from camera-based frame to terrain-based frame is $\mathbf{r}_{CT}$ and is calculated as the average of all $(\mathbf{r}_{CP})_i$ vectors, as Eq. 2–14 states.

$$\mathbf{r}_{CT} = \frac{1}{N} \sum_{i=1}^{N} (\mathbf{r}_{CP})_i \tag{2–14}$$

A deviation matrix $\mathbf{X} \in \mathbb{R}^{3\times N}$ is then formed to capture the error between $\mathbf{r}_{CT}$ and each point, shown in Eq. 2–15.

$$\mathbf{X} = [(\mathbf{r}_{CP})_1 - \mathbf{r}_{CT} \quad (\mathbf{r}_{CP})_2 - \mathbf{r}_{CT} \quad \dots \quad (\mathbf{r}_{CP})_N - \mathbf{r}_{CT}] \tag{2–15}$$

The eigenvector of $\mathbf{X}\mathbf{X}^T$ corresponding to the smallest eigenvalue defines the unitary normal vector $\mathbf{n}_T$ which is perpendicular to the regression plane, originating from the centroid of N points.

The local frame of the regression plane in the terrain Σ_T is determined by selecting unit vectors from the centroid of the regression plane. The unit z-axis of Σ_T, $\mathbf{z}_T$, aligns with the normal vector of the regression plane. The normal vector $\mathbf{n}_T$ points either from the plane to the sky or sub-ground. The direction of $\mathbf{n}_T$ is chosen based on the camera position, leading to the definition of $\mathbf{z}_T$ as:

$$\mathbf{z}_T = \begin{cases} \mathbf{n}_T & \text{if} \quad \mathbf{r}_{CT} \cdot \mathbf{n}_T < 0 \\ -\mathbf{n}_T & \text{if} \quad \mathbf{r}_{CT} \cdot \mathbf{n}_T \geq 0 \end{cases} \tag{2–16}$$

Additionally, unit vectors $\mathbf{y}_T$ and $\mathbf{x}_T$ in the local terrain frame are defined as shown in Eq. 2–17 and Eq. 2–18. This ensures the alignment of the robot's back-to-front direction with the x-direction.

$$\mathbf{y}_T = \frac{\mathbf{r}_{CT} \times \mathbf{z}_T}{\|\mathbf{r}_{CT} \times \mathbf{z}_T\|} \tag{2–17}$$

$$\mathbf{x}_T = \mathbf{z}_T \times \mathbf{y}_T \tag{2–18}$$

The transformation matrix $\mathbf{T}_{CT}$ is constructed using $\mathbf{x}_T$, $\mathbf{y}_T$, and $\mathbf{z}_T$, depicted in Eq. 2–19.

$$\mathbf{T}_{CT} = \begin{bmatrix} \mathbf{x}_T & \mathbf{y}_T & \mathbf{z}_T \end{bmatrix} \tag{2–19}$$

Using $\mathbf{r}_{CT}$ and $\mathbf{T}_{CT}$, a transformation can be established to convert the point cloud from the camera frame Σ_C to the terrain frame Σ_T. This transformation is represented by the equation 2–20.

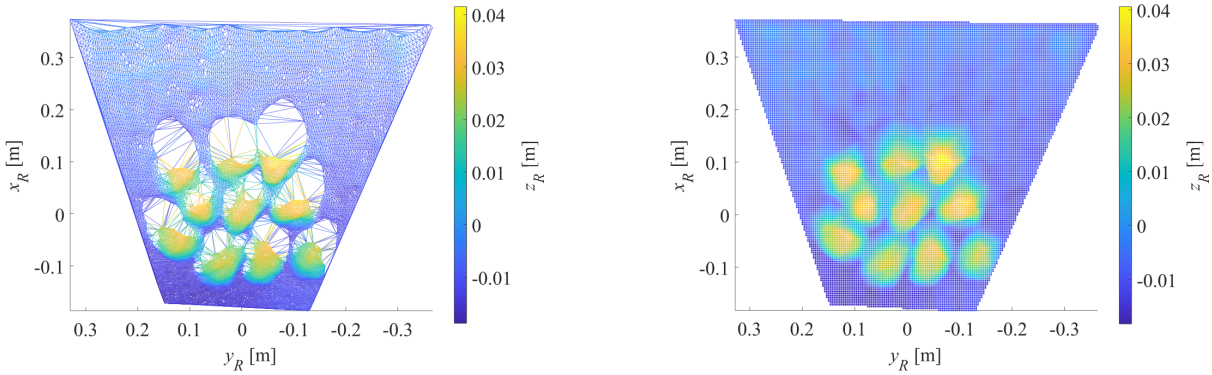
$$\begin{aligned} \begin{bmatrix} \mathbf{P}_T \\ \mathbf{1}^\top \end{bmatrix} &= \mathbf{T}_{CT}^{-1} \begin{bmatrix} \mathbf{P}_C \\ \mathbf{1}^\top \end{bmatrix} \\ &= \begin{bmatrix} & \mathbf{T}_{CT}^\top & & -\mathbf{T}_{CT}^\top \mathbf{r}_{CT} \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \mathbf{P}_C \\ \mathbf{1}^\top \end{bmatrix} \end{aligned} \tag{2–20}$$

Here, $\mathbf{1}$ denotes a vector of ones. This transformation results in Eq. 2–21, which presents the transformed terrain point cloud $\mathbf{P}_T$.

$$\mathbf{P}_T = \mathbf{T}_{CT}^{\top}\mathbf{P}_C - \mathbf{T}_{CT}^{\top}\mathbf{r}_{CT}\mathbf{1}^{\top} \tag{2–21}$$

2.5.3 Interpolation of Occlusion Zones and Homogenization

Certain parts of the terrain that cannot be seen by the robot's camera, which are known as occluded areas. These areas are behind the convex sections of the terrain, hidden from the camera's view. To enhance accuracy in detecting convex shapes and to create a more uniform point cloud density, the algorithm interpolates the terrain shape within these occluded regions.



(a) Delaunay triangulated point cloud. **(b) Linearly interpolated point cloud.**

Fig. 2–8: The point cloud is processed into the Delaunay triangles (a) and the occluded areas are compensated (b) [55].

A Delaunay triangulation mesh [56, 57]is generated based on the transformed point cloud, as shown in Fig. 2–8a. This mesh effectively divides the point cloud into smaller triangles formed by connecting nearby points. Each of these triangles allows the creation of an approximate surface that passes through all the points within it.

Subsequently, the $x_T - y_T$ plane in the terrain based frame is subdivided into a grid. At each intersection point (X_T, Y_T) of the grid, the corresponding z_T coordinate is determined by interpolating within the polygon formed by the Delaunay triangulation. This step ensures that the point cloud's coordinates (X_T, Y_T) align with the grid, and the grid size is adjusted to maintain a similar point density to the original point cloud. The compensated and homogenized point cloud is shown in Fig. 2–8b.

This transformation through grid meshing results in thinning out densely populated areas of the point cloud and complementing sparser regions. This process achieves a uniform point cloud density, improving the overall representation of the terrain.

As the robot moves closer to convex sections of the terrain, the backside of these convex areas gradually becomes visible to the camera. When this happens, linear interpolation is again applied to estimate the shape accurately. If the occluded region does not become visible as the robot approaches, it suggests that the convexity is too large, and the convex section can be treated as an obstacle to avoid.

2.5.4 Voxelization

Voxelization is the process of converting a continuous three-dimensional space into discrete cubic units called voxels. Each voxel represents a specific volume within the space. In the context of terrain analysis, this involves dividing the terrain into small, regularly sized cubes. The conception of terrain array is illustrated in Fig. 2–9.

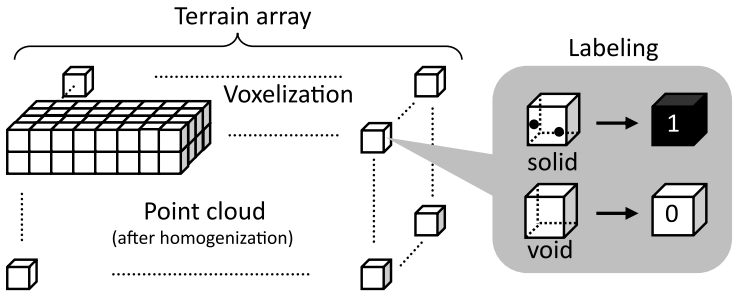


Fig. 2–9: Schematic representation of the generation of the voxelized terrain array [7].

Labeling, in this context, refers to assigning a value or characteristic to each voxel based on certain criteria. When voxelizing a terrain, the labeling process involves determining whether each voxel contains terrain surface points or not.

In simpler terms, voxelization involves making a grid of small cubes over a region, and then for each cube, we decide if it contains part of the ground or not. If it does, we label it as solid, and if it doesn't, we label it as empty. This way, we create a structured representation of the terrain's surface in terms of these labeled cubes.

2.5.5 Gripper Mask

The gripper mask plays a crucial role in the graspable target detection method. It's a 3D array aligned with the terrain array's dimensions. Its purpose is to define the specific positions that the gripper's spine tip can reach. Within the mask, a value of "1" is assigned to locations reachable by the gripper, while "0" is assigned to non-reachable areas. Fig. 2–10 shows the concept of gripper mask designing, where the cross-section of the dark-colored region represents the range that the gripper spine tip can reach from the maximum opening state to the minimum closing state [55].

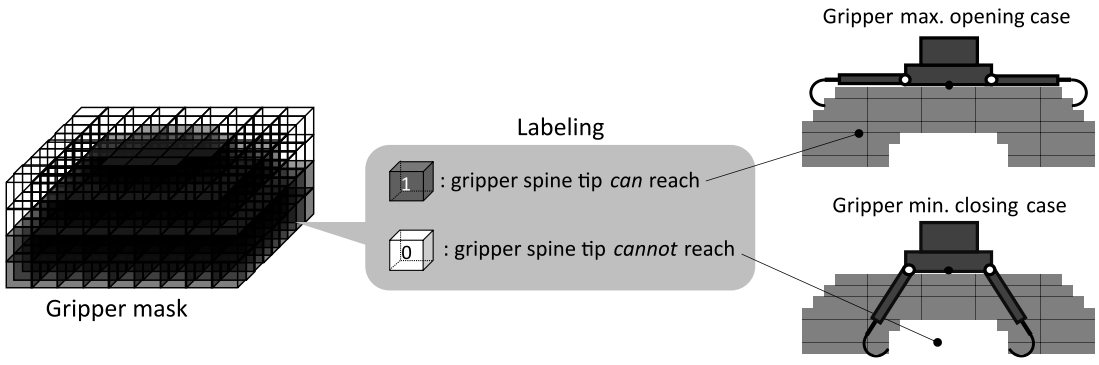


Fig. 2–10: Illustration of the design of the gripper mask[7].

Creating an effective gripper mask involves accounting for various factors. For instance, the pitch of the voxelized mask, i.e. the distance between every two elements, should match the pitch of the voxelized terrain array. For an axisymmetric gripper, like one with circularly arranged fingers, the mask takes a truncated conical shape based on finger motion range.

Regarding the graspable terrain it can be said, while convex shapes are generally preferable, overly flat or sharp conical shapes are suboptimal for grasping, as confirmed through grip and pull tests. A suitable mask considers both geometrical design and gripper performance evaluation.

2.5.6 Voxel Matching

The finished gripper mask is utilized in the final step for scanning the terrain array to identify correspondences. Once a solid voxel in the terrain array has been identified, the algorithm pivots it to the top center of the mask, and extracts a partial array matching the mask's size. By checking if all solid terrain voxels align with the gripper mask's solid regions, the algorithm determines the graspable convex summit. This process is repeated for all solid voxels throughout the entire terrain array.

Fig. 2–11 elaborates on how the algorithm identifies graspable points while excluding overly sharp or flat convex regions. In successful cases (Fig. 2–11(a)), all solid voxels within the extracted terrain array conform to the gripper mask's solid volume. In cases of excessively sharp or flat terrain shapes (Fig. 2–11(b) and Fig. 2–11(c)), certain solid voxels fall outside the gripper mask, indicated in red. The gripper mask's Λ-shaped design detects such mismatches, offering tunable customization of allowable convexity sharpness by adjusting void voxel arrangements.

For complex cases (Fig. 2–11(d) and Fig. 2–11(e)) with solid voxels matching the mask's region but unsuitable for gripping due to inclines or concavities, a threshold helps discard cases with a lower number of solid voxels in the extracted array compared to the successful case (Fig. 2–11(a)). This selective thresholding prevents erroneous selections.

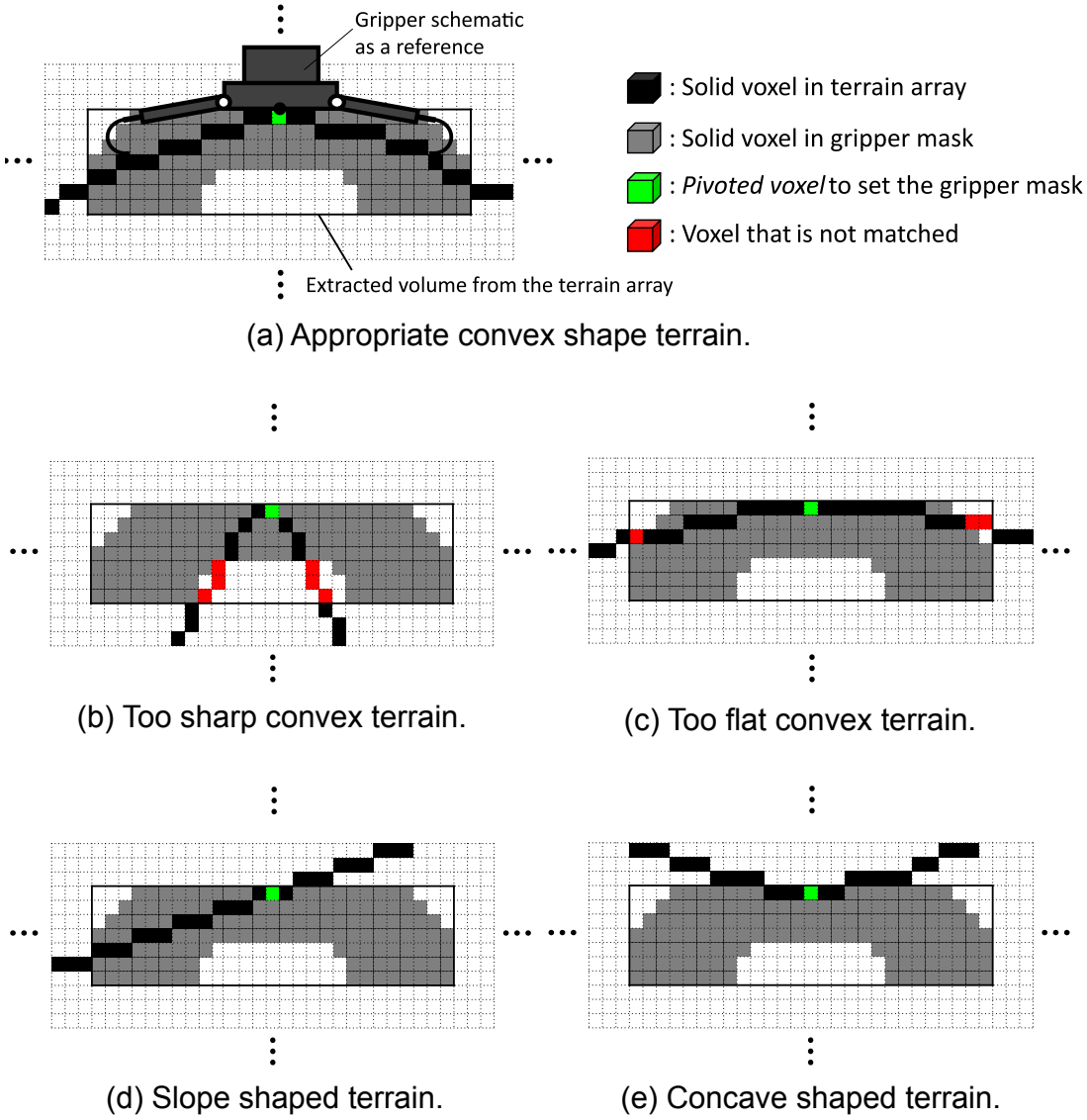


Fig. 2–11: Case study of the convex region detection principle over various terrain. Part of the terrain array and the gripper mask are illustrated from the horizontal cross-sectional view [7].

3 Methodology and Setup

This chapter deals with the process of implementation of the Graspable Target Detection Algorithm for SCAR-E, along with the design and construction of the simulation environment and the field test area. The focus here is on the implementation and adaptation of ClimbLab for panoramic environment scanning in a multi-camera setting.

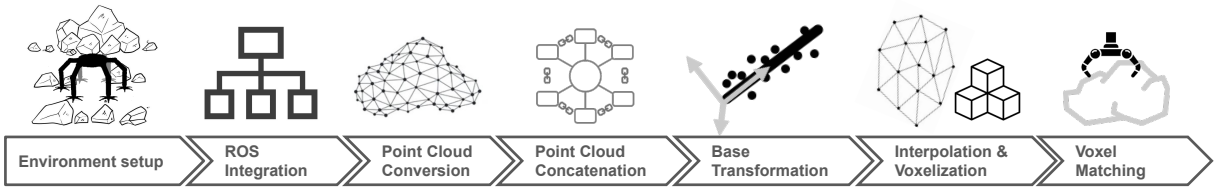


Fig. 3–1: Schematic chart of the workflow.

Fig. 3–1 illustrates the workflow of the process. After setting up the test environment, scanning and refining the data using RTAB-Map, and building a simulation in CoppeliaSim, integration with ROS is achieved using C++. The depth images from the six sensors in the CoppeliaSim scene are individually transformed into 3D point clouds and combined into a panoramic view. The base transformation is the transition from a camera-centred to a terrain-centred image. Occluded areas are interpolated and the point cloud is voxelised to create a terrain array. The generation of the gripper mask and the crucial step of voxel matching, which includes several criteria for the identification of a graspable point. The final step is the visualisation of the results to make them accessible for evaluation.

3.1 Simulation Setup

The initial preparatory measures include the setup of the test environment. Using a hand held Kinect v2 device, the 3 m×3 m test area consisting of fake rocks is scanned through RTAB-Map, followed by a refinement process to enhance the quality of the collected data.

3.1.1 Test Area Setup

The test area (shown in Fig. 3–2a) consists of 10 handmade polystyrene rock models (artificial rocks) coated with acrylic paints in different shades of gray, which gives the surface a rock-like appearance and also increases the hardness of the surface. The vertical height of the largest rock is approx. 67 cm and the one of smallest is about 20 cm. In addition, two real river cobbles, both about 15 cm high, are added to the setting. These are included in the collection only because of their similar shape and texture to the fake rocks, and play no particular role in the evaluation. The distribution of the rocks can be seen in Fig. 3–2b.

(a) Photo of the test area. **(b) Distribution of the rocks.**

Fig. 3–2: Design of the test area. The border of the area is marked with white band, while each rock is assigned a number.

The 12 rocks delimit a flat and empty area of about 1.5 m×1.5 m, which is intended for the placement of the SCAR-E robot, i.e. the model of the robot in the simulation. All rocks are placed in such a way that a certain part be "seen" by the robot.

3.1.2 Kinect Scanning of Test Side

The Kinect v2 is larger and heavier compared to other ToF-sensors, but has a higher depth camera resolution of 512 pixels × 424 pixels. It operates using a ToF measurement principle, with infrared light illuminating the scene and reflecting off obstacles. The device estimates distances via wave modulation and phase detection. Its depth measurement method is detailed in [58].

Integrating the sensor into a robotic system requires understanding its coordinate system. The IR camera axis is assumed to be perpendicular to the Kinect v2's front plate. The open-source Kinect v2 driver libfreenect2 [59] and associated ROS packages [60] were used, with contributions for easy installation on Ubuntu 20.04 LTS. This integration supports accelerated depth processing through Open Graphics Library (OpenGL) and Open Computing Language (OpenCL), in addition to the standard CPU-based method.[61]

The intrinsic parameters of the IR camera are calibrated with the help of a checkerboard and the software by Wiedemeyer [60], which uses the Open Source Computer Vision Library (OpenCV) calibration. According to [61], the average reprojection error after calibration is below 0.5 px.

3.1.3 RTAB-Map Parameter Adjustment

The `rtabmap_ros` package for the ROS version Noetic is used as an interface for the mapping. All parameters can be entered via the command line when starting the RTAB-Map software. Alternatively, ROS launch files are available in which the desired parameters can be set. For Kinect v2, a default launch file is conveniently available in

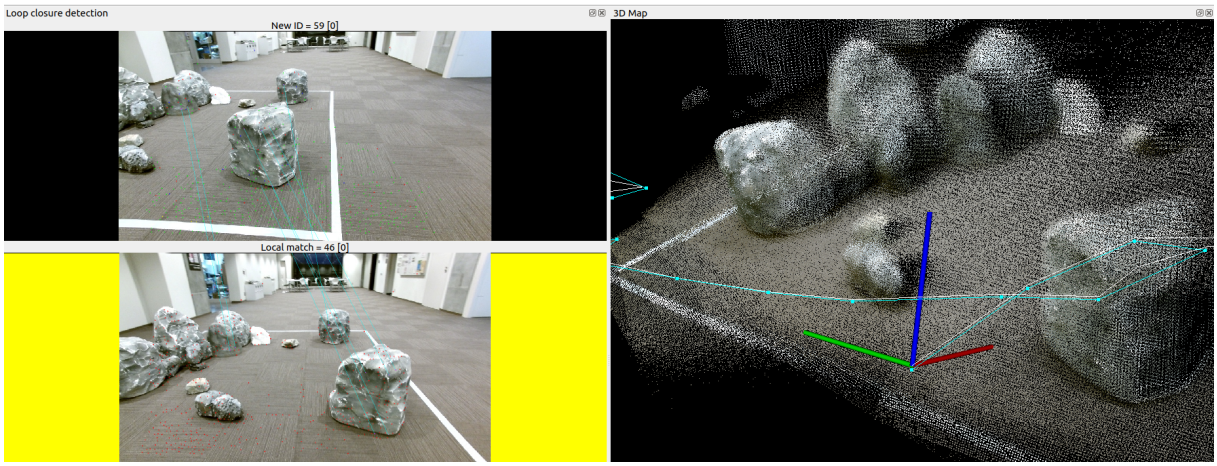


Fig. 3–3: Live footage of the RTAB-Map scanning procedure. Pictures on the left are the RGB images. The resulting point cloud and the camera trajectory (blue line) can be seen in the right half.

the Github repository of `rtabmap_ros`[1]. Additionally, the g2o [2] algorithm is used for minimization non-linear errors.

During the scanning process, RTAB-map automatically recognizes the loop closures in the RGB images of the sensor, shown as green dots in the image above left in Fig. 3–3, and uses those as reference points to estimate the odometry, i.e. the position and orientation of the sensor and the trajectory of the movement (blue line on the right image). If the movement is too fast, or in case of featureless environments or misalignment between the RGB and depth images, the odometry can no longer be calculated and the mapping aborts, which is indicated by a red screen. To recover from this situation, the camera has to be replaced where it failed to compute the odometry. Alternatively, more texture or objects can be added to the scene to add more visual features to track.

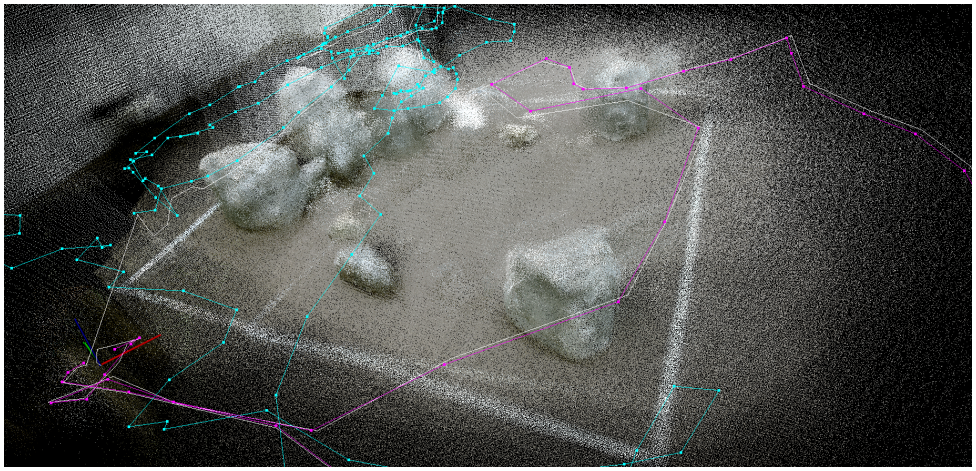
3.1.4 Post Processing and Meshing

The resulting raw point cloud can be exported in Point Cloud Data (PCD) or Polygon File Format (PLY) formats. The whole mapping procedure including the odometry and every image taken by the sensor can also be stored as a database, which can be re-played as input source to test different mapping and loop closure detection parameters.

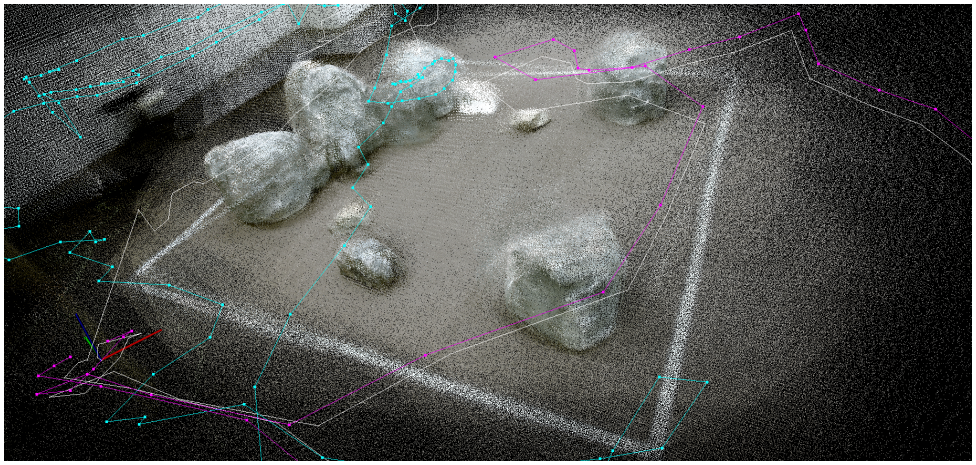
Some misalignments can be clearly seen in the raw point cloud, as shown in Fig. 3–4a. Subsequently, postprocessing measures such as SBA and ICP are undertaken to refine the result. Furthermore, five iterations of detecting more loop closures are carried out in order to find more distinctive points in the cloud within a cluster radius of 50 cm (which is the approximate range of the largest misalignment). In case of visible orientation error, enabling ICP will increase precision of the loop closure and the visual odometry. However, in case of correctly aligned clouds, ICP can decrease

[1] https://github.com/introlab/rtabmap_ros
[2] https://github.com/RainerKuemmerle/g2o.git

the precision, which is why ICP was not enabled. As shown in Fig. 3–4a, there are minor alignment errors in the resulting raw point cloud. Using the refinement steps, Fig. 3–4b shows a overall better alignment, even though the errors could not be eliminated completely.



(a) Raw point cloud before post processing.



(b) After post processing.

Fig. 3–4: Post processing using SBA and loope closure detection results in an overall better alignment of the point cloud.

For the export of the refined point cloud as a textured mesh, the pictures taken by the RGB camera need to be first stored separately in *Bundler* format [62]. Bundler is a *Structure-from-Motion* (SfM) system and uses a modified version of the SBA package, taking a set of images, image features, and image matches as input, and produces a 3D reconstruction of camera and (sparse) scene geometry as output. To limit the number of cameras, we will first conduct the pose filtering by comparing the poses in the same area and only export one picture in a fixed range of 30° and a radius of 50 cm, which results in a cluster of RGB pictures (Bundler cluster) ready to use for the texturing of the mesh.

Next, we can export the 3D cloud. The meshing algorithm using Poisson surface reconstruction [63] is an integrated part of RTAB-Map. The maximum depth is set to 4 m

to filter out points that are further away. The voxel size is set to 1 cm. When meshes are assembled, voxel size is the range in which the vertices of the polygons are merged. This measure also removes outliers and homogenizes the cloud. Finally, the resulting 3D mesh can be imported into MeshLab [64], and the aforementioned Bundler cluster can be applied on the raw mesh using the *"Parametrization + texturing from registered rasters"* tool in MeshLab. The edges are cropped to the size of the test area, and the result is shown in Fig. 3–5.

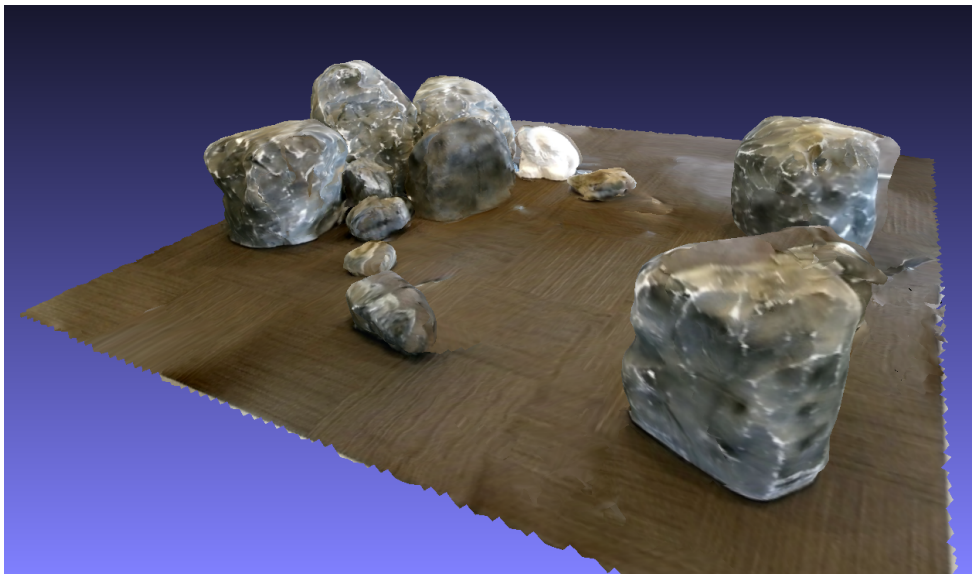


Fig. 3–5: Resulting 3D mesh. Cropped and textured using Bundler cluster.

3.1.5 CoppeliaSim Simulation

The 3D mesh obtained by the Kinect mapping can be imported as terrain in CoppeliaSim. CoppeliaSim [65] is a open source robot simulator built around a distributed control architecture having Python or Lua scripts acting as individual, synchronous controllers. The ROS interface plugin enables full integration into ROS. The SCAR-E model and its inverse kinematics are described in [9] and for ease of use, we use the ROS package `Keystroke` to control the motion of the robot with the keyboard. The hexapod model of SCAR-E is shown in Fig. 3–6.

Six perspective vision sensor models are attached to the main body of SCAR-E model and equipped with the same specifications as the RealSense D435i cameras, as shown in Tab. 3–1. Due to the large size of the concatenated point cloud in the later course, the resolution of the depth image is limited to 640 pixels×360 pixels.

The controller of the depth sensor is written in Lua and is shown in Appx. A.7. It generates the depth image and publishes it in ROS2 in `sensor_msgs/msg/Image` format using the `simVision` and `simROS2` APIs, as shown in Appendix A.7. The terrain mesh model is set to detectable and respondable so that the robot can interact with it. The RGB output of the vision sensor is not needed to generate the point clouds, so it is initially neglected in the ROS integration. Similarly, the terrain texture is omitted as

Tab. 3–1: Specifications of the RealSense camera model used in CoppeliaSim.

Specification	Depth	RGB
Resolution	640 pixels × 360 pixels	1280 pixels × 720 pixels
Field of view	87° × 58°	69° × 42°
Minimum distance	0.28 m	-
Maximum distance	1.50 m	-

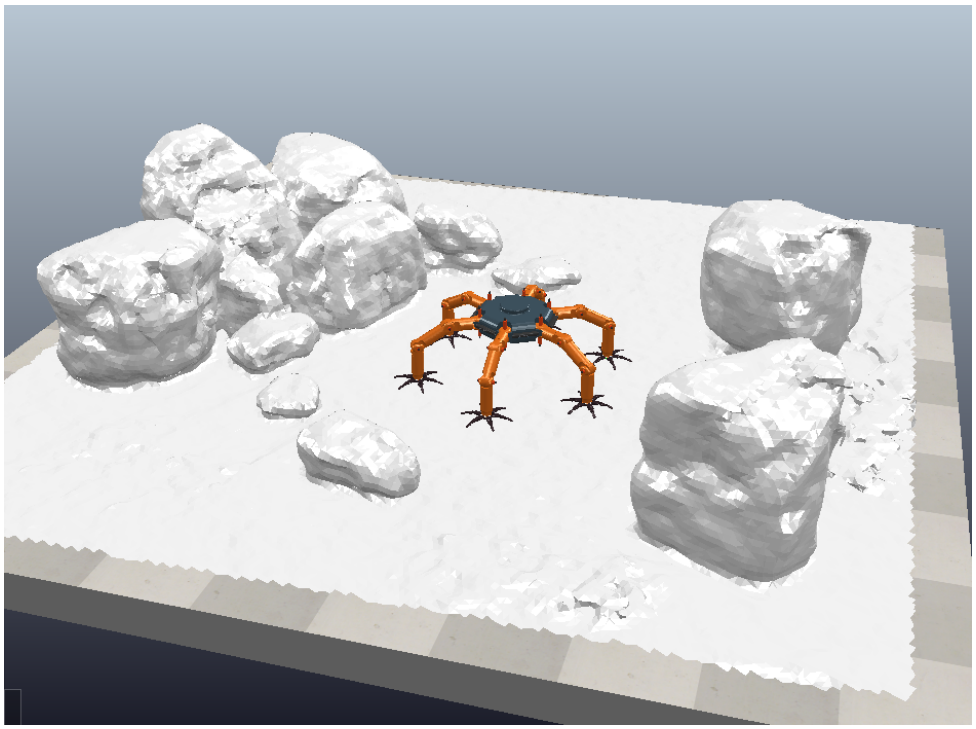


Fig. 3–6: CoppeliaSim scene with the SCAR-E model, equipped with six RGB-D cameras and standing on the terrain mesh model generated using the Kinect device.

it slows down the simulation, but kept in reserve as these may be needed for other applications in the simulation.

3.1.6 ROS Integration

The ROS is a set of open source framework for robotic software libraries and tools. A detailed description of ROS can be found in [66]. One of the key aspects of ROS is its message-passing architecture, which enables communication between different processes in a distributed system. This messaging system is crucial for building modular and scalable robotic applications.

In ROS, software components are called "nodes". Each node represents a distinct process that performs a specific task or computation. Nodes can communicate with each other by exchanging messages. Messages are structured data types used to transmit information between nodes. ROS provides a variety of predefined message

types (e.g., integers, strings, poses, images) that we can use. Nodes communicate by publishing and subscribing to "topics." A topic is a named message channel where nodes can send and receive messages. A node that produces messages on a topic is called a "publisher", while a node that consumes those messages is called a "subscriber". Nodes that generate data (publishers) do not need to know which nodes will use that data. Subscribing nodes express interest in specific topics and receive messages when publishers publish on those topics. This decoupling promotes modularity and flexibility in the system design.

The key advantage of ROS2 over ROS is, among others, it's real-time capabilities. It includes features to better support hard real-time and soft real-time systems, making it more suitable for robotics applications that require precise timing and responsiveness. The inverse kinematics of SCAR-E are programed within a ROS2 framework. However, ROS2 is still under development and some libraries and applications such as RTAB-Map or Point Cloud Library (PCL), which are crucial for this study, are not available in a ROS2 framework, or the compatibility is not sufficient. In these cases, a ROS-ROS2-bridge will be used, which "translates" ROS messages and makes them subscribable for ROS2 applications and vice versa.

The following sections all describe the implementation of applications within a ROS/ROS2 framework in C++ language. The first step involves synchronising incoming raw depth images with camera intrinsic parameters using a message filter. For this purpose, a node is built that bundles the camera intrinsics into an array and publishes them as a message. On the other hand, it filters incoming depth image messages from the six RGB-D cameras in CoppeliaSim, which arrive with different time delays due to different sizes, with the package `image_filter/time_synchronizer`. Subsequently, the node re-publishes the messages with a frequency of 1 Hz.

3.1.7 Point Cloud Conversion

Next, the point cloud conversion node subscribes to the synchronized depth images and camera intrinsic information and performs the conversion, as shown in Listing 3.1.

`u` and `v` are the horizontal and vertical pixel coordinates within an image. The maximum and minimum detection distance are set according to Tab. 3–1. The value of `depth` is given in uint16 color code, where the color ranges from 0 to 65535. The final calculation of the world coordinates of the point cloud requires the center coordinates of the image `c_x` and `c_y` and the focal lengths `f_x` and `f_y`, taken from the camera intrinsics. The resulting point cloud from a depth image is shown in Fig. 3–7.

3.1.8 Multi Point Clouds Concatenation

Fig. 3–8 shows the arrangement of the six RGB-D cameras on the main body of SCAR-E. The six cameras are offset by 60 degrees and positioned 20 cm from the center. With a horizontal field of view of 87°, there is an overlap of 27° between two cameras. The origin of the reference frame for each camera is set to the center of the camera, with the z-axis pointing up perpendicular to the ground and the y-axis pointing forward. This camera-based reference frame Σ_C is to be transferred to a common reference frame

Listing 3.1: Point cloud conversion from depth image using C++.

```
...
    // define maximum and minimum detection distance
    float max_dist = 1.5;
    float min_dist = 0.28;
    // define maximum and minimum depth in uint16 color code
    int min_depth = 0;
    int max_depth = 65535;
    // calculation of the depth to meter ratio
    float depth_to_meter_ratio = (max_dist - min_dist)/(
       ↪ max_depth - min_depth);
    static float toMeters(uint16_t depth) {return depth *
       ↪ depth_to_meter_ratio;}
    // conversion of pixel coordinates in world coordinates
    x = (u - c_x) * depth * f_x;
    y = (v - c_y) * depth * f_y;
    z = toMeters(depth);
...
```

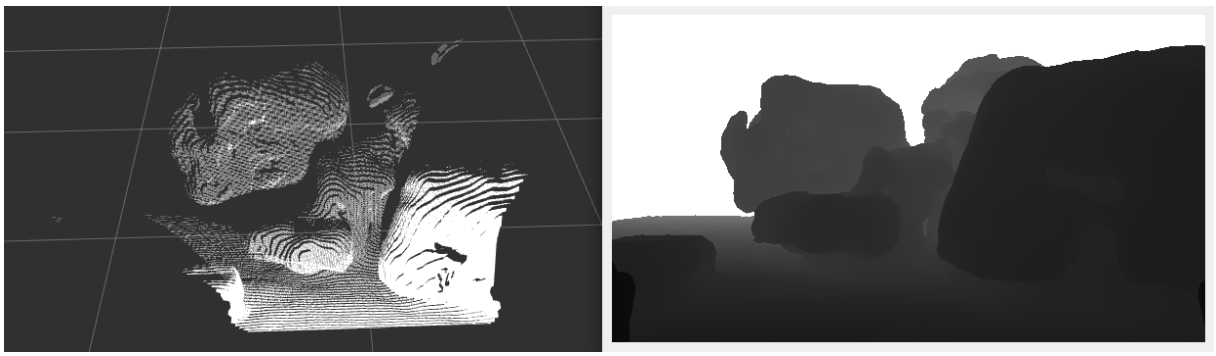


Fig. 3–7: Resulting point cloud (left) from the subscribed depth image (right) message.

with the origin in the center of the body, which we will call subsequently robot-based reference frame Σ_R.

The new reference frame is set to the center of the body, with the y-axis pointing down perpendicular to the ground and the z-axis pointing forward. In case of camera no. 2, for instance, the Euler angle of rotation around x is set to $\alpha = -\pi$, $\beta = 0$ and $\gamma = \pi/3$. Furthermore, the translation in x is $t_1 = -0.173$ and $t_2 = 0.1$ in y direction. Translation in z is $t_3 = 0$.

Eq. 3–1 depicts the resulting rotation matrix $\mathbf{R}$.

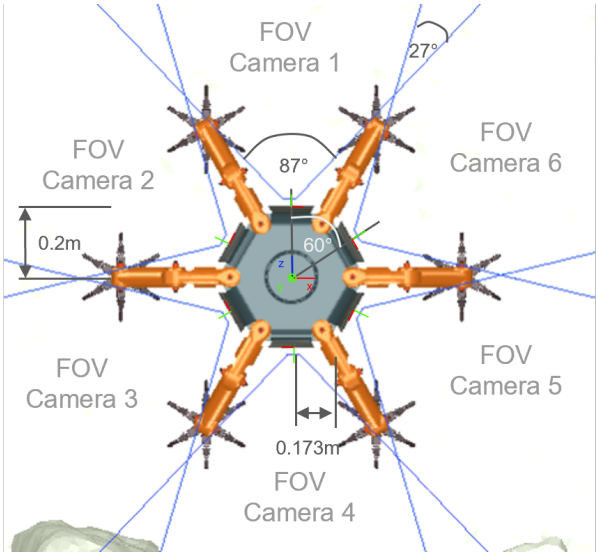


Fig. 3–8: Arrangement of the six RGB-D cameras on the main body of SCAR-E.

$$
\begin{aligned}
\mathbf{R} &= \mathbf{R}_z(\gamma)\mathbf{R}_y(\beta)\mathbf{R}_x(\alpha) \\
&= \begin{bmatrix} \cos\gamma & -\sin\gamma & 0 \\ \sin\gamma & \cos\gamma & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\beta & 0 & \sin\beta \\ 0 & 1 & 0 \\ -\sin\beta & 0 & \cos\beta \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha \\ 0 & \sin\alpha & \cos\alpha \end{bmatrix} \\
&= \begin{bmatrix} \cos\beta\cos\gamma & \sin\alpha\sin\beta\cos\gamma - \cos\alpha\sin\gamma & \cos\alpha\sin\beta\cos\gamma + \sin\alpha\sin\gamma \\ \cos\beta\sin\gamma & \sin\alpha\sin\beta\sin\gamma + \cos\alpha\cos\gamma & \cos\alpha\sin\beta\sin\gamma - \sin\alpha\cos\gamma \\ -\sin\beta & \sin\alpha\cos\beta & \cos\alpha\cos\beta \end{bmatrix}
\end{aligned}
\tag{3–1}
$$

and Eq. 3–2 shows the translation vector $\mathbf{t}$

$$
\mathbf{t} = \begin{pmatrix} t_1 \\ t_2 \\ 0 \end{pmatrix} \tag{3–2}
$$

set into the extrinsic matrix $\mathbf{T} = [\mathbf{R}|\mathbf{t}]$ (see Eq. 2–1), we obtain the transformed point cloud $\mathbf{P}_R$ stated in Eq. 3–3.

$$
\mathbf{P}_{R,i} = \mathbf{P}_{C,i} \cdot \mathbf{T}_i, \qquad i = 1, ..., 6, \tag{3–3}
$$

where i is the indexing of the camera. The concatenation of the six individual point clouds is the addition of the transformed point clouds in Eq. 3–4.

$$\mathbf{P}_{\mathbf{concatenated}} = \sum_{i=1}^{6} \mathbf{P}_{R,i} \tag{3–4}$$

The resulting concatenated point cloud is shown in Fig. 3–9 and is published under the topic `/merged_pcd`.

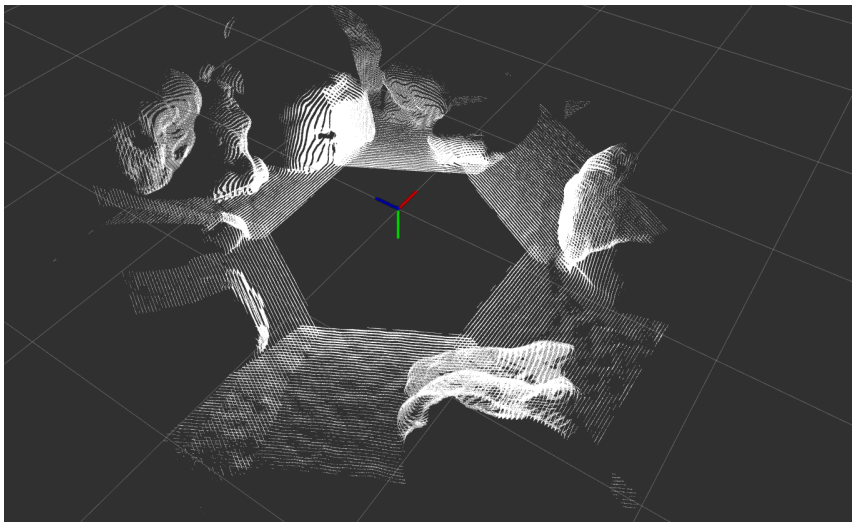


Fig. 3–9: The panoramic point cloud after transformation and concatenation.

3.2 Implementation of Target Detection Algorithm in C++

The following section covers the implementation of the Graspable Target Detection algorithm in C++. Since some essential libraries for processing point clouds are not available or not mature in Robot Operating System 2 (ROS2), such as the Point Cloud Library (PCL), the algorithm is developed in a ROS Noetic environment.

First, we need to introduce some terminologies in order to prevent confusion:

- **Raw Cloud**: Input point cloud of the terrain which is in PCD file format.
- **Transformed Cloud**: Coordinate transformed point cloud aligned with the normal of the regression plane.
- **Interpolated Cloud**: Point cloud with occluded areas compensated by the interpolation and homogenization step.
- **Solid Voxel**: Voxel denoted with 1 which represents a solid space
- **Void Voxel**: Voxel denoted with 0 which represents an empty space.
- **Voxelized Terrain Array**: Voxel array which is the point cloud transformed into discrete equally sized cubes.
- **Gripper Mask**: Voxel array which represents the gripping extent of the gripper prototype.

- **Subset**: A portion of the Voxelized Terrain Array that has the same dimensions as the Gripper Mask.
- **Threshold of Solid Voxels**, denoted as TSV: Threshold of the minimum number of solid terrain voxels that is required to be in a Subset to prevent erroneous predictions like in Fig. 2–11(d) and Fig. 2–11(e).

The ROS node `detect_graspable_points` is initialized in the `main()` function by the `ros::init()` function and the interface for creating subscribers and publishers is created with `ros::NodeHandle`. The node subscribes to a single topic, namely the aforementioned concatenated point cloud `/merged_pcd`. The node runs in a loop by using `ros::spin()` in the end of the node.

A header file `.hpp` adds all necessary libraries to the code and initializes and explains all functions and required variables. Furthermore, it predefines all ROS topics to be published.

The constructor of the node calls all necessary functions to process and analyze the raw point cloud based on the original concept of ClimbLab, which was described in Chapter 2.5, adding some improvements. These functions are presented in detail below.

3.2.1 Coordinate Transformation

The coordinate transformation is divided into two functions.

- **pcd_least_squares_plane()** computes the centroid and normal axis of the raw point cloud's regression plane.
- **pcd_transform()** aligns the axes of the new terrain-based reference frame using the normal vector of the regression plane.

The quintessence of the function `pcd_least_squares_plane()` is shown in Listing A.1 corresponding to the following steps:

1. Compute the Centroid c using pcl::compute3DCentroid.
2. Substract the each element of the centroid vector from each column of the cloud matrix. Deviation Matrix $\mathbf{X}(i,j) = \mathbf{P}(i,j) - \mathbf{c}(i)$.
3. Compute the product of Deviation Matrix.
4. Compute the eigenvectors and eigenvalues of PD.
5. Take only the real part of the eigenvalues and eigenvectors.
6. Find the smallest eigenvalues and its corresponding eigenvector.
7. Normal Vector of the Regression Plane is the Eigenvector corresponding to the smallest eigenvalue.

We first compute the centroid of the raw point cloud using a PCL library, which returns a vector pointing to the centroid. The subsequent for loop corresponds to Eq. 2–15

and derives the deviation matrix $\mathbf{X}$. We will then calculate the eigenvalues of $\mathbf{X}\mathbf{X}^T$. For symmetric matrices, we have a basis of eigenvectors, and every eigenvalue is real. Subsequently, the algorithm takes the real part of the eigenvalues and eigenvector, and the eigenvector corresponding of the smallest eigenvalue is assigned to the normal vector $\mathbf{n}_T$, which is also the z-axis of the new reference frame.

Listing A.2 shows the function for the coordinate transformation `pcd_transform()`. y- and x-axis of the new reference frame are computed following the Equations 2–17 and 2–18. The elements of the rotation matrix are then assigned to the coordinates of the new frame. Finally, the frame transformation according to Eq. 2–21 is performed, resulting in the new, Transformed Cloud.

3.2.2 Point Cloud Interpolation

The function `pcd_interpolate()` compensates the occluded areas and is presented in simplified pseudocode in Listing A.3. The function first calculates a suitable grid size, based on the size of the input cloud. This grid size GS is crucial for creating a uniform grid structure for interpolation.

$$GS = \frac{1}{\sqrt{\frac{N}{w_x \cdot w_y} \cdot \frac{5}{3}}} \tag{3–5}$$

Here, the point cloud size N represents the number of points in the input point cloud, w_x and w_y denote the dimensions of the point cloud area in the x and y directions, respectively. The coefficient of $5/3$ is chosen to maintain a comparable point density in the interpolated data, akin to the input point cloud.

Next, the function generates grid vectors for both the x and y dimensions. These vectors define the coordinates of the grid points where interpolation will be performed.

Finally, a linear Delaunay triangle interpolator from the C++ library `LibInterpolate` is utilized to perform the interpolation.

The code performs the interpolation on the grid points, yielding interpolated values for the z-coordinate. It filters out any points interpolated to $z = 0$, as they are considered outside the original range of data. The resulting interpolated grid vector, denoted in the code as `x_grid_vector`, `y_grid_vector` and `interp_z` are then transformed back into a 3-column matrix format, which is the finished Interpolated Cloud.

3.2.3 Voxelization

The function `pcd_voxelize()` takes an input point cloud and the voxel cube size cs (which is set to 0.01 m by default) as parameters. The main steps of the function involve data preparation, calculating the grid size, applying voxel grid filtering, and determining voxel occupancy based on the cube size and point coordinates. It utilizes the `pcl::VoxelGrid` library and voxelizes the input point cloud by dividing it into equally sized cubes and retaining only the centroids of the cubes that contain points. The resulting voxelized data is stored in a 3D matrix.

Listing A.4 demonstrates the voxelization of the input point cloud by marking the corresponding voxel as occupied based on the cube size and the coordinates of each point in the filtered point cloud. Next, the voxel indices are assigned to the current point. These indices determine which voxel in the 3D grid the point belongs to. The calculation involves dividing the difference between the point's coordinates and the minimum coordinates by the cube size. The addition of $cs/4$ helps ensure that the point is assigned to the voxel with its centroid, reducing the likelihood of a point falling on the edge of a voxel.

The corresponding voxel in the 3D grid is marked as *solid* by setting its value to 1. This indicates that there is at least one point within that voxel.

3.2.4 Gripper Mask

The dimensions of the gripper model used to design the gripper mask is shown in Fig. 3–10. This is a simplification of the soft grip solution for the SCAR-E rover. The number of segments per finger has been limited to two and only the foremost segment has a spine. Although the capability to grip and adapt to the terrain increases with the number of finger segments and spines, the basic operation and functionality of the gripper remains consistent. This simplification is intended to ensure the reliability of the concept. All parameters and their corresponding values are listed in Tab. 3–2.

The C++ function `creategrippermask()` generates a 3D mask representing a gripper's geometry within a voxelized space. It calculates and sets the elements of the mask to 1 or 0 to represent solid and void regions, respectively, based on the gripper's parameters and voxel size.

The side length of the gripper mask L_{GM}, denoted as `gripper_mask_size`, is computed by adding the radius of the gripper palm between the first finger joints and the sum of the lengths of the finger segments:

$$L_{GM} = (\frac{\varnothing_{FJ}}{2} + L_F + L_S) \cdot 2. \tag{3–6}$$

The maximum height of the gripper mask H_{GM} is equal to the vertical distance between the tip of the spine and the bottom of the palm when the gripper is closed:

$$H_{GM} = H_c. \tag{3–7}$$

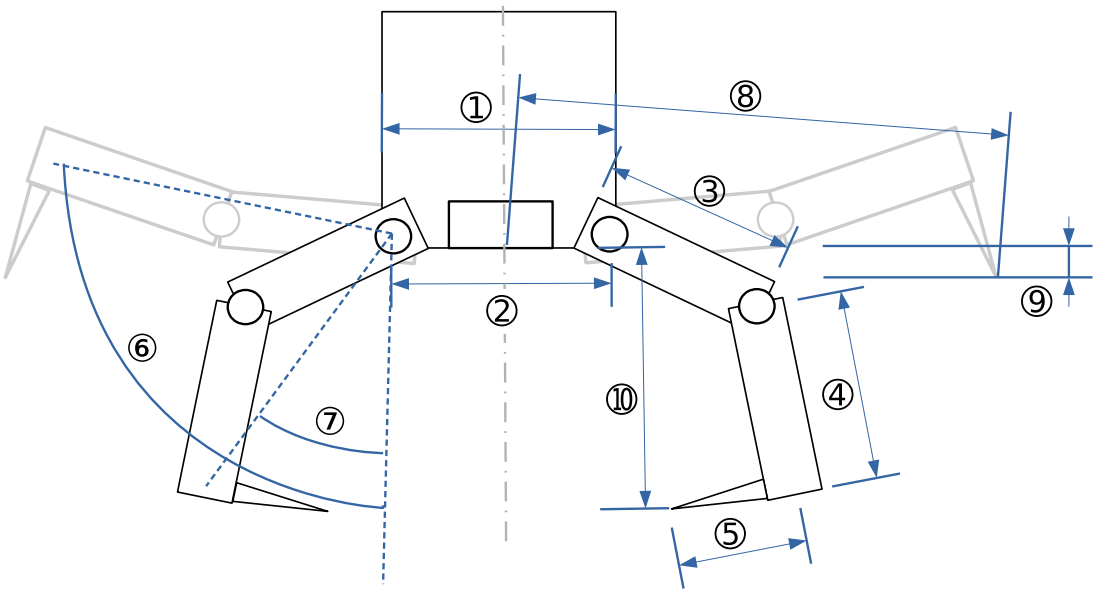


Fig. 3–10: Geometrical parameters of the gripper which are relevant for the creation of the gripper mask.

Tab. 3–2: Parameters of the gripper model and their value.

No.	Variable	Denoted as	Explanation	Value
1	Palm diameter	$\varnothing_P$	Diameter of gripper's palm	71 mm
2	Palm diameter of finger joints	$\varnothing_{FJ}$	Distance between two opposite first finger joints	92 mm
3	Finger length	L_F	Length of the first finger segment	40 mm
4	Spine length	L_S	Length of the last finger segment	41 mm
5	Spine depth	D_S	Length of the spine itself	5 mm
6	Opening angle	θ_o	Maximum opening angle	85 deg
7	Closing angle	θ_c	Maximum closing angle	10 deg
8	Opening spine radius	R_{os}	Distance from the center of palm to the tip of the furthest spine	136 mm
9	Opening spine depth	D_{os}	Distance from the horizontal plane to the tip of the spine when opened	5 mm
10	Closing height	H_c	Vertical distance between the tip of the spine and the bottom of the palm when closed	90 mm

Additional parameters are required to determine the solid and void areas of the gripper mask. These parameters include the top solid radius, clearance, and bottom void radius. The top solid radius R_T of the gripper mask is half of the palm's diameter:

$$R_T = \frac{\varnothing_P}{2}. \tag{3–8}$$

The maximum height of the void space H_{cl} in the upper part of the gripper mask, denoted as `gripper_mask_clearance` is computed as follows:

$$H_{cl} = \frac{L_{GM} - \varnothing_P}{2} \cdot tan(90° - \theta_o). \tag{3–9}$$

The radius of the void space R_B in the lower part of the gripper mask, denoted as `gripper_mask_bottom_void_radius`, is computed as follows:

$$R_B = \frac{\varnothing_P}{2} + (H_{GM} \cdot tan(\theta_c)). \tag{3–10}$$

The creation of the gripper mask as an array is done by the loop in Listing A.5. The radius of the grippable area R_g changes as the z-value changes and is computed by the equation in Eq. 3–11:

$$R_g = R_T + (\frac{L_{GM}}{2} - R_T)\frac{z}{H_{cl}}. \tag{3–11}$$

The Inner Unreachable Radius $R_{u,i}$ is formed following the formula for half chord length of a circle [3].

$$R_{u,i} = \sqrt{2 \cdot R_B(R_B - (H_{GM} - z)) - (R_B(H_{GM} - z))^2} \tag{3–12}$$

The radius of the top unreachable area $R_{u,t}$ is optional, it is only necessary if the gripper's maximum opening angle is larger than 90°

$$R_{u,t} = \frac{L_{GM}}{2} - (\frac{L_{GM}}{2} - (R_{os} + D_S)) \cdot \frac{z}{D_{os}}. \tag{3–13}$$

The distance of each voxel from the centers of the respective layer is computed by:

$$DC = \sqrt{(\frac{L_{GM}}{2} - x)^2 + (\frac{L_{GM}}{2} - y)^2}. \tag{3–14}$$

In order to convert the parameters specified in mm into discrete parameters P_{voxel} in voxel, all continuous parameters Eq. 3–6 - Eq. 3–14 P_{float} are multiplied by a voxel ratio R_{voxel} that depends on the voxel cube size cs.

[3] Chord length of a circle: $2\sqrt{2Rh - h^2}$

$$R_{voxel} = \frac{1}{cs \cdot 1000} \tag{3–15}$$

$$P_{voxel} = \lfloor P_{float} \cdot R_{voxel} \rceil \tag{3–16}$$

The loop in Listing A.5 iterates through the voxels within the gripper mask and for each voxel, it calculates the distance from the center of the mask layer and determines whether it falls within the grippable, unreachable, solid, or void region based on the calculated radii and clearances. Voxels that belong to the solid regions are marked as 1 in the Gripper Mask, while others remain 0.

The dimensions of the finalized gripper mask is shown in Fig. 3–11.To prevent erroneous predictions in the areas of depressions or steep side walls, a feature in the form of an additional void layer H_{add}, denoted as `extra_sheet` is added to the Gripper Mask. The auxiliary top area will penalize the assessment of the graspability in the case that terrain voxels are detected in this area.

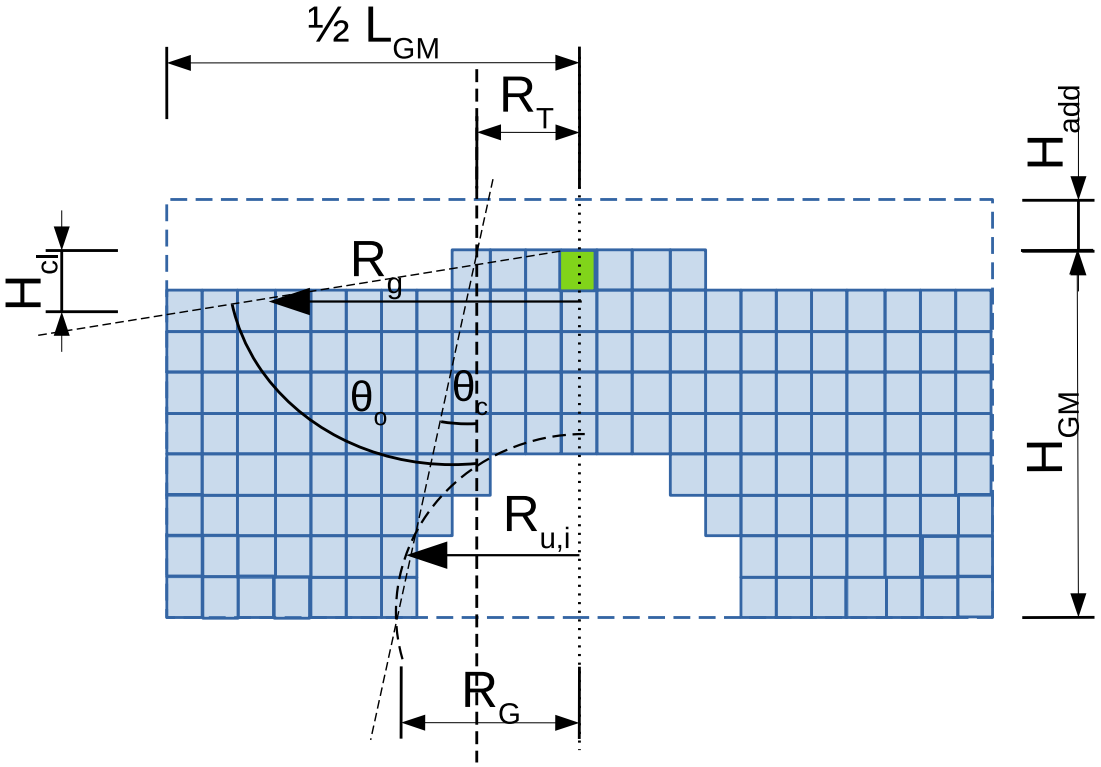


Fig. 3–11: Resulting Gripper Mask and dimensions.

3.2.5 Voxel Matching

The final step in graspable target detection is the voxel matching function, which is divided into several steps and sub-functions:

1. **Voxel_Clip()** prevents the gripper mask from protruding from the Voxelized Terrain Array (VTA) during the matching algorithm. It crops the VTA at all sides by the half size of Gripper Mask and sets the values to zero.

2. **Find Solid Voxels**: find all "ones" (solid voxels) in the search voxel array and change indices to subscripts of solid voxels.

3. **Pivot Point Transformation** Transforms the Pivot Point of the Gripper Mask to the upper center of the array.

4. **Voxel_Extract()** extracts a **Subset** $\mathbf{P}_{subset}$ array of voxels of the same size as the Gripper Mask from the larger VTA.

5. **Voxel_Compare()** compares the number of Solid Voxels within the Subset ("Subset Solid Voxels", denoted as SSV) with the number of intersections of SSV and Solid Voxels within the Gripper Mask $\mathbf{P}_{GM}$ which corresponds to the gripping extent of the gripper. This intersection is called in the following "Gripper Solid Voxels", denoted as GSV. It outputs the probability of **Graspability** score[4] G which is defined as:

$$G = \frac{\mathbf{P}_{subset} \cdot \mathbf{P}_{GM}}{SSV} = \frac{GSV}{SSV} \tag{3–17}$$

   The Graspability score thus represents the proportion of the intersection between the Subset and the Gripper Mask within the total set of Solid Terrain Voxels within the Subset.

6. **Thresholding**: Subsets that have a lower number of Solid Voxels than the Threshold of Solid Voxels TSV are penalized so that the **Penalized Graspability** value G_P of the Pivot Voxel is reduced according to the difference between SSP and threshold value TSV. For initial testing, TSV is set to 120, which is an empirical value.

$$G_P = G - \frac{TSV - SSV}{TSV} \tag{3–18}$$

7. **Output**: The function return a four columns array called **Voxel Coordinates of Graspable Points** where column 1, 2 and 3 denote the x, y and z coordinates of the point and the fourth column the corresponding Graspability value ranging from 0.0 to 1.0.

Listing A.6 demonstrates the function using a simplified pseudocode. The value of Graspability is therefore formed from the ratio of the intersection of the grippable extent of the Gripper Mask with the terrain and the total terrain voxels present in the Subset.

[4] In the following we refer to the variable G when Graspability is written with a capital letter. Otherwise, graspability means the general suitability of the terrain for grasping with the gripper.

3.2.6 Convex Shape Detection

In order to limit the range of graspable areas to only convex surfaces and to limit the false positive rate, a combined measure was taken to find optimally graspable surfaces. It is assumed that the intersection of graspable areas found using the gripper mask and the peaks of the convex surfaces provides optimal graspability:

$$\text{Optimum graspability} = \text{Graspability score maxima} \times \text{peaks of convex surfaces}$$

The implementation of convex surface detection using the principal curvature will be explained in the following briefly. A more detailed view on differential geometry and in particular, curvature of surfaces is provided by Kühnel [67].

3.2.6.1 Principal Curvature

Consider a differentiable surface in 3-dimensional Euclidean space. At each point P on this surface, it is possible to define a unit surface normal vector $\mathbf{n}$, which is perpendicular to it. The normal curvature is the curvature of a plane curve that results from a normal section, i.e. from an intersection of the given surface with the plane determined by the surface normal vector and the given tangential direction.

The normal curvature is assigned to each tangential direction, i.e. each direction in that a tangential vector can be assumed at this point on the surface. The normal curvature can be seen as a continuous periodic function of the angle between a fixed tangential vector and the direction. The minimum and maximum values of these curvatures are called principal curvatures k_1 and k_2.

The Gaussian curvature (K) is the product of the principal curvatures:

$$K = k_1 \cdot k_2 \tag{3–19}$$

If both principal curvatures are of the same sign, K will be positive: $K > 0$, then corresponding point is called an elliptic point. At such points, the surface will be dome shaped.

The implementation of the curvature estimation is shown in Listings 3.2 in pseudocode.

The searching radius for curvature is set to 0.09 m, which is roughly the mean radius of the gripper's gripping extent. In a point cloud of a terrain surface, a uniform orientation of the normal vectors is not defined, e.g. it is not clear to the algorithm which is the "air side" and which is the "rock side" of the surface. This problem is only partly solved by sorting out those points with minus z oriented normal vectors, which eliminates erroneous concave shapes, but would also lead to some incompleteness in practice, as shown later in Fig. A2–2.

Listing 3.2: Function for the detection of convex shapes. In simplified pseudocode.

```
In {voxelized_terrain_array, searching_radius}
Out {convex_peak_cloud}

curvature1 =// compute the maximum normal curvature within the
   ↪ searching radius;
curvature2 =// compute the minimum normal curvature analogously
   ↪ ;
normal_z =// compute the normal vector of the surface and take
   ↪ the z;

for (all points p in the voxelized_terrain_array) {
    //the normal z value should point outward, so in order to
       ↪ find convex shaped peaks and avoid concave areas, we
       ↪ require points whose normal z value is positive.
    if ((curvature1 * curvature2) > 0 && normal_z > 0.0) {
        convex_peak_cloud.push_back(p);
    }
}
```

3.2.6.2 Intersection Criterion

The implementation of the intersection of graspable areas and convex shapes maxima is shown in Listings 3.3. It takes those points with a higher Graspability score

Listing 3.3: Function for the intersection of convex shapes and graspable areas.

```
In {voxel_coordinates_of_graspable_points, convex_peak_cloud,
   ↪ graspability_threshold, distance_threshold}
Out {intersection_cloud}

for (all points p in voxel_coordinates_of_graspable_points) {
            if (graspability >= graspability_threshold){
                cloud1.push_back(p);
            }
        }
for (all points p in cloud1) {
        for (all points q in cloud2) {
            distance = pcl::euclideanDistance(p, q);
            if (distance < distance_threshold) {
                intersection.push_back(p);
                intersection.push_back(q);
            }
        }
    }
```

The Graspability Threshold is set according to the analysis in Chapter 5.2.1.5. For the initial testing, it is set to 0.8.

3.2.7 Visualization

The resulting Voxel Coordinates of Graspable Points is published as a ROS point cloud message, where the ranges of the Graspability score represent a variety of colors ranging from red (non-graspable) to dark green (fully graspable). The color white is chosen for the ground below a certain z value (Ground Threshold). The flat ground is defined to be parallel to the computed regression plane of the point cloud. This colored point cloud is called subsequently "Graspability Map".

The results of the convex shape detection and the intersection criterion are visualized in a separate visualization as blue and dark green points on top of the raw point cloud. This map is called subsequently "Curvature Map".

The visualization is carried out with RVIZ, which is a visualization program within the ROS framework.

4 Results

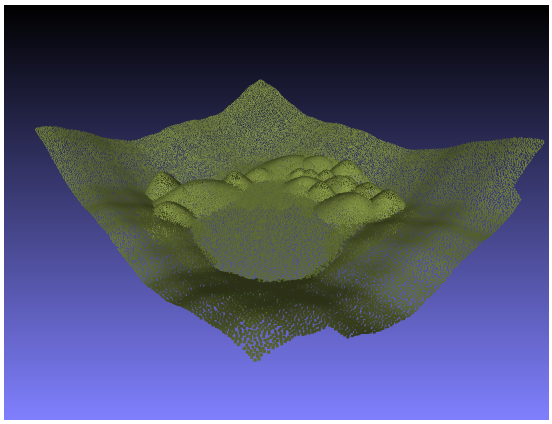
This chapter presents the results of the development of the graspable target detection algorithm and the test results on several test scenarios. The aim is to present the ability of the algorithm to recognize graspable targets in different terrains, both on a previously scanned map and for in-situ cases, especially using the setup for the SCAR-E robot in CoppeliaSim described in Chapter 3.1.5.

A simple field test is carried out to verify the statements about the graspability of the target and to derive the accuracy of the algorithm. The running time of the algorithm is then briefly presented. The comparison of the results between the individual cases and the inclusion of the field test in the evaluation of the performance of SRL-GTD is conducted in the Discussion Chapter 5.

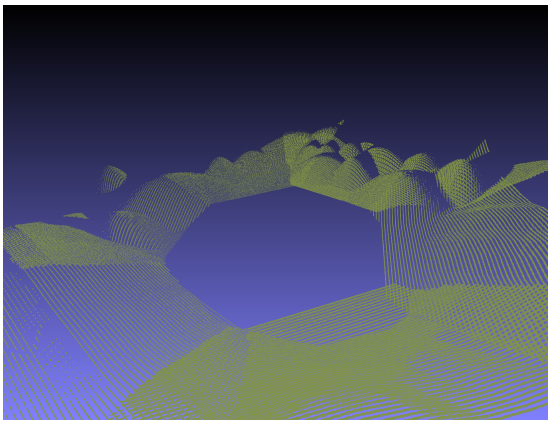
4.1 Test Scenarios

For lunar and planetary exploration, as well as for terrestrial applications in difficult terrain, there are two ways to process the terrain information to be examined. In case of previously known terrain, the 3D model can be converted into a suitable point cloud, which can then be examined in its entirety by the graspable target detection algorithm.

However, terrain information obtained by remote sensing often has the problem of low resolution rates. Thus, once the robot enters unknown terrain, the rover must decide *in situ* where to go next. To evaluate the algorithm, two different cases are considered: **pre-scan** and **in-situ** (or **real-time**) application. This is illustrated using two different point clouds from the same scenario as the input point cloud for the algorithm, as shown in Fig. 4–1.



(a) Pre-scanned point cloud from remote sensing.

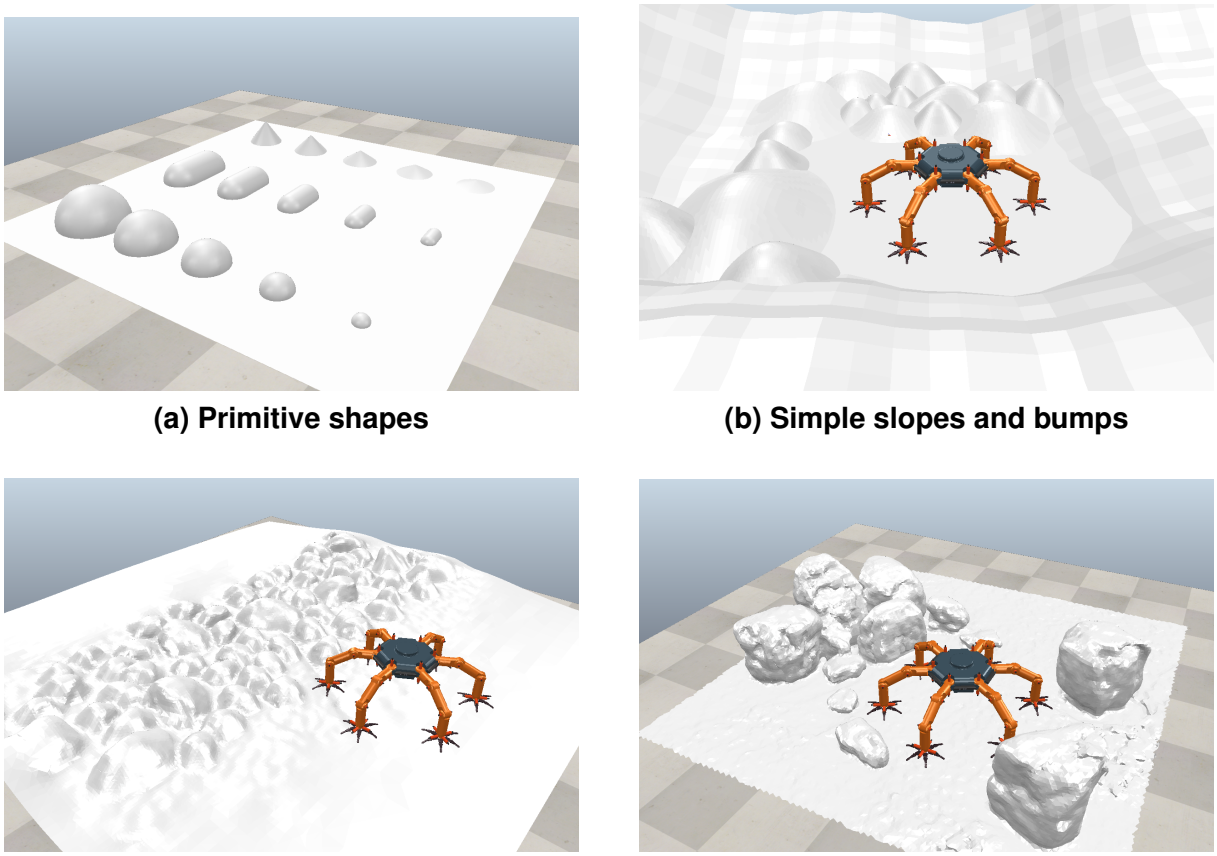


(b) Real-time point cloud from in-situ exploration.

Fig. 4–1: Input point clouds obtained from different sensing methods.

Four distinct test scenarios are conducted to evaluate the algorithm's performance, as shown in Fig. 4–2. The first scenario represents a simple plane with three different

geometries, each in five different sizes. This primitive scenario is intended to test the basic function of the gripper mask, as well as the function of calculating the regression plane. The second scenario involves a simple simulated terrain created in CoppeliaSim with slopes and minor bumps. The point cloud shown in 4–1b will be used as input. The aim of this scenario is to test the occlusion compensation step of the algorithm. In the third scenario, a scan of a bouldering wall with climbing holds of different sizes placed close to each other is used. The terrain in such constellation does not represent a difficult obstacle for a hexapod robot to overcome, but each step must still be taken carefully so that the gripper arm can fix itself on one of the climbing holds and does not get stuck between two of them.



(a) Primitive shapes **(b) Simple slopes and bumps**

(c) Bouldering wall **(d) Artificial rocks**

Fig. 4–2: Four scenarios of different terrain on which the algorithm is tested.

In the last scenario, the Kinect pre-scanned map consisting of artificial rocks, as presented in Chapter 3.1.1, is analyzed using the algorithm, as well as the in-situ case of the same scenario for comparison. This scenario represents a difficult terrain with some high and almost insurmountable obstacles in the form of vertical rocks. In order to find a way over it, the algorithm should be capable to show the graspable targets on the rocks so that the robot can climb over them. Tab. 4–1 briefly sums up the four scenarios and their purposes.

The algorithm's parameters used for the respective scenarios differ only minimally from each other and are listed in Tab. 2–1.

Tab. 4–1: Test scenarios and their source and purposes.

Test scenario	Pre-scan	In-situ	Purpose
1. Primitive shapes	✓		Test the basic function and constraints of the algorithm.
2. Slopes and bumps		✓	Test the occluded area compensation in real-time.
3. Bouldering wall	✓	✓	Test the algorithm in a low obstacle terrain.
4. Artificial rocks	✓	✓	Test the algorithm in a high obstacle terrain.

4.2 Algorithm Results

4.2.1 Primitive Shapes

Scenario No. 1 with primitive geometric shapes, as shown in Fig. 4–2a, consists of five hemispheres, capsules and cones, which are of different sizes and lie in three rows:

- First row: Hemispheres with varying radii from 0.05 m to 0.25 m.
- Second row: Half capsules with varying dimensions from 0.1 m×0.2 m to 0.3 m×0.6 m.
- Third row: Cones with the same radii of 0.15 m and varying heights from 0.05 m to 0.2 m.

The intuitively expected Graspability Map of scenario no. 1 is shown in Fig. 4–3. The smallest geometries would hardly be marked as graspable, since they take up a fraction of the volume of the gripper mask. In the case of the largest hemisphere, the surface of the peak would be considered as graspable in reality, but due to its flatness it is suboptimal for fixation. This fact should be reflected in the Graspability Map. For the medium sized and large half-capsules, the graspability maxima should be at both ends, while the ridge of the capsules would also be marked green.

The resulting Graspability Map is shown in fig 4–4. The ground level is manually set to 0.01 m. The regression plane has been calculated correctly, since all relevant elevations are above the zero plane. The area below the zero plane is marked in white and the z-axis corresponds to the normal vector of the regression plane.

The distribution of the calculated graspable targets largely corresponds to the expected distribution. In the case of a hemisphere with a smooth and homogeneous surface, the distribution of graspability emanates symmetrically from the center. The graspability maxima on the surfaces of the larger hemispheres are slightly offset. For the largest half-capsule, the expected maxima should be at the both ends of the geometry, which corresponds with the resulting Graspability Map. In case of some geometries, the cutting edges with the base are declared as graspable as well, which is an error caused by the high density of solid points at those sharp edges.

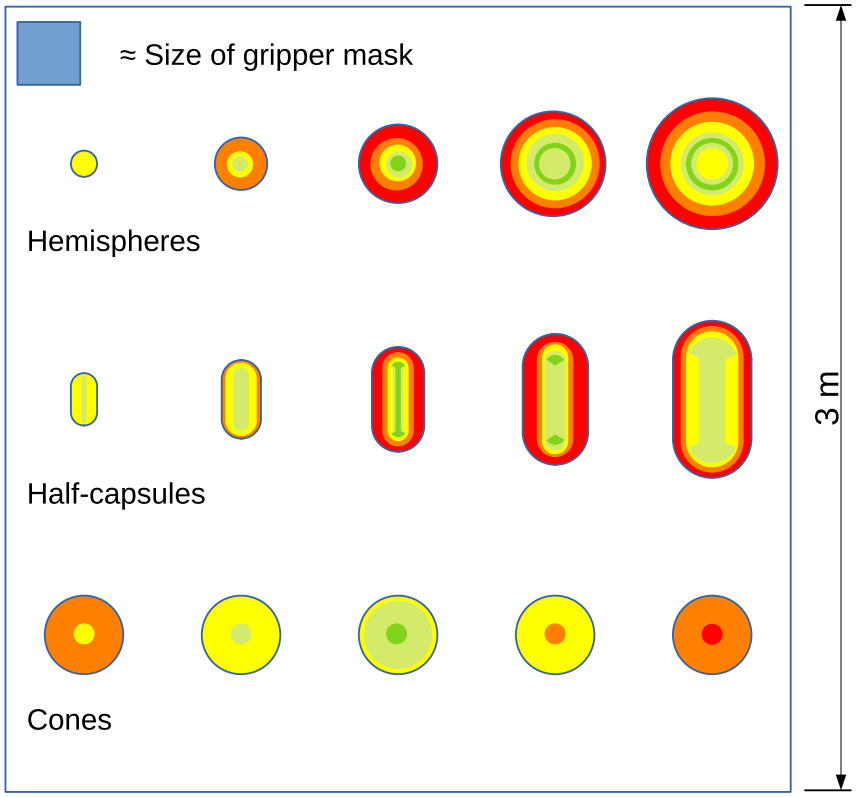


Fig. 4–3: Expected Graspability Map of the primitive geometries.

For hemisphere no. 2, 3 and 4, all capsules and cone no. 3, the result mostly agree with the expectations. the graspability maxima are placed at the top center of the geometry. The highest cone is expected to be not graspable, since the peak appears to be too pointed for the gripper. But apparently this cone fits perfectly into the shape of the mask.

A look at the area of the maximum curvature and its intersection with the maximum graspability reveals some misalignments, as shown in Fig. 2–2. The distribution of curvature, shown here in blue, should be symmetrical about the centre of a geometry, but the distribution appears to be truncated towards the coordinate centre, so that only half of the semicircles and cones are ever declared as curvature. This also has an effect on the intersection of curvature and graspability, shown in dark green, which is more in line with expectations for geometries closer to the coordinate centre.

4.2.2 Slopes and Bumps

Scenario 2 corresponds to a terrain with slight slopes and some bumps. The concatenated point cloud composed of six individual point clouds from the RGB-D cameras is shown in Fig. 4–5a. The robot stands in the center of a depression and can recognize the shape of the terrain on all sides, but the view of shadowed areas is hidden behind the bumbs, as well as the immediate area where the robot is located. This occluded area and the hexagonal hole in the middle of the terrain are compensated by the interpolation function. The resulting connected terrain is shown in Fig. 4–5b. At the same

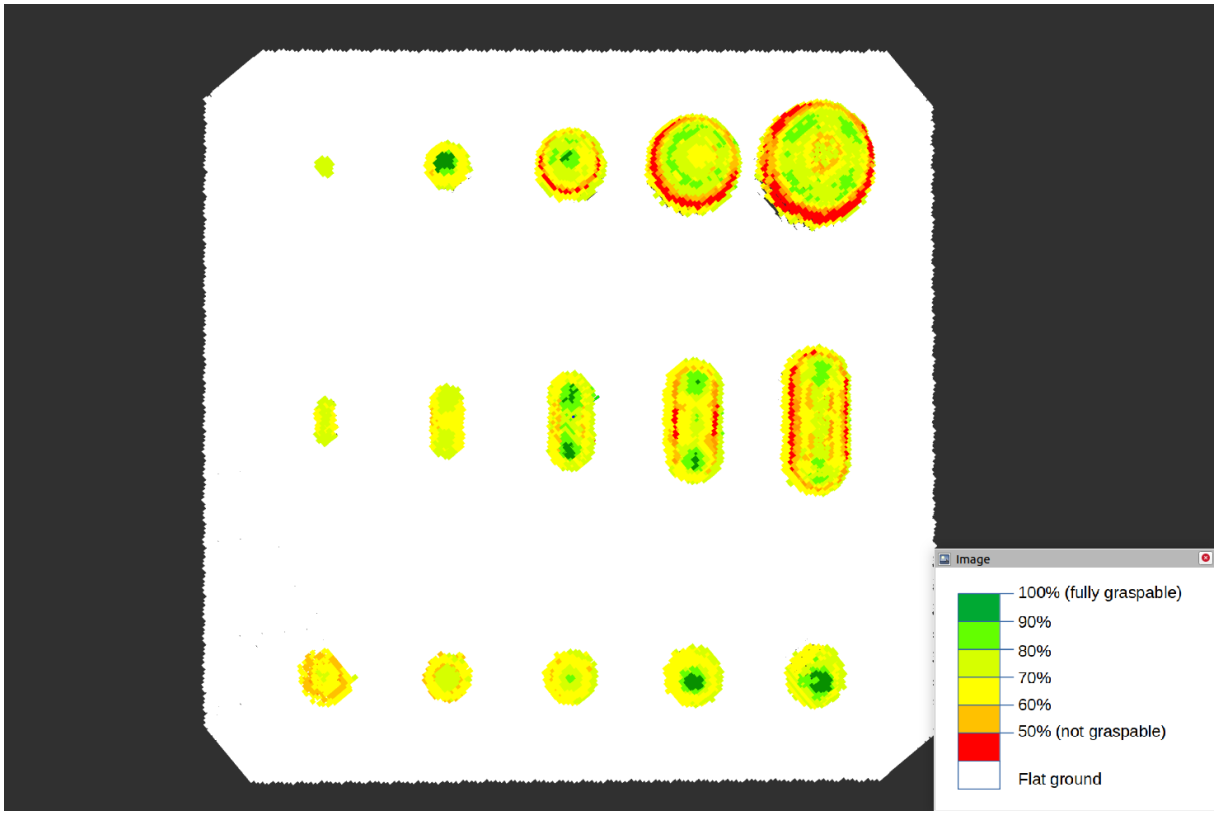


Fig. 4–4: Resulting Graspability Map of scenario no. 1.

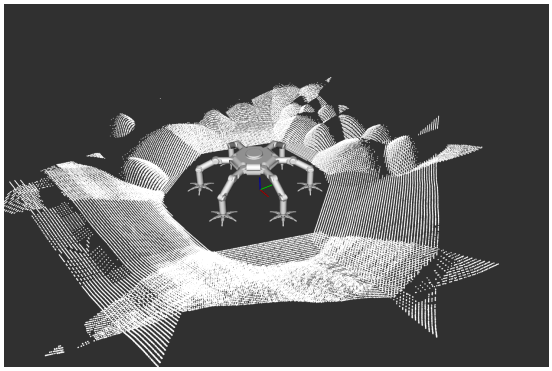
time, the distribution of points in the interpolated point cloud is homogenized by the algorithm, which in total corresponds to the original number of points in the raw point cloud.

The results of the non-interpolated point cloud are shown in Fig. 4–6. All bumps are recognized as such and marked in green, in contrast to the "vales" between the bumps, which are marked in yellow and red. A large part of the slopes are marked as graspable at those positions with a slightly convex shape. Most of the terrain behind the slopes, which can only be partially captured by the cameras, is marked in red because there are not enough points available for an assessment. While isolated hills are largely recognized as such and declared as graspable.

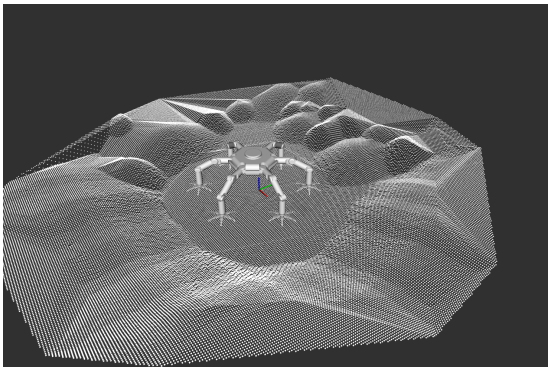
The compensation of the occluded areas does not change the result of the Graspability Map very much. In some areas, interpolation makes the assessment of graspability more accurate and limits the extent of the green area. Larger areas that lay in shadow and that are compensated by the algorithm should not be considered for the robot's movement or fixation anyway.

4.2.3 Bouldering Wall

In the third scenario, we primarily compare the performance of the algorithm between a pre-scanned point cloud and real-time exploration in a terrain of climbing rocks. The results of the algorithm on the two point clouds are presented below. The significance



(a) Input raw point cloud of scenario 2.



(b) Point cloud after interpolation.

Fig. 4–5: Occluded zone compensation and homogenization of the terrain to be examined.

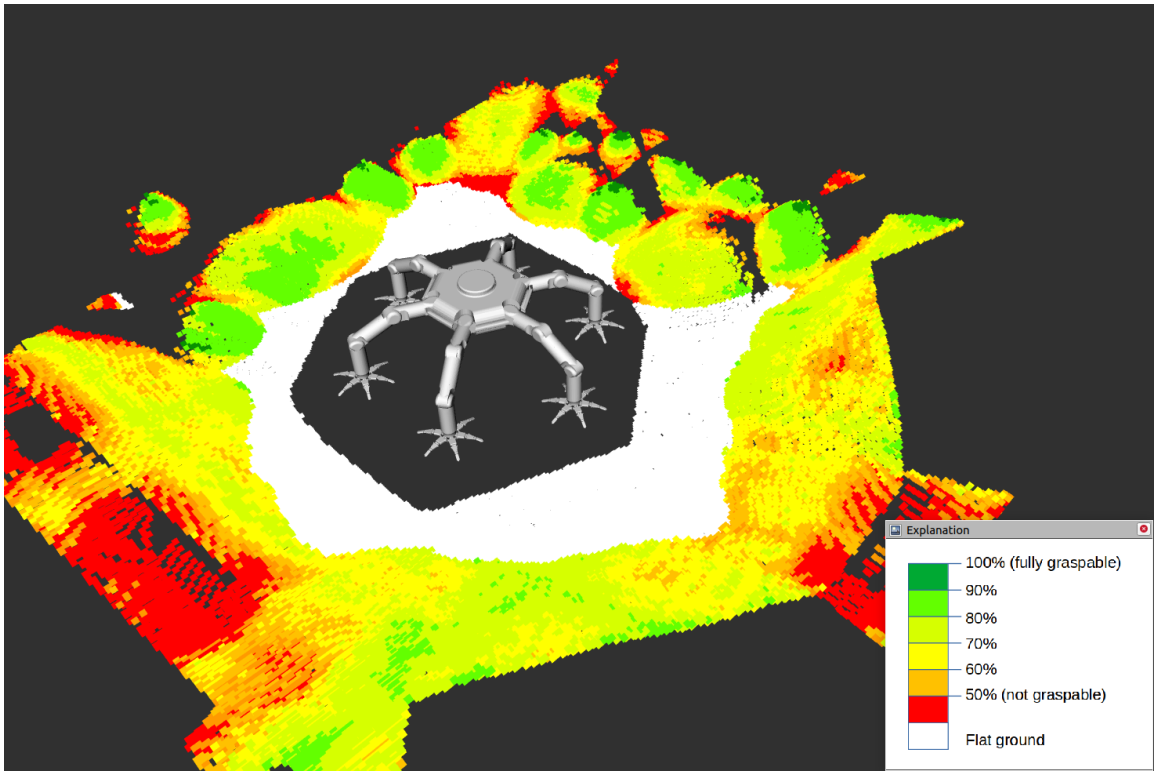


Fig. 4–6: Graspability Map of scenario 2 without occlusion compensation.

of the test results is assessed in the Discussion chapter.

4.2.3.1 Pre-scanned Map

The 58 climbing holds can be clearly seen in the Graspability Map in Fig. 4–8a. The tops of the boulders are largely marked as graspability maxima and the "vales" around the boulders are marked in red as inaccessible terrain.

A problem that has already been noticed in previous scenarios also exists here in the area of the cutting edge of the climbing holds with the ground. These areas are

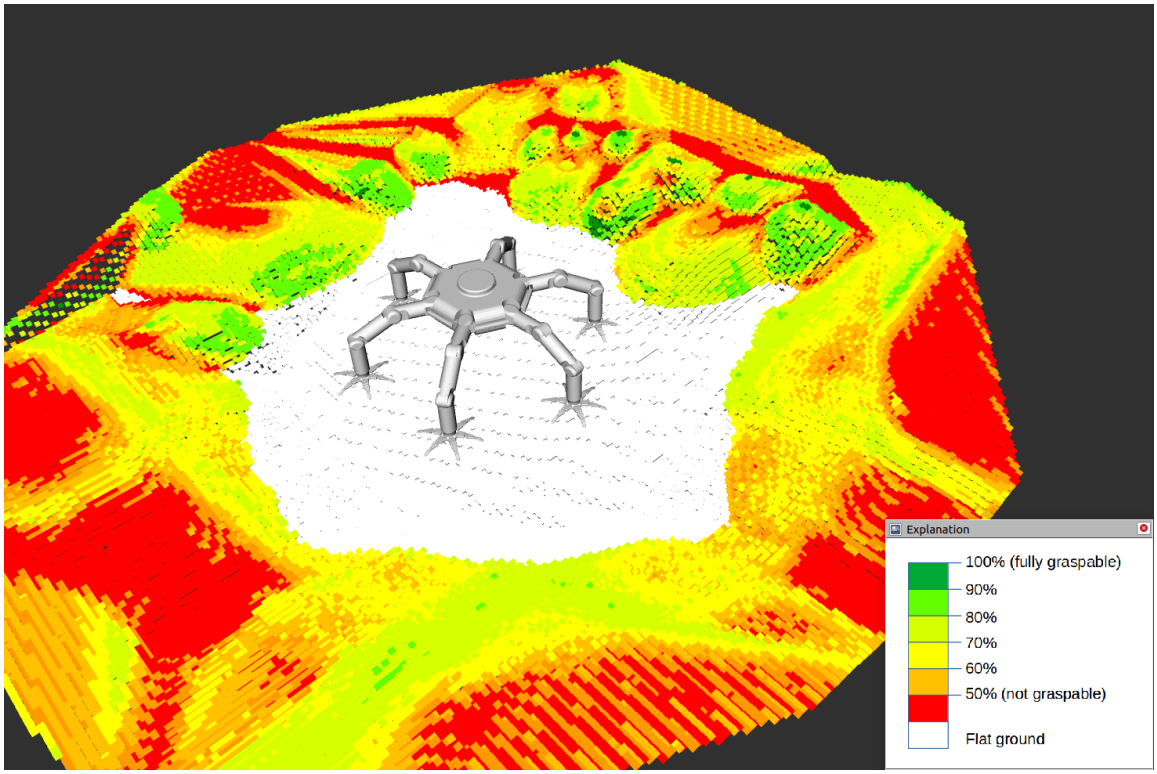


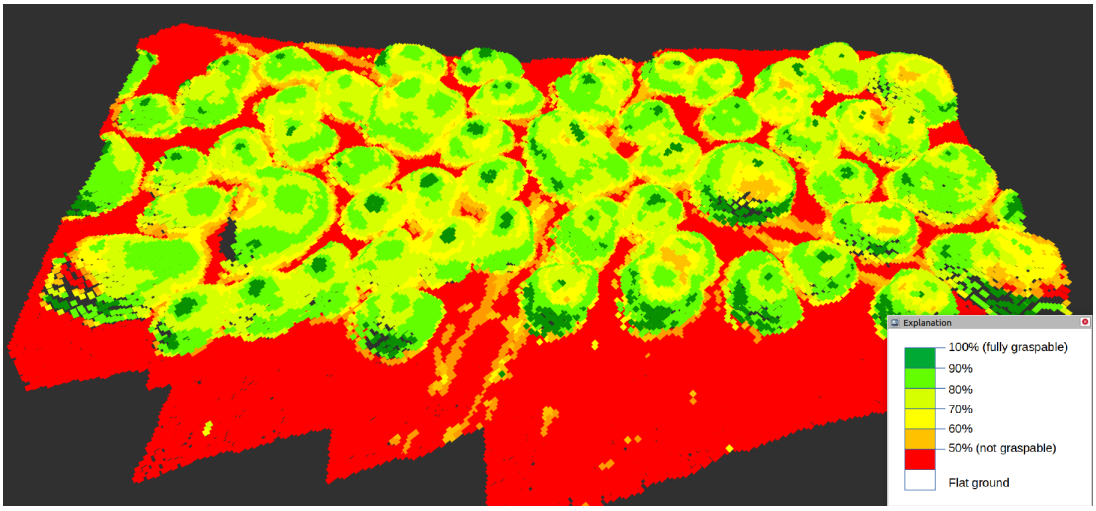
Fig. 4–7: Graspability Map of scenario 2 after interpolation of occluded areas.

declared as graspable because the number of solid voxels that fit into the grippable area of the gripper mask is greater here than in the area of the side walls higher up. This error can be hidden by setting a correct z-limit for the zero plane, see Figure 4–8b.

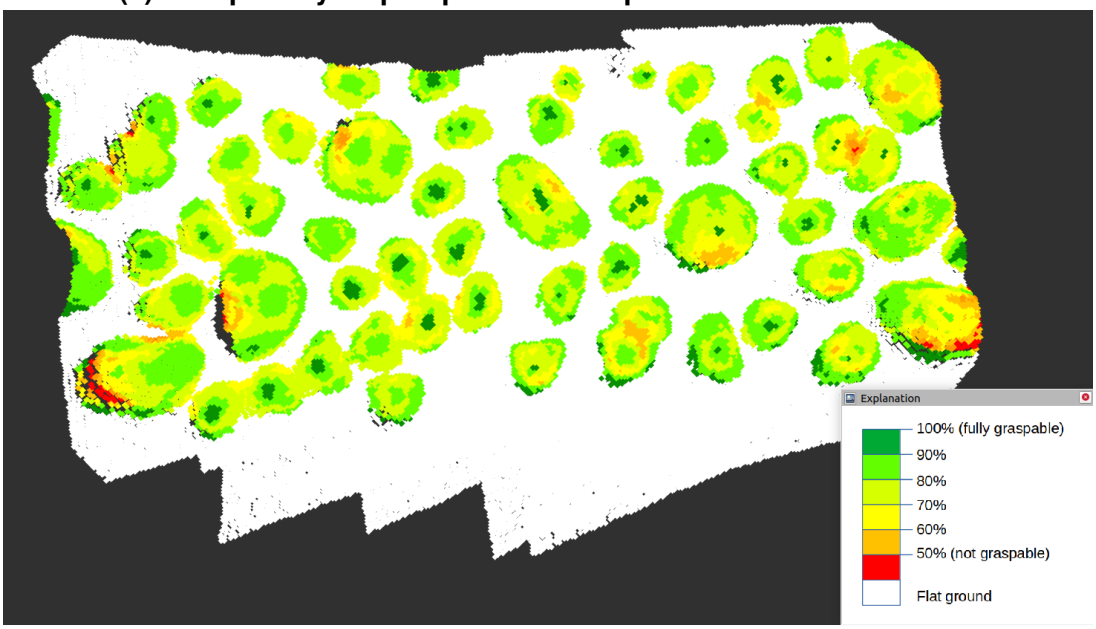
The Curvature Map of the scenario is shown in 4–8c. The intersection of the convex areas and the maxima of graspability should be at the top of each cliff that is declared graspable. The resulting intersection area is largely consistent with the result of the Graspability Map. While it cannot be said with certainty whether a region is graspable to the gripper or not without conducting a field test in the climbing gym with the gripper prototype, each climbing hold can be intuitively understood as a convex region with high curvature. However, since the curvature recognition algorithm leaves out those parts of the climbing holds whose surface normal vectors point toward the center, some of the summits of the climbing stones are not recognized as peaks.

4.2.3.2 Real-time Map

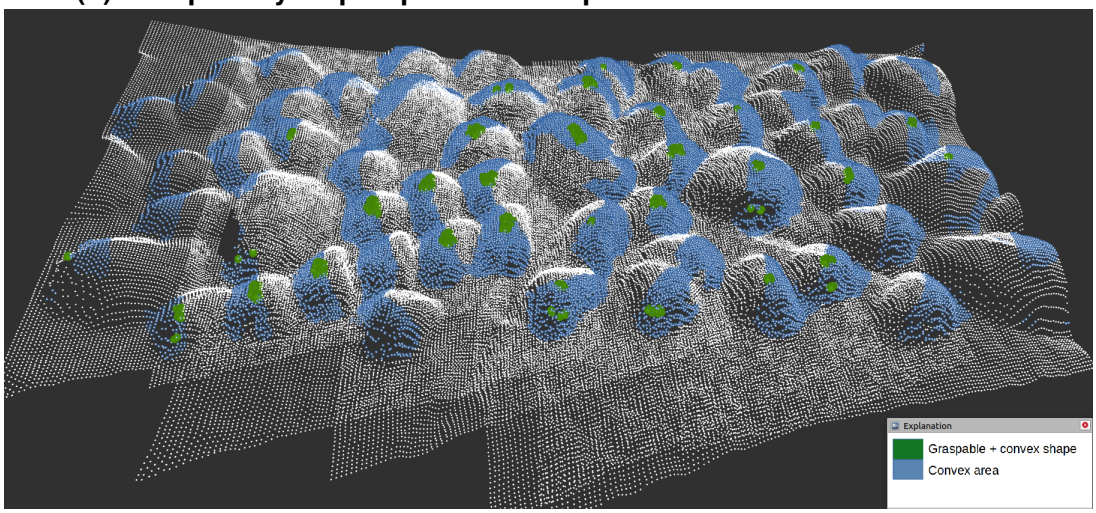
For the real-time case, The terrain to be examined only includes a part of the boulder field. Since the robot stands outside of the formation of climbing holds and the visibility of the cameras is limited to 1.5 m, the relevant part of the point cloud to be examined only includes the field of view of three of the six cameras. In contrast to the previous scenario, the cameras are located at an elevated point in relation to the climbing holds and therefore have a good overview of the stones, so that only a small part of the occluded areas need to be compensated.



(a) Graspability Map of pre-scanned point cloud of scenario 3.



(b) Graspability Map of pre-scanned point cloud with z-threshold set.



(c) Curvature Map and intersections with graspability maxima. Notice the left-out area whose surface normal vectors point toward the center.

Fig. 4–8: Resulting Graspability and Curvature Map of scenario 3.

The resulting Graspability Map of the raw input cloud is shown in 4–9a, which differs not much from the result of the interpolated map, shown in 4–9b. The graspable areas are largely congruent with the pre-scanned map. At the edges of the map, the interpolation has caused deformations in the terrain that are incorrect. However, these would have little influence on the evaluation.

It should also be noted that due to the uneven distribution of elevations on the map, the regression plane does not run perfectly parallel to the flat ground, but as seen from the robot, the ground slopes slightly towards the direction of the boulder field. The zero plane cuts some distant climbing stones horizontally in the middle. However, this also has only a minor influence on the evaluation, since the terrain below the zero plane is also considered for the computation of the graspability.

4.2.4 Artificial Rocks

The formation of artificial rocks described in Chapter 3.1.1 is examined below using the algorithm, whereby a distinction is also made here between the pre-scanned and the in-situ case as input.

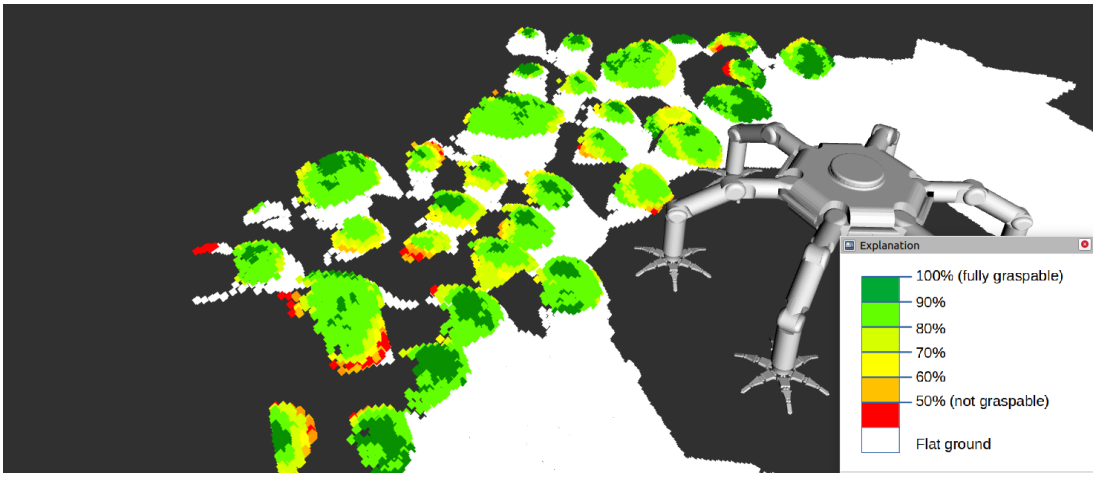
4.2.4.1 Pre-scanned Map

In this scenario, several rocks are significantly higher than the previously examined elevations and have steeper walls. The input point cloud should therefore also take these walls into account and map them in detail. Consequently, there is no interpolation and homogenization of the terrain, as the interpolation algorithm redistributes the points on an x-y grid map and thus, the vertical wall surfaces would be "thinned out". Instead, the point cloud is derived directly from the surface mesh (as shown in Fig. 3–5) using the Poisson-disk sampling method [68].

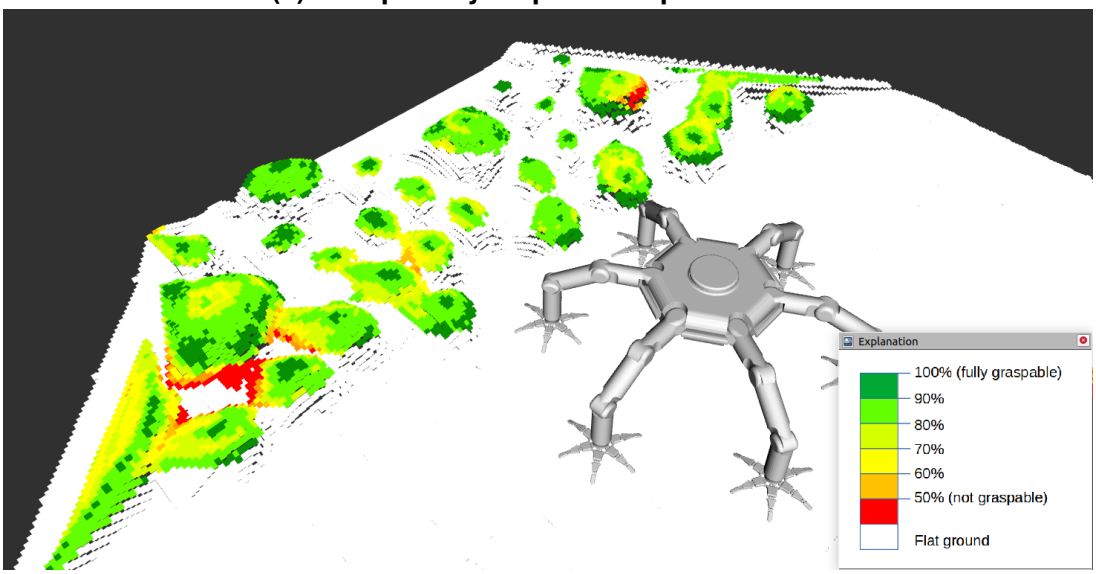
Fig. 4–10a shows the resulting Graspability Map. Flat surfaces on the top of the rocks are marked as non-graspable areas, but the edges of the high plateaus usually are marked green. The very pointed peaks of rocks no. 2 and 3 are marked orange to yellow. Concave surfaces are usually marked as red, although this only applies to the vertical depressions whose normal vector points in the z direction. The concave surfaces on the vertical surfaces are rarely recognized as such.

Again, due to the uneven distribution of rocks in the terrain, the regression plane was computed with a slight offset angle of < 5 degrees, so that the upper part of the map (in the image) is tilted towards -z. This can be seen in the red marked corner at the bottom right of the image, which protrudes from the zero plane, although the flat ground should be in theory perpendicular to the z-axis.

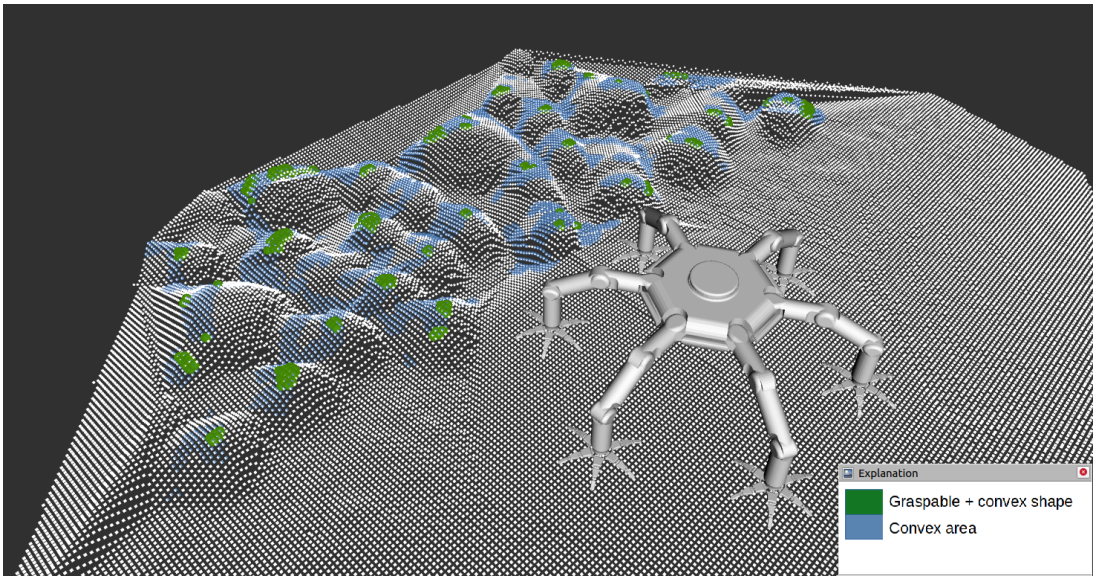
The curvature map in Fig. 4–10b shows most of the convex surfaces in the terrain, with some limitations: First, not all parts of a convex surface are visualised due to the previously mentioned error in the curvature estimation algorithm. This error arises because surface normal vectors do not have an unique orientation, as the algorithm cannot tell which is the "rock side" and which is the "air side" of a surface. Second, due to the fixed search radius of normals and curvature, which is $\sim$1/2 the side length



(a) Graspability Map on raw point cloud.

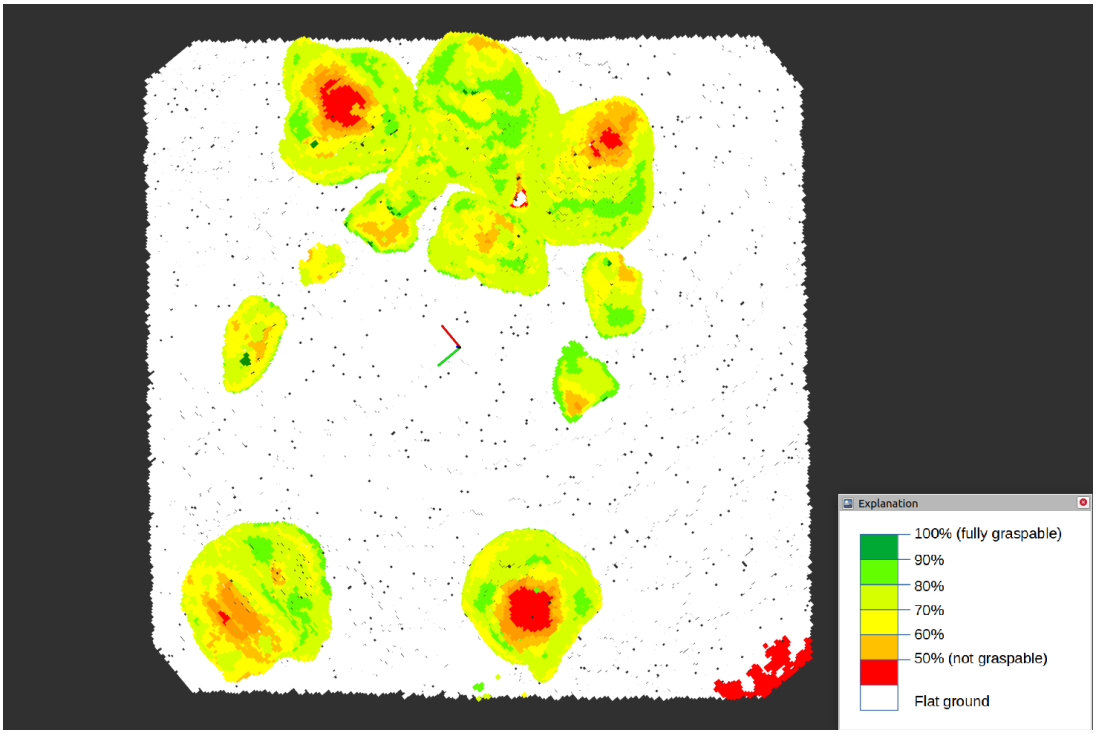


(b) Graspability Map on interpolated point cloud.

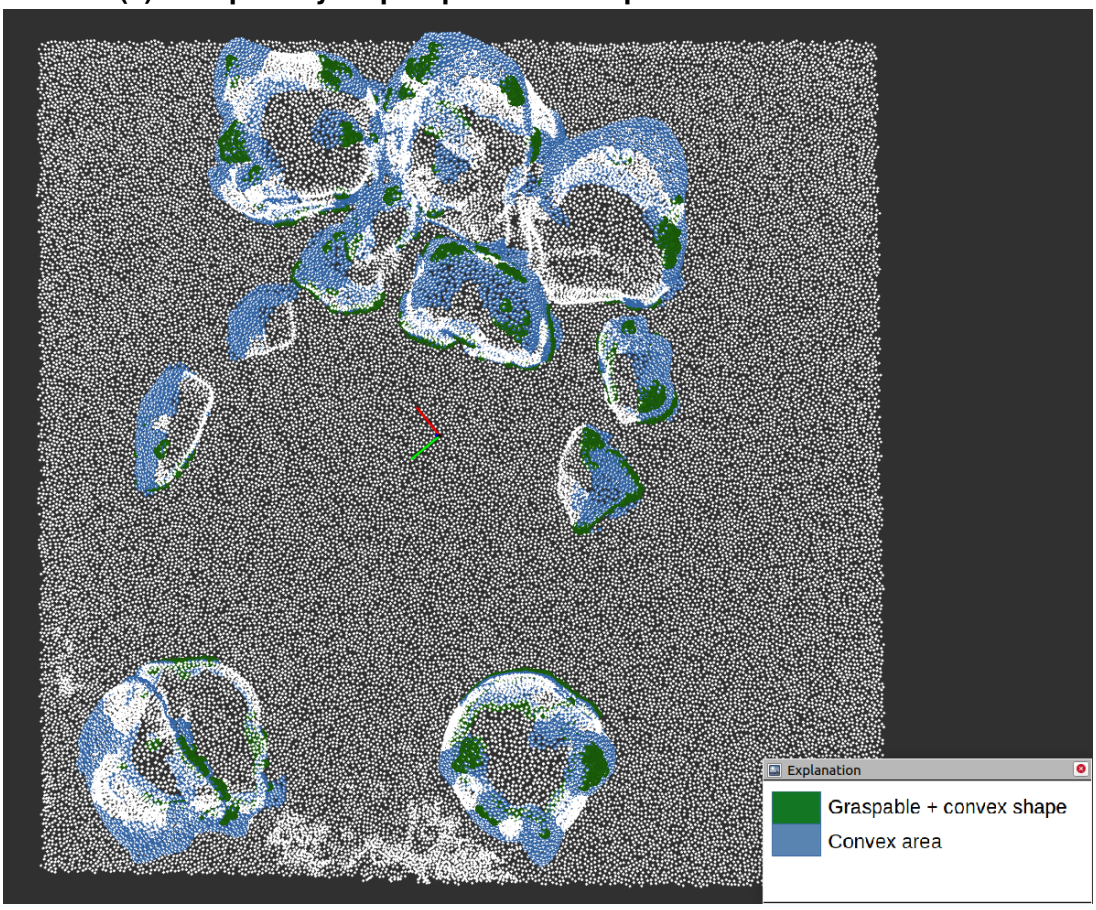


(c) Curvature Map and intersections with graspability maxima.

Fig. 4–9: Results of the in-situ approach of scenario 3.



(a) Graspability Map of pre-scanned point cloud of scenario 4.



(b) Curvature Map and intersections with Graspability maxima.

Fig. 4–10: Resulting graspability and curvature map of scenario 4.

of the gripper mask, curvatures with too large or too small a radius are not displayed. The convex surfaces located on vertical rock faces are made visible by the algorithm, which is a clear advantage over the gripper mask. The intersections with the tangibility maxima are all at prominent convex points in the map, and the grouping of the points also appears reasonable.

4.2.4.2 Real-time Map

The analysis of the real-time case encounters more challenges than in the previous scenarios. First, due to the height of some of the rocks and the proximity of the obstacles to the robot, it is not possible to see behind them. High rocks faces with the vertical faces are not an optimal target for the gripper mask as it cannot tilt or conform to the shape of the terrain. The results of the Graspability Map on the vertical surfaces should therefore be viewed with caution.

The Graspability Map of the non-interpolated point cloud is shown in Fig. 4–11(a). The flat areas on top of the high rocks no. 1, 12, 6 and 7 are hardly visible from the robot; these are only partially imaged and should not be considered for the evaluation, even if the algorithm predicts full graspability for these areas. The only surfaces that are relevant are the transition area from the vertical walls to the top area of those rocks.

For the interpolated map, shown in Fig. 4–11(b), the Graspability of the areas that were previously marked dark green has been severely restricted. This limitation stems from two primary factors. Firstly, the overall density of points within the relevant regions was reduced due to the grid-based homogenization process. Secondly, the correction of concealed areas behind rocks resulted in complete surface elevations connected to the ground. The modified geometries of the rocks have also exerted an influence on the resultant Graspability.

The interpolation has a positive effect on the lower rocks no. 4, 5, 8, 9 and 10, most of which were already largely visible without the interpolation. The interpolation estimates the trailing edge of these rocks quite adequately, so that the Graspability Map for these rocks is quite similar to the results of the pre-scanned map.

Finally, let's take a look at the curvature map, shown in Fig. 4–11(c). Here, the convex surfaces (blue) are first computed on the interpolated real-time map in order to then visualize the intersection of the convex surfaces with the Graspability maxima (shown in dark green). The distribution of the intersection areas is surprisingly clean, as the grouping of the intersections is well-arranged. These are often located at the front edges of the rocks (as seen by the robot), which is definitely a desirable result.

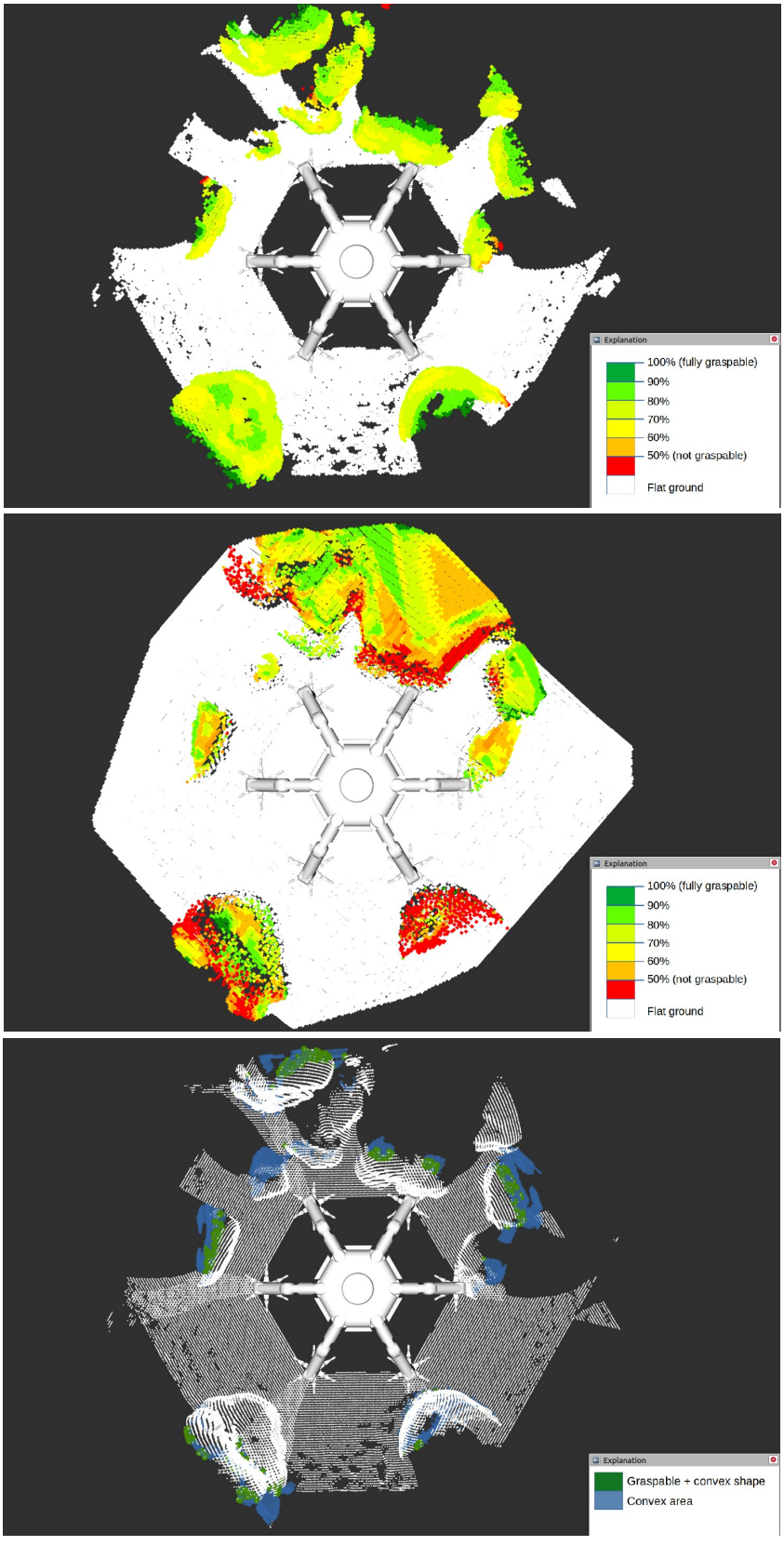


Fig. 4–11: Graspability Map of real-time raw (a), interpolated (b) and curvature combined (c) point cloud of scenario 4.

4.3 Field Test

For scenario 4, a small-scale field test was carried out. 30 points on the rocks were selected, as shown in 4–12a. The following criteria played a role in selecting the points:

- The set includes points on different parts of the artificial rocks, such as flat surfaces, edges, corners, and irregular surfaces.
- Points that are too close to the ground have been avoided.
- The set includes points that vary in terms of the difficulty of grasping. Some points appear to be easy to grasp, while others could be more challenging.
- A point that can be used by the robot for locomotion or for fixation in a microgravity environment is considered graspable.

The field test was carried out by a lab colleague. A set of 30 images where the positions of the points to be examined are precisely marked is provided for analysis. The results are listed in Tab. 2–3. The positions of the examined points are marked on Fig. 4–12a and highlighted in green (graspable) and red (non-graspable) on the textured mesh of the test area, as shown in Fig. 4–12b.

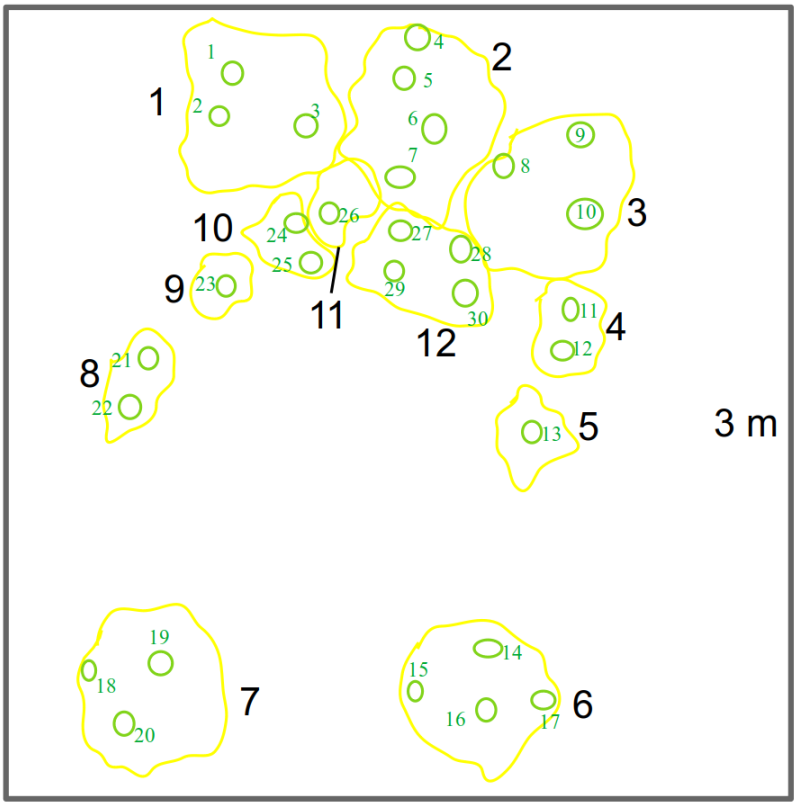
4.4 Running Time

SRL-GTD was developed based on the existing climbing robot simulator ClimbLab, which was written in MATLAB. In terms of temporal resource utilization, it is anticipated that C++ would deliver superior performance in comparison to Matlab.

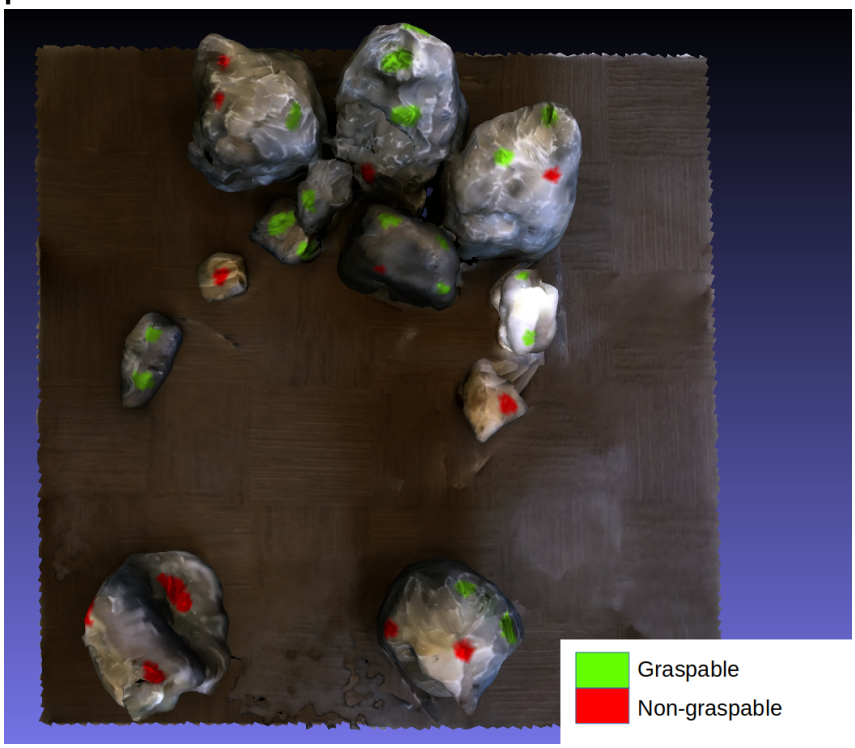
In the MATLAB algorithm, a notable delay is encountered during the *Voxel Matching* segment, whereby this phase exceeds around one second in duration. As shown in Tab. 4–2, this extended duration is attributed to the slower execution of the two subroutines, namely voxel extraction and voxel comparison, nested within the search loop. These subroutines handle extensive data variables and exhibit diminished swiftness in the MATLAB implementation. Given that the loop iterates roughly 50,000 times, analogous to the quantity of solid voxels within the map, the protracted duration materializes.

The C++ implementation on the other hand consistently shows significantly lower execution times for each core function compared to MATLAB. The most significant difference in runtime is observed in the most crucial *Voxel matching* function, with the C++ implementation being substantially faster. The total time for the C++ implementation of graspable target detection is **3.90 seconds** or approximately one-third of the time taken by the MATLAB implementation.

The entire process of a cycle from the creation of the depth images to the conversion into a point cloud to the detection of the graspable targets and their visualization in RVIZ takes approximately **4.5 seconds**, as shown in Tab. 4–3 using the example of scenario 4. The running time generally also depends heavily on the performance of the CPU. In the case of the point cloud conversion step, the frequency of receiving the depth images is deliberately limited to one second per image in order not to overload the computer.



(a) Numbering of the rocks and the numbered positions of the points to be tested.



(b) Results of the field test. Points which are graspable are marked green, the non-graspable ones are marked red.

Fig. 4–12: Positions of the positions on the artificial rocks to be tested and the test results visualized on the textured mesh of the test field.

Tab. 4–2: Comparison of the time consumption for each core function of SRL-GTD in µs. Input is a pre-scanned point cloud with ∼65 000 points.

Function	MATLAB	C++
Coordinate transformation	3866 µs	3132 µs
Interpolation	298 211 µs	87 564 µs
Voxelization	40 992 µs	15 388 µs
Gripper mask	378 µs	66 µs
Voxel matching (subfunctions):		
Voxel clip	95 386 µs	20 696 µs
Voxel extract	12 µs	12 µs
Voxel compare	10 µs	1 µs
Voxel matching overall	1 525 386 µs	865 696 µs
Total time	∼11.4 s	∼3.90 s

Tab. 4–3: Running time of one cycle of target detection in a real-time scenario (∼37 000 points), starting from the depth image generation in CoppeliaSim until the visualization of the Graspability Map.

Operation	Time elapsed
Depth image generation & publication	0.25 s
Point cloud conversion	1.69 s
Graspable target detection	2.54 s
Total time	4.48 s

5 Discussion

In this chapter, among the various aspects of the Graspable Target Detection algorithm, we extendedly provide an analysis to the core component, the gripper mask and the performance of the algorithm. The chapter is structured into several sub-chapters, each addressing specific elements of SRL-GTD's design and functionality.

5.1 Geometry of the Gripper Mask

The geometry of the gripper mask is crucial in determining the ability to detect graspable targets. An alternative geometric design is proposed and compared with the chosen design, highlighting both strengths and limitations.

5.1.1 Alternative proposal

In order to better represent the gripping extent of the gripper, an alternative concept for the geometry of the Gripper Mask was developed, which is shown in Fig. 5–1. The SCAR-E gripper has up to eight segmented fingers. Therefore, the corresponding mask should be rotationally symmetrical, with the region between the maximum and minimum gripping extent represented as solid voxels. This extent is limited by the maximum and minimum gripping angles θ_o (opening angle) and θ_c (closing angle).

In addition to the already known side lengths and height of the gripper mask, the geometry is represented by three crucial radii.

The Grippable Radius R_g is formed, analogous to the current design, from the top solid radius R_T and increases in z-direction.

$$R_g = R_T + \frac{z}{tan(90° - \theta_o)} \tag{5–1}$$

The Outer Unreachable Radius $R_{u,o}$ describes the outermost extent of the fingers and is formed following the formula for half chord length of a circle [1] and takes the minimal gripping angle θ_c into account.

$$R_{u,o} = R_T + \sqrt{2 \cdot \frac{H_{GM}}{cos(\theta_c)}(H_{GM} - z) - (H_{GM} - z)^2} \tag{5–2}$$

5.1.2 Comparison with Current Design

The alternative concept is a more accurate representation of the gripper and reflects the reach of the fingers well and omits those areas that do not represent the gripper extent. In the current algorithm, however, the alternative mask has some drawbacks

[1] Chord length of a circle: $2\sqrt{2Rh - h^2}$

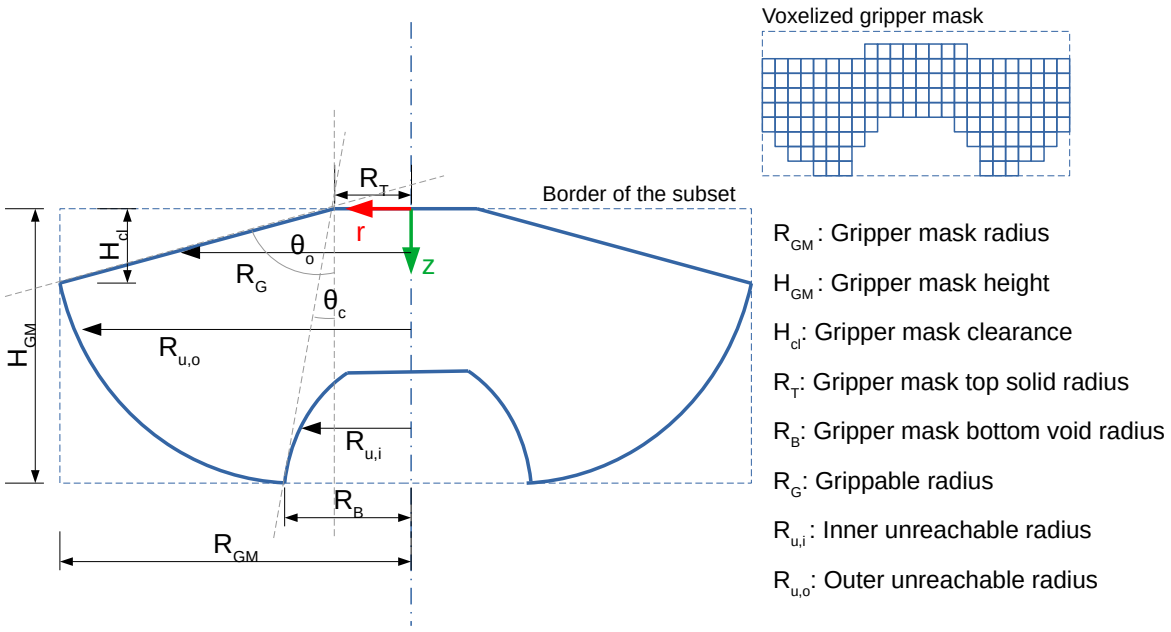


Fig. 5–1: Alternative concept of the Gripper Mask's design.

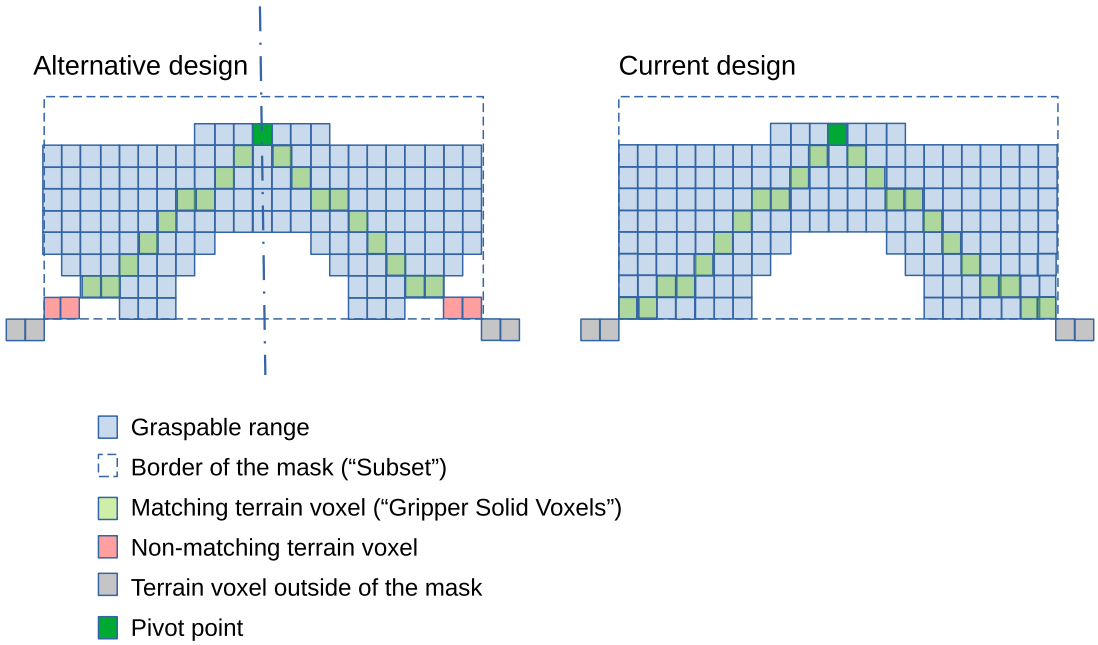


Fig. 5–2: Comparison of the alternative gripper mask with the current design in case of an ideal graspable cone shaped terrain. With the alternative mask, this shape would not be declared as fully graspable.

in detecting the actual graspability. As Fig. 5–2 shows, parts of shapes that could actually be perfectly gripped by the gripper would be considered outliers due to the non-matching area outside the Outer Unreachable Radius $R_{u,o}$. These void areas are of no relevance for the actual graspability of the pivot point, but influence the Graspability score. Since the Subset is a cuboid, terrain voxels within the newly emerged non-matching area would penalize the Graspability score. But in reality, these areas do not pose any obstacles to grasping.

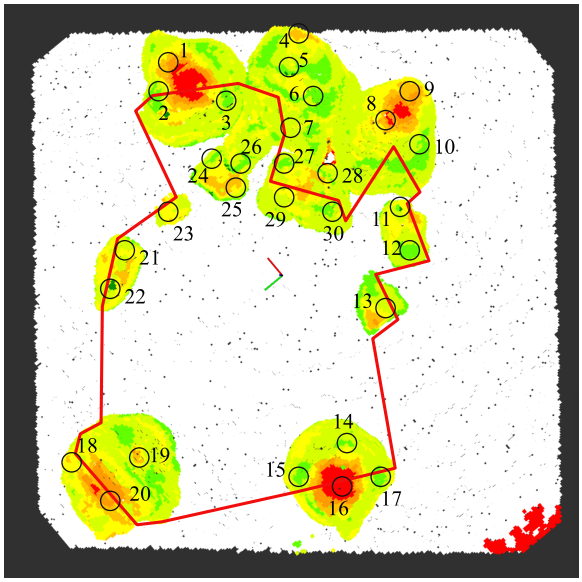
A small change to the algorithm could potentially solve this problem: by excluding the previously mentioned non-matching void area outside $R_{u,o}$ when calculating the Subset Solid Voxels (SSV) and the Gripper Solid Voxels Gripper Solid Voxels (GSV) (see Chapter 3.2.5), the influence of this non-matching region on the result would be minimized. This would probably lead to promising results, but was no longer tested within the scope of this work. The current design is square in plan, and the nominal graspable area includes the area beyond the reach of the fingers. It is not clear from the available results whether this actually represents a disadvantage.

5.2 Performance of the Graspable Target Detection

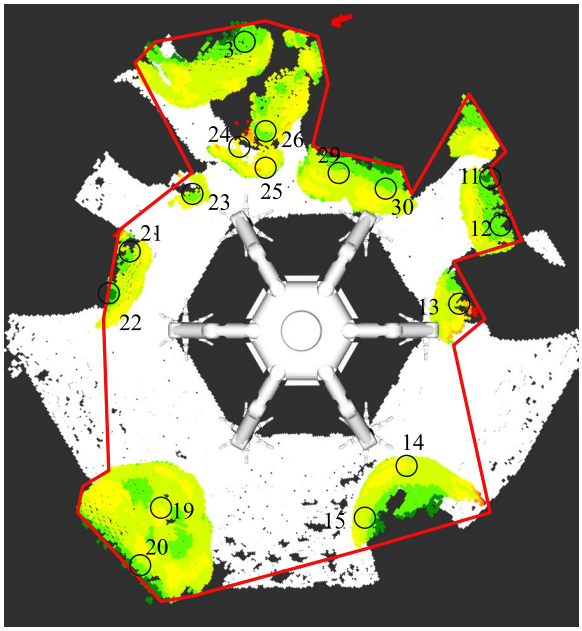
The following sections encompass the utilization of a binary classifier and the Receiver Operating Characteristic (ROC) curve for assessing of the discriminative performance of the test, focusing scenario 4, which was the most comprehensive. Subsequently, we will analyze the accuracy of the findings, incorporating the bouldering wall scenario in our evaluation.

5.2.1 Receiver Operating Characteristics

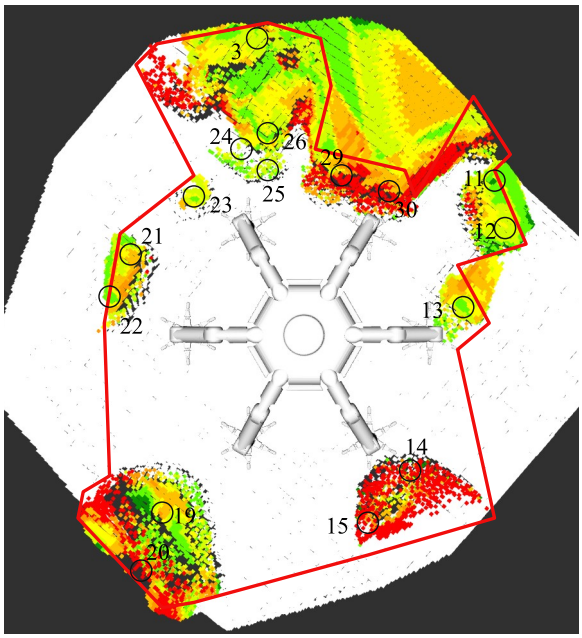
The outcomes of the field test hold significance, as they establish a reference for the subsequent assessment of the algorithm. In Fig. 5–3a, the points examined in the field test are marked on the Graspability Map of the pre-scanned map of scenario 4. Fig. 5–3b and Fig. 5–3c show the real-time point clouds in their raw state and in their interpolated state. Here, all points are marked here whose Graspability score is clearly visible. Thus, the border of the visible area for the real-time case is highlighted in red. The corresponding Graspability score read from the map are listed in Tab. 2–3.



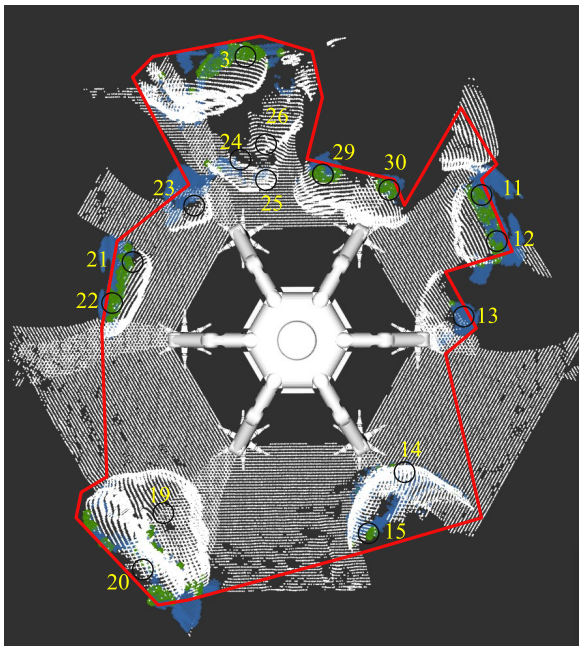
(a) Pre-scanned map.



(b) Real-time map raw.



(c) Real-time map interpolated.



(d) Real-time map combined.

Fig. 5–3: Points examined in the field test marked in the graspability maps of scenario 4. The red line marks the limited area examined for the real-time cases.

5.2.1.1 Sensitivity and Specificity

To assess how well a logistic regression model fits a dataset, we can look at the following metrics:

- **Sensitivity** (True Positive Rate (TPR)): The probability that the algorithm predicts a positive outcome for a testing point when indeed the outcome is positive.

$$TPR = \frac{TP}{TP + FN} \tag{5–3}$$

- **Specificity** (True Negative Rate (TNR)): The probability that the algorithm predicts a negative outcome for a testing point when indeed the outcome is negative.

$$TNR = \frac{TN}{TN + FP} \tag{5–4}$$

- **Fall-out** (False Positive Rate (FPR)): The probability that the algorithm incorrectly predicts a positive outcome for a testing point when the outcome is negative.

$$FPR = 1 - TNR \tag{5–5}$$

- **Miss Rate** (False Negative Rate (FNR)): The probability that the algorithm incorrectly predicts a negative outcome for a testing point when the outcome is positive.

$$FNR = 1 - TPR \tag{5–6}$$

Which includes the following terminology:

- **True Positives (TP)**: Entirely of results of the algorithm which indicates the presence of a graspable point.
- **True Negatives (TN)**: Entirely of results of the algorithm which indicates the absence of a graspable point.
- **False Positives (FP)**: Entirely of results of the algorithm which wrongly indicates that a graspable point is present.
- **False Negatives (FN)**: Entirely of results of the algorithm which wrongly indicates that a graspable point is absent.

5.2.1.2 ROC curve

The Receiver Operating Characteristic (ROC) [69] is a graphical representation of a binary classification model's performance across various discrimination thresholds. It plots the sensitivity against the false positive rate (1 - specificity) as the threshold

for classifying instances is varied. The ROC curve helps visualize the trade-off between sensitivity and specificity and assesses the model's performance across different threshold settings.

5.2.1.3 Area under the Curve

The Area Under the Curve (AUC) is a single scalar value that quantifies the overall performance of a binary classification model. AUC measures the ability of the model to discriminate between positive and negative instances across *all* possible threshold settings. A model with an AUC of 1.0 is perfect, while random guessing yields an AUC of 0.5. An AUC between 0.5 and 1.0 indicates the model's discrimination ability, with higher values indicating better discrimination.

5.2.1.4 Discriminative Power

Tab. 5–1 - Tab. 5–3 shows ten threshold ranges and the respective sum of positive and negative points and the correspondent TPR and FPR for the three different cases of scenario 4. This is followed by the corresponding sensitivity and specificity values for the threshold ranges.

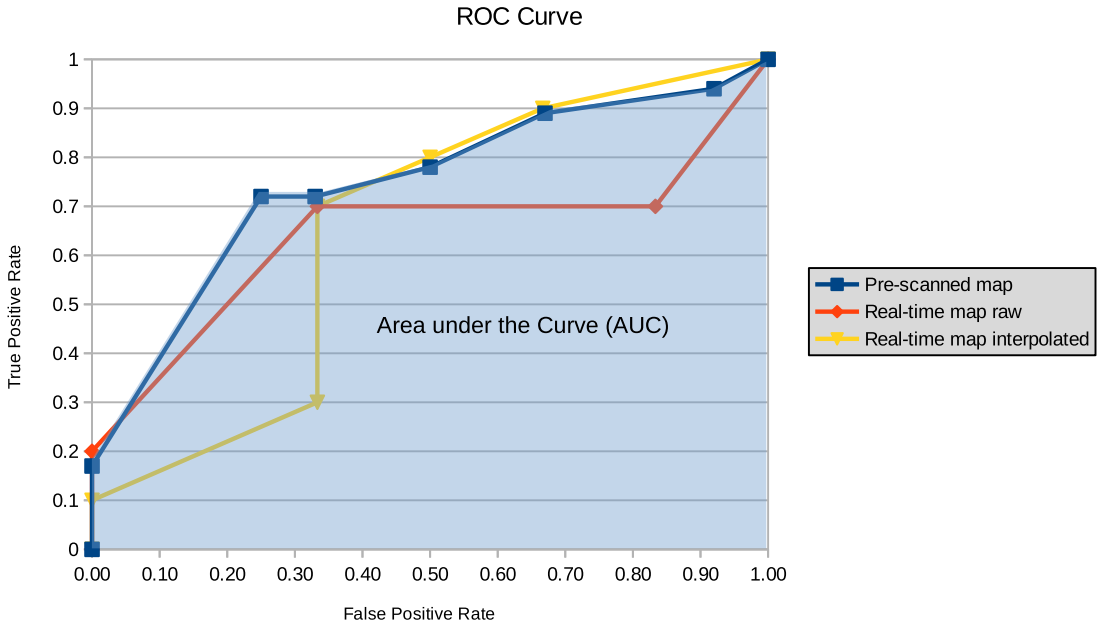


Fig. 5–4: ROC curve of the field test. Optimum graspability threshold is highlighted in yellow in Tab. 5–1 - 5–3. The pre-scanned case (blue) has the largest area under the curve AUC.

The ROC curve for the three cases is illustrated in Fig. 5–4. A high value in AUC signifies that the model has a high true positive rate (sensitivity) while maintaining a low false positive rate, indicating strong discrimination. Based on the AUC value of 0.84 in the pre-scanned case, which demonstrates high discriminatory power, we can clearly see that this test scenario is superior to the others. This is not surprising since the case provides Grapability score for all 30 the points tested in the field test. On the contrary, the real-time interpolated case provides the least discriminative power.

5.2.1.5 Graspability Threshold

The Graspability score represents a probability of how likely a point is to be graspable. In order to define an exact threshold value, from which one can safely assume a graspability, we take another look at the ROC analysis.

The choice of an operating point on the ROC curve depends on individual priorities. The optimum threshold represents a trade-off between sensitivity and specificity. In the case of identifying graping points on rocks, false positive results can lead to much more severe errors, including disastrous accidents. Therefore, a high true positive rate is selected with a prioritized low false positive rate. Therefore, the Graspability Threshold is set to **0.8** in general.

ROC and AUC may not provide a complete picture when dealing with imbalanced datasets. ROC and AUC do not reflect changes in the TPR and FPR if the class distribution changes, or if one class greatly outnumbers the other. This was the case when only part of the terrain is visible, as in the real-time cases where the proportion of negative field test values is lower. ROC and AUC do not reveal how well a model generalizes to new data.

Tab. 5–1: ROC table and AUC of scenario 4, pre-scanned case.

Cutoff	Predicted positives	Predicted negatives	True Positive Rate	False Positive Rate	AUC
0.1	30	0	1.00	1.00	0.00
0.2	30	0	1.00	1.00	0.00
0.3	30	0	1.00	1.00	0.08
0.4	28	2	0.94	0.92	0.24
0.5	24	6	0.89	0.67	0.15
0.6	20	10	0.78	0.50	0.13
0.7	17	13	0.72	0.33	0.06
0.8	16	14	0.72	0.25	0.18
0.9	3	27	0.17	0.00	0.00
1	0	30	0.00	0.00	0.00
Total					**0.84**

Tab. 5–2: ROC table and AUC value of scenario 4, real-time raw case.

Cutoff	predicted positives	Predicted negatives	True Positive Rate	False Positive Rate	AUC
0.1	16	0	1.00	1.00	0.00
0.2	16	0	1.00	1.00	0.00
0.3	16	0	1.00	1.00	0.00
0.4	16	0	1.00	1.00	0.00
0.5	16	0	1.00	1.00	0.00
0.6	16	0	1.00	1.00	0.17
0.7	12	4	0.70	0.83	0.35
0.8	9	7	0.70	0.33	0.23
0.9	2	14	0.20	0.00	0.00
1	0	16	0.00	0.00	0.00
Total					**0.75**

Tab. 5–3: ROC table and AUC of scenario 4, real-time interpolated case.

Cutoff	predicted positives	Predicted negatives	True Positive Rate	False Positive Rate	AUC
0.1	16	0	1.00	1.00	0.00
0.2	16	0	1.00	1.00	0.00
0.3	16	0	1.00	1.00	0.33
0.4	13	3	0.90	0.67	0.15
0.5	11	5	0.80	0.50	0.13
0.6	9	7	0.70	0.33	0.00
0.7	9	7	0.70	0.33	0.00
0.8	5	11	0.30	0.33	0.10
0.9	1	15	0.10	0.00	0.00
1	0	16	0.00	0.00	0.00
Total					**0.72**

5.2.2 Accuracy of the Results

To evaluate the accuracy of the statements that SRL-GTD provides, two additional metrics are introduced: Positive Predictive Value (Precision) and Accuracy.

- **Positive Predictive Value (PPV)**, also called **Precision**, measures the ability of a binary classifier to correctly identify positive instances from all instances it classifies as positive. ("Of all the points that the model predicted as graspable, how many were actually graspable?")

$$PPV = \frac{TP}{TP + FP} \tag{5–7}$$

- **Accuracy (ACC)** is how close a given set of measurements are to their true value. It measures the overall correctness of a binary classifier, considering both positive and negative classifications. ("Of all points in the dataset, how many were correctly classified?")

$$ACC = \frac{TP + TN}{TP + TN + FP + FN} \tag{5–8}$$

The sensitivity and specificity of the cases in the 4th scenario can be read directly from Tab. 5–1 - Tab. 5–3 The resulting accuracy is around 70% for every case, which is satisfactory. The result of the pre-scanned map clearly shows the best predictive power among all cases. The pre-scanned cases include all 30 testing points. It is noticeable that in case of the combined analysis of curvature and graspability, the false positive rate is significantly lower, which is in favor of this method.

In the real-time cases, the true positive rate is moderately high, but with a high false positive rate of 1/3. However, the available data set of 16 points is small due to limited visibility.

For scenario no. 3, no field test results are available. However, this is also a fairly easily accessible area, and it can be assumed that all climbing holds are within the robot's gripping capability. The peaks of all climbing holds have a Graspability score of more than 0.8. It's more a matter of finding the *ideal* points for locomotion. Therefore, the pre-scanned case is used as a reference because it provides the most comprehensive information about the graspability. The Graspability threshold is set to 0.9, which is a stricter criterion than in scenario 4. Fig. 2–1 shows the numbering of the holds and the results for each case. Positions with graspability above the threshold are marked in dark green. In addition, the visible range for real-time cases is highlighted with a red line. The results of the five examined cases are shown in Tab. 2–4.

The evaluation of the five examined cases reveals that the method using combined criterion of the pre-scanned map has the overall lowest false positive rate and a high accuracy.

In the real-time cases, 30 of 58 points were examined. The rate of false positive results here is very high, which is due to the strict Graspability threshold of 0.9, but also to the

very low number of only 9 negative points in the reference map. The results here are therefore only of limited significance.

Overall, the evaluation shows a satisfactory result in terms of the accuracy of the algorithm

Tab. 5–4: Accuracy analysis across the investigated cases for scenario 3 and 4. Terminology: TPR: True Positive Rate (Sensibility); TNR: True Negative Rate (Specificity); FPR: False Positive Rate (Fall-out); FNR: False Negative Rate; PPV: Positive Predictive Value (Precision); ACC: Accuracy.

Scenario	Case	TPR	TNR	FPR	FNR	PPV	ACC
Bouldering wall	Pre-scanned curvature	80.00%	88.89%	11.11%	20.00%	94.12%	71.14%
	Real-time raw	90.48%	22.22%	77.78%	9.52%	73.08%	70.00%
	Real-time interpolated	100.00%	44.44%	55.56%	0.00%	80.77%	83.33%
	Real-time curvature	90.48%	33.33%	66.67%	9.52%	76.00%	73.33%
Artificial rocks	Pre-scanned raw	72.22%	75.00%	25.00%	27.78%	81.25%	73.33%
	Pre-scanned curvature	66.67%	83.33%	16.67%	33.33%	85.71%	73.33%
	Real-time raw	70.00%	66.67%	33.33%	30.00%	77.78%	68.75%
	Real-time interpolated	70.00%	66.67%	33.33%	30.00%	77.78%	68.75%
	Real-time curvature	70.00%	66.67%	33.33%	30.00%	77.78%	68.75%

5.2.3 Robot in a Different Pose

So far, we have only examined one pose of the robot in the real-time scenario. The robot always stands on a flat surface from where it can overlook a large part of the terrain. One question that can be asked is how the target detection results change if the robot is in a different pose.

This is demonstrated in Fig. 5–5. Here the main body of the robot is in an approx. 25 degree inclined position to the ground. From this position, only 15 of 30 points examined in the field test are visible. The evaluation of the results, shown in Tab. 5–5 provides moderate accuracy. Two thirds of the predictions are correct. Despite

the crooked position of the robot, the regression plane was calculated approximately correctly and is parallel to the flat ground. The error angle is less than 5°.

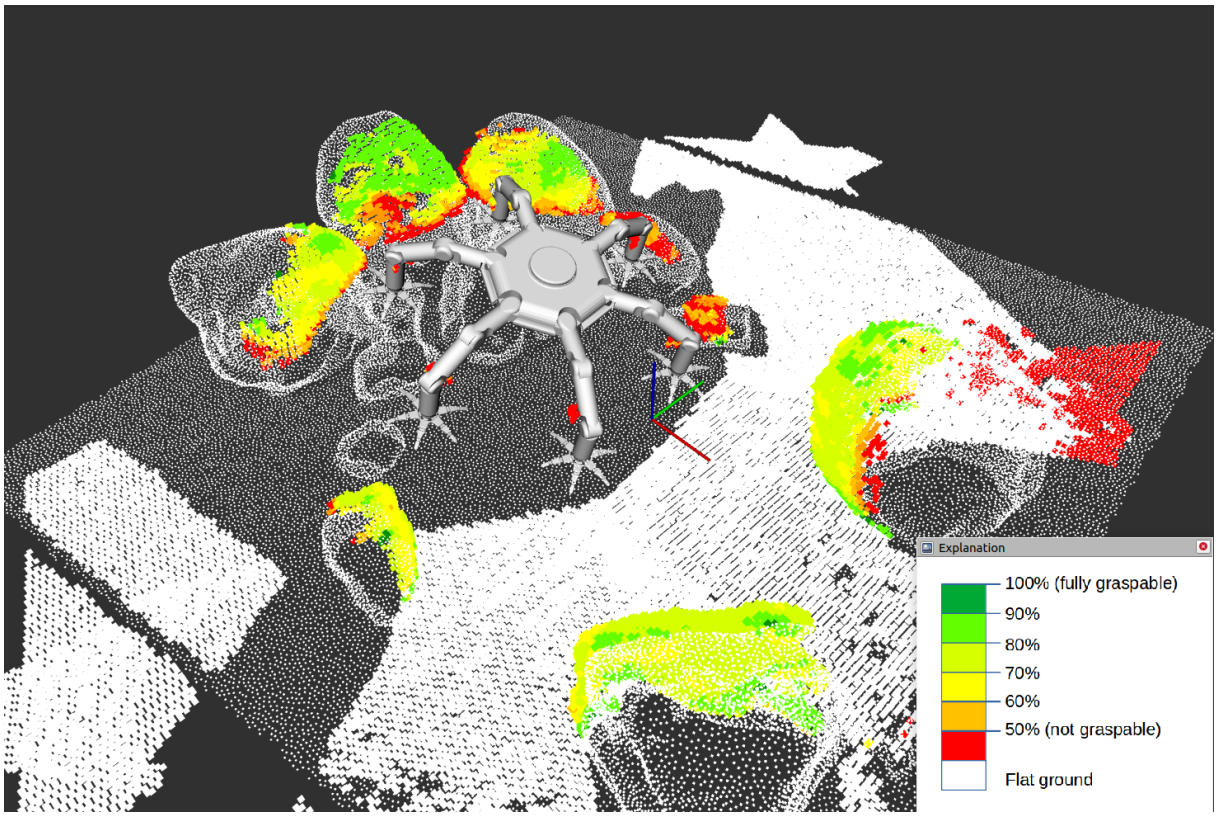


Fig. 5–5: Graspability Map of scenario 4 with the robot in a crooked pose.

Tab. 5–5: Evaluation of the graspable points examined in a different pose of the robot.

Scenario	Case	TPR	TNR	FPR	FNR	PPV	ACC
Artificial Rocks	Real-time, different pose	66.67%	66.67%	33.33%	33.33%	75.00%	66.67%

5.2.4 Thresholding

Two different measures were taken to minimize incorrect assessments in the areas of depressions and side walls:

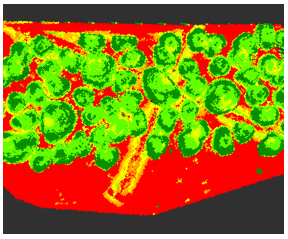
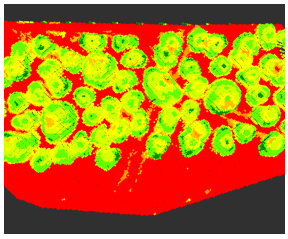
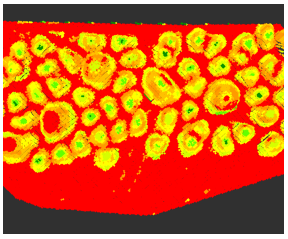
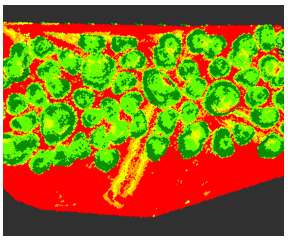
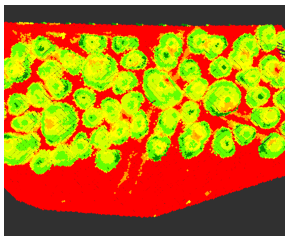
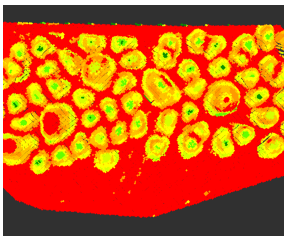
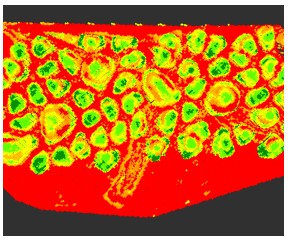
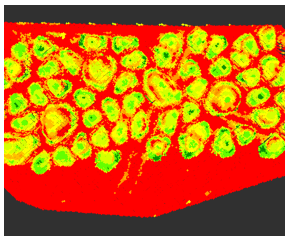
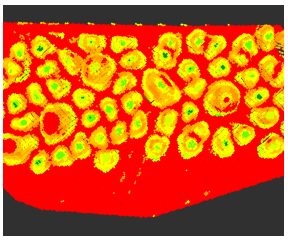
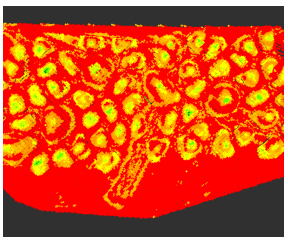
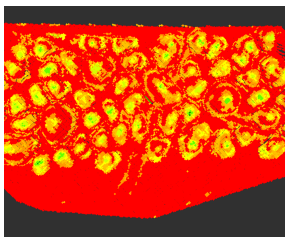
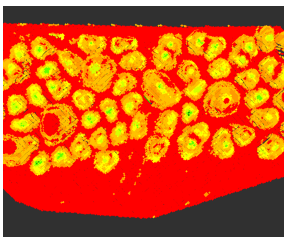
- **Threshold of Solid Voxels** (TSV), which leads to a penalty of the Graspability score for those voxels whose subsets contain too few solid terrain voxels.
- **Additional Void Gripper Mask Layers** H_{add}, which adds void voxel layers above the pivot point.

Both thresholds counteract the erroneous predictions to a certain extent. For the previous tests, an empirical value of $TSV = 120$ and H_1 were used, which deliver the best

results for the dimensions of the SCAR-E gripper and eliminates most false positive results.

To explore the impact of these two threshold values on the distribution of graspability, we conducted a total of 12 tests on the pre-scanned, interpolated map in scenario 3, as shown in Tab. 5–6. This scenario has been chosen, since all climbing holds are considered graspable and the ideal climbing point can be assumed to be on the peak of each hold.

Tab. 5–6: Study of distribution of graspability with varying Threshold of Solid Voxels TSV and Additional Void Gripper Mask Layers H_{add}

	$H_{add} = 0$	$H_{add} = 1$	$H_{add} = 5$
$TSV = 0$			
$TSV = 120$			
$TSV = 250$			
$TSV = 350$			

5.2.4.1 Discussion of Thresholding

In the case of no threshold, large parts of the side walls of the rocks are declared graspable and no clear marker for the peak is visible. As the Threshold of Solid Voxels (TSV) value increases, the peaks become more prominent, but the edges of the rocks and the actually inconspicuous trench on the ground (groove between two bouldering wall panels) are still visible. With increasing H_{add} value, the result becomes

more specific, as the Graspability maxima are assigned to the peaks, which is desirable. However, the peak surfaces of larger size rocks are completely excluded from the accessible range.

Judging from the graphic, the threshold optimum is between $TSV = 120 \sim 250$ and $H_{add} = 1 \sim 5$. Results with threshold values that go beyond those described are not listed here as they tend to lead to worse results.

This is only a rough estimation of the threshold values, especially since only one scenario was used for testing. For a more precise estimate, a ROC analysis is required on standardized forms, such as those in scenario 1. The Graspability score obtained in defined areas can then be compared with expected values in order to find the optimum. This could not be addressed within the scope of this work.

5.2.5 Conclusion about the performance of the algorithm

Based on the results of the evaluation of the ROC analysis, the following statements can be made:

- From the ROC analysis of Scenario No. 4, using the field test results as a reference, it can be determined that in difficult terrain a Graspability Threshold of 0.8 provides the best discriminatory power.
- Pre-scanned maps provide better reliability in terms of accuracy and higher sensitivity with low fall-out compared to real-time explorations.
- The combined criterion of convex shapes and Graspability maxima provides a more conservative, but in terms of low fall-out rate, reliable method to find securely graspable terrain and avoid non-graspable areas.
- Regarding real-time cases, no significant differences can be found between the three methods used, due to the small number of samples. However, the accuracy of these methods are all in an acceptable range, although with significant higher false positive rates.
- In scenario 3, a terrain that is easily surveyed by robots from an elevated position and free of very tall rocks, the algorithm provides reliable results in terms of the position and graspability of the elevated points. Here, the Graspability threshold must be increased to 0.9 for any difference between the cases to be noticeable.
- However, in the case of the bouldering wall, no field tests were carried out, but the ROC analysis was carried out with the most complete result (pre-scanned case) as a reference. This reduces the comparability of the two scenarios.
- The algorithm also provides acceptable results even when the main body of the robot is in an inclined position that is not parallel to the ground. Also in this case the zero level is calculated correctly.
- The values for the Threshold of Solid Voxels and Additional Void Gripper Mask Layers lay between $TSV = 120 \sim 250$ and $H_{add} = 1 \sim 5$.

Overall, a sufficient metric was found to adequately evaluate the test points given a comparatively small amount of reference data.

5.3 Data Set Accuracy

Within the scope of this study, there were no metrics developed to determine the accuracy of the field test itself. The field test is also a singular test that would need to be repeated for a more meaningful result, the algorithm would need to be tested extensively in other scenarios.

Accuracy with respect to the data set means the difference of the input raw terrain point cloud used for the algorithm's input compared to the ground truth of the actual test side. Regarding the 3 m $\times$ 3 m map of the artificial rocks data set, the following information are known for the accuracy. The terrain was scanned using the Kinect v2 device, which has a constant accuracy in the form of an offset of 18 mm [70] and a sensor accuracy of $\pm$ 0.5 px [61].

The SLAM method used is RTAB-Map, which in itself has a low accuracy, with Root Mean Square Error (RMSE) between estimated and ground truth trajectories in indoor environments hit sometimes several meters [71]. In large-scale outdoor tests, conducted by the SRL, the RMSE reaches an average value of 12 cm in the xy plane [72]. These errors can be reduced significantly with the combination of graph optimisation algorithms (see Chapter 3.1.4). The resulting colored and textured mesh of the test site, as shown in Fig. 3–5, shows a high degree of correspondence with the photos taken of the artificial rocks. Although the resulting simulation environment have not been systematically evaluated, an approximate error can be estimated. A spotcheck comparison along the four edge lengths of the resulting simulation test site with the ground truth of field test site shows that the average absolute error in z is 40 mm, as shown in Tab. B2–2.

When creating the surface mesh of the optimised point cloud, it is subsequently filtered with a grid that has a voxel size of 0.01 m. The inaccuracy of the RGB-D camera RealSense D435i used for the robot's prototype was not implemented in the subsequent CoppeliaSim simulation when creating the depth camera model, thus, the input data set for the real-time scenarios has the same accuracy as the terrain mesh used in the simulation.

Obtaining the necessary size of data for the significance of a binary classification evaluation typically requires a in-depth statistical power analysis, which is to be done for a more systematic testing of the algorithm.

6 Conclusion and Outlook

This work introduced the SRL Graspable Target Detection Algorithm (SRL-GTD), developed for use with hexapod climbing and exploration robots, with a particular focus on its application to the SCAR-E robot, designed for planetary and terrestrial exploration tasks. SRL-GTD is a geometry-based salient shape recognition algorithm that processes depth information perceived by the robot's depth sensors and predicts terrain accessibility using a gripper-based geometric mask.

6.1 Results Roundup

To establish a representative testing environment for simulating conditions approximating the rugged nature of an asteroid, and to subsequently conduct tests on this environment to validate the performance of SRL-GTD, we constructed a 3 m × 3 m test area comprised of artificial rocks. This terrain is scanned with a Kinect v2 device in combination with the SLAM method RTAB-Map. Further map refinement was accomplished using Robust Graph Optimization algorithms g2o and SBA, which results in a highly accurate emulation of the test side with an estimated mean error of $\sim$0.04 m. Subsequently, a simulation environment is created in CoppeliaSim, which included a controllable model of SCAR-E. Furthermore, a small-scale field test was carried out, involving the deployment of the prototype of the gripper at 30 locations on the artificial rock formation, the result of which is used as a reference in the evaluation of the algorithm.

This elaborate replica of the test site was examined for graspable points employing the SRL-GTD methodology. This analysis showed that the algorithm can provide us detailed information about the probability of being suitable for gripping throughout the field test. The Receiver Operating Characteristic (ROC) analysis shows a robust discriminative performance for this test scenario. Simultaneously, we devised other scenarios with better accessibility. The examination of primitive geometric shapes with the algorithm shows an extensive agreement with the expected result. In a terrain with gentle slopes and elevations, the algorithm demonstrates the capability to compensate occluded areas behind obstacles, thereby meaningfully limiting the accessible area. On a bouldering wall with 58 climbing holds, all elevations could be easily localized. All depressions and the smooth, flat ground were successfully excluded from the graspable region.

Compared to the original ClimbLab concept, SRL-GTD brings several innovations. First, it introduces a new criterion that computes the probability of graspability of the terrain. This criterion assigns a Graspability score to each point and subsequently provides a visual representation. Secondly, the algorithm examines each point's principal curvature in the terrain, identifying convex regions. The overlap of the maxima of Graspability score and the peaks of convex regions is considered as safe graspable points. This method achieves a accuracy rate of 73% in difficult terrain with a low false positive rate of 17%.

When comparing pre-scanned terrain maps, typically derived from remote sensing, with simulation-generated in-situ scenarios, it becomes evident that the pre-scanned scenarios exhibit a notably high weighted arithmetic mean of accuracy, exceeding 70%. These scenarios also demonstrate a remarkable mean precision rate of 83%, at very low mean false positive rate of 16%. In contrast, the real-time cases yield similarly high values regarding accuracy, but they are accompanied by a comparatively higher weighted mean of false positive rate of $\sim$50%.

Regardless of the pose of the robot in the terrain, the algorithm aligns the depth information processed into the point cloud according to its regression level. Furthermore, in easy accessible terrain, the algorithm exhibits its ability of compensating occluded areas through interpolation. In this case, the result for the interpolated real-time point cloud has an accuracy of over 80 percent, while in difficult terrain this value drops to 69%.

Furthermore, SRL-GTD is designed to work seamlessly in a multi-sensor environment, processing and evaluating data from individual depth images into a coherent, panoramic point cloud. Its integration with ROS facilitates high connectivity and adaptability, while also significantly improving its runtime by using C++.

6.2 Possible Optimization Approaches

The future development aspect of SRL-GTD includes the improvement of the algorithm in steeper elevated terrain, the integration of an enhanced gripper mask design, which superiorly represents the gripper's extent, and further runtime acceleration

High rocks with vertical walls are a comparatively major obstacle for the algorithm, because the steep surface leads to erroneous results due to the geometry of the Gripper Mask. Therefore, high obstacles that are difficult to overcome should be detected and marked in the next step so that they can be bypassed by the robot.

In case of limited accessibility of high rocks, or if climbing steep walls is explicitly desired, the orientation of the Gripper Mask should be adjusted accordingly so that it adapts to the slope of a surface. A more enhanced cylindrical design of the gripper mask could also improve the specificity of the algorithm, and therefore needs to be intensively tested.

The algorithm's capacity for real-time operation is notably facilitated by its minimal data requirements, limited to the depth images from the cameras for point cloud generation and parameter provision for the gripper mask. In this aspect, it could potentially outperform contemporary deep learning approaches by eliminating the need for extensive datasets in graspable target detection. Despite these advantages, significant potential for enhancing the algorithm's runtime remains untapped. The extensive utilization of pointers stands as a promising way for substantial runtime improvements. Furthermore, the incorporation of multiprocessing strategies, such as CUDA or OpenMP, should be explored. These parallel processing frameworks have the potential to dramatically accelerate computation, particularly for computationally intensive sections of the algorithm.

As of November 2023, SCAR-E has successfully reached significant milestones in its development, including the establishment of a comprehensive Inverse Kinematics system and the completion of the prototype assembly, inclusive of all electronic components. The forthcoming critical phase entails the seamless integration of the SRL-GTD algorithm with the robot's inverse kinematics, marking a pivotal step towards the incorporation of the algorithm within the SCAR-E system. This integration represents a crucial convergence of advanced robotic capabilities, contributing to the continued advancement of SCAR-E's functionality and mission readiness.

Bibliography

[1] ESA, Nov 2014, accessed: 2023-08-22. [Online]. Available: https://www.flickr.com/photos/europeanspaceagency/4781143008/

[2] T. R. Watters, P. C. Thomas, and M. S. Robinson, "Thrust faults and the near-surface strength of asteroid 433 eros," *Geophysical Research Letters*, vol. 38, no. 2, 2011.

[3] T. Yoshimitsu and T. Kubota, "Asteroid surface exploration by minerva-ii small rovers," in *18th Annual Meeting of the Asia Oceania Geosciences Society: Proceedings of the 18th Annual Meeting of the Asia Oceania Geosciences Society (AOGS 2021)*. World Scientific, 2022, pp. 180–182.

[4] Y. Tsuda, T. Saiki, F. Terui, S. Nakazawa, M. Yoshikawa, and S. ichiro Watanabe, "Hayabusa2 mission status: Landing, roving and cratering on asteroid ryugu," *Acta Astronautica*, vol. 171, pp. 42–54, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0094576520301089

[5] T. Yoshimitsu, T. Kubota, T. Adachi, and Y. Kuroda, "Advanced robotic system of hopping rovers for small solar system bodies," in *Proc. 11th Int. Symp. Artif. Intel., Robot. Autom. in Space*, 2012.

[6] K. Uno, N. Takada, T. Okawara, K. Haji, A. Candalot, W. F. Ribeiro, K. Nagaoka, and K. Yoshida, "Hubrobo: A lightweight multi-limbed climbing robot for exploration in challenging terrain," in *2020 IEEE-RAS 20th International Conference on Humanoid Robots (Humanoids)*. IEEE, 2021, pp. 209–215.

[7] U. Kentaro, K. Yusuke, H. Keigo, K. Maximilian, H. Simon, J. William, N. Kenji, and Y. Kazuya, "Non-periodic gait planning based on salient region detection for a planetary cave exploration robot," 2020.

[8] A. Candalot, "Soft gripping solution and performances for a multi-limbed climbing robot," master thesis, Tohoku University, 2021.

[9] E. Neppel, "Adaptive gait sequence planning on hexapod walker for foothold limited terrain," master thesis, Tohoku University, 2023.

[10] A. Cellino, S. J. Bus, A. Doressoundiram, D. Lazzaro *et al.*, "Spectroscopic properties of asteroid families," *Asteroids III. Univ. of Arizona Press, Tucson*, pp. 633–643, 2002.

[11] F. DeMeo, C. Alexander, K. Walsh, C. Chapman, R. Binzel *et al.*, "The compositional structure of the asteroid belt," *Asteroids iv*, vol. 1, p. 13, 2015.

[12] NASA, Jul 2021. [Online]. Available: https://solarsystem.nasa.gov/asteroids-comets-and-meteors/asteroids/in-depth/

[13] J. N. Abrahams and F. Nimmo, "Ferrovolcanism: Iron volcanism on metallic asteroids," *Geophysical research letters*, vol. 46, no. 10, pp. 5055–5064, 2019.

[14] P. Vernazza, M. Ferrais, L. Jorda, J. Hanuš, B. Carry, M. Marsset, M. Brož, R. Fétick, M. Viikinkoski, F. Marchis *et al.*, "Vlt/sphere imaging survey of the largest main-belt asteroids: Final results and synthesis," *Astronomy & Astrophysics*, vol. 654, p. A56, 2021.

[15] I. Belskaya and C.-I. Lagerkvist, "Physical properties of m class asteroids," *Planetary and Space Science*, vol. 44, no. 8, pp. 783–794, 1996.

[16] J. C. Gradie, C. R. Chapman, and E. F. Tedesco, "Distribution of taxonomic classes and the compositional structure of the asteroid belt." *Asteroids II*, pp. 316–335, 1989.

[17] K. M. Cannon, M. Gialich, and J. Acain, "Precious and structural metals on asteroids," *Planetary and Space Science*, vol. 225, p. 105608, 2023.

[18] Y. Marrocchi, D. V. Bekaert, and L. Piani, "Origin and abundance of water in carbonaceous asteroids," *Earth and Planetary Science Letters*, vol. 482, pp. 23–32, 2018.

[19] Y. Oba, T. Koga, Y. Takano, N. O. Ogawa, N. Ohkouchi, K. Sasaki, H. Sato, D. P. Glavin, J. P. Dworkin, H. Naraoka *et al.*, "Uracil in the carbonaceous asteroid (162173) ryugu," *Nature Communications*, vol. 14, no. 1, p. 1292, 2023.

[20] M. Martínez-Jiménez, C. E. Moyano-Cambero, J. M. Trigo-Rodríguez, J. Alonso-Azcárate, and J. Llorca, "Asteroid mining: mineral resources in undifferentiated bodies from the chemical composition of carbonaceous chondrites," in *Assessment and Mitigation of Asteroid Impact Hazards: Proceedings of the 2015 Barcelona Asteroid Day*. Springer, 2017, pp. 73–101.

[21] R. Jaumann, N. Schmitz, T.-M. Ho, S. Schröder, K. A. Otto, K. Stephan, S. Elgner, K. Krohn, F. Preusker, F. Scholten *et al.*, "Images from the surface of asteroid ryugu show rocks similar to carbonaceous chondrite meteorites," *Science*, vol. 365, no. 6455, pp. 817–820, 2019.

[22] K. T. Smith, "Landing on the surface of ryugu," *Science*, vol. 365, no. 6455, pp. 769–769, 2019.

[23] M. Adachi, H. Maezono, and H. Kawamoto, "Sampling of regolith on asteroids using electrostatic force," *Journal of Aerospace Engineering*, vol. 29, no. 4, p. 04015081, 2016.

[24] S. Watanabe, M. Hirabayashi, N. Hirata, N. Hirata, R. Noguchi, Y. Shimaki, H. Ikeda, E. Tatsumi, M. Yoshikawa, S. Kikuchi *et al.*, "Hayabusa2 arrives at the carbonaceous asteroid 162173 ryugu—a spinning top–shaped rubble pile," *Science*, vol. 364, no. 6437, pp. 268–272, 2019.

[25] R. I. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*, 2nd ed. Cambridge University Press, ISBN: 0521540518, 2004.

[26] K. A. Tychola, I. Tsimperidis, and G. A. Papakostas, "On 3d reconstruction using rgb-d cameras," *Digital*, vol. 2, no. 3, pp. 401–421, 2022.

[27] D. Scharstein and R. Szeliski, “High-accuracy stereo depth maps using structured light,” in *2003 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2003. Proceedings.*, vol. 1. IEEE, 2003, pp. I–I.

[28] “Intel realsense d435i datasheet,” Jan 2019. [Online]. Available: https://www.intel.com/content/dam/support/us/en/documents/emerging-technologies/intel-realsense-technology/IntelRealSenseTrackingT265Datasheet.pdf

[29] M. Labbé and F. Michaud, “Memory management for real-time appearance-based loop closure detection,” in *2011 IEEE/RSJ international conference on intelligent robots and systems*. IEEE, 2011, pp. 1271–1276.

[30] ——, “Appearance-based loop closure detection for online large-scale and long-term operation,” *IEEE Transactions on Robotics*, vol. 29, no. 3, pp. 734–745, 2013.

[31] ——, “Online global loop closure detection for large-scale multi-session graph-based slam,” in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2014, pp. 2661–2666.

[32] ——, “Long-term online multi-session graph-based splam with memory management,” *Autonomous Robots*, vol. 42, pp. 1133–1150, 2018.

[33] ——, “Rtab-map as an open-source lidar and visual simultaneous localization and mapping library for large-scale and long-term online operation,” *Journal of field robotics*, vol. 36, no. 2, pp. 416–446, 2019.

[34] ——, “Multi-session visual slam for illumination-invariant re-localization in indoor environments,” *Frontiers in Robotics and AI*, vol. 9, p. 801886, 2022.

[35] G. Grisetti, R. Kümmerle, C. Stachniss, and W. Burgard, “A tutorial on graph-based slam,” *IEEE Intelligent Transportation Systems Magazine*, vol. 2, no. 4, pp. 31–43, 2010.

[36] R. Kümmerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard, “g 2 o: A general framework for graph optimization,” in *2011 IEEE International Conference on Robotics and Automation*. IEEE, 2011, pp. 3607–3613.

[37] F. Dellaert, “Factor graphs and gtsam: A hands-on introduction,” *Georgia Institute of Technology, Tech. Rep*, vol. 2, p. 4, 2012. [Online]. Available: https://github.com/borglab/gtsam

[38] M. I. Lourakis and A. A. Argyros, “Sba: A software package for generic sparse bundle adjustment,” *ACM Transactions on Mathematical Software (TOMS)*, vol. 36, no. 1, pp. 1–30, 2009.

[39] K. Konolige and W. Garage, “Sparse sparse bundle adjustment.” in *BMVC*, vol. 10, 2010, pp. 102–1.

[40] Y. Chen and G. Medioni, “Object modelling by registration of multiple range images,” *Image and vision computing*, vol. 10, no. 3, pp. 145–155, 1992.

[41] P. J. Besl and N. D. McKay, “Method for registration of 3-d shapes,” in *Sensor fusion IV: control paradigms and data structures*, vol. 1611. Spie, 1992, pp. 586–606.

[42] S. Rusinkiewicz and M. Levoy, “Efficient variants of the icp algorithm,” in *Proceedings third international conference on 3-D digital imaging and modeling*. IEEE, 2001, pp. 145–152.

[43] F. Spenrath, A. Spiller, and A. Verl, “Gripping point determination and collision prevention in a bin- picking application,” in *ROBOTIK 2012; 7th German Conference on Robotics*, 2012, pp. 1–6.

[44] J. Aleotti, D. Lodi Rizzini, and S. Caselli, “Perception and grasping of object parts from active robot exploration,” *Journal of Intelligent & Robotic Systems*, vol. 76, pp. 401–425, 2014.

[45] G. Kahn, P. Sujan, S. Patil, S. Bopardikar, J. Ryde, K. Goldberg, and P. Abbeel, “Active exploration using trajectory optimization for robotic grasping in the presence of occlusions,” in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, 2015, pp. 4783–4790.

[46] B. Zhao, H. Zhang, X. Lan, H. Wang, Z. Tian, and N. Zheng, “Regnet: Region-based grasp network for end-to-end grasp detection in point clouds,” in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, 2021, pp. 13 474–13 480.

[47] E. Corona, G. Alenyà, A. Gabas, and C. Torras, “Active garment recognition and target grasping point detection using deep learning,” *Pattern Recognition*, vol. 74, pp. 629–641, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0031320317303941

[48] A. Iriondo, E. Lazkano, and A. Ansuategi, “Affordance-based grasping point detection using graph convolutional networks for industrial bin-picking applications,” *Sensors*, vol. 21, no. 3, 2021. [Online]. Available: https://www.mdpi.com/1424-8220/21/3/816

[49] H. Liu, Y. Yuan, Y. Deng, X. Guo, Y. Wei, K. Lu, B. Fang, D. Guo, and F. Sun, “Active affordance exploration for robot grasping,” in *Intelligent Robotics and Applications: 12th International Conference, ICIRA 2019, Shenyang, China, August 8–11, 2019, Proceedings, Part V 12*. Springer, 2019, pp. 426–438.

[50] K. Uno, W. F. Ribeiro, Y. Koizumi, K. Haji, K. Kurihara, W. Jones, and K. Yoshida, “Climblab: Matlab simulation platform for legged climbing robotics,” in *Robotics for Sustainable Future: CLAWAR 2021 24*. Springer, 2022, pp. 229–241.

[51] W. Wang, Q. Lai, H. Fu, J. Shen, H. Ling, and R. Yang, “Salient object detection in the deep learning era: An in-depth survey,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 44, no. 6, pp. 3239–3259, 2021.

[52] W. Wang, S. Zhao, J. Shen, S. C. Hoi, and A. Borji, "Salient object detection with pyramid attention and salient edges," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019, pp. 1448–1457.

[53] M. Hisada, A. G. Belyaev, and T. L. Kunii, "A skeleton-based approach for detection of perceptually salient features on polygonal surfaces," in *Computer Graphics Forum*, vol. 21, no. 4. Wiley Online Library, 2002, pp. 689–700.

[54] K. Nagaoka, H. Minote, K. Maruya, Y. Shirai, K. Yoshida, T. Hakamada, H. Sawada, and T. Kubota, "Passive spine gripper for free-climbing robot in extreme terrain," *IEEE Robotics and Automation Letters*, vol. 3, no. 3, pp. 1765–1770, 2018.

[55] K. Uno, "Autonomous limbed climbing robots for challenging terrain exploration," doctoral thesis, Tohoku University, 2021.

[56] S.-W. Cheng, T. K. Dey, and J. Shewchuk, *Delaunay mesh generation*. CRC Press, 2012.

[57] T. J. Baker, "Automatic mesh generation for complex three-dimensional regions using a constrained delaunay triangulation," *Engineering with Computers*, vol. 5, pp. 161–175, 1989.

[58] J. Sell and P. O'Connor, "The xbox one system on a chip and kinect sensor," *IEEE Micro*, vol. 34, no. 2, pp. 44–53, 2014.

[59] J. Blake, F. Echtler, C. Kerl, and L. Xiang, "libfreenect2: Open source drivers for the kinect for windows v2 device," https://github.com/OpenKinect/libfreenect2, 2015.

[60] T. Wiedemeyer, "IAI Kinect2," https://github.com/code-iai/iai_kinect2, Institute for Artificial Intelligence, University Bremen, 2014 – 2015, accessed June 12, 2015.

[61] P. Fankhauser, M. Bloesch, D. Rodriguez, R. Kaestner, M. Hutter, and R. Siegwart, "Kinect v2 for mobile robot navigation: Evaluation and modeling," in *2015 international conference on advanced robotics (ICAR)*. IEEE, 2015, pp. 388–394.

[62] A. Koutsoudis, B. Vidmar, G. Ioannakis, F. Arnaoutoglou, G. Pavlidis, and C. Chamzas, "Multi-image 3d reconstruction data evaluation," *Journal of Cultural Heritage*, vol. 15, no. 1, pp. 73–79, 2014. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1296207412001926

[63] M. Kazhdan, M. Bolitho, and H. Hoppe, "Poisson surface reconstruction," in *Proceedings of the fourth Eurographics symposium on Geometry processing*, vol. 7, 2006, p. 0.

[64] P. Cignoni, G. Ranzuglia, M. Callieri, M. Corsini, F. Ganovelli, N. Pietroni, M. Tarini *et al.*, "Meshlab," 2011.

[65] E. Rohmer, S. P. Singh, and M. Freese, “V-rep: A versatile and scalable robot simulation framework,” in *2013 IEEE/RSJ international conference on intelligent robots and systems*. IEEE, 2013, pp. 1321–1326.

[66] A. Koubâa *et al.*, *Robot Operating System (ROS)*. Springer, 2017, vol. 1.

[67] W. Kühnel, *Lokale Flächentheorie*. Wiesbaden: Springer Fachmedien Wiesbaden, 2013, pp. 39–94. [Online]. Available: https://doi.org/10.1007/978-3-658-00615-0_3

[68] M. Corsini, P. Cignoni, and R. Scopigno, “Efficient and flexible sampling with blue noise properties of triangular meshes,” *IEEE transactions on visualization and computer graphics*, vol. 18, no. 6, pp. 914–924, 2012.

[69] J. Fan, S. Upadhye, and A. Worster, “Understanding receiver operating characteristic (roc) curves,” *Canadian Journal of Emergency Medicine*, vol. 8, no. 1, p. 19–20, 2006.

[70] O. Wasenmüller and D. Stricker, “Comparison of kinect v1 and v2 depth images in terms of accuracy and precision,” in *Computer Vision – ACCV 2016 Workshops*, C.-S. Chen, J. Lu, and K.-K. Ma, Eds. Cham: Springer International Publishing, 2017, pp. 34–45.

[71] N. Ragot, R. Khemmar, A. Pokala, R. Rossi, and J.-Y. Ertaud, “Benchmark of visual slam algorithms: Orb-slam2 vs rtab-map,” in *2019 Eighth International Conference on Emerging Security Technologies (EST)*, 2019, pp. 1–6.

[72] D. Vincent, “Internship report 09/2023,” test report, Tohoku University, 2023.

A Appendix: Listings

Listing A.1: Function least_square_plane() computes the centroid and normal vectors of the point cloud's regression plane.

```
In {raw_cloud}
Out {normal_vector_of_regression_plane, centroid_vector}
  //compute the centroid
  centroid_vector_of_plane = pcl::compute3DCentroid(raw_cloud);
  // substracting the centroid (vector) to each column of the
     ↪ matrix, one column being a point
  for(int i=0; i< raw_cloud.size(); i++)
   {
    deviation_matrix(0,i)=(raw_cloud(0,i) -
       ↪ centroid_vector_of_plane(0));
    deviation_matrix(1,i)=(raw_cloud(1,i) -
       ↪ centroid_vector_of_plane(1));
    deviation_matrix(2,i)=(raw_cloud(2,i) -
       ↪ centroid_vector_of_plane(2));
  }

  //compute the product of deviation matrix
  deviation_matrix_transposed = deviation_matrix.transpose();
  product_deviation_matrix=deviation_matrix*
     ↪ deviation_matrix_transposed;

  //EigenSolver computes the eigen vectors and eigen values
  Eigen::EigenSolver<Eigen::MatrixXf> es(
     ↪ product_deviation_matrix);
  Eigen::Vector3cf X_eigen_values = es.eigenvalues();

...

  //sort the smallest eigenvalues and its corresponding index
  float a = X_eigen_values(0);
  int index=0;
  if (X_eigen_values(1) < a) {
    index = 1;
    a=X_eigen_values(1);
  }
  if (X_eigen_values(2) < a) {
    index = 2;
  }

  //choose vector corresponding to the smallest eigenvalue
```

```
  Eigen::Vector3cf normal_vector_of_plane = es.eigenvectors().
     ↪ col(index);

...
```

Listing A.2: Function pcd_transform() transforms coordinate of pointcloud from camera based to terrain based frame.

```
In {raw_cloud, centroid_vector,
   ↪ normal_vector_of_regression_plane}
Out {transformed_cloud}

    // cross product of normal and centroid vector
    y_vector_of_plane = centroid_vector.cross(
       ↪ normal_vector_of_regression_plane);
    // x - axis : cross product of y and normal vector
    x_vector_of_plane = normal_vector_of_regression_plane.cross
       ↪ (y_vector_of_plane);
    // z axis is the normal vector
    z_vecor_of_plane = normal_vector_of_regression_plane;
    rotation_matrix = [x, y, z];
    // Frame transformation
    transformed_cloud = rotation_matrix_transposed * raw_cloud
       ↪ - rotation_matrix_transposed * centroid_vector;
```

Listing A.3: Execution of the interpolation on the meshgrid.

```
In {transformed_cloud}
Out {interpolated_cloud}

std::vector<double> x,y,z;
pcl::PointXYZ point;

// assign points into vectors for interpolation
  for (all indices i in raw_cloud)
  {
    point = raw_cloud.points[i];
    x.push_back(point.x);
    y.push_back(point.y);
    z.push_back(point.z);
  }
...
// Compute the grid size using the width of raw cloud in x and
   ↪ y
// multiply and devide 10000 is for the suitable decimal places
   ↪ !
 double grid_size = 1/(round(sqrt(raw_pcd.size() / (x_width *
```

```
    ↪ y_width)*(5/3) ) * 10000) / 10000);
//create grid vectors
for (double i = min_y; i < max_y; i+=grid_size)
  {
    y_grid_vector.push_back(i);
  }

for (double i = min_x; i < max_x; i+=grid_size)
  {
    x_grid_vector.push_back(i);
  }
// Create a Delaunay triangle interpolator and add known data
    _2D::LinearDelaunayTriangleInterpolator<double>
      ↪ delaunay_interpolator;

//Interpolate values based on x and y vectors like a grid
    for (all indices i in x_grid_vector)
    {
        for (all indices j in y_grid_vector)
        {
            interp_z = delaunay_interpolator(x_grid_vector[i],
              ↪ y_grid_vector[j]);
            if (interp_z != 0)
            {
                interpolated_point_cloud.push_back(pcl::
                  ↪ PointXYZ(x[i], y_grid_vector[j], interp_z
                  ↪ ));
            }
        }
    }
```

Listing A.4: Key loop for the creation of the voxelized terrain array in pcd_voxelize().

```
In {transformed_cloud}
Out {voxelized_terrain_array}
...
// First, create a voxel grid filter
pcl::VoxelGrid<pcl::PCLPointCloud2> voxel_grid;
voxel_grid.setInputCloud(transformed_cloud); // set input into
  ↪ the grid
voxel_grid.setLeafSize(cube_size,cube_size,cube_size); // set
  ↪ leaf size
pcl_after_filtering = voxel_grid.filter

min_x = transformed_cloud.x.min();
min_y = transformed_cloud.y.min();
```

```
min_z = transformed_cloud.z.min();

int voxel_index_x, voxel_index_y, voxel_index_z;

// Iterate through the filtered point cloud to determine voxel
   ↪ occupancy.
for (all indices i inside the voxel grid filter)
{
    point = pcl_after_filtering.points[i];

    // Calculate voxel indices for the current point.
    voxel_index_x = trunc((point.x - min_x) / cube_size +
       ↪ cube_size / 4);
    voxel_index_y = trunc((point.y - min_y) / cube_size +
       ↪ cube_size / 4);
    voxel_index_z = trunc((point.z - min_z) / cube_size +
       ↪ cube_size / 4);

    // Mark the corresponding voxel as occupied.
    voxelized_terrain_array[voxel_index_x][voxel_index_y][
       ↪ voxel_index_z] = 1;
}
...
```

Listing A.5: Key loop for the creation of the gripper mask array in function creategrippermask().

```
In {gripper_parameters}
Out {gripper_mask}
// first, compute the voxel ratio and multiply & round all
   ↪ inputs with it...
...
for (all z subscripts in gripper_mask_height) {

        grippable_radius = (Eq. 3-11)
        inner_unreachable_radius = (Eq. 3-12)

        for (all y in gripper_mask_size) {
            for (all x in gripper mask size) {

                distance_from_center_of_layer = (Eq. 3-14)

                if ((z_subscript <= gripper_mask_clearance
                    && distance_from_center_of_layer <=
                       ↪ grippable_radius) ||
                    (z_subscript > gripper_mask_clearance
                    && z_subscript <= (gripper_mask_height-
```

```
                        ↪ gripper_mask_bottom_void_radius))
                    ||
                    (z_subscript > (gripper_mask_height-
                        ↪ gripper_mask_bottom_void_radius)
                    && z_subscript != gripper_mask_height
                    && distance_from_center_of_layer >
                        ↪ inner_unreachable_radius)
                    ||
                    (z_subscript == gripper_mask_height
                    && distance_from_center_of_layer >
                        ↪ gripper_mask_bottom_void_radius))
                    {
                    // flip in z
                    gripper_mask[x][y][gripper_mask_height - z]
                        ↪  = 1;
                    }
            }
        }
    }

     for (all indices i in gripper_mask_size) {
        for (all indices j gripper_mask_size) {
            gripper_mask[i][j].insert(gripper_mask[i][j].end(),
                ↪  extra_sheet, 0);
        }
    }
```

Listing A.6: Function voxel_matching(). In simplified pseudocode.

```
In {voxelized_terrain_array, gripper_mask}
Out {voxel_coordinates_of_graspable_points}

// Voxel Clip
clip_x = size_of_gripper_mask[0] / 2;
clip_y = size_of_gripper_mask[1] / 2;

// Crop in x-direction}
for (all points p in voxelized_terrain_array != clip_x :
   ↪ voxelized_terrain_array.size[0] - clip_x)
  {
    p = 0;
  }
// Analogue in y-direction
...

//Find solid voxels
```

```
for (all indices i,j,k in voxelized_terrain_array)
  {
    if(voxelized_terrain_array[i][j][k] != 0){
        subscripts_of_solids.x = i;
        subscripts_of_solids.y = j;
        subscripts_of_solids.z = k;
    }
  }

// Pivot point transformation
subscripts_of_solids.x = subscripts_of_solids.x - clip_x;
subscripts_of_solids.y = subscripts_of_solids.y - clip_y;

// Great loop
for(all indices i in subscripts_of_solids.x){

    // Voxel Extract
    pos_x = subscripts_of_solids.x[i];
    pos_y = subscripts_of_solids.y[i];
    pos_z = subscripts_of_solids.z[i];

    for(all indices x,y,z in size_of_gripper_mask){
        extracted_voxel_array[x][y][z] =
          ↪ voxelized_terrain_array[pos_x + x][pos_y + y][
          ↪ pos_z + z];
    }

    // Voxel Compare
    // Count the number of Solid Voxels within the Subset
    subset_solid_voxels = 0;
    for (all points p in extracted_voxel_array) {
        subset_solid_voxels += p;
    }
    for (all indices x,y,z in extracted_voxel_array) {
        gripper_solid_voxels += extracted_voxel_array[x][y][z]
          ↪ * gripper_mask[x][y][z];
    }
    graspability = gripper_solid_voxels / subset_solid_voxels;

    // Build outout array
    voxelized_coordinates_of_graspable_points[0][i] =
      ↪ subscripts_of_solids.x;
    voxelized_coordinates_of_graspable_points[1][i] =
      ↪ subscripts_of_solids.y;
    voxelized_coordinates_of_graspable_points[2][i] =
```

```
        ↪ subscripts_of_solids.z;

    // Thresholding
    if (subset_solid_voxels > solid_voxels_threshold) {
        voxelized_coordinates_of_graspable_points[3][i] =
            ↪ graspability;
      }
    // if Subset Solid Voxels less than threshold AND
        ↪ graspability too high, then penalize. otherwise no
        ↪ need to do that.
    else if (subset_solid_voxels <= solid_voxels_threshold &&
        ↪ graspability >= 60) {
        voxelized_coordinates_of_graspable_points[3][i] =
            ↪ graspability - (solid_voxels_threshold -
            ↪ number_of_matching_voxels)/solid_voxels_threshold
            ↪ ;
     }
    else {
      voxelized_coordinates_of_graspable_points[3][i] =
          ↪ graspability;
    }

}
```

Listing A.7: Depth sensor controller in Lua. Depth image generation and ROS2 integration.

```
-- Generation of the depth image.
function sysCall_vision(inData)
    local retVal={}
    retVal.trigger=false
    retVal.packedPackets={}
    simVision.sensorDepthMapToWorkImg(inData.handle)
    --  Transforms the work image into an intensity
        ↪ representation.
    -- start: the value representing the minimum intensity. end
        ↪ : the value representing the maximum intensity.
    simVision.intensityScaleOnWorkImg(inData.handle,0.000000,1
        ↪ .000000,true)
    simVision.workImgToSensorImg(inData.handle)
    return retVal
end

-- ROS2 interface initialization
function sysCall_init()

    if simROS2 then
```

```
        sim.addLog(sim.verbosity_scriptinfos,"ROS interface was
            ↪  found.")
        -- Create a topic publisher.
        depthPub=simROS2.createPublisher('spherical2/depth', '
            ↪ sensor_msgs/msg/Image')
        -- treat uint8 arrays as strings (much faster, tables/
            ↪ arrays are slow in Lua)
        simROS2.publisherTreatUInt8ArrayAsString(depthPub)
    else
        sim.addLog(sim.verbosity_scripterrors,"ROS interface
            ↪ was not found. Cannot run.")
    end

end

-- ROS2 message specification and publishing
function sysCall_sensing()

     if simROS2 then
        -- Publish the image of the active depth sensor:
        local data,w,h=sim.getVisionSensorCharImage(depthCam)
        sim.transformImage(data,{w,h},4)
        d={}
        d.header={stamp=simROS2.getTime(), frame_id='depth_mert
            ↪ '}
        d.height=h
        d.width=w
        d.encoding='rgb8'
        d.is_bigendian=1
        d.step=w*3
        d.data=data
        simROS2.publish(depthPub,d)
    end

end
```

B Appendix: Figures and Tables

Tab. 2–1: Paramter setting used for the scenarios tested with the algorithm. H_{add}: Additional Void Gripper Mask Layers; TSV: Threshold of Solid Voxels

	Scenario 1	Scenario 2	Scenario 3		Scenario 4	
Parameters			pre-scan	real-time	pre-scan	real-time
Interpolation	yes	yes	no	yes	no	yes
Zero-plane threshold [m]	0.00	-0.07	0.02	0.02	-0.05	-0.05
Voxel size	0.005	0.01	0.01	0.01	0.01	0.01
H_{add}	5	1	1	1	1	1
TSV	120	120	120	120	120	120

Tab. 2–2: A spotcheck of z-offset along the four edge length a, b, c and d of the Kinect scanned and refined point cloud of the test side shows the difference to the ground truth (0.00 m).

Edge	Error	[m]					
a	0.031	0.023	0.029	0.023	0.028	0.042	0.029
b	0.043	0.018	0.013	0.014	0.005	0.006	0.006
c	0.093	0.068	0.067	0.066	0.077	0.065	0.041
d	0.045	0.050	0.041	0.044	0.052	0.055	0.052
Average							**0.040**

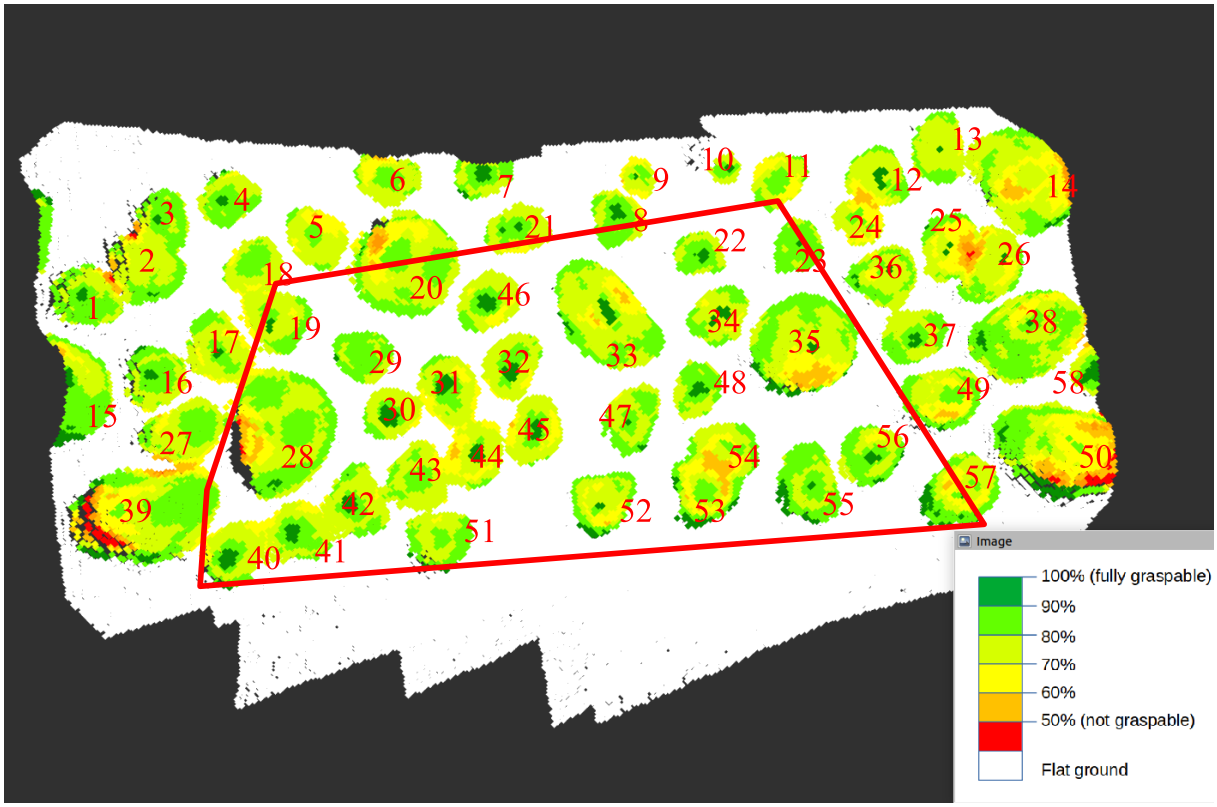


Fig. 2–1: Positions investigated in scenario 3 and corresponding results of the five cases. The area visible in the real-time cases is highlighted in red.

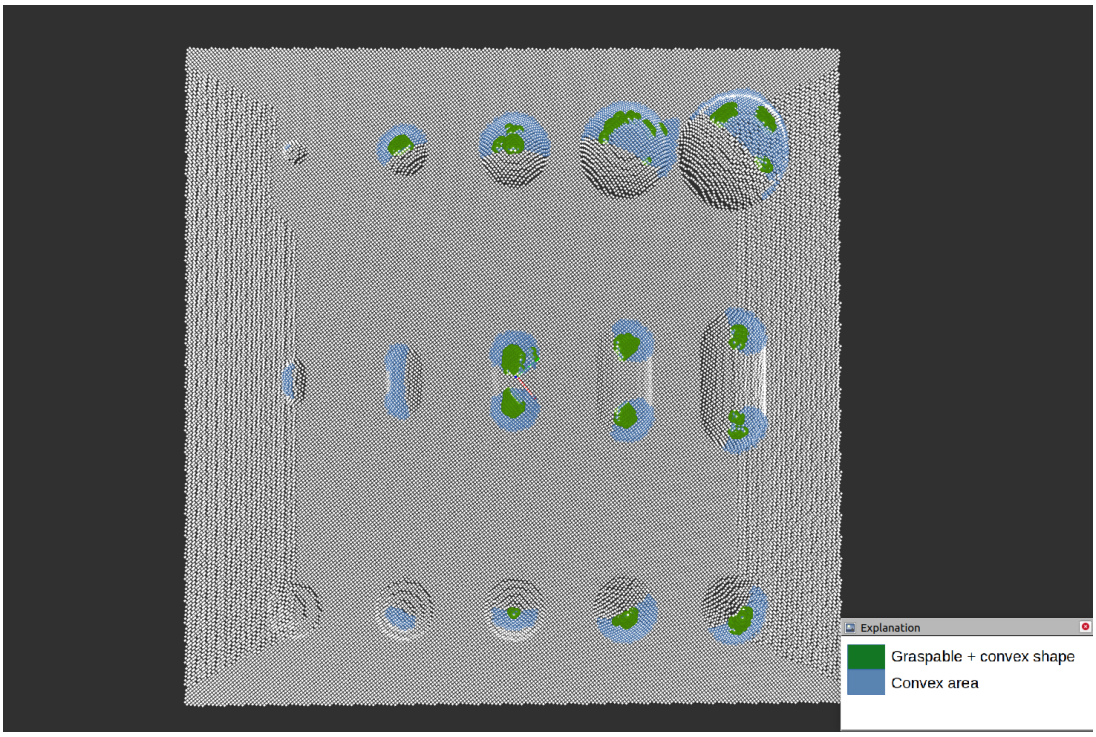


Fig. 2–2: Curvature map (blue) and intersections (green) with regions of high graspability. The erroneous convex shape detection can be noticed, since the convex point's normal vector never points toward the center.

Tab. 2–3: Graspability values read from the map at the corresponding points examined in the field test. 1 denotes full graspabiliy. 0 indicates that it cannot be grasped.

Point no.	Field test	Pre-scaned map	Real-time map raw	Real-time map interpolated	Realtime map combined
1	0	0.4			
2	0	0.8			
3	1	0.8	0.8	0.7	1
4	1	0.5			
5	1	0.9			
6	1	0.8			
7	0	0.7			
8	1	0.3			
9	1	0.4			
10	0	0.8			
11	1	0.9	0.9	0.9	1
12	1	0.8	0.8	0.7	1
13	0	0.6	0.6	0.8	0
14	1	0.8	0.6	0.3	0
15	0	0.8	0.7	0.3	1
16	0	0.3			
17	1	0.8			
18	0	0.5			
19	0	0.4	0.7	0.5	0
20	0	0.4	0.8	0.3	0
21	1	0.6	0.8	0.5	1
22	1	0.9	0.9	0.8	1
23	0	0.6	0.8	0.8	0
24	1	0.8	0.6	0.7	1
25	1	0.5	0.6	0.7	0
26	1	0.8	0.8	0.8	0
27	1	0.8			
28	1	0.8			
29	0	0.5	0.7	0.4	1
30	1	0.8	0.8	0.4	1

Tab. 2–4: Results of the five cases in scenario 3. I: Pre-scanned raw map (reference); II: Pre-scanned curvature map; III: Real-time raw map; IV: Real-time interpolated map; V: Real-time curvature map. 1 denotes full graspabiliy. 0 indicates that it cannot be grasped.

Hold no.	I	II	III	IV	V	Hold no.	I	II	III	IV	V
1	1	0				30	1	1	0	1	0
2	0	0				31	1	1	1	1	1
3	1	0				32	1	1	1	1	0
4	1	0				33	1	1	1	1	1
5	0	0				34	1	1	1	1	1
6	0	0				35	1	1	1	1	1
7	1	0				36	1	1			
8	1	1				37	1	1			
9	1	1				38	1	1			
10	1	1				39	0	1			
11	0	0	1	0	0	40	1	1	1	1	1
12	1	1				41	1	1	1	1	1
13	1	1				42	1	1	1	1	1
14	0	0				43	0	0	1	1	1
15	1	0				44	1	1	1	1	1
16	1	0				45	1	1	1	1	1
17	1	0				46	1	1	1	1	1
18	0	0				47	1	1	1	1	1
19	1	1	1	1	1	48	1	1	1	1	1
20	0	0	1	1	1	49	0	0	1	0	0
21	1	1				50	0	0			
22	1	1	1	1	1	51	0	0	1	1	1
23	1	1	1	1	1	52	1	1	0	1	1
24	0	0				53	0	1	1	1	1
25	1	0				54	0	0	0	0	0
26	1	1				55	1	1	1	1	1
27	0	0				56	1	1	1	1	1
28	0	0	1	0	1	57	1	1	1	1	1
29	0	0	0	1	1	58	1	1			