Analysis and applications of reachability and capability maps for robotic manipulators

Oliver Porges

DLR

RMC, Institut für Robotik und Mechatronik	BJ.: 2013
	IB.Nr.: 572-2014/34

MASTERARBEIT

ANALYSIS AND APPLICATIONS OF REACHABILITY AND CAPABILITY MAPS for robotic manipulators

Freigabe:	Der Bearbeiter:	Unterschriften
	Oliver Porges	
	Betreuer:	
	Dr. Máximo A. Roa	
	Der Institutsdirektor	
	Prof. Alin Albu-Schäffer	

Dieser Bericht enthält 63 Seiten, 48 Abbildungen und 20 Tabellen

Ort: Oberpfaffenhofen	Datum: 11.12.2014	Bearbeiter: Oliver Porges	Zeichen:

Luleå Technical University

Julius-Maximilians University Wurzburg

Analysis and applications of reachability and capability maps for robotic manipulators

Author: Oliver Porges, Space Master, round 7

Supervisors:

German Aerospace Center (DLR): Dr. Maximo A. Roa-Garzon

Julius Maximilian University of Würzburg: Prof. Dr.-Ing. Sergio Montenegro

Lulea Technical University: Dr. Thomas Kuhn

Abstract

Robotic arms are the most popular robots on the market. Technology behind the arm manipulators and their sensors is getting more accessible, which results in an increased interest from both commercial and research communities. Autonomous applications of robotic arms needs a combination of good sensory input and previous knowledge to speed up application development. Path planning for the robotic end effector, required to design a trajectory to move from an initial to a goal position, requires both knowledge of the kinematic structure of the robot as well as sensory information coming from the environment, that helps to identify key elements like the objects to manipulate, surfaces of support or possible obstacles. The kinematic structure that results from a design stage is fixed, and knowledge of the workspace of the robot and its dexterity in this space can be preprocessed to speed up on-line applications.

This thesis proposes a novel way to address the problem of computing the workspace of a robotic arm and its dexterity within this space. Our proposed off-line analysis of reachability is designed to answer questions about workspace shape and quality. We show how to use the precomputed structure for on-line grasp selection, operational workspace selection, collision free path planning, path validation or robot pose selection.

Our approach builds on top of the concept of reachability and capability maps. Traditional methods to generate such maps use forward or inverse kinematics, and we investigate the advantages and limitations of both. We later propose a hybrid method which combines their advantages while maintaining low generation time. Quality of the results is evaluated by a prediction accuracy test. The structure of the map is designed such that it is also suitable for on-line applications. Real-time visual information is incorporated into the map's data structure to improve real-world interactions such as grasp selection or collision-free path planning. To conclude, several examples of real applications that illustrate the usefulness of the map are presented and discussed.

Contents

1 Introduction **1**

1.1 Overview . . . 1

1.2 State of the art . . . 2

1.3 Outline of the thesis . . . 2

2 Generation and use of reachability maps **4**

2.1 Grid structure for reachability maps . . . 5

2.2 $\mathbb{R}^3$ mapping and indexing . . . 6

2.3 Dynamic map expansion . . . 7

2.4 $SO(2)$ mapping and indexing . . . 9

2.5 Mapping of TCP orientation into its bin . . . 10

2.6 Underlying memory structure . . . 12

2.7 Computation of the map . . . 13

2.7.1 Inverse kinematics method . . . 13

2.7.2 Forward kinematics method . . . 14

2.7.3 Hybrid method . . . 17

2.8 Collision detection and environment perception . . . 19

2.8.1 Collision primitives . . . 20

2.8.2 On-line collision detection . . . 21

2.9 From reachability to capability map . . . 21

2.9.1 Extended reachability index . . . 22

2.9.2 Fast implementation of Reachability index calculation . . . 23

2.9.3 Visualization . . . 23

3 Map quality assessment **26**

3.1 Prediction accuracy and error classification . . . 26

3.2 Prediction accuracy, generation time and error structure . . . 27

3.2.1 Forward kinematics . . . 28

3.2.2 Inverse kinematics . . . 30

3.2.3 Hybrid method 31
3.2.4 Results comparison 32
3.3 Source of prediction error 37
3.3.1 Error related to Cartesian position in the workspace 38
3.3.2 Error related to voxel structure 38
3.3.3 Error related to generation method 39

4 Applications **42**
4.1 On-line detection of reachable points 42
4.2 Grasp selection 43
4.3 Off-line workspace analysis for a free-floating manipulator 44
4.4 Workspace selection using capability maps 47
4.5 On-line workspace boundary estimation of a surgical robot 48
4.6 Path planning 50

5 Conclusions and future work **54**

A System architecture **56**

1. Introduction

1.1 Overview

There are a number of methods in literature to analyze the workspace of a robotic arm, which are usually applicable to simple kinematic chains. As the complexity of a chain increases with the number of joints, these methods either don't work or are computationally very expensive. Methods for static analysis are typically not suitable for on-line use. As an alternative, the concepts of reachability and capability maps have been proposed.

Reachability map is a commonly used concept in robotic manipulation to define the workspace and determine the best pose of a robot base to execute a particular task. It is a discretized representation of reachable poses of the robot's end effector frame (tool center point, TCP). A 6D voxelized structure is overlayed on the robot's workspace. Voxels are used to discretize the Cartesian space, and each voxel holds an inner structure that discretizes the remaining three dimensions for orientations. At the end we obtain a binary representation where each bin represents a small range of orientations and positions of the end effector. Bin's reachability is described by a binary value that indicates if the TCP can reach that particular position and orientation. Additional indices can be used to interpret certain quality measures of a group of bins or voxels. The most basic one is called reachability index which, when computed for the whole workspace, creates the capability map. Therefore, capability map is a reachability map plus a non-directional quality measure of elementary volume (voxel) quality. It reflects the robot's dexterity in a particular volume.

Currently, two methods are used to generate the reachability map. One approach uses inverse kinematics to determine whether a certain position is reachable or not. Each bin of the 6D grid is represented by its central vector (TCP position and orientation). If the inverse kinematics is able to solve for joint values to reach the TCP pose, the bin is marked as reachable. The generation and query of TCP poses can be systematically guided throughout the complete workspace. This is the guarantee of completeness of the method. A significant drawback is its computational requirements. Depending on the IK solver used, a solution might not always be found even if one exists. With increasing complexity of the kinematic chain, the method becomes computationally more expensive, resulting in lower number of poses verified per unit of time. The other common approach uses forward kinematics. This method consists of choosing random joint values, creating transformation matrices and multiplying them to obtain a random TCP pose. This process is, in general, computationally less expensive and easily vectorizable. It can generate a significantly larger number of samples in time compared to inverse kinematics. On the other hand, as the process is random, it is hard to guarantee a complete exploration of the workspace. Since the workspace exploration cannot be guided, this process is ran for a period of time. Random TCP samples are processed and represented in the map. As no stopping criteria have been proposed yet, we provide guidelines on when to stop the process. This document also describes a novel, hybrid approach for reachability map generation. We show that we can use the benefits of both methods to reduce the generation time and to guarantee a complete workspace exploration.

A large amount of data has to be processed during the off-line phase and a large data structure has to be loaded during the on-line phase. To be able to generate and operate such maps effectively, a cleverly designed data structure has to be used. Strong emphasis was put on low memory footprint as well as fast access times. This allows us to attend a large number of queries per unit of time. Moreover, achieving such performance opens possibilities for new on-line applications. Most common applications include quick examination and analysis of the workspace boundaries, identifying regions of high dexterity that are suitable for manipulation tasks or on-line validation for path execution.

1.2 State of the art

A proper understanding of robotic manipulator capabilities and their representation helps to solve planning and manipulation tasks more easily. Representation of the reachable poses of a robotic arm is commonly known as a reachability map, and it was introduced in [1]. This concept was later extended in [5] where a reachability index was introduced. An associated quality measure (as reachability index) provided a tool to examine the local workspace quality. An analytical approach to compute the boundaries of the workspace was introduced in [3], but for many applications we would need more information than just the bounding volumes of the reachable space.

Reachability map can be generated using forward or inverse kinematics. For forward kinematics generation, no guidelines have been provided on how to generate maps of certain quality. Inverse kinematics is generally a computationally demanding method, especially with large number of degrees of freedom of the kinematic chain [6]. Such knowledge representation then helps to validate reachable poses, validate feasibilities of trajectories and determine a good position of a mobile robot base with respect to a specific task like object manipulation, door or drawer opening [1], [7], [8], [9].

We examine the properties of both generation methods and we propose a hybrid method to reduce the generation time. A suitable memory structure is introduced to facilitate fast queries into the map and to have an on-line availability of capability map without storing it in the memory separately from the reachability map. With a memory structure similar to [13] we are able to extend the functionality of such knowledge representation to path planning, collision detection and scene perception.

1.3 Outline of the thesis

This document is organized as follows. Chapter 2 describes the memory structure for reachability and capability maps, generation methods and other necessary modules as collision and scene perception. Chapter 3 provides guidelines on how to evaluate the obtained data, and compares the three generation methods. Chapter 4 describes several applications to tackle some common robotic problems using the reachability and capability maps. We conclude the work in Chapter 5. Appendix A provides an overview of the system architecture and its implementation.

2. Generation and use of reachability maps

The maps we are about to present are complex hierarchical structures. In order to describe them precisely we shall define a common nomenclature that will be used throughout the document. A robot is a set of connected rigid bodies or so called links. Any two links are connected with a joint. Each joint has a range of motion, rotational or translational. A reference frame is a 6 dimensional vector describing the position and orientation of a rigid 3D object. Robot base is the reference frame of the first link, and robot's end effector is the reference frame at the other end of the serial kinematic chain, described with respect to the robot's base.

- **Robot workspace** - Set of all reachable end effector frames
- **Workspace bounding box** - The smallest box containing all the workspace, with sides aligned with the axis of the base reference frame
- $\mathbb{R}^3$ **space** - Space that describes the Cartesian position - 3D translational component
- **SO(2) group** - Space that describes all possible rotations achievable with two perpendicular axis of rotations
- **SE(3)** - Space for the Cartesian position and three degrees of freedom in rotation - a configuration space that describes all the possible positions and orientations of a rigid object
- **Grid** - Discretization structure of any of the spaces previously mentioned
- **Voxel** - A small cubical volume of $\mathbb{R}^3$ space
- **Approach direction** - A vector of orientation in $SO(2)$ space
- **Approach area** - A range of approach directions, a range in $SO(2)$
- **Bin** - A binary value which in our case represents a range of positions and orientations in $SE(3)$ space
- **TCP** - Tool Center Point or the robot's end effector
- **Sample** - or TCP sample is a particular position and orientation reached by a robot's end effector

Each one of the terms is important to understand and clearly define the hierarchy of discretization used. This chapter describes the discretization parameters and the underlying grid structure of reachability maps. Following is the description of the mapping process and query creation. Once the data manipulation is covered, we give details on the three generation methods.

2.1 Grid structure for reachability maps

Our goal is to create a structure that represents all reachable TCP positions and orientations. In order to maintain fast access times we have to avoid any searching, hence, the data needs to be index accessible. To achieve this, we have chosen the robot base as the map's reference frame. Any two maps we create will always contain the robot base coordinate as one vertex of one cell (voxel), therefore, the grid structures are always aligned. It allows us to directly compare identical grids obtained by different generation methods. We follow a scheme for splitting up the 6D space as follows. The workspace $\mathbb{R}^3$ is divided into finite number of small cubical elements - voxels. Each voxel contains a virtual inscribed sphere. The spherical surface is divided into a number of regions that provide a two-dimensional division grid (approach areas splitting up the pan - tilt parameters of a TCP). Each one of these regions is paired with one additional one-dimensional grid to divide the remaining parameter (roll angle). As any discretization method, the user needs to specify the desired resolution parameters. Fig. 2.1 illustrates these grids, and the parameters are defined as follows:

- **Voxel size -** v_s
  Voxel size refers to a side length of a cube. These cubes are inscribed inside the workspace and provide the Cartesian division grid in R^3
- **SO(2) divisions -** so_2
  Each voxel is divided into a number of approach directions that represent SO(2) space. This parameter maps the pan and tilt angles into a 2D grid
- **Number of layers -** n_l
  Additional parameter that allows the user to control how are the SO(2) bins distributed on the surface of the sphere
- **Number of roll divisions -** n_r
  Describes the discretization of the tool's roll angle

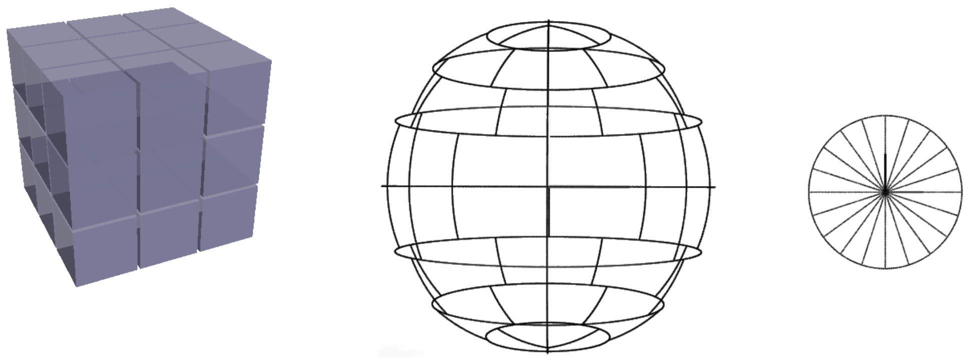


Figure 2.1: Overview of grid types and their hierarchy, $\mathbb{R}^3$, $SO(2)$, roll angle

2.2 $\mathbb{R}^3$ mapping and indexing

The continuous workspace volume W_c has a bounding box of dimensions $D_x \times D_y \times D_z$. We discretize W_c into n voxels as $\bigcup_{i=0}^{n-1} V_i$, where i denotes the voxel index. With the discretization parameter v_s we express the number of voxels in the map as,

$$n = \left\lceil \frac{D_x}{v_s} \right\rceil \times \left\lceil \frac{D_y}{v_s} \right\rceil \times \left\lceil \frac{D_z}{v_s} \right\rceil = d_x \times d_y \times d_z \in \mathbb{N} \tag{2.1}$$

where $\lceil x \rceil$ denotes the ceiling operator which rounds to the smallest integer that is not less than x.

Ideally, every voxel should be accessed with a single index i. We denote the Cartesian position of the TCP sample as $\vec{t}_{TCP} = \{t_x, t_y, t_z\}$. To determine a particular voxel to which the sample belongs to we need to define additional variables $\vec{m} = \{m_x, m_y, m_z\}$. They represent the lowest Cartesian coordinates that the map contains, to represent the position of voxel of index zero - V_0. The values of the vector $\vec{m}$ are always a multiple of the discretization parameter v_s. Now we are able to define the mapping of a sample's position in the 3D grid coordinates $\varrho(\vec{t}_{TCP}) \rightarrow \vec{t}_{grid}$ with $\vec{t}_{grid} = \{x, y, z\}$ where $x, y, z \in \mathbb{N}$, with $x, y, z \geq 0$. Sample's coordinates within the grid are

$$\vec{t}_{grid} = \left\lfloor \frac{\vec{t}_{TCP} - \vec{m}}{v_s} \right\rfloor \tag{2.2}$$

where $\lfloor x \rfloor$ stands for the floor operator which rounds to the highest integer value not larger than x. The three-dimensional vector can be wrapped into a one-dimensional index which directly corresponds to voxel indices. The mapping $\iota(\vec{t}_{grid}) \rightarrow i$ is expressed as follows

$$i = x + y \times d_z + z \times d_y \times d_z \tag{2.3}$$

where the three following conditions must hold for the mapping to be unique:

$$x < d_x \qquad y < d_y \qquad z < d_z \tag{2.4}$$

2.3 Dynamic map expansion

A naive method to build a reachability map will start by computing the workspace size using the TCP samples generated. This means that the size of the map is iteratively adjusted at the beginning of the sampling process. Before we can add a sample to the map, we need to check whether it lays within the mapped range. Two conditions must hold:

$$t_{TCP-x} \geq m_x,\ t_{TCP-y} \geq m_y,\ t_{TCP-z} \geq m_z \tag{2.5}$$

$$t_{grid_x} < d_x,\ t_{grid_y} < d_y,\ t_{grid_z} < d_z \tag{2.6}$$

If these constrains are met, we can proceed by setting an appropriate position for the sample in the map. If one of the conditions fails we have to enlarge the boundaries of the map before proceeding. Enlarging the boundary values leads to changes in voxel index i, as can be seen from Eq. (2.3). We need to reallocate the memory to maintain continuity and consistency with the indexing mechanism. The memory requirements can easily grow close to the limit of one process or computer memory, therefore it is important that the reallocation happens without large memory overhead. It is not viable to create two instances, the old and the new map at the same time. The new map needs to be allocated while the old structure is being deallocated. To be able to reallocate the old map on the fly, we have to obtain the indexing parameters of the new map. New map dimensions are determined as a difference of the new sample coordinates and current mapping limits. The difference of parameters also determines the re-mapping criteria. Previously allocated voxels do not occupy a continuous range in the new coordinate system. The reallocation process works in three steps:

1. Allocate new memory block
2. If applies, copy valid data from previous map, otherwise, initialize the values to zero
3. If applies, deallocate the block from previous map, otherwise, skip this step.

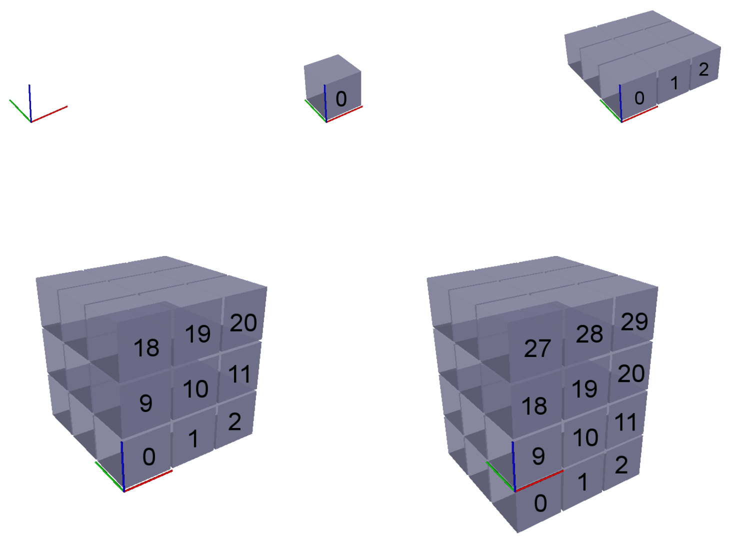


Figure 2.2: Example of index change during map expansion

Fig. 2.2 shows how the index of a voxel changes with map expansion. A voxel initially placed at the origin has an index of zero. If the map expands in any positive direction, the index is still valid. In cases when the map is expanded in any negative direction, new voxels are placed in the memory ahead of the original ones, and the index changes. For instance, for Fig. 2.2 the voxel with index $i = 0$ is remapped to $i = 9$, $1 \to 10$, $2 \to 11$ and so on.

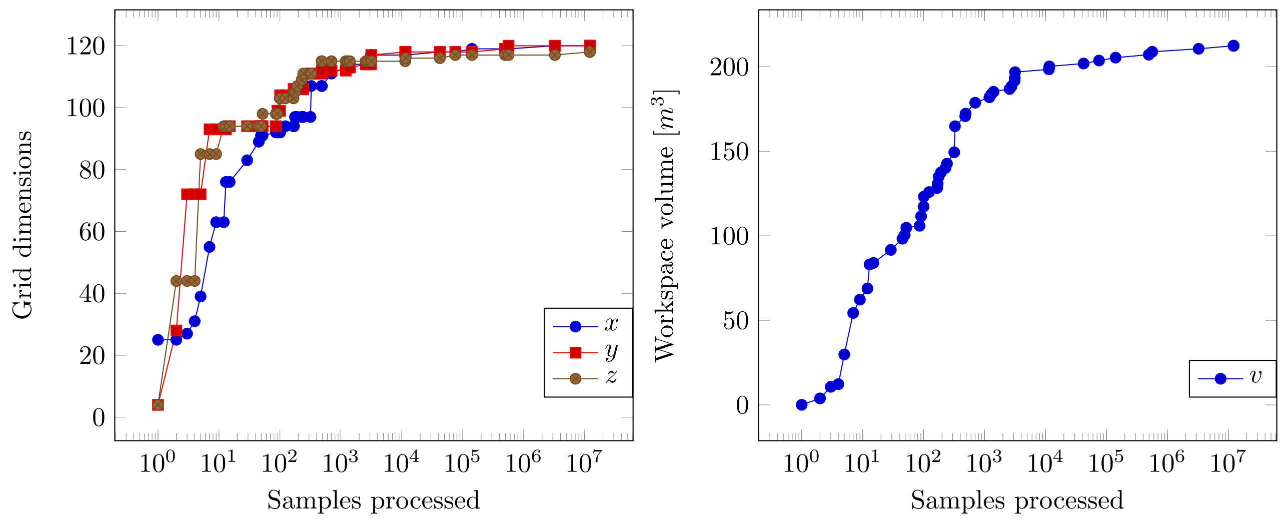


Figure 2.3: Grid size expansion at the beginning of generation and corresponding workspace volume ($v_s = 0.05m$)

Fig. 2.3 shows an example of how the map grows with the number of samples, both in dimensions and in volume. Note that this step does not take more than a few seconds. At high voxel resolutions the expansion can occasionally happen later during the process, however, the structure is built to handle even such cases.

2.4 $SO(2)$ mapping and indexing

Once we have have obtained a valid voxel index i we can proceed with mapping the remaining three components that represent the orientation of TCP. This step is divided into two parts. First part consists of dividing the $SO(2)$ space into the user defined number of bins. It is important that these bins occupy equal areas - $SO(2)$ approach directions [20]. The relative ratio of these areas determines a directional preference of the mapping. The $SO(2)$ grid can be visualized as a spherical surface divided by vertical and horizontal lines, parallel to longitude and latitude lines. An example of such division can be seen in Fig. 2.1. There are two control parameters so_2 and n_l. The spherical surface gets divided at equator, and the equatorial plane is parallel to the X-Y plane of the global reference frame. The hemispheric division is mirrored through equatorial plane. A hemisphere is divided into n_l regions of equal height, and the division planes are parallel to the equatorial plane.

We have to determine the number of bins in each layer. The user specified binning capacity parameter so_2 has to be distributed among the layers to maintain a reasonable area ratio between the bins. We define a distribution parameter M_r that determines the bin count ratio between layers.

$$M_r = \sum_{i=1}^{n_l} \sin\left(i \times \frac{\pi/2}{n_l}\right) \tag{2.7}$$

Using the M_r parameter we now create an allocation table and calculate the number of bins (layer bin capacity $c_i, i \in \mathbb{N}$) inside each layer as follows

$$c_i = \left(\frac{so_2}{M_r}\right) \times \sin\left((i+1) \times \frac{\pi/2}{n_l}\right) \tag{2.8}$$

Layer indexing starts from zero, where the zero layer is always the one covering the sphere's caps. The index increases in direction towards the equatorial plane. Each layer is split in the X-Y plane into c_i equally sized areas. Figure 2.4 shows examples of the spherical surface being split into 50 areas in 4 layers, 100 areas in 5 layers and 200 areas in 5 layers, respectively. These bins represent the discretization of $SO(2)$ (approach direction - pan and tilt of the end effector). Fast access to the layer binning is achieved using a lookup table that will be described in section 2.6.

The last missing coordinate is the roll angle of the tool. In some cases, when the robot is built with full roll capability $(-\pi, \pi)$, we do not need to consider roll angle for mapping since all rotations are reachable. In such cases it is sufficient to have a five-dimensional division grid to fully describe the robot capabilities. In case of limited tool's roll rotation we have to impose one more division. Each bin of the $SO(2)$ grid is split by its roll angle into n_r equally sized approach areas.

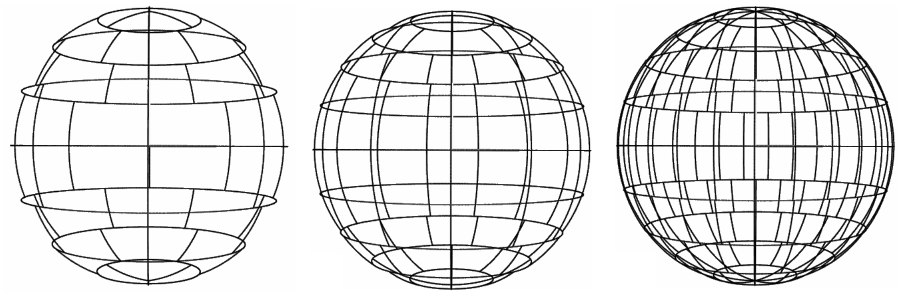


Figure 2.4: Examples of division areas on the spherical surface

2.5 Mapping of TCP orientation into its bin

We now describe the mapping from a homogeneous matrix describing the TCP sample to the bin position in our 6D grid. We use a general 4×4 matrix definition,

$$TCP(t, R) = \begin{bmatrix} R_{1,1} & R_{1,2} & R_{1,3} & t_x \\ R_{2,1} & R_{2,2} & R_{2,3} & t_y \\ R_{3,1} & R_{3,2} & R_{3,3} & t_x \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.9}$$

Cartesian coordinates t_{TCP} are directly readable from the matrix in Eq. (2.9). The sample's location in the Cartesian grid is then determined by Eq. (2.2). The remaining binning within the voxel is determined by examining the rotational matrix. Three parameters α, β, γ are extracted. Parameter α corresponds to tilt angle, β to pan angle and γ to roll angle.

$$\alpha = \text{atan2}(R_{3,3}, \sqrt{R_{1,3}^2 + R_{2,3}^2}) \tag{2.10}$$

$$\beta = \text{atan2}(R_{1,3}, \sqrt{R_{2,3}^2}) \tag{2.11}$$

In order to obtain the tool roll angle we have to transform the TCP sample in such way that the z axis will be aligned with the main coordinate frame. After that we can express the projection in X-Y plane and obtain the roll angle. Let us have $\vec{v_z}$ as vector of z axis of the TCP and $\vec{z_0} = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}^T$ as the z vector of the world frame. First we calculate the angle γ' between $\vec{v_z}$ and $\vec{z_0}$

$$\gamma' = \arccos(\vec{v_z} \cdot \vec{z_0}) \tag{2.12}$$

The axis of rotation is perpendicular to both vectors and can be expressed as

$$\vec{a} = |\vec{z_0} \times \vec{v_z}| \tag{2.13}$$

where $|\vec{v}|$ denotes a vector of direction $\vec{v}$ and length 1. The transformation matrix we are looking for is defined by the angle axis representation as $R(\gamma', \vec{a})$. Applying the transformation, we obtain

$$TCP_{\text{aligned}} = R(\gamma', \vec{a})^{-1} TCP(t, R) \tag{2.14}$$

Finally the tool roll angle can be expressed as

$$\gamma = \text{atan2}(TCP_{\text{aligned}}(R_{2,1}), TCP_{\text{aligned}}(R_{1,1})) \tag{2.15}$$

These angles need to be aligned with the reference coordinate frame first. The tool roll angle is aligned onto a $[0, 2\pi]$ range. For angles $\gamma < 0$ we perform the alignment as follows

$$\gamma := \gamma + 2 \cdot \pi \tag{2.16}$$

Where := is an assignment operator. The hemispheric assignment h_s depends on the angle α. For angles $\alpha < 0$, we mark the location as southern hemisphere, $h_s = 1$, and remap the values as

$$\alpha := \alpha + \frac{\pi}{2} \tag{2.17}$$

In case $\alpha > 0$, we mark the location on the northern hemisphere, $h_s = 0$, and remap the value as

$$\alpha := \frac{\pi}{2} - \alpha \tag{2.18}$$

The parameter α also determines the layer that the sample belongs to:

$$\text{layer} = \alpha \times \frac{1}{\frac{\pi}{2 \times n_l}} \tag{2.19}$$

The angle β is aligned as

$$\beta := \frac{3 \cdot \pi}{4} + \beta \tag{2.20}$$

unless the $R_{2,3}$ (the y component on Z vector) is negative. In such case, the alignment is performed as

$$\beta := \frac{\pi}{2} + |\beta| \tag{2.21}$$

for $\beta < 0$ and

$$\beta := \frac{\pi}{2} - \beta \tag{2.22}$$

for $\beta > 0$.

A particular approach direction for the selected layer is calculated as

$$\text{bin} = \beta \times \frac{1}{\frac{2 \times \pi}{c_{layer}}} \tag{2.23}$$

Finally, the bit memory offset is calculated with the help of the look up table generated in Eq. (2.28).

$$\text{bit} = (\delta_{layer-1} + \text{bin}) \times n_r + \left(\gamma \times \frac{1}{\frac{2\times\pi}{n_r}}\right) \tag{2.24}$$

The voxel memory is arranged symmetrically such that northern hemisphere is mirrored by the equatorial plane. In case we are accessing bins in the southern hemisphere (if $h_s = 1$) we only need to offset the calculated bit position by a constant

$$\text{offset} = \sum_{i=0}^{n_l} \left(\frac{so_2}{M_r}\right) \times \sin\left((i+1) \times \frac{\pi/2}{n_l}\right) \tag{2.25}$$

Now we have all the information necessary to complete the $TCP \rightarrow SE(3)$ mapping.

2.6 Underlying memory structure

Given the discretization parameters and robot workspace volume, we can calculate how much memory we need to store all the information. If we want to achieve a high precision of the map we have to increase the resolution of the grids, which in turns increases the memory footprint. Our representation is binary by nature. We cannot afford to use more than one bit for one bin in the map, nor can we use a common templated data structure which eases the data handling but for a price of additional memory overhead. In case of handling larger or higher resolution maps we are also constrained by the largest continuous memory block we can allocate. Therefore, we have to use a two-dimensional memory structure. We split the map memory into blocks, where each block is allocated individually. Smaller block allocation does not suffer from the effect of memory fragmentation, and it is still fast to access. The total memory requirement in bits M_b for a map can be calculated as

$$M_b = 2 \times d_x \times d_y \times d_z \times so_2 \times n_r \tag{2.26}$$

Equation (2.26) shows that the memory requirement grows with the third power of voxel resolution $\frac{1}{v_s}$. It is easy to reach the target device memory limit by such large memory footprint. There might not be a region in the memory that would allow us to allocate all data at once. To combat this factor, our underlying data structure is a two-dimensional array. The block size is controlled by a fragmentation parameter B_s that expresses the size of one memory block in number of elements (voxels in our case). Our voxel structure is then easily mapped into these blocks. The block can be, while remapping, allocated and deallocated in a serial manner, without creating a significant overhead to the overall process. The mapping $\nu(i \subset \mathbb{R}^1, B_s) \rightarrow m_x, m_y \subset \mathbb{R}^2$ is performed as follows

$$\begin{aligned} x &= \left\lfloor \frac{i}{B_s} \right\rfloor \\ y &= (i \mod B_s) \times 2 \times so_2 \times n_r \end{aligned} \tag{2.27}$$

where mod stands for modulus operator. After being able to address the desired voxel's memory location we proceed with addressing the individual bits representing a particular bin. This is done using a look-up table. Instead of traversing or recalculating the individual bin mapping we store the offset δ_i of layer i. The table is created during the initial grid creation process as

$$\delta_i = \sum_{l=0}^{i} \left(\frac{so_2}{M_r}\right) \times \sin\left(l \times \frac{\pi/2}{n_l}\right) \tag{2.28}$$

2.7 Computation of the map

There are three methods for generating reachability maps. The original method used by [6] uses an inverse kinematics (IK) solver. Because of the computational complexity of IK solvers, forward kinematics (FK) seems to be a more feasible way to fill in the map, although it might not explore all the workspace. We propose a third method, Hybrid method (HK), which combines the speed of generation of forward kinematics and guarantees the complete workspace exploration as the inverse kinematics.

These methods do not produce exactly identical results, therefore, it is important to understand the differences. For instance, the inverse kinematics solver might be unable to find a solution in some cases (even if it exists), but there is no ground truth to compare the results to. The comparison of results produced by the methods and the quality assesment procedure will be presented in chapter 3.

2.7.1 Inverse kinematics method

The inverse kinematics method is based on a single vector representation of each bin. As described in section 2.5, each bin represents a small range of TCP poses. We choose one pose (the middle point) within this range to represent the entire bin. The translational component is chosen as follows.

$$\begin{aligned} t_x &= m_x + x \times v_s + \frac{v_s}{2} \\ t_y &= m_y + y \times v_s + \frac{v_s}{2} \\ t_z &= m_z + z \times v_s + \frac{v_s}{2} \end{aligned} \tag{2.29}$$

The rotational component is also chosen as the mid point on the spherical and roll angle grids:

$$\alpha_{\text{layer}} = \left(\frac{\pi/2}{n_l} \times \text{layer} + \frac{\pi/2}{n_l} \times (\text{layer}+1)\right)/2 \tag{2.30}$$

$$\beta_{\text{bin}} = \left(\frac{2\times\pi}{c_{\text{bin}}} \times \text{bin} + \frac{2\times\pi}{c_{\text{bin}+1}} \times (\text{bin}+1)\right)/2 \tag{2.31}$$

$$\gamma_{\text{roll bin}} = \left(\frac{2\times\pi}{n_r} \times \text{roll bin} + \frac{2\times\pi}{n_r+1} \times (\text{roll bin}+1)\right)/2 \tag{2.32}$$

Once the parameters α, β, γ have been determined, we can form the rotational matrix R. These parameters correspond to tilt, pan, roll angles respectively, therefore we can write

$$R_{TCP} = R_{roll} \times R_{tilt} \times R_{pan} \tag{2.33}$$

Now we can form a homogeneous transformation matrix T_q for each bin. These configurations are generated and stored at the map initialization time to reduce the computational costs.

The generation process, however, is first started using a random forward kinematics generator described in detail in section 2.7.2. The purpose of this step is to determine the workspace dimensions. At the beginning, these random samples force our memory structure to expand. The memory expansion does not happen with every new sample, rather, the frequency of boundary expansion diminishes with every sample, eventually the expansion stops once the whole workspace is inscribed inside the grid's bounding box. This can be seen in Fig. 2.2.

Once the boundary expansion has stopped we can proceed with actual map generation. Knowing the workspace boundaries saves time, because we don't need to explore the space outside. We start by traversing all voxels and all poses within them. Each TCP pose is passed to the inverse kinematics solver to determine whether a solution exists or not. If at least one solution exists (assuming a redundant manipulator) we mark the bin as occupied (we set the bit from 0 to 1). After we traversed all voxels V_i in the map we can state that we explored the whole workspace and the map is complete. Depending on which type of solver we have used, the performance might change. In particular, some solvers can have longer response time while querying a pose that is not reachable. In general, the performance is stable over time and the map generation time t_g can be estimated using t_q as average time per inverse kinematics query

$$t_g = n \times so_2 \times n_r \times t_q \tag{2.34}$$

In this work, we use the IK solver proposed in [15], developed at DLR and commonly used to solve the IK for the Light Weight Robot arm (LWR) arm of KUKA Robotics.

2.7.2 Forward kinematics method

Since we already use this method to obtain the workspace boundaries, we can also use it as a generation method. A random TCP pose is obtained by randomly sampling the joint space of the robot. This suggests that another sampling parameter has to be specified. If we were to cover all possible combinations of sampled joint values we would have to process an unrealistic volume of data. In the case of a 7-joint robot with an average joint angle range of 300 degrees, sampled at 1 degree resolution we would get

$$\left(\frac{300}{1}\right)^7 = 2.187 \times 10^{17} \text{ configurations} \tag{2.35}$$

Assuming we can generate 500 samples per millisecond, it would still take us 1.21×10^8 hours to process all the samples. Therefore, we only take a subset of all possible combinations using random sampling. After choosing a random value for each joint we create a series of homogeneous transformation matrices representing joints and links of the robot. Multiplying them we obtain a random TCP sample to be directly mapped into our structure. The computational complexity is much lower and the process is also easily vectorisable. In comparison, our IK solver was able to verify 10 poses per millisecond for the LWR; a forward kinematics approach would generate around 500 poses per millisecond. The main disadvantage is that we cannot guide the workspace exploration process, therefore, we cannot guarantee the thorough exploration in a finite time.

Unlike inverse kinematics, which tests one sample per one bin, forward kinematics can generate two samples close to each other such that the second sample does not change the information already stored in the map, and therefore the time spent on its generation is useless. We propose a stopping criteria to depict the time when new sample generation is not effective any more. When this criteria is fulfilled we stop the generation and the map is ready to be used. When the percentage of newly set bins drops below a percent δ of last million samples generated, the generation is stopped. Typically we would choose δ to be 1% or 0.1% for FK generated maps.

In other words, the generation process randomly pulls samples from the total number of possible configurations (Eq. (2.35)). The stopping criteria δ tells us that we have pulled enough representative samples out of the configuration sample space to represent the given workspace at the given quality.

Note that a uniform random distribution in configuration space does not lead to a uniform distribution in the workspace. Samples accumulate in regions close to singular positions. In other words, more samples end up in areas where larger joint velocities result in smaller changes in TCP poses. This is demonstrated in figure 2.5.

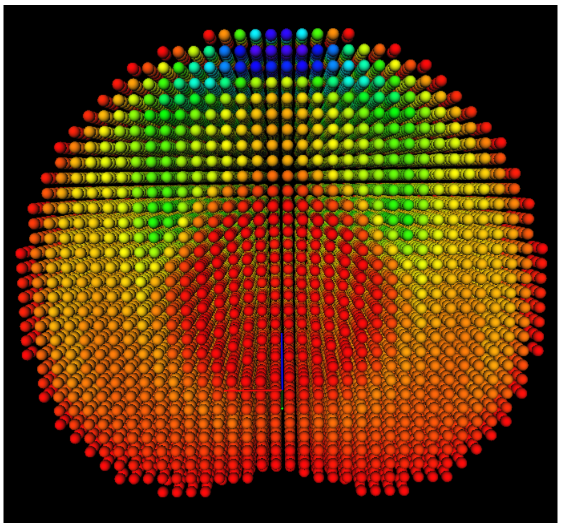


Figure 2.5: Sample accumulation in workspace after 1 billion samples. The color indicates the number of samples in particular voxel, blue indicates the highest and red the lowest number of samples.

We can plot a relationship between the number of randomly generated samples and a number of samples that resulted in a change in the map (new occupancies), as shown in Fig. 2.6. This figure and the following ones in this chapter are produced for the generation of the capability map for the LWR. The longer we run the generation, the less samples are relevant for the reachability map creation.

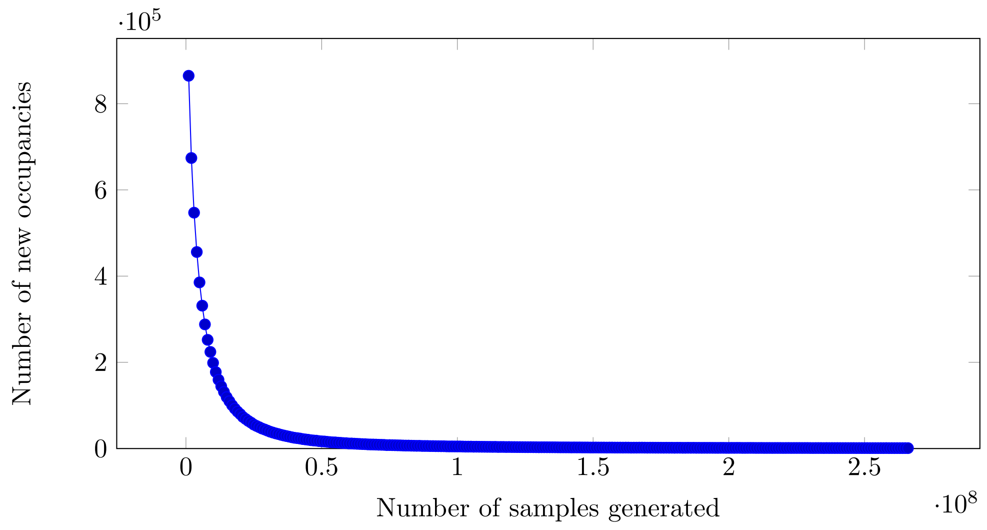


Figure 2.6: Newly marked bins with respect to total number of samples

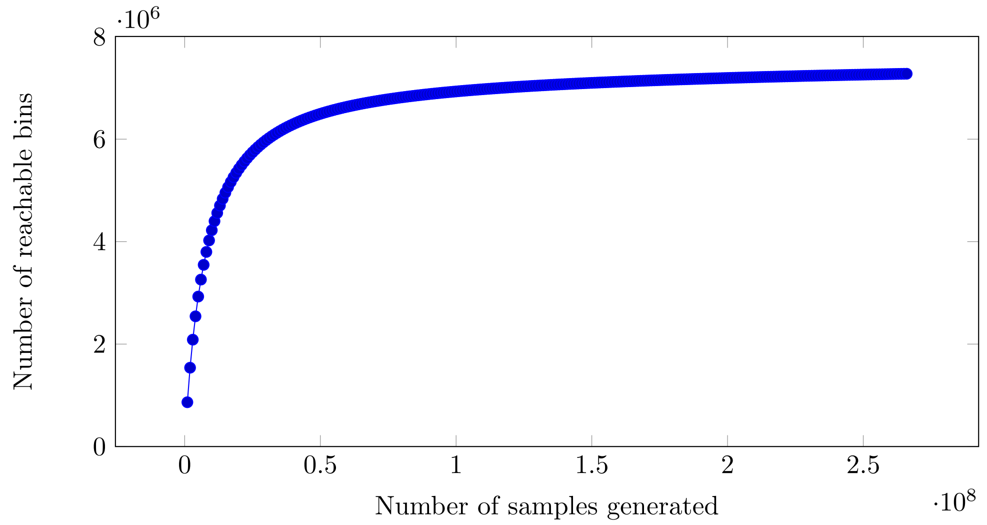


Figure 2.7: Number of reachable bins in the map with respect to number of samples generated

Figure 2.6 shows how the performance of the method diminishes with the number of generated samples. Over time, the performance of FK method drops below the performance of IK method and it still does not guarantee the map's completeness. This effect gave rise to the hybrid approach described in section 2.7.3. There is one more important issue when it comes to randomly generated samples. Every random generator has a period, after which it starts repeating the sequence. Because we might be close to the limits due to the large number of samples required to generate the map, it is important that our generated sequence is out of phase with the generator's limits. For example, a common period of standard *rand*() function is 2^{32}, which allows us to generate $2^{32}/6 = 7.1583 \times 10^8$ unique configurations for a 6 DOF robot and $2^{32}/8 = 5.36 \times 10^8$ for 8 DOF robot. After this many samples, the solutions would start to repeat and would not lead to any new occupancies. For case of high resolution maps, this is a limit that should be taken into account.

2.7.3 Hybrid method

We propose a combined method to generate reachability maps in a fast manner that guarantees the whole workspace exploration. The hybrid method starts the generation with forward kinematics, which pushes the workspace boundary expansion as well as generates a significantly higher number of valid samples than inverse kinematics. We measure the generation performance by iteratively comparing the change in total number of occupied bins in comparison with the number of generated samples. As the performance degenerates over time, we can identify the point when switching to inverse kinematics would yield higher performance. After switching to inverse kinematics, we start querying bins as described in section 2.7.1. If a particular bin is already set by forward kinematics we can skip it, hence, we only query the bins which are still unclear. If a solution is found we set the bin to 1, otherwise, it is already set correctly to 0. Traversing a map in such manner significantly reduces the time needed to complete the information using inverse kinematics, guarantees a complete workspace exploration and significantly reduces the map generation time.

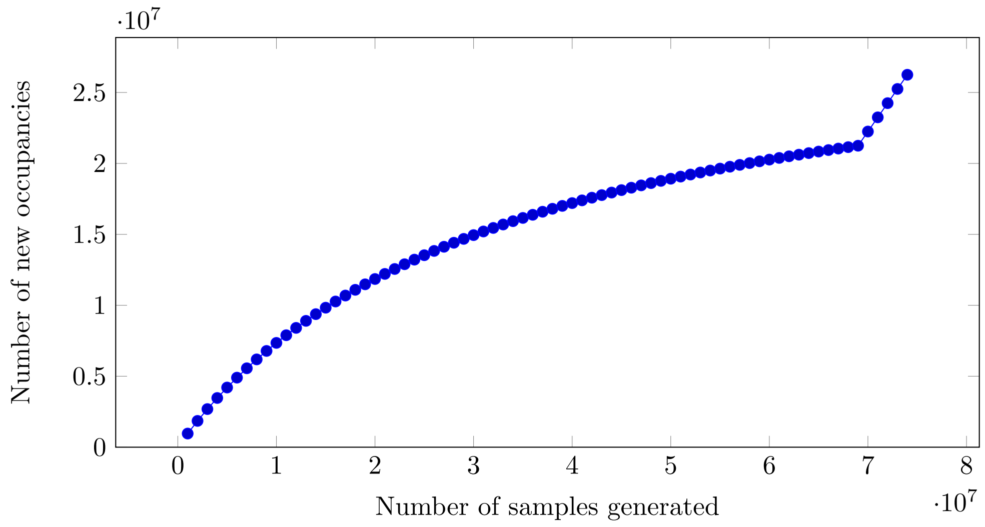


Figure 2.8: Newly marked bins with respect to total number of samples

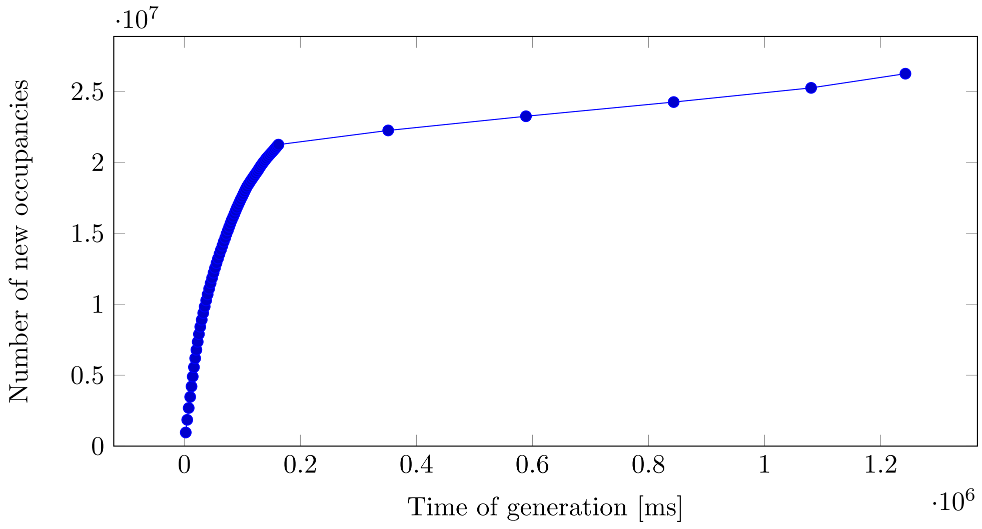


Figure 2.9: Newly marked bins with respect to generation time

Fig. 2.8 illustrates the generation process using the hybrid method; two clear segments are visible, corresponding to the FK and IK methods. We already mentioned that IK generation has approximately constant sample verification time. This can be seen in Fig. 2.9 as the linear segment of the plot. The generation is stopped when the complete workspace has been explored. If we were to only use the IK method we would see a plot only consisting of the linear segment. The FK part at the beginning rapidly rises the number of new occupancies and reaches a plateau (as in Fig.2.7). By switching the methods at the right time we can minimize the generation time by having a large number of samples generated at the beginning and only completing the workspace exploration with inverse kinematics. Fig. 2.10 compares the generation times of the individual methods, but for different voxel sizes. There are no options to change the inverse kinematics generation time, but the forward and the hybrid methods can be influenced by the stopping criteria δ. For purely FK generated maps we would normally choose $\delta = 1\%$ or less, and for hybrid generated maps we typically choose values around $\delta = 10\%$.

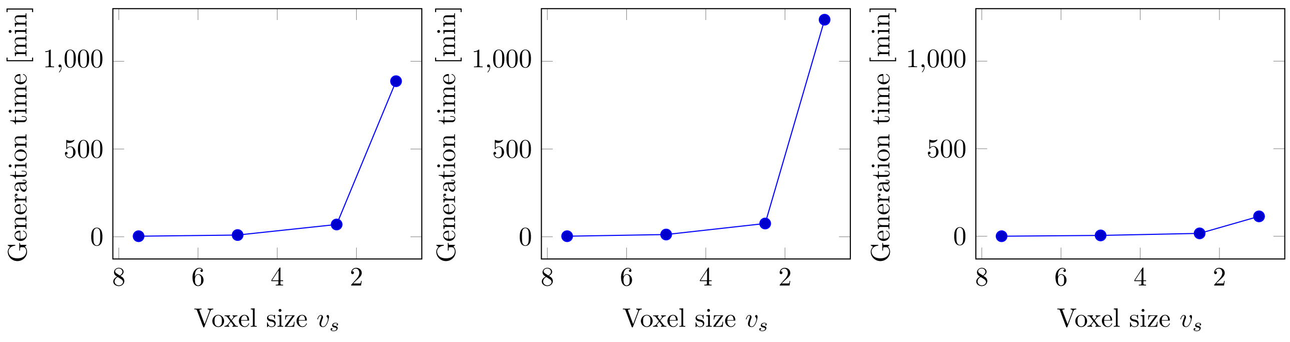


Figure 2.10: Generation times of FK ($\delta = 0.1\%$), IK, and HK ($\delta = 10\%$) with increasing voxel resolution

2.8 Collision detection and environment perception

No robotic system is complete without a proper collision detection subsystem. Whether we are creating an off-line analysis or implementing an on-line path planner, collisions (or self collisions) are always present and they significantly influence the number of reachable poses. In particular, the number of reachable poses considering collisions is a subset of a collision-free reachability map. It is a general rule that the collision detection system is computationally very expensive. In an actual application we are usually left with a trade off between precision and speed performance. Due to the large amount of data we need to process in our off-line generation phase, we need a solution which is as fast as possible. With this requirement in mind we have chosen a simple yet precise point to primitive collision detection mechanism. The advantage of such approach is its scalability in two manners,

1. We can tweak the performance by choosing the number of primitive shapes that approximate the robot. We can either create a low primitive-count model that creates a bounding volume around the robot, or we can have a higher precision high primitive-count model which would be slower to operate but would work better for off-line analysis applications.
2. The number of collision points can be reduced (downsampled) according to the needs of the application, which significantly speeds up the performance.

Another major advantage of this scheme is the ease of fusion with visual information. Nowadays, commonly used sensors for environment perceptions include Microsoft Kinect, Asus Xtion and other PrimeSens units, time of flight sensors or stereo camera systems. All of these sensors have a common form of output. They all function as 2.5 D systems, which means that their primary output is a depth map. Depth map is a two-dimensional array where each pixel value represents a distance measured from chip surface to object. Using the camera's projection parameters (given its field of view) we can transform the depth map from $\mathbb{R}^2$ to the Cartesian space $\mathbb{R}^3$. These parameters are offset u and pixel increment v. There are different values for the horizontal and vertical projection planes, therefore, there are 4 projection parameters. Let a single pixel value in a depth map be $z(x, y)$ where x and y are coordinates in a two-dimensional grid, then the projection $\mathbb{R}^2 \to \mathbb{R}^3$ is defined as follows,

$$\begin{aligned} p_x &= z(x,y) \cdot (u_h + x \cdot v_h) \\ p_y &= z(x,y) \cdot (u_v + y \cdot v_v) \\ p_z &= z(x,y) \end{aligned} \tag{2.36}$$

Our collision detection can be operated in two distinct modes. The focus while generating a reachability map in on precision, therefore, the algorithm operates on a full cloud quality during the off-line phase. When operating in on-line mode we can use several techniques to reduce the point count to achieve on-line performance.

2.8.1 Collision primitives

Two primitive types have been implemented so far. A simple and easy shape to implement is a sphere model. Any sphere can be defined by its center point $c_s = [x, y, z]$ and its radius r_s. To speed up the performance, we use r_s^2; in this way we avoid an unnecessary computation of the square root of the point-sphere distance. The collision is evaluated by

$$d_s^2 = (p_x - c_{s,x})^2 + (p_y - c_{s,y})^2 + (p_z.c_{s,z})^2 \tag{2.37}$$

where $p = [x, y, z]$ is the point in question. A collision condition is avoided when

$$d_s^2 > r_s^2 \tag{2.38}$$

otherwise we detect a collision. The second type of collision primitive used is a cylinder. Most of the robot links are easily represented by one cylinder element, which results in computationally faster evaluation than a set of spheres representing the same link [16]. The cylinder is defined by

- Center point of the base of the cylinder $c_c = [x, y, z]$
- Vector from the center point of the base to the center point of the opposite face $\vec{v}_c = [x, y, z]$
- Height of the cylinder squared l_c^2
- Radius of the cylinder squared r_c^2

The square distances are again used to improve the performance by avoiding unnecessary square root calculation. Letting p being our test point we first calculate the vector v_1 as

$$\begin{aligned} v_{1,x} &= p_x - c_{c,x} \\ v_{1,y} &= p_y - c_{c,y} \\ v_{1,z} &= p_z - c_{c,z} \end{aligned} \tag{2.39}$$

and we proceed by evaluating the dot product

$$d_p = \vec{v}_c \cdot \vec{v}_1 \tag{2.40}$$

If the test point lies below the base of the cylinder, the dot product d_p will be negative. On the other hand, if the point lies over the upper face of the cylinder, d_p will be larger than the cylinder's height squared. If the conditions in equation (2.41) hold then we do not need to proceed further because we are sure that the test point p is lying outside the cylinder caps, therefore, no collision with this point is possible. This mechanism is referred to as early outlier rejection.

$$\begin{aligned} d_p &< 0 \\ d_p &> l_c^2 \end{aligned} \tag{2.41}$$

If the conditions don't hold we have to check the distance of test point from the cylinder's main axis. The distance is calculated as

$$D = v_{1,x}^2 + v_{1,y}^2 + v_{1,z}^2 - \frac{d_p^2}{l_c^2} \tag{2.42}$$

If the following condition holds we do not detect a collision:

$$D > r_c^2 \tag{2.43}$$

2.8.2 On-line collision detection

We have already mentioned that the above described collision detection methods are suitable for on-line usage as well. First order of simplification is to remove points that are outside of the workspace boundaries. Two other strong performance improvements are possible:

1. Point reduction to grid resolution
2. Parallel GPU implementation

Point reduction is an effective method to choose the number of points such that we process those that are the most representative of the scene. The method is rough but very fast. We create a parallel structure to our grid. The structure maps each voxel to one binary value. If the value is not set, the bin is considered not occupied by the perceived scene. Vice versa, if at least one point of the sensor's point cloud falls within the voxel's volume, the representative bin is set to one. Mapping the point cloud data into the grid takes one traversal and it is essentially the same process as determining the voxel that a TCP belongs to. Equations (2.2) and (2.3) apply here as well to map the points in the point cloud to the voxel structure. We can process a full quality PrimeSens data stream on-line, which means depth map at 640×480 resolution at a rate of 30 frames per second. Naturally, a transformation has to be applied to the point cloud to align the sensor frame into the grid reference frame so that the equations can be applied. Once the collision structure has been filled, it can be represented in a point-wise manner. Each voxel is then represented by the voxel centroid obtained in equation (2.29), which is directly fed into the collision detection pipeline. Also, our path planning sub-system only plans within the voxels that are not occupied by the collision structure, therefore, this structure serves as an on-line filter mechanism for the path planner.

If necessary, this method of point to primitive collision detection can be highly parallelized on modern GPU hardware. Micro-kernels representing different shape primitives can then evaluate the collision on large number of points in one instant.

2.9 From reachability to capability map

Once the reachability map is built we can directly evaluate the existence of a TCP pose. The organized spatial structure of the map allows us to extend its use beyond the pure reachability information. The most basic extension of reachability map is a capability map. Capability map interprets the reachability map by defining a reachability index as follows

$$R_i = \frac{\sum_{a=1}^{so_2 \times n_r} V_i(a)}{so_2 \times n_r} \times 100 \tag{2.44}$$

where $V_i(a)$ refers to a particular bin within the voxel V_i. Reachability index defined in this way has a value in the range [0,100], 0 meaning that no poses are reachable inside the voxel and 100 meaning that all discretized directions are reachable. In other words, the higher the reachability index R_i is, the more dexterous the robot is in that region. Hence, it is a non-directional measure reflecting an overall dexterity within a small region of $\mathbb{R}^3$.

Such measure can be used to compare the quality of workspaces for a design or evaluation phase, as will be presented in Section 4.3. An off-line analysis of a capability map can identify good and bad regions for manipulation, or it can guide the manipulator design to maximize the high dexterity workspace volume. In other cases, this measure allows us to make on-line decisions such as choosing a better way of grasping objects for subsequent manipulation, as will be presented in Section 4.2.

2.9.1 Extended reachability index

We build on top of the reachability index concept and we propose its extension to be suitable for more applications. We define $co(i,j)$ as a function that evaluates the number of common directions between voxels V_i and V_j. With the central voxel labelled 0, the extended reachability index is defined as follows

$$R_i = \frac{\sum_{i=0}^{k} co(0,i)}{k \cdot co(0,0)} \tag{2.45}$$

Adjacent voxels are evaluated in several combinations according to fig. 2.11.

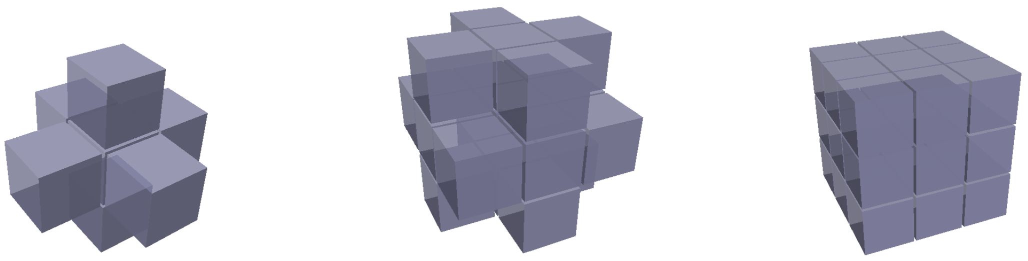


Figure 2.11: Various options to adjacency setting. From left to right: 6-, 18- and 26-adjacent voxels

The extended dexterity index is suitable for applications that need to examine the neighbourhood of a small area. Such methods include path planning or obstacle avoidance; extended reachability allows these methods to count with the conditions that are ahead of the path. A fast implementation is performed in the same way as described in section 2.9.2.

2.9.2 Fast implementation of Reachability index calculation

The memory holds the reachability map structure, which in general, holds more information than the Capability map, as the capability map only stores one index per voxel. Depending on a mode of operation, we have two main implementation methods for the Reachability index. If the application allows to drop the information contained in reachability map, we can create a separate structure holding the reachability index only. The indexing and voxelization structure stays the same, but now it contains a single floating point value per voxel. According to Eq. (2.26) we substitute the term $2 \times so_2 \times n_r$ which describes one voxel memory memory footprint by a size of one floating point value. The size can change according to the used platform, in our case, each float value is stored as 4 bytes. Therefore, the memory requirement to store a plain Capability map is

$$M_b = d_x \times d_y \times d_z \times 4 \times 8 \tag{2.46}$$

If necessary, this can be further reduced. If we allow the reachability index to only contain integer values we can store only one byte per voxel and the Capability map's memory footprint becomes

$$M_b = d_x \times d_y \times d_z \times 8 \tag{2.47}$$

The situation changes if we wish to be able to use the Capability and Reachability maps at the same time. Creating a separate instance might not be possible with the whole system design, or it might be too much load for one process. To solve this issue, we present a way to calculate the Capability map on the fly. Since the introduction of Intel's Nehalem microcode, every modern desktop CPU is equipped with a dedicated circuit called POPCNT. This instruction is then implemented in GCC as **__builtin_popcount**. In case the target platform supports POPCNT, a dedicated circuit is used. In case the target platform does not support POPCNT, a substitute code is compiled. The longest version available can calculate the number of bits set in an unsigned long long type and is implemented as **__builtin_popcountll** function. We only store the reachability map, yet we are able to get a quick response on the reachability index of a particular voxel by copying its bit content into unsigned long long blocks, and integrating their population count.

2.9.3 Visualization

Given its high dimensionality, it is hard to visualize all the information contained in the reachability map. On the other hand, capability map is a suitable visual representation of reachability maps as well. We are encoding four parameters, three of them are the Cartesian position of a voxel represented by its centroid. The remaining parameter is a color coded value of the index R_i. We code the index value using HSV encoding. The color scale is shown in Fig. 2.12.

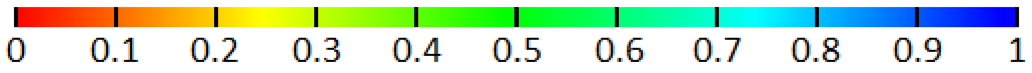


Figure 2.12: Color encoding of reachability index in visualizations

Rendering a map for fluent visualization is demanding on the graphics performance. To speed the rendering process and to ease the interpretation of the map we normally use a colored point representation. To simplify even more we also use a plane-sprite rendering, which is very effective but requires a moving scene to properly transmit the depth sensation to the user. For the purpose of paper presentation we render spheres instead of points.

A typical hands-on representation is pictured in Fig. 2.13 alongside with its equivalent representation using spheres.

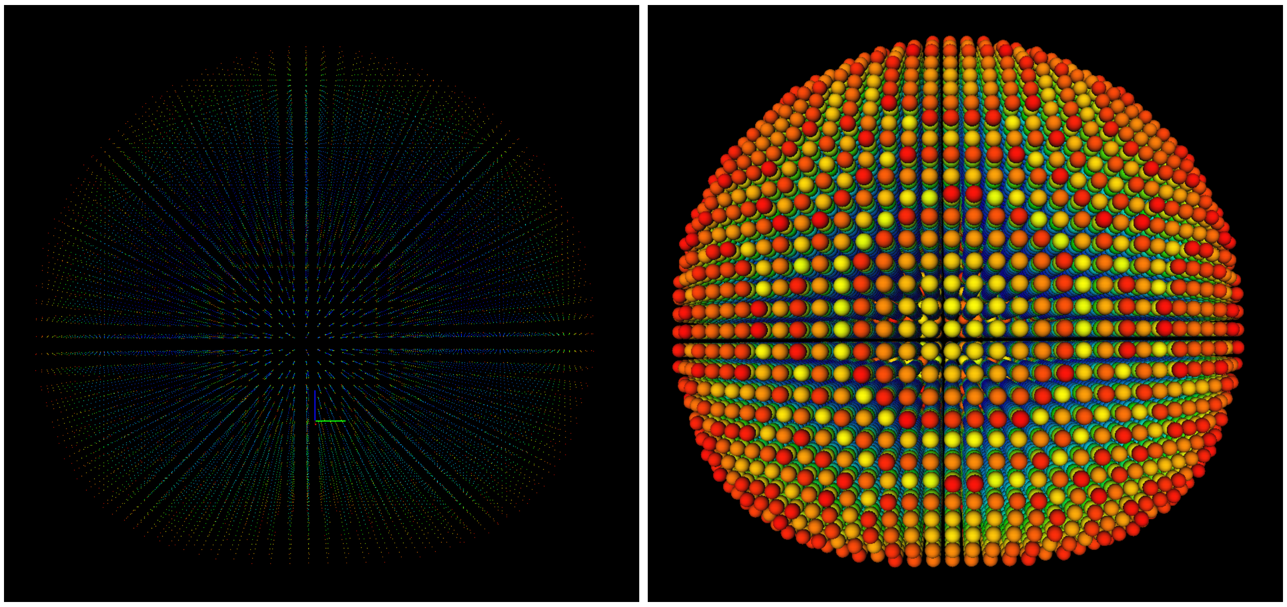


Figure 2.13: Point versus sphere representations of KUKA LWR workspace

Note that it is not possible to perceive the various layers contained in the sphere representation. To reveal the inner structure of workspace geometry and reachability, we would usually provide a cross section as in Fig. 2.14, which is more suitable for reader's comprehension.

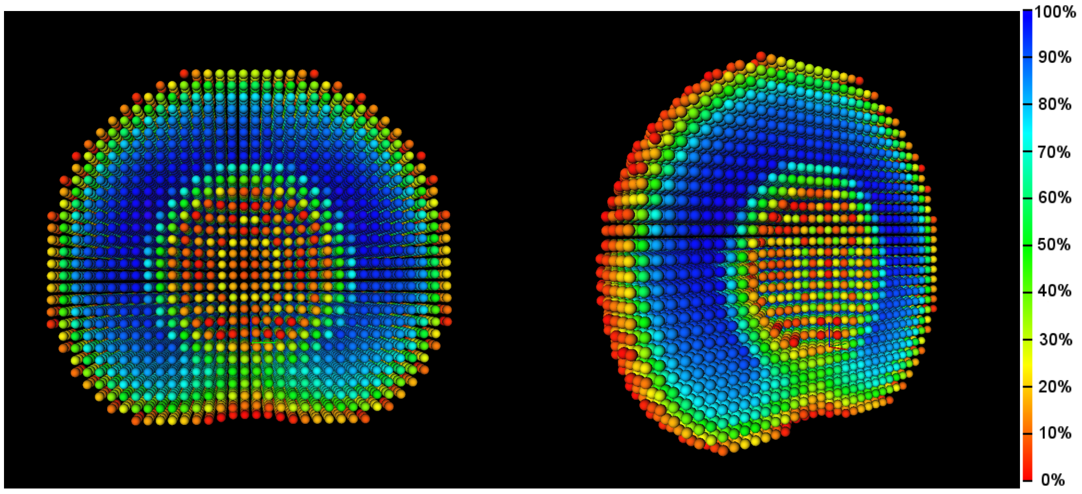


Figure 2.14: Cross sections of KUKA LWR workspace with color scale indicating the value of voxel's reachability index

3. Map quality assessment

Reachability maps use approximations that affect the quality of the reachability predictions obtained with them. With IK, a representative point of a bin (the middle point) is assumed to represent the reachability for all the points in the bin. For FK, even if one sample falls close to the boundary of a specific bin it assumes that all the points in the bin are reachable. Therefore, there is no ground truth for the maps we are creating. This chapter defines a methodology of quality assessment for the evaluation of the accuracy of the generated maps. The metric is derived from the primary purpose of use of the maps, i.e. analysis of reachability of TCP poses. We investigate the error present in the maps with respect to the discretization parameters and generation method used. This allows us to formulate several criteria for stopping the generation, switching to different method or choosing the right method for a particular application.

3.1 Prediction accuracy and error classification

As already mentioned, there is no ground truth which we can compare our results to. In order to create a consistent evaluation framework we have decided to evaluate the map in terms of its use. The main goal of the reachability map is to predict whether or not a given TCP pose is reachable. If it is reachable, it is still necessary to query the inverse kinematics to obtain the joint values. To evaluate the robustness of inverse kinematics, we first compare it to forward kinematics. Every random sample generated by forward kinematics is a valid and reachable sample. Inverse kinematics must therefore find a solution of joint values to the TCP pose. Such consistency should ideally be above 99%. Note that since inverse kinematics will be used for map quality assessment, our prediction accuracy cannot be higher than IK-FK consistency.

We generate an arbitrary large number of random positions and orientations within the bounding box of the workspace. It is important that the samples are located within the boundaries, because the prediction outside of workspace is always consistent with inverse kinematics - always unreachable by both methods, therefore they would increase the prediction accuracy. As the TCP samples are generated without a relation to the kinematic chain, we do not know if they are reachable or not. For each generated sample we:

1. Look for an IK solution for the TCP pose, and
2. Query the reachability of the pose using the map. To solve a query we need to assign the TCP to the 6D grid. We repeat the process from sections 2.2 and 2.5 to obtain a representative bin. Afterwards,

we calculate the appropriate memory indices as described in section 2.6 to obtain a binary value, 1 for reachable and 0 for unreachable pose.

Here we assume that if inverse kinematics cannot find a solution the pose is not reachable. Therefore, we take it as a ground truth and we can evaluate the prediction accuracy of the reachability map. For each generated sample we query the inverse kinematics and the reachability map simultaneously and compare the results. Treating the results as a classification problem we can split them into four categories according to Table 3.1.

	IK = 1	IK = 0
RM = 1	True positive	False positive
RM = 0	False negative	True negative

Table 3.1: Classification of results (RM - reachability map, IK - inverse kinematics)

Interpreting the table of classification we see that if inverse kinematics and reachability map agree about some sample's reachability we can have truly positive or truly negative classification. If our two classifiers don't agree we can have falsely positive or falsely negative classification. We would experience only true positive and negative samples in a map with 100% prediction accuracy. If the prediction accuracy p is less than 100%, we get $100\% - p(\%)$ samples that are in disagreement, falsely positive or negative. In consequence, we will asses the quality of reachability maps by investigating three parameters

1. (true positives + true negatives) out of all samples tested
2. false positives out of all samples tested
3. false negatives out of all samples tested

The first parameter tells us how good the map is if we want to use it for real applications; we will call it prediction accuracy from now on. Ideally it should be close to 100%. The remaining two parameters tell us more about the character of error that the map produces. We will call error structure to the pair false positives/false negatives. Ideally it should be 0%/0%. For certain applications we might wish to shift the type of error towards one of them to speed up the performance of other methods.

3.2 Prediction accuracy, generation time and error structure

The following data were generated for the KUKA Lightweight Robot pictured in Fig. 3.1. The technology behind LWR was developed at the laboratories of DLR and is now commercially available through KUKA.

Figure 3.1: KUKA Lightweight Robot - LWR (Courtesy of KUKA Robotics)

The following tables represent a matrix of combinations of different parameters used to generate the maps. The header of the table indicates the voxel size (v_s) in centimeters. The left column indicates the two parameters (so_2 - n_r). For future references, we label the maps in similar format v_s - so_2 - n_r to identify their resolutions from the label. Table 3.2 shows the memory required to store such maps. These values are independent of the method of generation used or of the prediction accuracy.

	7.5	5	2.5	1
50 - 10	0.69	2.3	18	258
100 - 20	2.9	9.7	73	1100
200 - 30	8.6	30	219	-

Table 3.2: Map memory requirements in MB

The following sections present data on prediction accuracy, generation times and error structure. The prediction accuracy test was carried out by generating 300 000 random poses and evaluating them using the reachability map and inverse kinematics. Generation times are recorded in minutes. Error structure shows what percentages belong to false positives and false negatives of the misclassified queries. Sections 3.2.1 to 3.2.3 present the raw data, and section 3.2.4 summarizes the results in a graphical way.

3.2.1 Forward kinematics

Tables 3.3 and 3.4 present the percentage of correctly identified reachability (true positives + true negatives) out of 300 000 cases, for different stopping criteria δ.

FK ($\delta = 1\%$)	7.5	5	2.5	1
50 - 10	89.552	91.548	91.506	93.111
100 - 20	90.477	92.515	93.653	94.330
200 - 30	90.886	92.94	94.22	-

FK ($\delta = 0.1\%$)	7.5	5	2.5	1
50 - 10	87.946	90.625	92.194	93.175
100 - 20	89.41	92.024	93.746	94.647
200 - 30	89.891	92.575	94.517	-

Table 3.3: Prediction accuracy of FK generated maps (in %)

FK ($\delta = 5\%$)	7.5	5	2.5	1
50 - 10	90.36	90.398	90.943	-
100 - 20	89.45	90.981	91.6	-
200 - 30	89.82	91.35	92.01	-

FK ($\delta = 10\%$)	7.5	5	2.5	1
50 - 10	88.092	89.097	89.407	-
100 - 20	88.198	89.324	89.594	-
200 - 30	8.252	89.311	89.661	-

Table 3.4: Prediction accuracy of FK generated maps (in %)

The generation process is evaluated iteratively. At each iteration we compare the number of samples generated per iteration and the number of samples that resulted in an actual change to the map. If the used samples drop below the δ percentage, the generation is stopped. To reveal the structure behind the error we evaluate the remaining two parameters - false positives / false negatives (presented in this order). They are shown in Tables 3.5 to 3.8.

FK ($\delta = 1\%$)	7.5	5	2.5	1
50 - 10	9.169 / 1.277	7.079 / 1.373	6.930 / 1.564	5.139 / 1.750
100 - 20	8.121 / 1.402	6.011 / 1.473	4.639 / 1.708	3.829 / 1.841
200 - 30	7.651 / 1.463	5.529 / 1.530	4.022 / 1.758	-

Table 3.5: Error structure of FK maps at $\delta = 1\%$ (in %/%)

FK ($\delta = 0.1\%$)	7.5	5	2.5	1
50 - 10	11.865 / 0.188	9.164 / 0.210	7.536 / 0.270	6.501 / 0.324
100 - 20	10.380 / 0.210	7.754 / 0.222	5.940 / 0.294	4.995 / 0.357
200 - 30	9.892 / 0.216	7.167 / 0.257	5.163 / 0.319	-

Table 3.6: Error structure of FK maps at $\delta = 0.1\%$ (in %/%)

FK ($\delta = 5\%$)	7.5	5	2.5	1
50 - 10	5.341 / 4.295	5.336 / 4.266	4.551 / 4.506	-
100 - 20	6.177 / 4.374	4.57 / 4.45	3.446 / 4.954	-
200 - 30	5.765 / 4.413	4.173 / 4.474	3.01 / 4.979	-

Table 3.7: Error structure of FK maps at $\delta = 5\%$ (in %/%)

FK ($\delta = 10\%$)	7.5	5	2.5	1
50 - 10	4.872 / 7.035	4.552 / 6.351	3.76 / 6.832	-
100 - 20	5.112 / 6.69	3.792 / 6.883	2.904 / 7.502	-
200 - 30	4.816 / 6.931	3.499 / 7.189	2.542 / 7.796	-

Table 3.8: Error structure of FK maps at $\delta = 10\%$ (in %/%)

These tables indicate that the method's error is mostly comprised of falsely identifying a pose to be reachable, rather than unreachable (there are, in general, more false positives than false negatives). This can be an important factor for certain applications. At last, we present the time needed to generate these maps in Tables 3.9 and 3.10.

FK ($\delta = 1\%$)	7.5	5	2.5	1
50 - 10	0.83	2.33	21.08	260.6
100 - 20	3.33	10.26	78.82	1093.6
200 - 30	9.21	28.91	217.6	-

FK ($\delta = 0.1\%$)	7.5	5	2.5	1
50 - 10	3.24	9.89	70.05	886.64
100 - 20	12.18	36.02	254.17	3477.4
200 - 30	34.37	99.91	700.58	-

Table 3.9: Generation time of FK method (in minutes)

FK ($\delta = 5\%$)	7.5	5	2.5	1
50 - 10	0.61	1.59	10.28	-
100 - 20	2.02	5.85	37.59	-
200 - 30	5.68	8.75	59.05	-

FK ($\delta = 10\%$)	7.5	5	2.5	1
50 - 10	0.3	0.9	6.9	-
100 - 20	1.13	3.43	20.62	-
200 - 30	2.97	8.75	59.05	-

Table 3.10: Generation time of FK method (in minutes)

3.2.2 Inverse kinematics

The same table structures are presented now for the inverse kinematics generated maps. There are no stopping criteria to tune, hence one table is sufficient to describe the performance. The structure of the error is presented in Table 3.12, and the generation times in minutes are presented in Table 3.13.

IK	7.5	5	2.5	1
50 - 10	94.845	95.830	96.123	96.201
100 - 20	95.474	96.497	97.072	97.335
200 - 30	95.659	97.542	97.488	-

Table 3.11: Prediction accuracy of IK generated maps (in %)

IK	7.5	5	2.5	1
50 - 10	2.581 / 2.573	2.092 / 2.078	1.909 / 1.908	1.915 / 1.883
100 - 20	2.227 / 2.299	1.752 / 1.750	1.458 / 1.469	1.324 / 1.340
200 - 30	2.166 / 2.173	1.229 / 1.229	1.281 / 1.230	-

Table 3.12: Error structure of IK generated maps (in %)

IK	7.5	5	2.5	1
50 - 10	3.1	12.66	75.52	1237
100 - 20	12.49	41.62	318.33	5179.7
200 - 30	37.9	125.43	969.38	-

Table 3.13: IK generation times (in minutes)

3.2.3 Hybrid method

Hybrid method combines the FK and IK methods, as described in section 2.7.3. Therefore, the results of the HK method also depend on the choice of the criterion δ used to switch from FK to IK.

HK ($\delta = 0,1\%$)	7.5	5	2.5	1
50 - 10	88.059	90.689	92.248	93.405
100 - 20	89.389	91.966	93.837	-
200 - 30	89.989	92.713	94.595	-

HK ($\delta = 1\%$)	7.5	5	2.5	1
50 - 10	91.935	92.278	93.049	94.188
100 - 20	90.612	93.401	94.175	-
200 - 30	91.035	93.911	-	-

Table 3.14: Prediction accuracy of HK generated maps for $\delta = 0.1\%$ and $\delta = 1\%$ (in %)

HK ($\delta = 5\%$)	7.5	5	2.5	1
50 - 10	91.935	93.299	93.163	-
100 - 20	92.501	94.253	93.968	-
200 - 30	92.997	94.772	-	-

HK ($\delta = 10\%$)	7.5	5	2.5	1
50 - 10	92.263	93.765	92.915	94.224
100 - 20	93.089	94.741	93.917	94.605
200 - 30	93.500	95.197	94.283	-

Table 3.15: Prediction accuracy of HK generated maps for $\delta = 5\%$ and $\delta = 10\%$ (in %)

The following tables describe the error structure of hybrid generated maps.

HK ($\delta = 0,1\%$)	7.5	5	2.5	1
50 - 10	11.753 / 0.187	9.189 / 0.122	7.512 / 0.239	6.429 / 0.165
100 - 20	10.415 / 0.196	7.886 / 0.147	4.698 / 1.127	-
200 - 30	9.800 / 0.210	7.166 / 0.121	5.072 / 0.332	-

Table 3.16: Error structure of HK maps at $\delta = 0.1\%$ (in %)

HK ($\delta = 1\%$)	7.5	5	2.5	1
50 - 10	6.950 / 1.115	7.269 / 0.453	5.986 / 0.964	5.366 / 0.445
100 - 20	8.212 / 1.175	6.222 / 0.377	4.698 / 1.127	-
200 - 30	7.791 / 1.174	5.728 / 0.360	4.082 / 0.332	-

Table 3.17: Error structure of HK maps at $\delta = 1\%$ (in %)

HK ($\delta = 5\%$)	7.5	5	2.5	1
50 - 10	7.521 / 0.852	5.899 / 0.802	4.714 / 2.123	-
100 - 20	6.699 / 0.801	5.084 / 0.663	3.718 / 2.314	-
200 - 30	6.281 / 0.722	4.648 / 0.579	3.292 / 2.191	-

Table 3.18: Error structure of HK maps at $\delta = 5\%$ (in %)

HK ($\delta = 10\%$)	7.5	5	2.5	1
50 - 10	6.582 / 1.154	5.245 / 0.990	4.218 / 2.866	2.859 / 2.917
100 - 20	5.886 / 1.025	4.413 / 0.845	3.247 / 2.836	2.810 / 2.585
200 - 30	5.525 / 0.975	4.063 / 0.741	2.843 / 2.874	-

Table 3.19: Error structure of HK maps at $\delta = 10\%$ (in %)

Finally, we present the generation time needed for each one of the maps.

HK ($\delta = 0,1\%$)	7.5	5	2.5	1
50 - 10	3.55	12.7	96.68	877.1
100 - 20	12.14	45.32	511.8	-
200 - 30	33.95	134.2	760.88	-

HK ($\delta = 1\%$)	7.5	5	2.5	1
50 - 10	0.89	3.25	59.23	488.9
100 - 20	3.21	13.46	152.37	-
200 - 30	18.06	71.37	392.39	-

Table 3.20: Generation times for HK maps at $\delta = 0.1\%$ and $\delta = 1\%$ (in minutes)

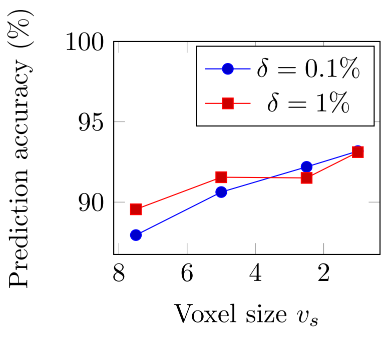
HK ($\delta = 5\%$)	7.5	5	2.5	1
50 - 10	0.37	14.8	25.997	-
100 - 20	5.1	18.39	135.63	-
200 - 30	14.79	55.22	314.67	-

HK ($\delta = 10\%$)	7.5	5	2.5	1
50 - 10	0.25	4.78	14.53	113.62
100 - 20	8.65	20.71	64.1	720.1
200 - 30	23.6	64.21	292.85	-

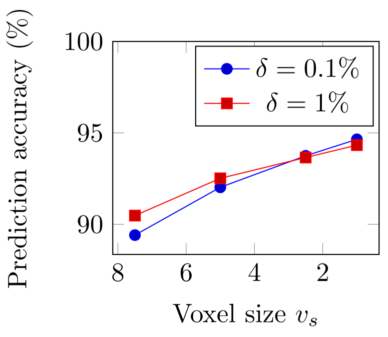
Table 3.21: Generation times for HK maps at $\delta = 5\%$ and $\delta = 10\%$ (in minutes)

3.2.4 Results comparison

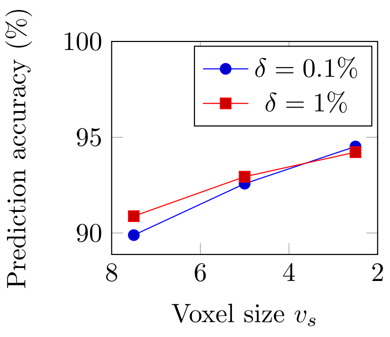
This section will describe the impact of individual parameters on the quality and generation time. When we intent to generate a map for one-time workspace analysis, we aim for high precision and we are not limited by the memory footprint. The main driving parameter behind the memory footprint is the voxel size v_s, as can be seen in Eq. (2.26). With increasing resolution in voxel space, we increase the memory footprint with third power. For each voxel resolution, there is an optimal $SO(3)$ discretization grid. For on-line applications we are constrained by the task given, and the voxel resolution is therefore always a compromise. On one hand, minimal memory footprint is desired while for example the overlaying capability map should still be able to distinguish between several poses for grasping applications. In this case, the voxel size needs to reflect the smallest distance between two different grasping poses for one object. For voxel guided path planning applications, we need to guarantee a minimum critical distance to achieve a safe interpolation between the samples. We tested four voxel resolutions to cover a full range of very coarse and very dense grids. For every voxel resolution we tested three different $SO(3)$ resolutions. Some of the combinations of high resolutions were not covered due to the memory limits.



(a) FK at 50-10

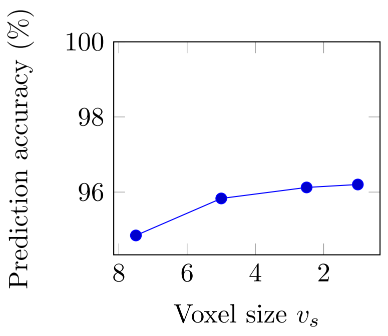


(b) FK at 100-20

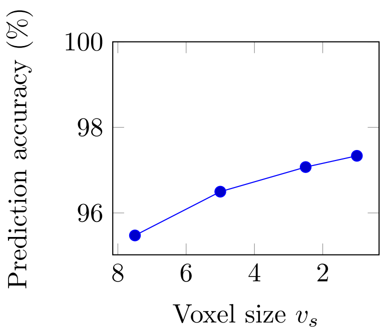


(c) FK at 200-30

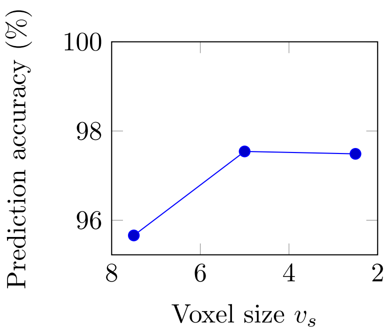
Figure 3.2: Prediction accuracy with respect to discretization parameters of FK generated maps



(a) IK at 50-10

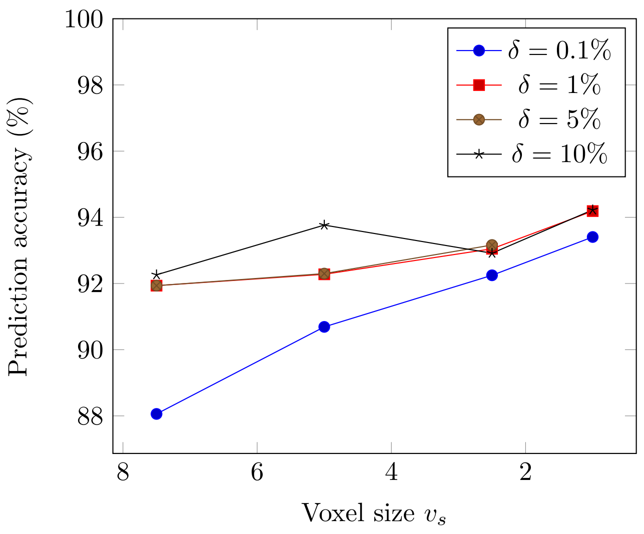


(b) IK at 100-20

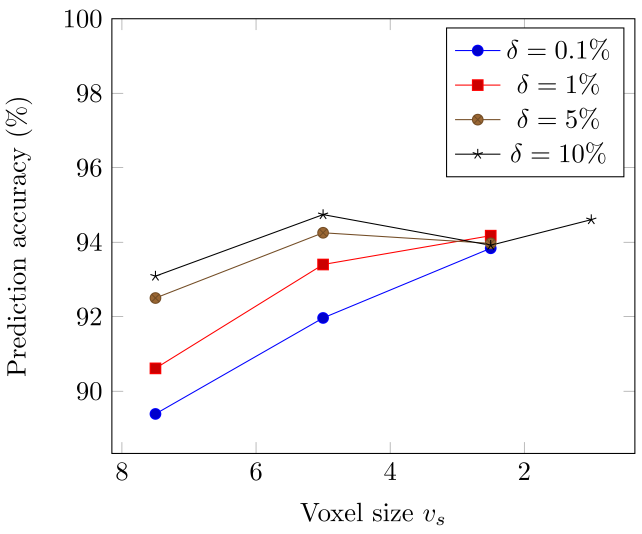


(c) FK at 200-30

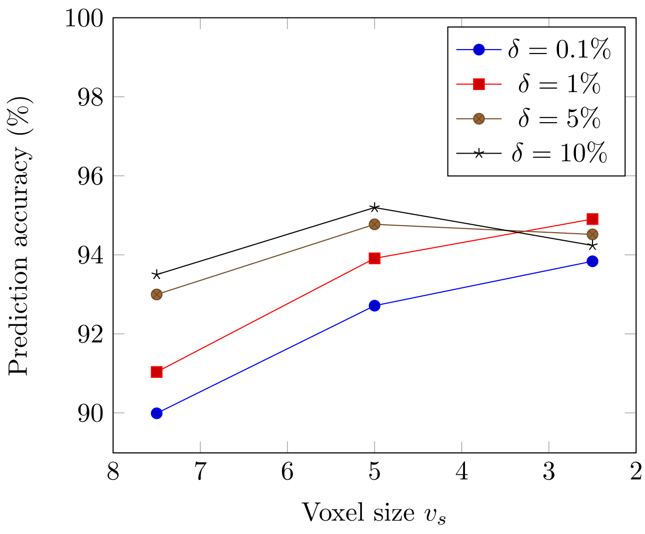
Figure 3.3: Prediction accuracy with respect to discretization parameters of IK generated maps



(a) HK at 50-10



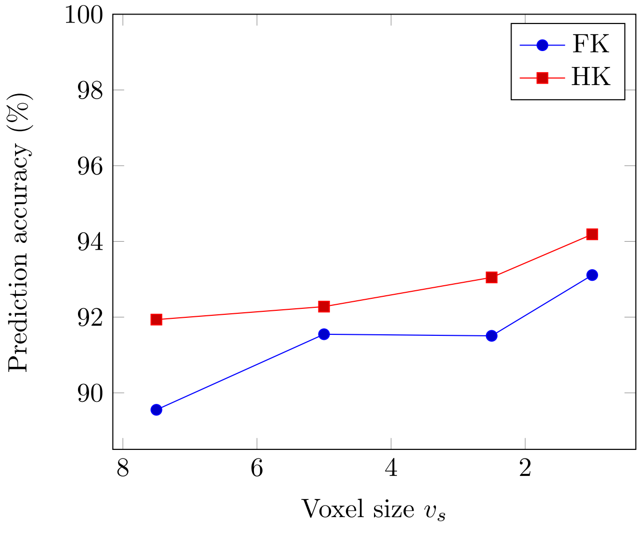
(b) HK at 100-20



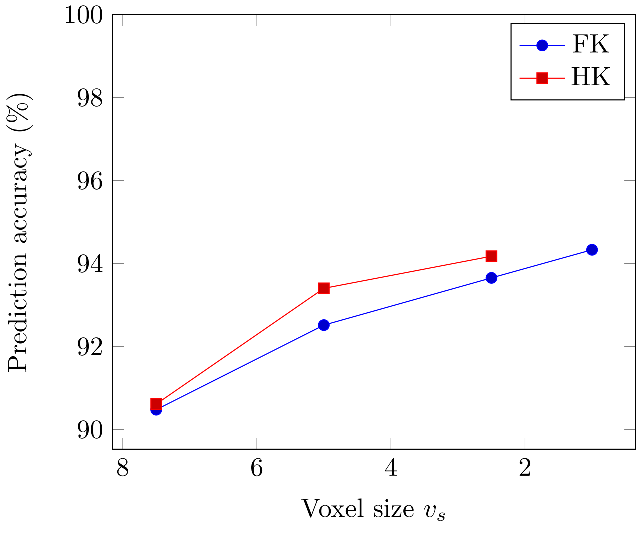
(c) HK at 200-30

Figure 3.4: Prediction accuracy with respect to discretization parameters of HK generated maps

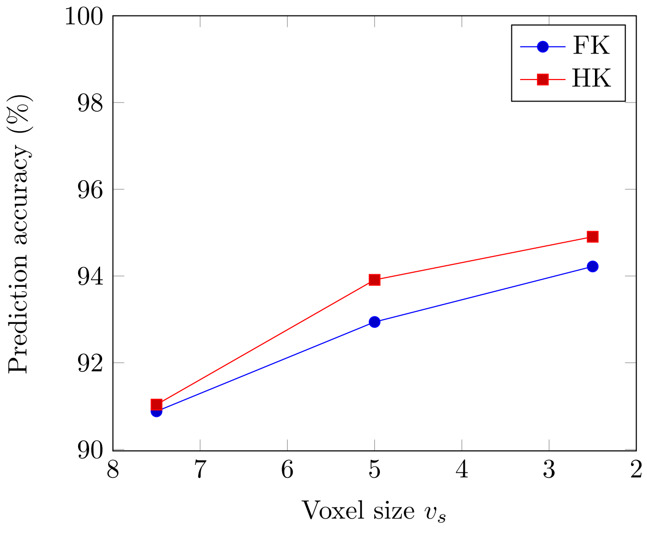
Now we put the two methods, forward and hybrid kinematics into perspective. We plot the prediction accuracy at equal resolutions and stopping criteria in Fig. 3.5. These plots were created for comparison only, to be able to asses the different behaviour of the generation methods. In real world application we would not use an FK generated map at the same stopping criteria as HK map, since the effect of fast generation would decrease.



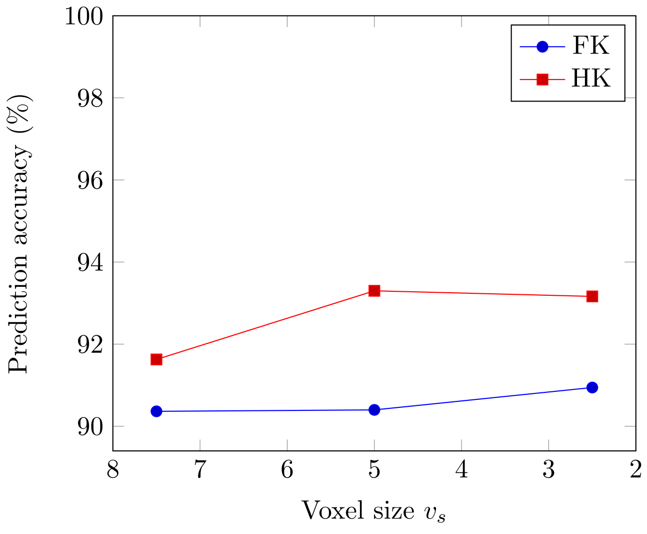
(a) HK vs FK at 50-10, $\delta = 1\%$



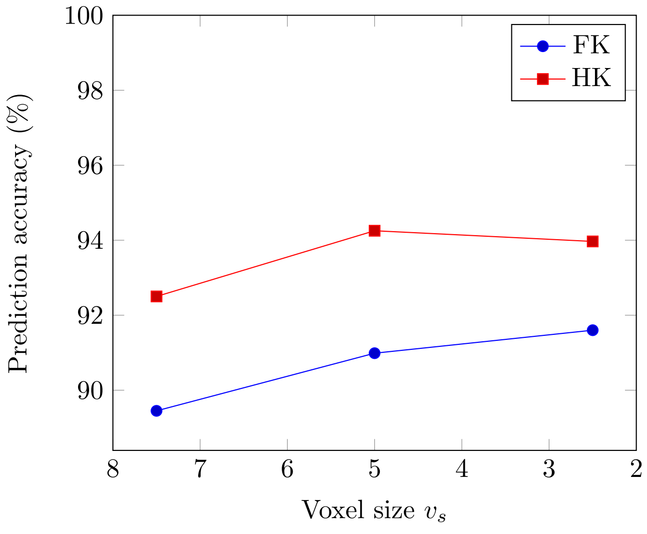
(b) HK vs FK at 100-20, $\delta = 1\%$



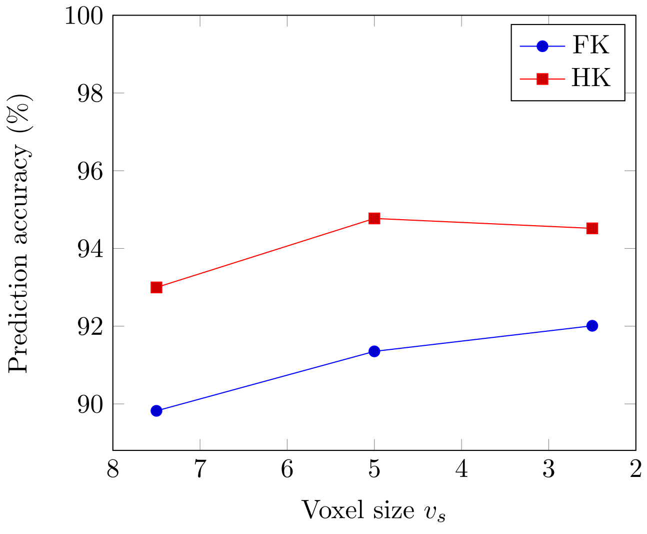
(c) HK vs FK at 200-30, $\delta = 1\%$



(d) HK vs FK at 50-10, $\delta = 5\%$



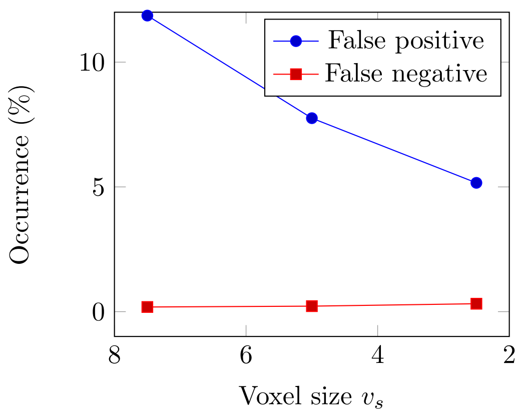
(e) HK vs FK at 100-20, $\delta = 5\%$



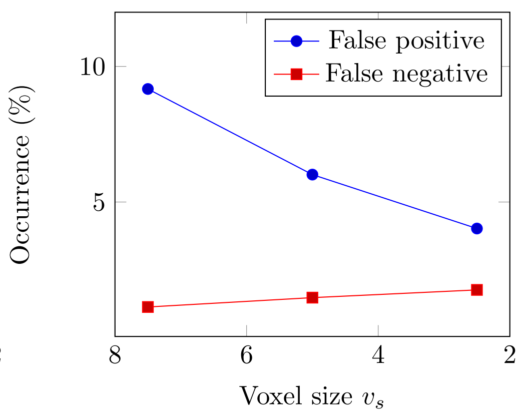
(f) HK vs FK at 200-30, $\delta = 5\%$

Figure 3.5: Comparison of prediction accuracy of HK and FK methods with equal stopping criteria and resolution

Plotting all the possible combinations of resolutions to illustrate the error characteristics is not necessary. We only plot three selected resolutions to show the structure, which is valid for all the other cases. Numerical results for complete reader's comparison are available in Tables 3.12 for IK maps, 3.5 and 3.6 for FK maps and 3.16, 3.17, 3.18 and 3.19 for HK maps. The resolutions we plot are on the diagonal of the tables, therefore we plot the values for 7.5-50-10, 5-100-20, 2.5-200-30 resolutions.

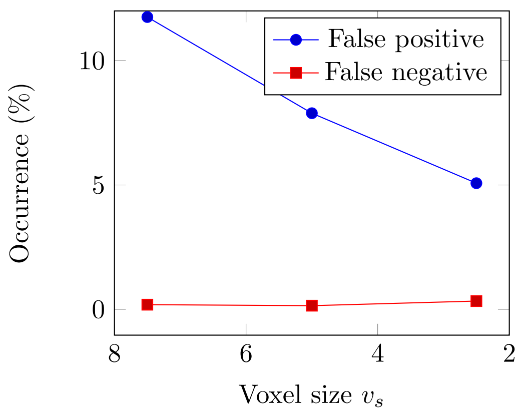


(a) FK with $\delta = 0.1\%$

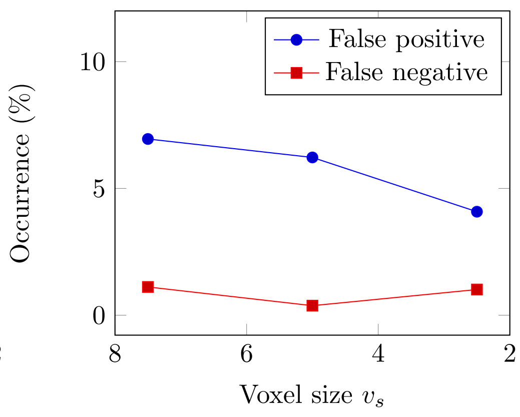


(b) FK with $\delta = 1\%$

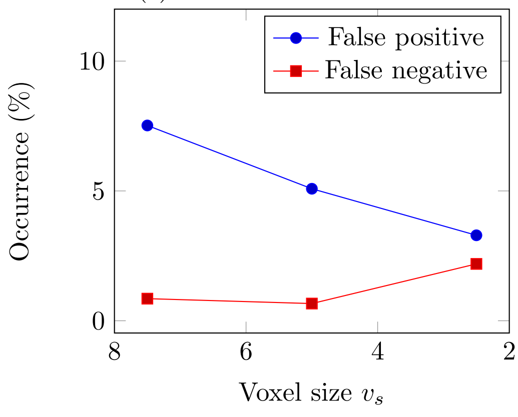
Figure 3.6: Error structure of FK maps for different δ parameter



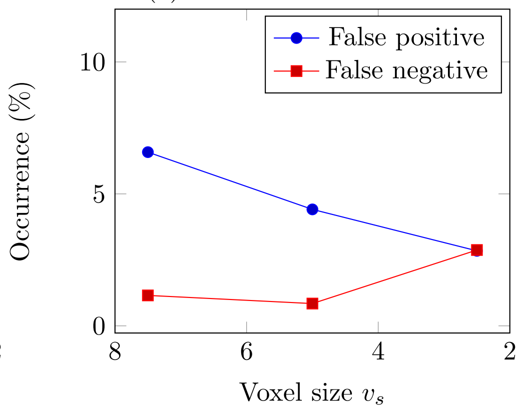
(a) HK with $\delta = 0.1\%$



(b) HK with $\delta = 1\%$



(c) HK with $\delta = 5\%$



(d) HK with $\delta = 10\%$

Figure 3.7: Error structure of HK maps for different δ parameter

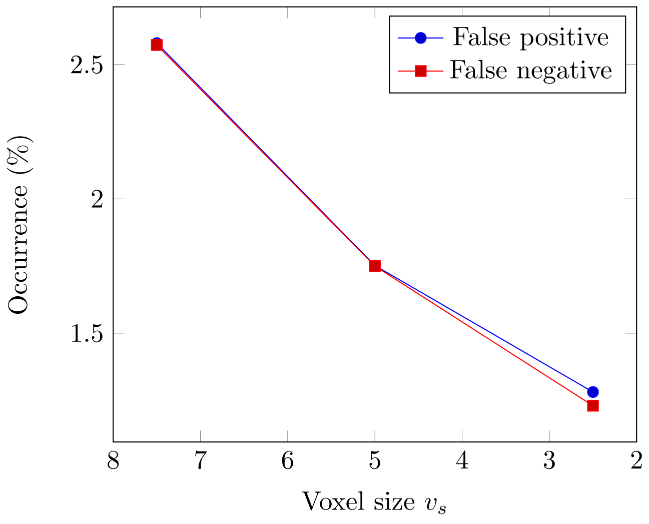
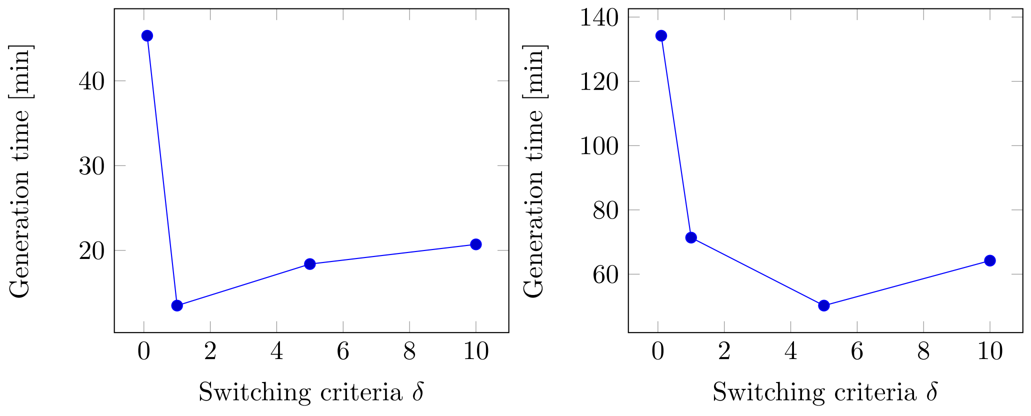


Figure 3.8: Error structure of IK maps

Figure 3.9 plots the values of generation times of hybrid approach at various resolution parameters. It shows the dependency of an optimality of the switching criteria with the chosen resolution. This optimal point is mainly dependent on the IK pose verification performance ratio to the FK random pose generation performance.



(a) $v_s = 5$, $so_2 = 100$, $n_r = 20$

(b) $v_s = 5$, $so_2 = 200$, $n_r = 30$

Figure 3.9: Generation times of HK method with respect to the switching criteria δ at various resolutions

3.3 Source of prediction error

This section examines the nature of the error. Discretizing higher dimensional spaces is not an easy task. The discretization method might favour some regions of the space more than others. Therefore, we examine the error queries with respect to their parameters such as Cartesian location within the voxel and within the whole workspace.

3.3.1 Error related to Cartesian position in the workspace

While the consistency test of reachability map and inverse kinematics was performed, we recorded the Cartesian positions of the queries. As each position can be assigned to a particular voxel, Fig. 3.10 indicates the relative number of errors per voxel. In this case, red voxels contain the least number of erroneous queries while the blue voxels contain the highest number of erroneous queries. Note that there are few errors in the regions close to the inner and outer boundaries of the workspace, while high number of errors appear in regions of medium dexterity (compare to Fig. 2.14).

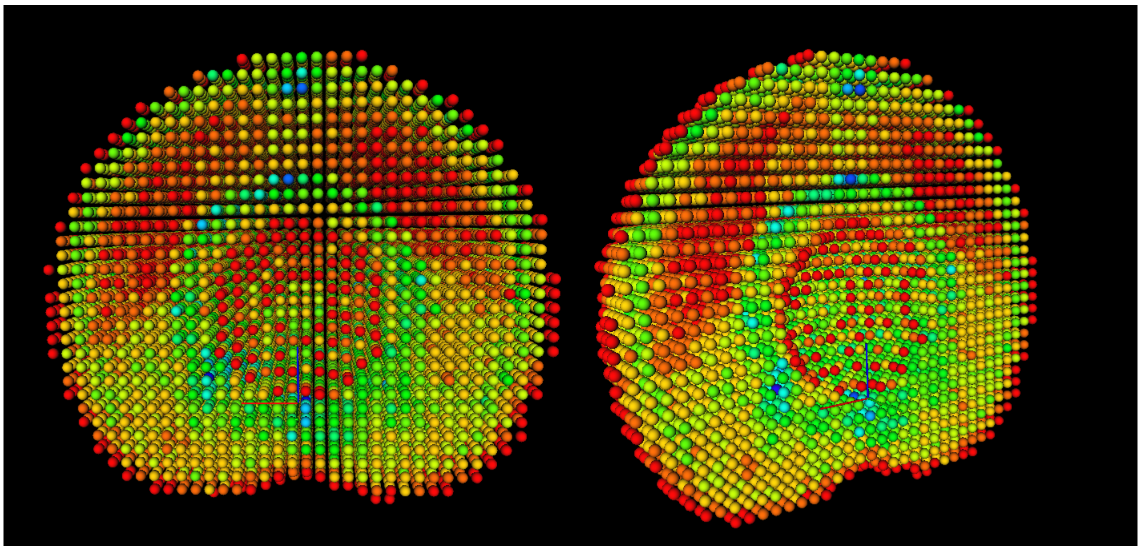


Figure 3.10: Crossection of an LWR workspace, depicting the voxels where an erroneous query was obtained

3.3.2 Error related to voxel structure

In order to have a better understanding of the error structure within one voxel we also recorded the distance between the query and the assigned centroid. The resulting histogram of the values is presented in Fig. 3.11.

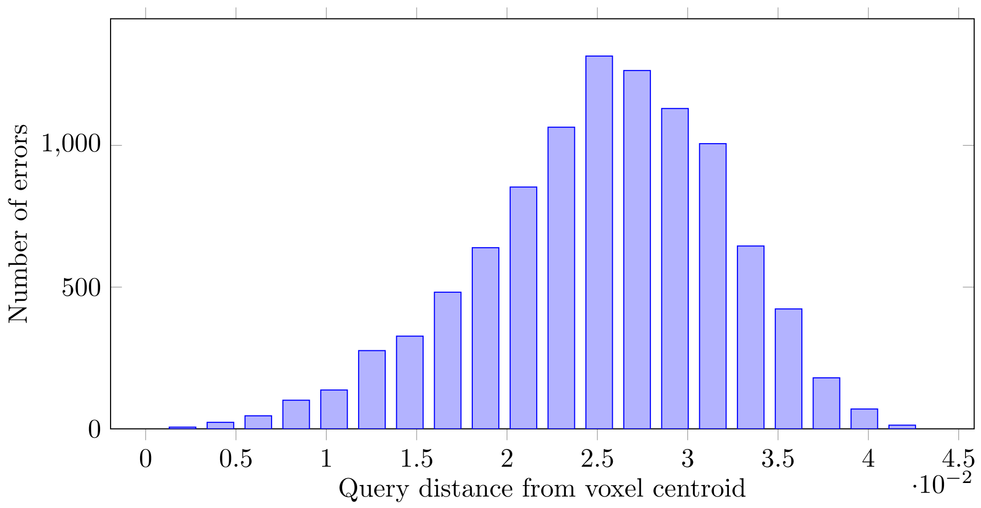


Figure 3.11: Faulty classifications related to query distance from the voxel centroid (at $v_s = 0.05$)

With voxel size v_s we have a maximum possible distance d_m between the query and the centroid given by Eq. (3.1). Given the voxelized grid, it is simply half a length of a spatial diagonal of a cube.

$$d_m = \frac{\sqrt{v_s^2 + v_s^2 + v_s^2}}{2} \tag{3.1}$$

For a voxel size of $v_s = 5cm$, $d_m = 4.33cm$. Note that about 50% of errors are produced a distances higher than $2.5cm$ (half of the voxel size).

3.3.3 Error related to generation method

We start analyzing the error data presented in tables 3.5 and 3.12. An FK generated map is in general always biased to produce more falsely positive error. Let's imagine the reachability map as a classifier, and the generation method as providing training examples. IK can provide positive and negative examples, while FK only provides positive examples. Our space discretization then labels the rest of the data as negative example. We show the consequences of such behaviour on a simple 2-dimensional case. We start with and empty grid - simplification to our 6D discretization, as shown in Fig. 3.12.

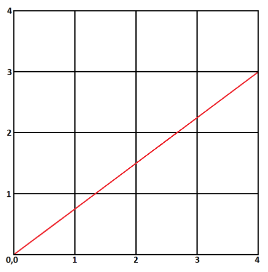


Figure 3.12: An empty 2D grid

The red line defines the boundary, separating reachable and unreachable poses. According to Fig. 3.12 we want to discretize the space into square regions of size 1. The reachable space would be described as $y < \frac{4}{3}x$. Generating representative samples using IK and collecting random FK samples is presented in Fig. 3.13.

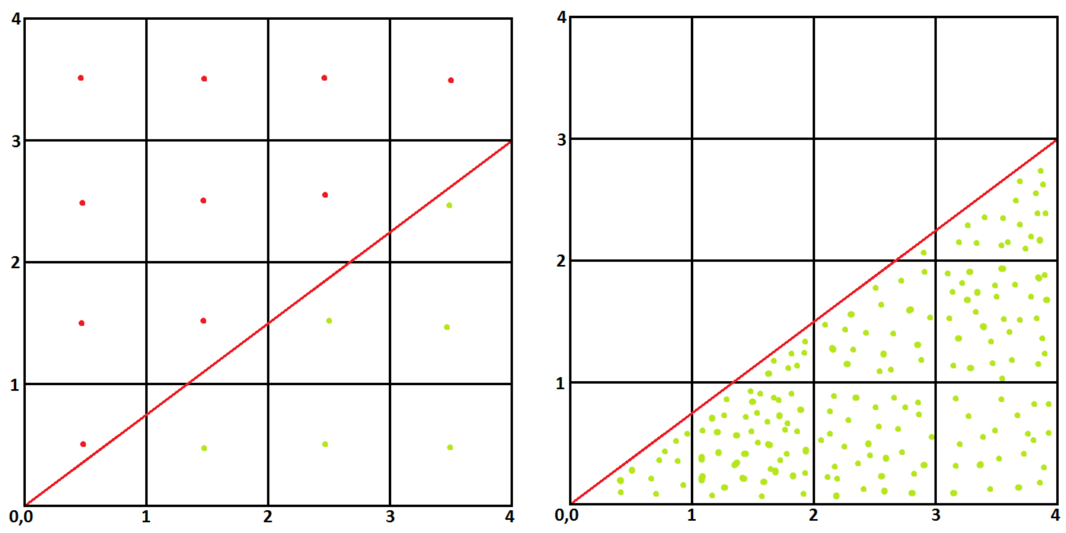


Figure 3.13: Positive and negative samples as collected by IK and FK methods

This example shows that using IK we are generating a bin-representative positive (green) and negative (red) examples. FK generation only generates positive examples. The negative examples after FK generation are simply the bins that were not visited. The boundary line cuts through the discretization grid, where all the boundary bins are split. Therefore, such bins cannot be represented by a single binary value. This is the main source of error of the $SO(3)$ and $\mathbb{R}^3$ grids. After processing the data from Fig. 3.13 we obtain the binary representation shown in Fig. 3.14.

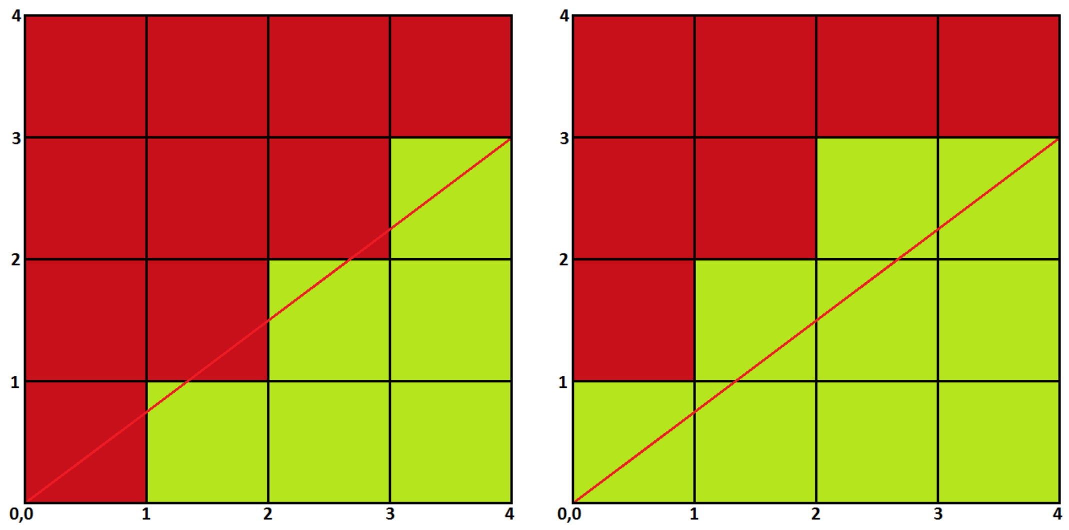


Figure 3.14: Classification result based on IK and FK approach

Figure 3.14 shows our final representation in the reachability map. Red areas are marked as not reachable and green areas as reachable. The grids and reachable regions are identical but the classification result is not. The longer the FK generation is running the more the error shifts towards false positive. In cases when most of the region of a bin is unreachable, it will eventually be marked as reachable by FK method but not by IK.

Note that the binary nature of the classifier creates this distortion. This result suggests that another method of sample classification, for instance a Support Vector Machine (SVM), might improve the boundary estimation, even with the same resolutions and number of samples used so far.

4. Applications

This chapter presents several applications, on and off-line, for the reachability or capability maps. Adapting to new on-line applications is generally only a matter of proper implementation of the higher layer that provides access to the raw data of the map. When the number of queries is not extensive, the underlying reachability / capability map can satisfy the constraints for on-line performance.

4.1 On-line detection of reachable points

Our first application was a real-time visualization of reachable points in the scene. We obtain a depth map and calibration parameters for projection. After projecting the depth map into a set of 3D points we estimate the normals in the scene. Normal estimation is a common technique, based on principal component analysis for a set of neighboring points. The ambiguity of estimated normals direction is resolved by the fact that we have a 2.5D sensor. All of the perceived surfaces must, therefore, point towards the camera. A freely available implementation can be found in the Point Cloud Library [17]. To demonstrate the method on a simple case, we placed a ball and a box on the floor. We also assume that the sensor and robot base frame are identical; in a robot, the difference between the frames can be easily obtained. The combination of point and a normal creates a 5D query information. Since the last missing parameter is not provided (tool roll angle), we consider a point to be reachable if at least one roll direction is reachable for the corresponding approach direction in the reachability map.

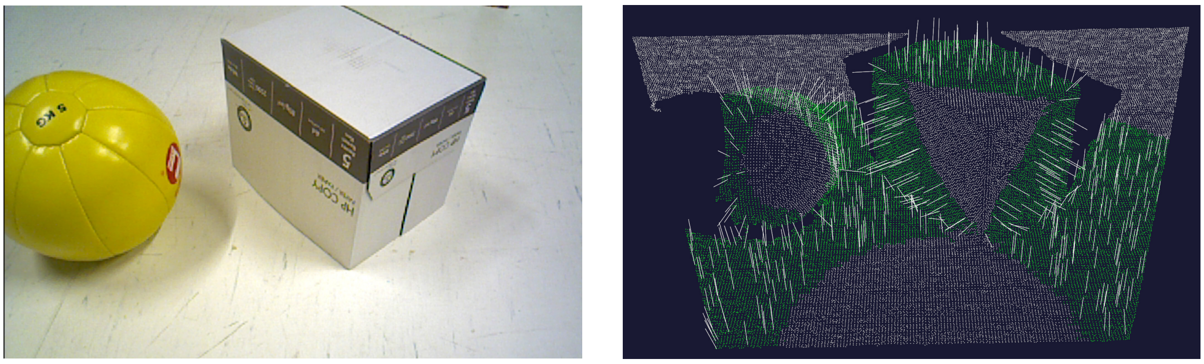


Figure 4.1: Reachable points in a simple scene

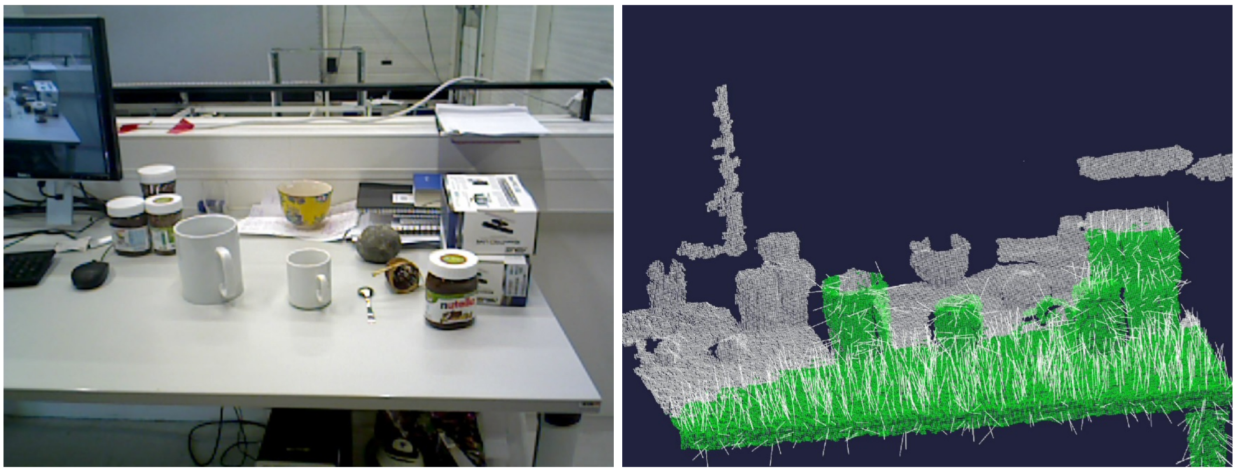


Figure 4.2: Reachable points in a complex table top scene

Points marked in green represent positions where the robot is able to apply force in the direction of the normal to the surface. The same method applied to a more complex table-top scene is shown in Fig. 4.2.

4.2 Grasp selection

Another on-line application uses the reachability/capability map to select possible grasps, or to choose the best grasp for subsequent object manipulation. To identify which grasp is more suitable for immediate manipulation, capability map provides a metric for selection. We start by identifying an object in front of the robot. Each object has a stored database of grasps, where a grasp is a defined as a pre-grasp pose of the end effector.

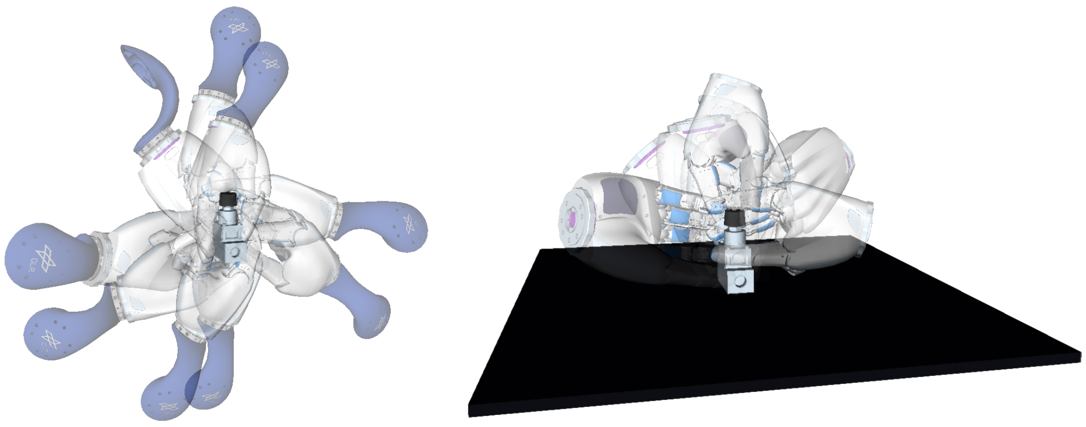


Figure 4.3: Possible grasps of an object

Fig. 4.3 shows the database of grasps for one free floating object. After filtering unfeasible grasps due to the supporting surface, we are still left with many options. These options are qualitatively different, since some poses cannot be reached with the current object position and some have very limited options for subsequent motions. Capability map and the reachability or extended reachability index allows us to make an educated decision on which grasp to choose.

4.3 Off-line workspace analysis for a free-floating manipulator

When performing off-line analysis, we mostly want to address the questions of workspace shape and quality. Reachability index represented in the capability map allows us to estimate volumes of certain quality and directly compare different scenarios or settings.

We demonstrate the utility of these methods in an off-line analysis performed for the ongoing space mission DEOS, which stands for DEutsche Orbital Servicing - a twin satellite mission to investigate capture and control techniques. DEOS is a mission governed by the German Aerospace Agency (DLR) and EADS Astrium. Two satellites will be put in orbit. One, called client, will simulate a non-responsive satellite body. The other is a universal servicer with the capability of on-orbit capture, servicing and release (or decommission) of the client. In this analysis we take a closer look on different possible mission designs for the servicer satellite.

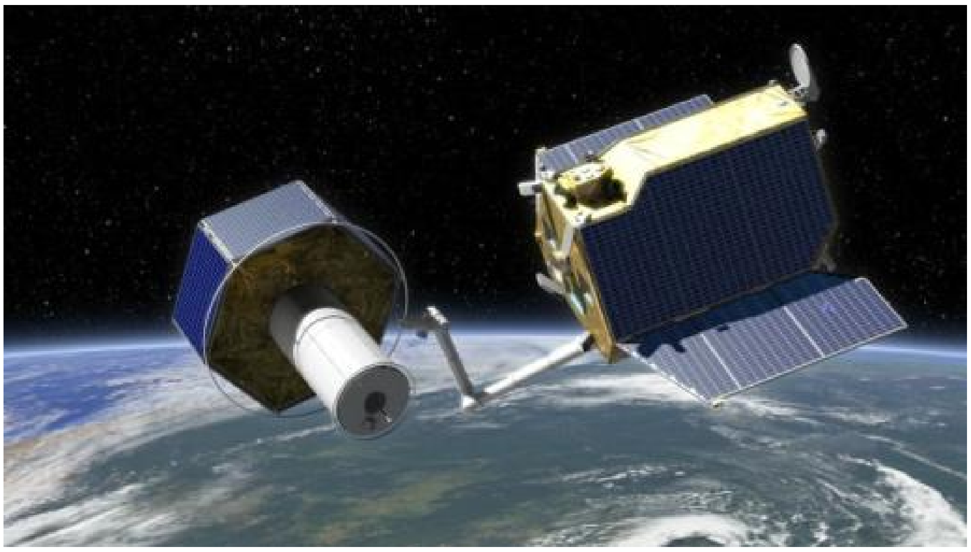


Figure 4.4: DEOS mission - client and the servicer (courtesy of Astrium)

The servicer is equipped with a custom designed robotic arm. The goal is to approach a tumbling target and capture it using the robot. The robot can be mounted on several spots on the satellite body. One of the steps in the mission involves grasping a tumbling target in front of the satellite, therefore, the mounting point position and the robot's design have to maximize the dexterous workspace and its volume at the frontal face. We create a capability map for two different mounting points of the robot and compare the dexterous workspace. Figures 4.6, 4.7, 4.8, 4.10, 4.9, 4.11 provide a comparison of these two mounting points. On the left side, the workspace for a robot mounted on the top face of the satellite is shown. On the right side of the figures, there is a corresponding visualization of the workspace when the robot is mounted on the frontal face of the satellite. Figure 4.5 presents the plain point cloud model of the satellite used for the collision detection.

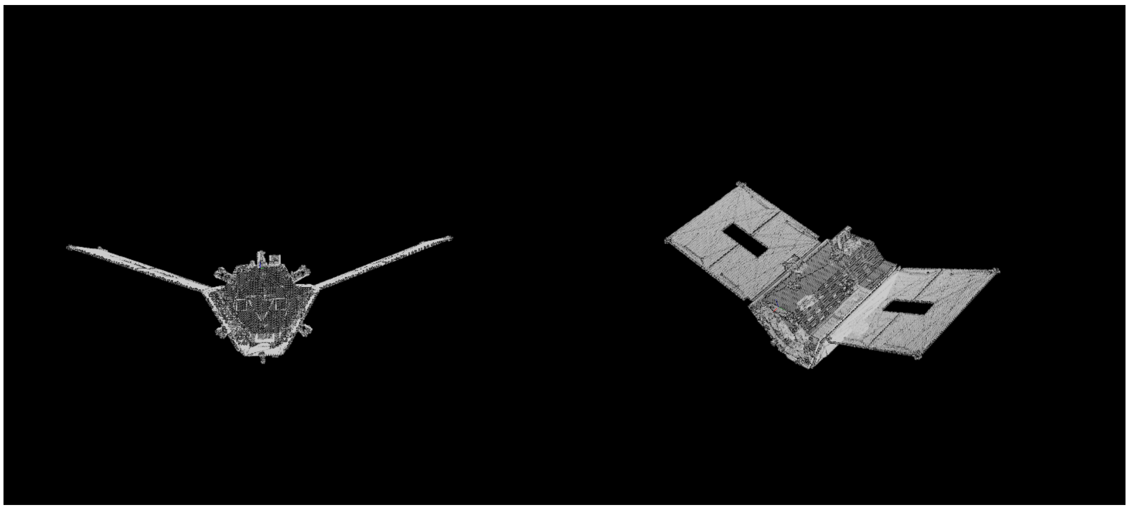


Figure 4.5: Views on DEOS servicer, point cloud model

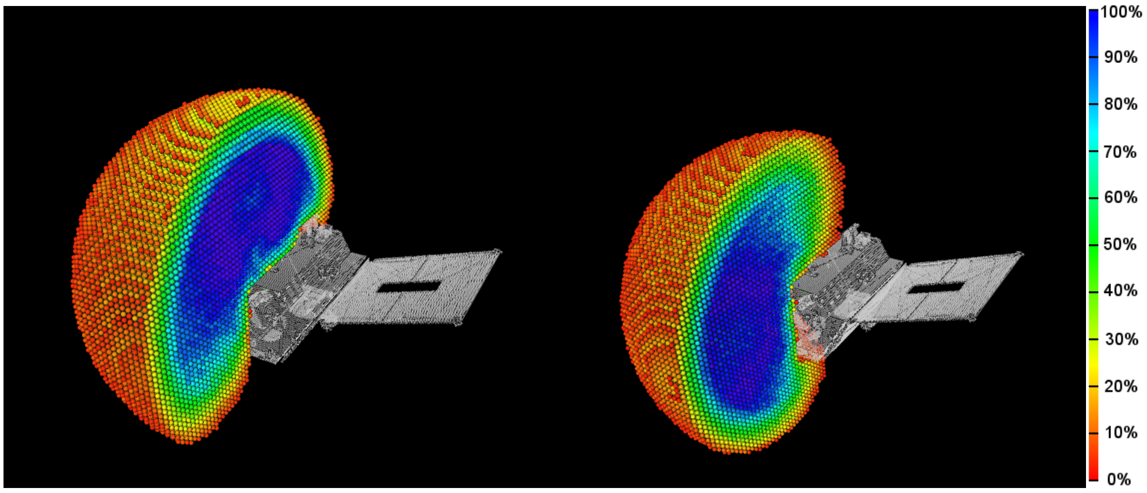


Figure 4.6: Cross sections of robot’s workspace

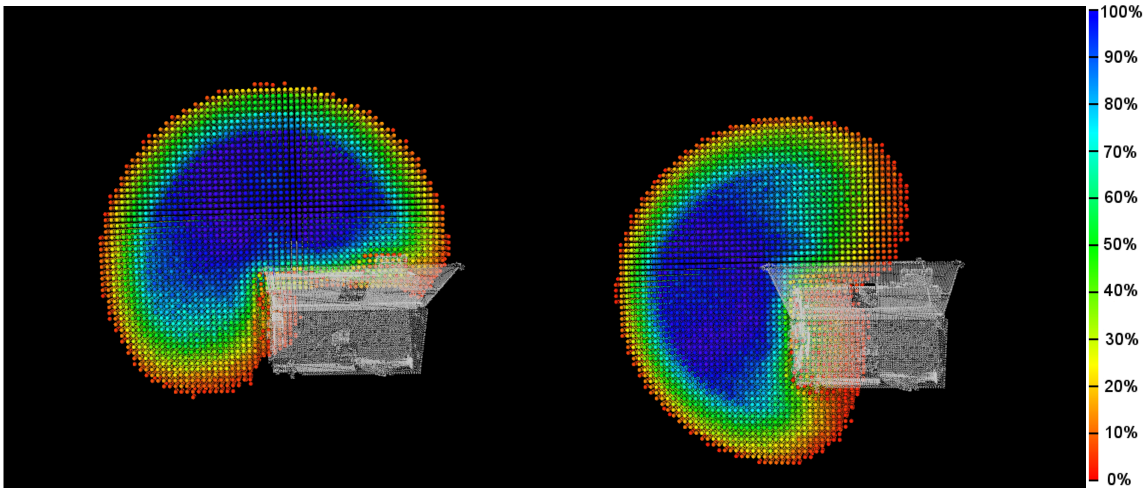


Figure 4.7: Cross sections of robot’s workspace - lateral view

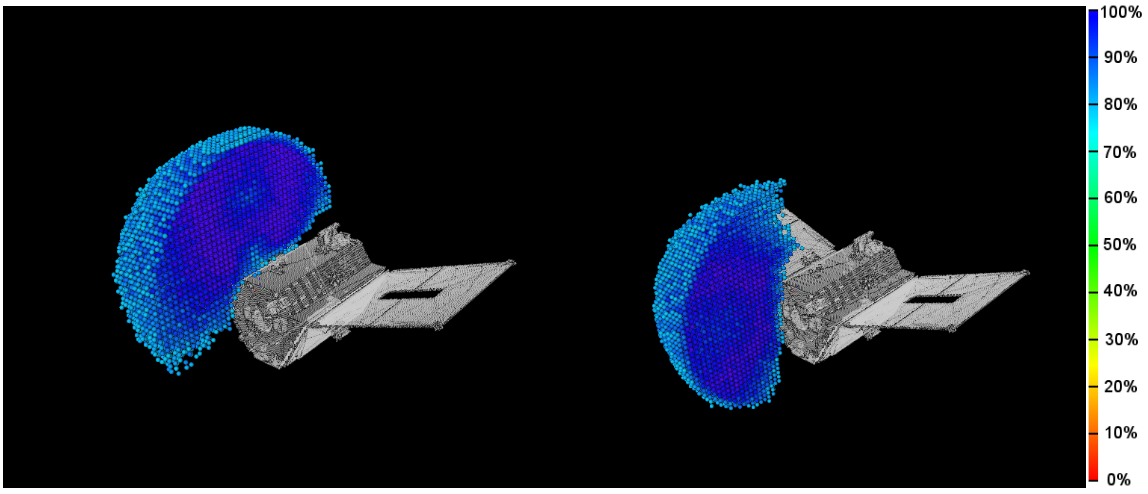


Figure 4.8: Cross sections of high dexterity workspace ($R_i > 75\%$)

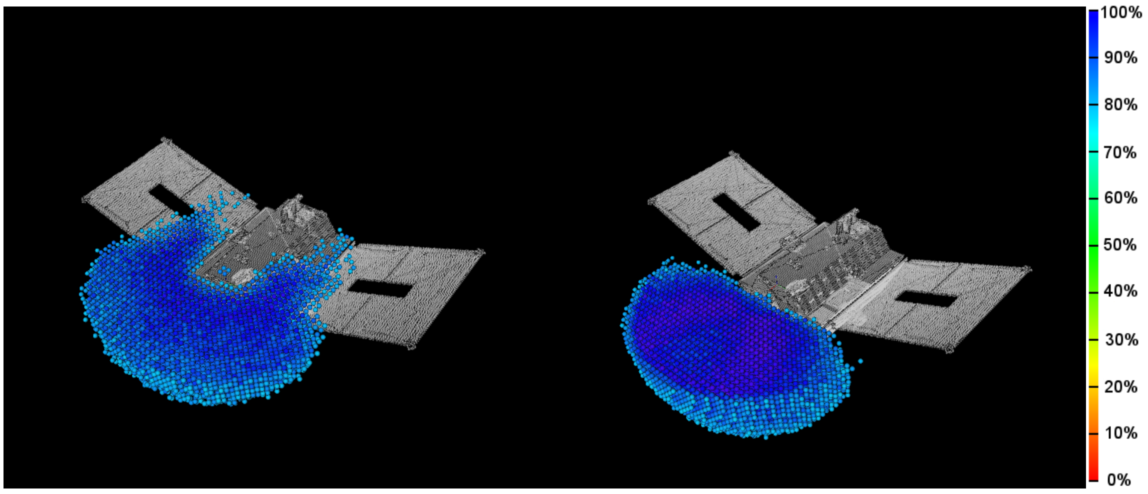


Figure 4.9: Lower cross sections of higher dexterity robot's workspace ($R_i > 75\%$)

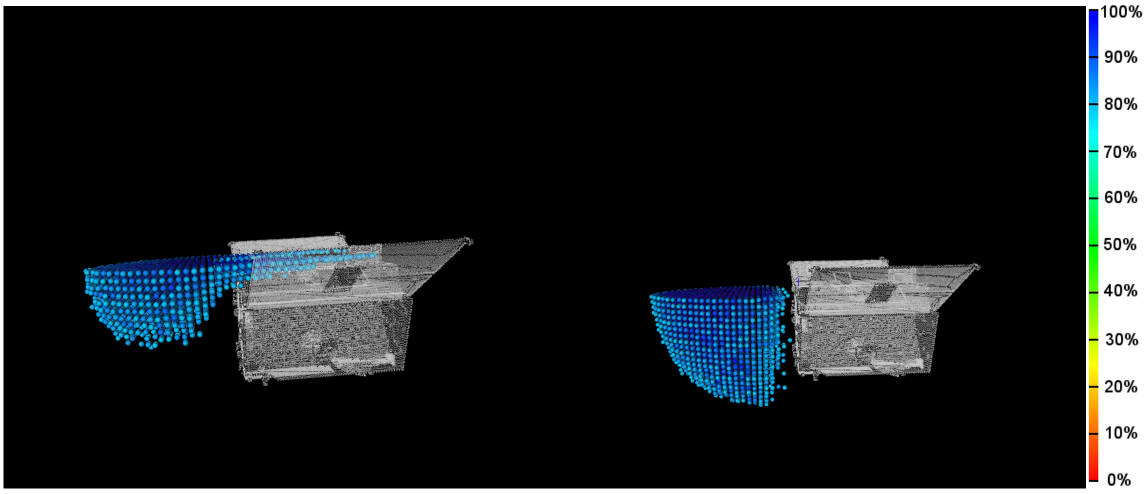


Figure 4.10: Lower cross sections of high dexterity robot's workspace - lateral view ($R_i > 75\%$)

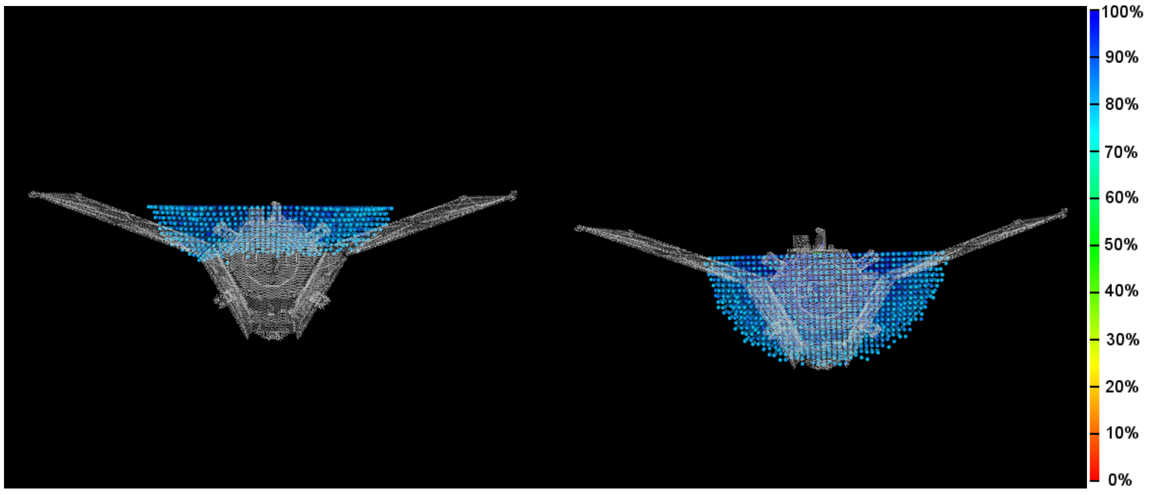


Figure 4.11: Rear view of the cross sections of high dexterity workspace ($R_i > 75\%$).

Fig. 4.7 and 4.8 show a cross-section of the robot's capability map with the servicer model. Fig. 4.10, 4.9, 4.11 show various cross sections of the high dexterity workspaces. A higher dexterity workspace is defined as $R_i > 75\%$, which means that more than 75% of the discretized $SO(3)$ directions are reachable. Figures above show such part of the robot's workspace, obviously avoiding collisions with the main body. There are approximately $30.66m^3$ of a high dexterous workspace in the figures where the robot is mounted on the top face of the servicer. However, only 5.05 m^3 lies at the frontal face, which is depicted in Figures 4.10 and 4.9. In the other case, when the robot is mounted on the frontal face of the servicer, the total volume of high dexterity space is around $23.46m^3$. However, there is a volume of 10.47 m^3 that lies at the frontal face. This means that the robot is able to perform dexterous movements in a volume two times bigger when the arm is mounted on the frontal face.

4.4 Workspace selection using capability maps

In this case we are given a constrained volume in which we want to perform a task or we have a defined trajectory the robot should execute. As example, we can have a defined manipulation task and we want to know how to position the robot in order to fulfill the task. In scenarios like opening doors or drawers, we want to position the robot such that it can execute the trajectory without repositioning, or with a minimal relocation effort. Identifying the optimal dexterous sub-workspace for a manipulation task can be performed off-line, because, the result is static with respect to the robot base. This allows, for instance, choosing the best height of a table for performing a sequence of motions, i.e. the work environment for the robot could be designed off-line. In other words, we would perform a sliding window averaging of the map to obtain suitable volumes for task execution, i.e. volumes that generate the maximum dexterity when performing the task. Such experiment was done for the DLR Humanoid robot TORO. Fig. 4.12 presents a collision free capability map for the right arm of the humanoid. The analysis revealed that an ideal height of a table for the robot is $120cm$ above the ground. A related problem, the computation of a suitable posture for executing a given trajectory, can be performed on-line using the same map.

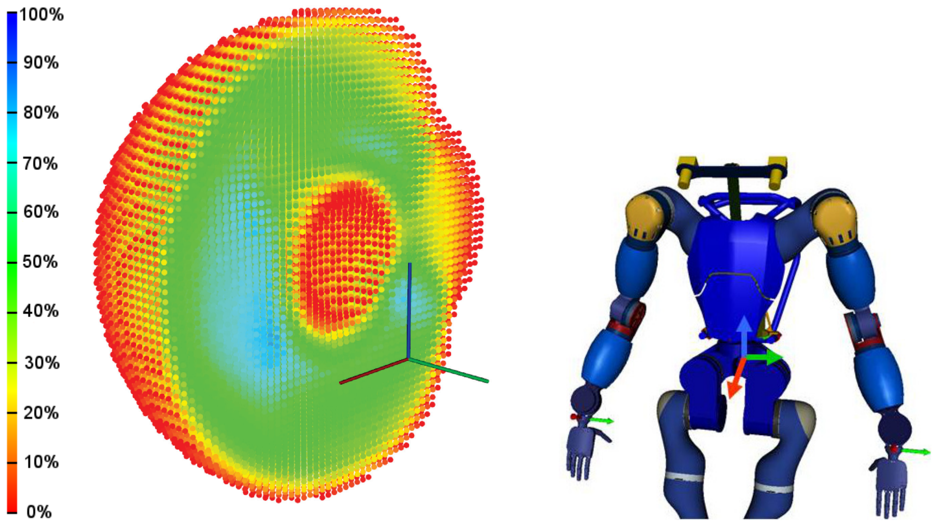


Figure 4.12: Capability map for DLR Humanoid

4.5 On-line workspace boundary estimation of a surgical robot

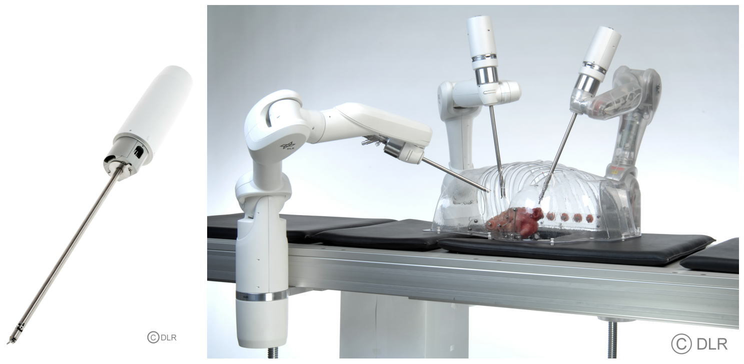


Figure 4.13: Surgical robot MIRO (Courtesy of DLR)

Fig. 4.13 shows the robotic surgery system developed at the laboratories of DLR. The surgical tool is shown in the left side, and the actual robotic system is shown to the right. The robot has to enter the subject's body through an incision point. This point cannot be changed after the operation has started, since if would defeat the idea behind a minimal invasive surgery. The capabilities of the kinematic chain before the incision point are projected, using the tool parameters, through the incision point into the subject's body. The system is designed such that the robots can be arbitrarily positioned around the operating table. It is a crucial task to determine the operating workspace shape, i.e. validate an incision point before the operation starts. We use a projection of the reachability map through a given point to determine the workspace shape and boundaries. The parameters required for the projection are the tool length t_l and the incision point $I = [i_x, i_y, i_z]$. For each voxel within the range of the instrument around the point I, we check the validity of the end effector pose in the reachability map. If the pose in the direction of the incision point is valid, we project it through the point with the given tool length t_l. A cross-section of the capability map of a surgical robot is shown in Fig. 4.14. The results of the projection for a sample incision point are pictured in Fig. 4.15, 4.16. The incision point is depicted in red and the projected points are depicted in white.

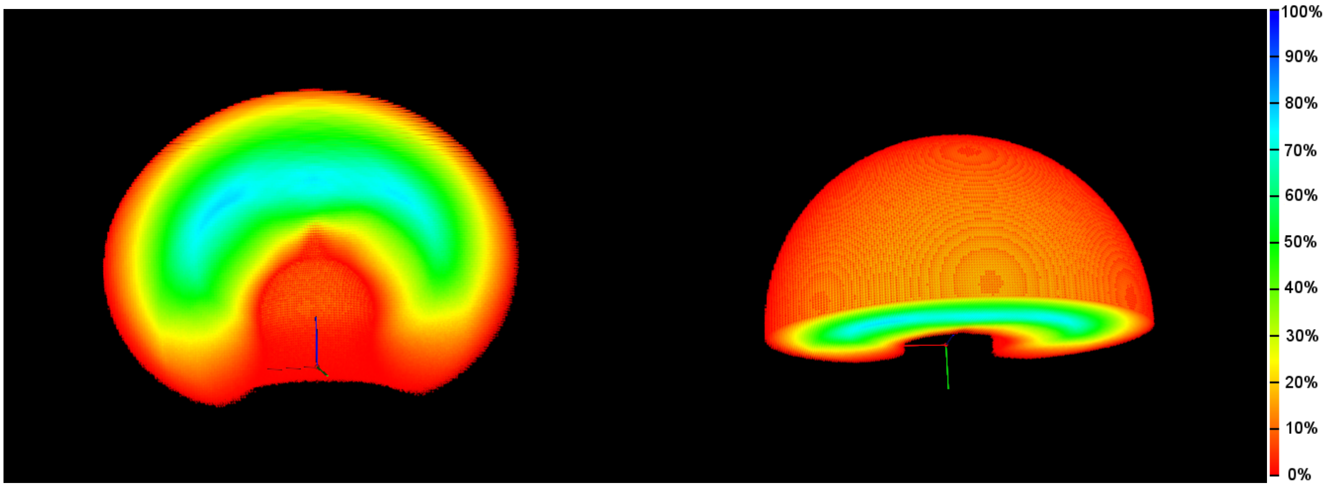


Figure 4.14: Cross sections of surgical robot's workspace

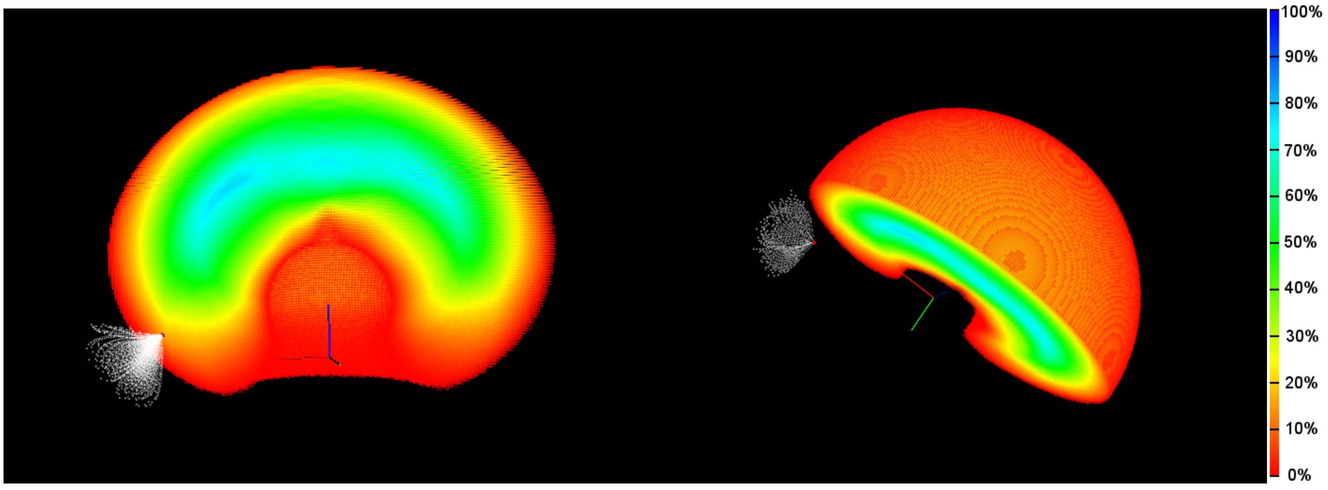


Figure 4.15: Projection of valid workspace through a fixed incision point

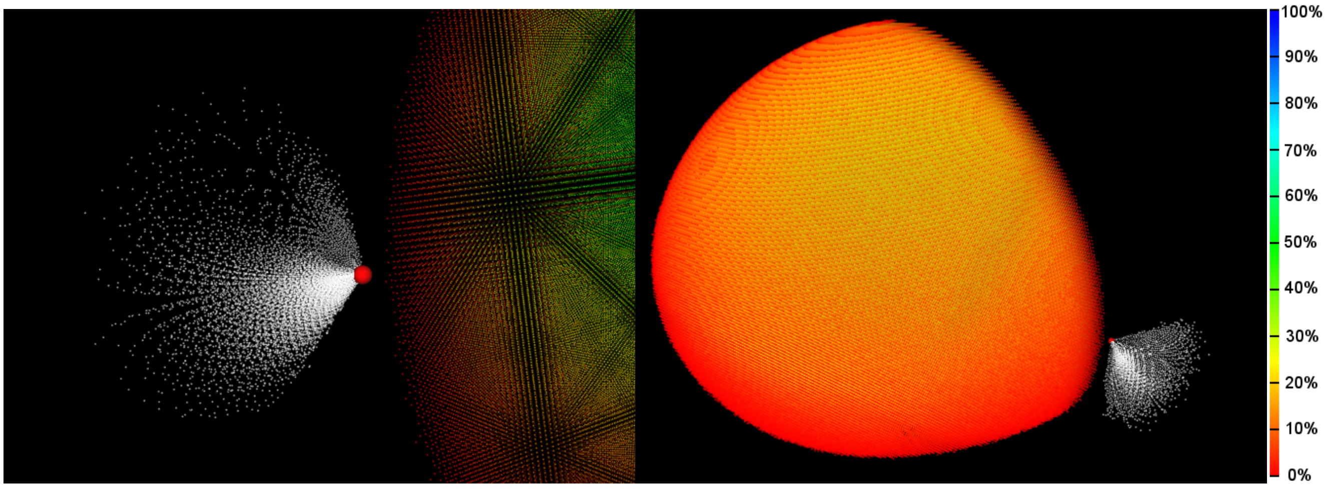


Figure 4.16: Projection of valid workspace through a fixed incision point

4.6 Path planning

Path planning belongs to the group of on-line applications. The options of incorporating reachability or capability maps with path planning are endless. We can for example use the reachability map to avoid querying the inverse kinematics, hence to speed up the overall planning. We can navigate a planner through more dexterous regions using capability map, where we are less likely to get stuck in case an obstacle is introduced. The voxelized structure can be used to downsample the perceived point set for collision detection into a minimal representative set which is just enough for the collision detection to work flawlessly. Hereby, we present a basic solution for map exploration and path finding for the TCP. Then we describe how to couple this path with inverse kinematics and produce a collision free path. We use the reachability map for query evaluation, capability map for exploration guidance and the voxelized structure to facilitate the integration of a fast environment perception.

The core of the method is governed by an A^* algorithm. The voxelized structure provides a natural A^* operating environment. There are several advantages and consequences with this method. A^* always finds a path, if one exists, and it is optimal with respect to the metric provided. We are navigating in a 6 dimensional environment, therefore we have to define how do we treat the rotational and translational components. The problem is defined as finding a path from start to goal, $TCP_A \rightarrow TCP_B$. To simplify for A^* we start by decoupling the rotational components as rotational matrices R_A and R_B. The rotational part will be handled separately by the reachability map (as one example) and the translational part will be handled by A^*. The start and end poses are given; however, the constrains on the rotational components might be task specific. A grasping task with an installed in-hand camera might require to keep the target object in the field of view. Manipulation with liquid containers can restrict the rotational path such that the TCP stays within a rotational range (to avoid tipping over the liquid). Our application follows the first track: We would like the orientation of the TCP to reach its final orientation R_B in about half the distance to the object. R_A and R_B are transformed into quaternions which, unlike rotational matrices, can be linearly interpolated [18]. This will be our desired path in the orientation space. Next, we use A^* algorithm to find a valid translational path through the voxelized structure, satisfying the interpolated rotations. The pose validity is first checked with the reachability map to avoid unnecessary inverse kinematics calculation. This ensures that we only explore the reachable space.

Use of A^*

A^* is a well known algorithm. We use the main concepts of keeping an open list for adjacent voxels of the path and a closed list for already explored and verified voxels. We also use the composed metric of distance already travelled, commonly denoted as g, and approximated distance h which in the basic case is the Manhattan distance. The metric is what allows us to bias the path towards more desirable regions. In general we scale the distance travelled g by an additional desired metric. It could be the reachability index, extended reachability index or a metric that describes distance to singularities.

The resulting path is composed by back propagation through the closed list of voxels with the lowest metric. Additional information on plausibility of the path is required. The actual inverse kinematics can be used during the space exploration or during the back propagation. In the first case, there will be many more queries for the inverse kinematics and it is incorporated during the validation of a sample. The later case is more sensitive to the quality of the reachability map. In case when the reachability map has a large number of false positive validations, it would slow down the back propagation process. We employed the inverse kinematics during the space exploration phase as part of the sample validation. In such case it is easier to integrate a collision detection as described in section 2.8. In case of using a redundant robotic arm we also need to ensure the continuity of the samples. There might be different possible configurations to reach the goal. Such metric can be reflected in the back propagation phase by simply constructing the path of a lowest distance in the configuration space.

Note on implementation

To achieve reasonable search times with higher resolution maps it is important to implement a suited memory structure for the open and closed lists. Such structure is commonly called binary heap. Binary heap is a memory structure implementing a variation of quick sort algorithm directly on the memory container. Every time a sample is added, it is placed in a particular offset in the memory. Such mechanism ensures that the values are ordered at all times and smallest or largest values can be pulled off the list without traversing all the values in the list.

Fig. 4.17 shows how we can bias the A^* metric in both space exploration and path propagation using an additional metric. The figure is showing two paths starting at the TCP pose marked by a red sphere and goal marked by a green sphere. The coordinate frames visualize the samples that A^* propagated through. The path indicated only by frames (without marking) was discovered by employing a distance metric only. The path with samples marked with a white sphere were discovered using a modified metric that takes into account the distance to singularities, given by the minimum singular value of the Jacobian of the corresponding arm pose [21] [22]. The effect of the metric is visualized in the cross section where blue spheres (further away from singularities) are the most preferred volumes to propagate through.

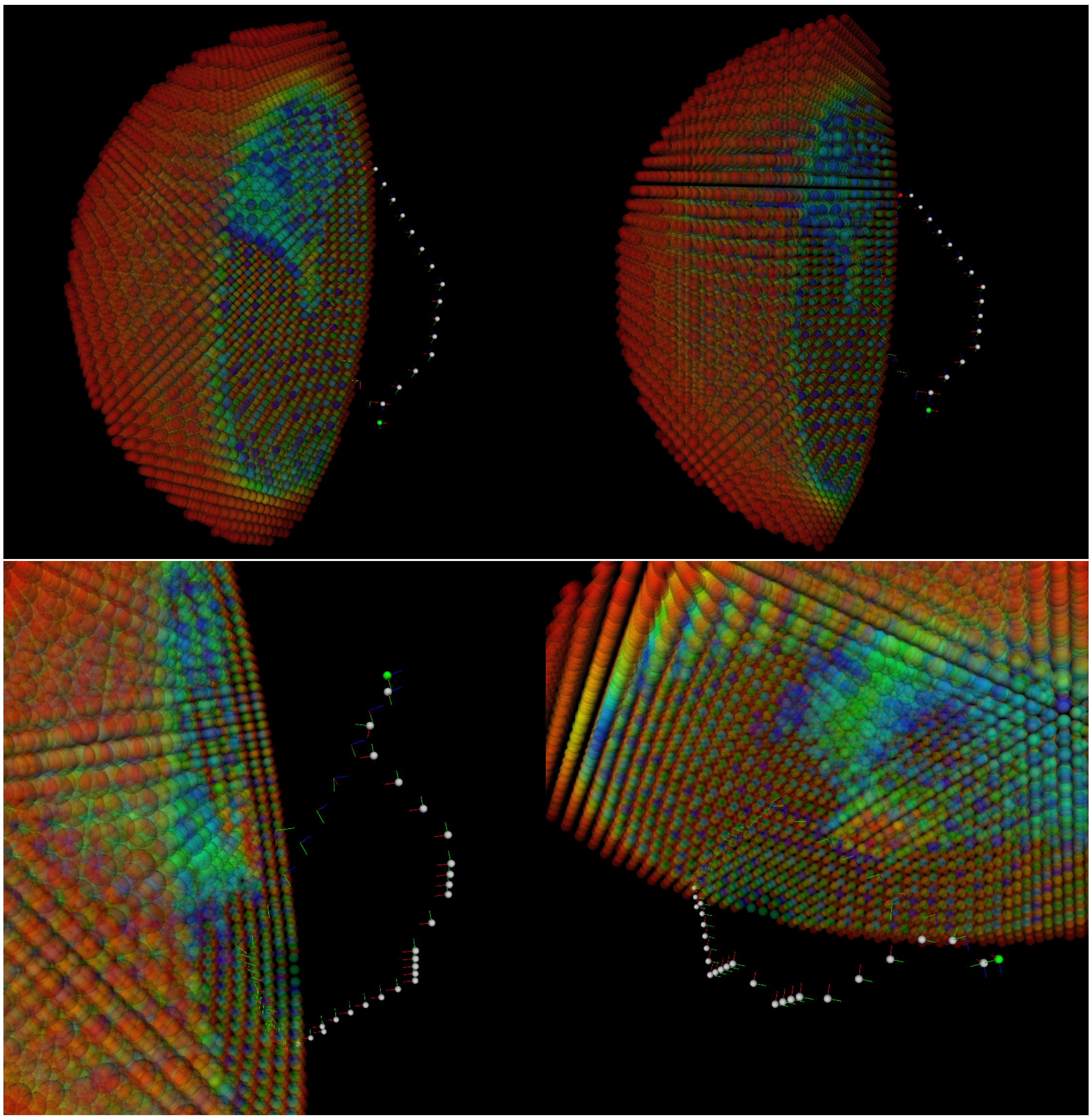


Figure 4.17: Comparison of pure and biased A^*

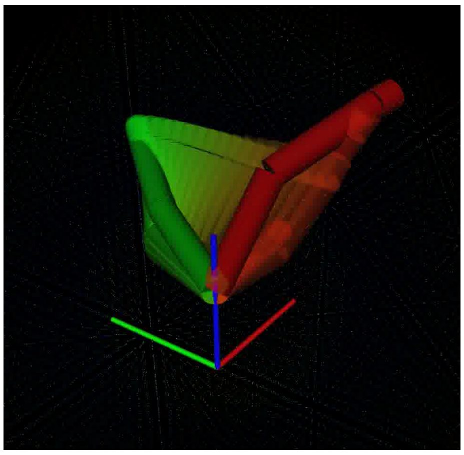


Figure 4.18: Path planning including the whole robot body

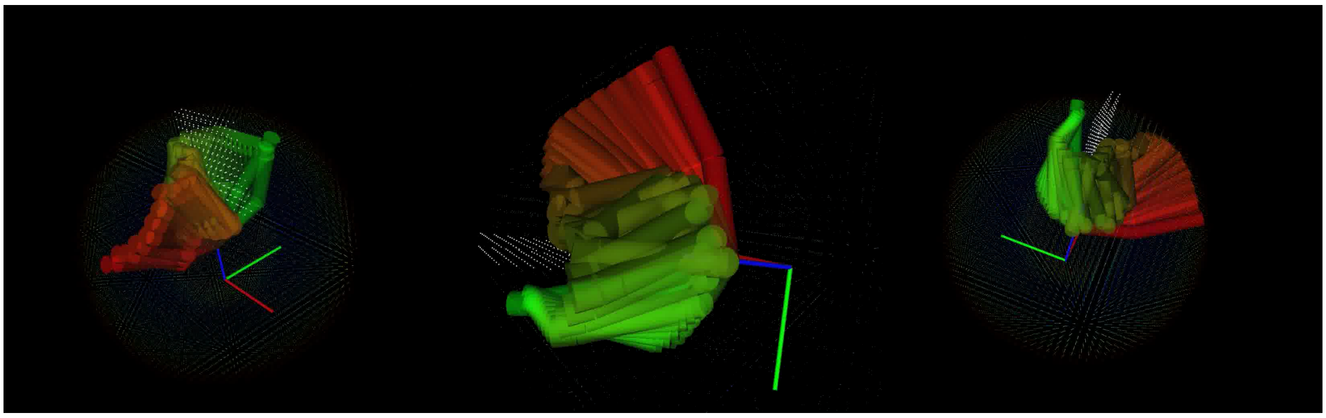


Figure 4.19: Multiple views on resulting path including the whole robot body and obstacle avoidance

Figures 4.18 and 4.19 visualize the path planning methods applied to the KUKA LWR. The planning starts from the most red configuration (given as TCP_A) and plans a valid path to the most green configuration (given as TCP_B). In Fig. 4.19 we introduced an artificial obstacle which is visualized as the set of white points. The planner successfully avoids the obstacle and reaches the goal pose. The time for obtaining a solution depends on the voxel resolution $1/v_s$ and it varies with the length of the path. On average, the computation time is around 0.5 seconds.

5. Conclusions and future work

We have demonstrated a method of discretizing a 6-dimensional configuration space, which is suitable for obtaining the reachability and capability maps that describe the size and quality of the workspace of a robot. Three methods of map generations were compared and studied - forward kinematics, inverse kinematics and a new hybrid approach. Given the results of prediction accuracy from Figures 3.2, 3.3 and 3.4 we can conclude that the most accurate approximation of all reachable points of a robotic manipulator is the inverse kinematics method. Maps of usable quality can be quickly generated using the forward kinematics method.

These maps suffer from an unclear boundary shaping as it was demonstrated in Sec. 3.3.3. A strong observation on the quality of this method is that the error characteristics shifts towards falsely positive classifications. This can degrade the performance of methods relying on such results. Due to the decoupled nature of $SE(3)$, it is not possible to simply interpolate the resulting query using the nearest neighbor bins. A trade-off of low generation time, complete workspace exploration with better defined boundary, therefore, better prediction accuracy is obtained with the hybrid approach. The grid resolution plays equal role in all the methods, the only additional parameter is the criterion δ for switching between forward and inverse kinematics. In general, $\delta = 10\%$ is a good choice for the hybrid methods.

An average query performance of the reachability map is 1950 queries per millisecond in the current implementation. Thanks to this we were able to deploy applications such as off-line workspace design and analysis, on-line determination of workspaces, selection of grasping poses and path planning. The most performance demanding application of on-line determination of reachable points can run at a full frame rate and resolution of the PrimeSens unit used which is 640×480 points at $30Hz$. The bottleneck to the method is the normal estimation of the point cloud. Section 4.3 demonstrates the usage of capability map in combination with collision avoidance to reveal the real usable workspace of a mounted robotic manipulator. A robot with otherwise large and high dexterity workspace can be severely limited by its mobile base or other permanent obstacles. In such case, collisions and self-collisions have to be taken into account. We provide an application example of the method for a free floating manipulator in a space-related scenario. For autonomous manipulation tasks, capability map provides a metric for narrowing down the grasp options with respect to the range of subsequent manipulation as described in section 4.2.

As any discretization method, reachability map suffers from an inevitable error of the grid resolution. Storing high resolution maps is not always possible for on-line applications, therefore increasing the resolution is not a feasible way to increase the precision. Every real rigid object can only be moved in a continuous manner in the $SE(3)$ space. By analogy, the boundaries between reachable and unreachable task subspaces should be continuous. One line of future work is using SVM for approximating each voxel's reachability to increase the overall prediction accuracy and hopefully reduce the memory footprint.

Initial results of this thesis have been presented in the First Iberian Robotics Conference, ROBOT 2013, that took place in Madrid, November 2013. The publication is listed as [19].

A. System architecture

The system has been split up into several modules to facilitate flexible future applications. Each part was implemented in C++, with minimum number of dependent libraries. The dependencies include Eigen, Boost, SensorNet (institutes's in-house library), Coin3D and QT.

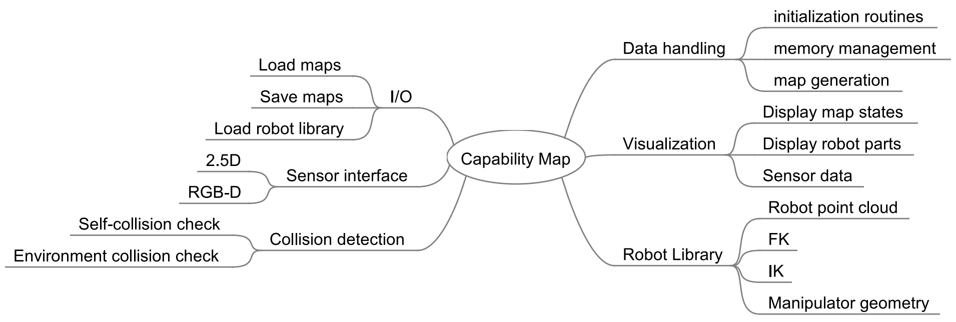


Figure A.1: Software modules - functional division

Bibliography

[1] Rosen Diankov, *Automated Construction of Robotic Manipulation Programs*, Carnegie Mellon University, Robotics Institute, 2010, CMU-RI-TR-10-29

[2] M. A. Roa, K. Hertkorn, F. Zacharias, C. Borst and G. Hirzinger, *Graspability Map: A Tool for Evaluating Grasp Capabilities*, IROS, 2011, p.1768-1774

[3] D. Kee and W. Karwowski, *Analytically derived three-dimensional reach volumes based on multijoint movements*, Human Factors: The Journal of the Human Factors and Ergonomics Society, 2002, p.530-544

[4] F. Zacharias, *Knowledge Representations for Planning Manipulation Tasks*, Springer, 2012

[5] F. Zacharias, C. Borst and G. Hirzinger, *Online Generation of Reachable Grasps for Dexterous Manipulation Using a Representation of the Reachable Workspace*, Proc. IEEE Int. Conf. Advanced Robotics, 2009, p.3229-3236

[6] F. Zacharias, C. Borst and G. Hirzinger, *Capturing Robot Workspace Structure: Representing Robot Capabilities*, IROS, 2007, p. 3229-3236

[7] F. Zacharias, W. Sepp, C. Borst and G. Hirzinger, *Using a Model of the Reachable Workspace to Position Mobile Manipulators for 3-D Trajectories*, HUMANOIDS, 2009, p.55-61

[8] N. Vahrenkamp, T. Asfour and R. Dillmann, *Robot placement based on reachability inversion*, ICRA, 2013, p. 1962-1967

[9] L. Guilamo, J. Kuffner, K. Nishikawi and S. Kagami, *Efficient Prioritized Inverse Kinematic Solutions for Redundant Manipulators*, IROS, 2005, p.3921-3926

[10] C. Ott, O. Eiberger, M.A. Roa and A. Albu-Schaeffer, *Hardware and Control Concept for an Experimental Bipedal Robot with Joint Torque Sensors*, Journal of the Robotics Society of Japan, 2012, 30-4, p.378-382

[11] T. Bodenmueller, *Streaming Surface Reconstruction from Real Time 3D Measurements*, Technical University of Munich - TUM, Lehrstuhl fuer Reakzeit - Computersysteme 2009,

[12] N. Vahrenkamp, T. Asfour, G. Metta, G. Sandini and R. Dillmann, *Manipulability analysis*, HUMANOIDS, 2012, p.568–573

[13] M. Sagardia, T. Hulin, C. Preusche and G. Hirzinger, *Improvements of the Voxmap-Pointshell Algorithm - Fast Generation of Haptic Data Structures*, Proc. 53rd Int. Wissenschaftliches Kolloquium, 2008

[14] R. Suarez, M.A. Roa and J. Cornella, *Grasp Quality Measures*, Technical University of Catalunya, 2006, IOC-DT-P-2006-10

[15] R. Konietschke and G. Hirzinger, *Inverse kinematics with closed form solutions for high redundant robotic systems*, ICRA, 2009

[16] R. Wang, K. Zhou, J. Snyder, X. Liu, H. Bao, Q. Peng and B. Guo, *Variational sphere set approximation for solid objects*, Visual computation, 2006, p.612-621

[17] R. Rusu and S. Cousins, *3D is here: Point Cloud Library (PCL)*, ICRA, 2011

[18] K. Shoemake, *Animating Rotation with Quaternion Curves* ,Computer Graphics, Volume 19, 1985

[19] O. Porges, T. Stouraitis, C. Borst and M.A. Roa, *Reachability and Capability Analysis for Manipulation Tasks*, ROBOT 2013: First Iberian Robotics Conference. Advances in Intelligent Systems and Computing 253, pp. 703-718. Eds: Manuel Armada et al. Springer Switzerland 2014

[20] P. Leopardi, *A partition of unit sphere into regions of equal are and small diameter*, Electronic transactions on Numerical Analysis, Kent state University, Volume 25, p. 309-327, 2006

[21] C. Klein and B. Blaho, *Dexterity measures for the design and control of kinematically redundant manipulators*, Int. J. Robotics Research, volume 6(2), p.72 - 83, 1987

[22] E. Staffetti, H. Bruyninckx and J. De Schutter *On the Invariance of Manipulability Indices*, Advances in Robot Kinematics, Ed. Springer, pp 57-66, 2002