Master Thesis
Intelligent Systems

Path Planning for a Redundant Robot Manipulator using Sparse Demonstration Data

Author:
Daniel Seidel

Supervisors:
apl. Prof. Dr. Jochen J. Steil
Dipl.-Math. Christian Emmerich

Research Institute for
Cognition and Robotics
—
Faculty of Technology
Bielefeld University

January 16, 2014

Abstract

The ability to plan and execute of movements to accomplish tasks is a fundamental requirement for all types of robot, whether in industrial or in research applications. This Master Thesis addresses path planning for redundant robot platforms. The research targets two major goals. The first is to bypass the need for an *explicit representation* of a robot's environment, which is strained with sophisticated computations as well as required expert knowledge. This bypass allows for a considerably more flexible use of a robot, being able to adapt its path planning data to an arbitrary new environment within minutes. The second goal is to provide a real-time capable path planning method, that utilizes the advantages of redundant robot platforms and handles the increased complexity of such systems. These goals are achieved by introducing *kinesthetic teaching* into path planning, which has already proven to be a successful improvement for single task methods dealing with redundancy resolution.

The thesis proposes an approach utilizing a topological neural network algorithm to construct an internal representation of a robot's workspace based on input data obtained from physical guidance of the robot by a user. In order to create feasible and safe movements, information from both *configuration space* of the robot and *task space* are employed. The algorithm is extended by heuristics to improve its results for the intended scenario. This modified network construction algorithm constructs a navigation graph similar to classical approaches with explicit modeling. It can be processed by means of conventional search algorithms from graph theory to generate paths between two arbitrary points in the workspace.

Contents

Chapter 1

Introduction

Path planning in robotics is the task to move a robot from an initial position to a given goal position. In mobile robotics this task is about finding a route in the environment along which the robot can travel. In the field of stationary robot manipulators one has to find a series of joint angles, which command the robot between the two endeffector positions [1]. This thesis deals with path planning for the latter robot type. By creating a series of joint angles, also called joint angle trajectory, one has to ensure a safe transition between individual positions. That is, the trajectory must not cause collisions with the environment or collisions of the robot with itself and hardware restrictions like joint limits have to be considered. Furthermore, in most cases one is also interested in fulfilling some quality criteria or constraints by minimizing a cost function. These are for example shortness of the path, distance to obstacles or low forces during movement. Once the criteria are defined for a given start and goal position, a path is calculated with respect to them.

Path planning and single task position control become especially challenging when working with redundant robots. They provide great advantages in flexibility and maneuverability, for example allowing robots to operate in cluttered environments by greatly boosting obstacle avoidance capabilities. For this reason, redundant robots have been the target of research for more then two decades, as for example Siciliano stated in 1990 that "The scientific and technological perspectives of robotics can be greatly enhanced by considering redundancy which has been recognized as offering greater flexibility and versatility in today's robot manipulators" [2]. However, these possibilities imply increased complexity in controlling these types of robot. Where for conventional robots the mapping between task space and configuration space is – with few exceptions – analytically solvable and distinct, redundant robot arms provide for every task space position an infinite number of choices for the redundancy resolution. This allows to have different configurations for a specific task to choose from, but requires new methods that are capable of calculating and choosing appropriate redundancy resolutions. In general, this is known as the task to calculate an inverse kinematics model, i.e., the ability to calculate a joint configuration for a given task space position.

In the research field of path planning such redundant robot manipulators offer great opportunities as well. They are able to approach the same place from different directions and in different ways, but depend on environmental or task specific constraints in order to operate well in such confined workspaces. This of course has a strong influence on the way path planning has to be done, since one has to choose an appropriate path from many to reach a certain goal position. This complexity of tasks and robots in real-world conditions places strict requirements on robustness, correctness and speed (real-time capability) of the algorithms employed to ensure safe usage of the robots.

1.1 Motivation

In many applications and scenarios path planning is done using a *sampling-based* approach [1]. The key idea in this type of planning is to determine whether single configurations cause collisions using a collision-detection system. A planner uses this primitive to sample different configurations to form a collision-free path. To achieve this, explicit modeling of constraints is required, which necessitates simulation software and detailed information about the environment and the robot. Information has to be transferred into a suitable internal representation like geometric models of the obstacles and the robot. For example, Behnisch *et al.* use a sampling-based random tree approach to search in the task space where distance based potential functions are calculated using a full simulation environment [3]. Using a suitable internal representation, a path is generated by means of exhaustive or heuristic search. The advantages of this approach are its high accuracy and the ability to simulate further details. Once the environment can be simulated, distance information and other geometric properties for objects in the scene can be obtained. However, this approach is costly in terms of computation time and detailed expert knowledge is required. The environment has to be known completely before the robot begins to move. For every change in the environment the model has to be adapted accordingly, which requires trained professionals with the respective expertise.

An approach that does not rely on explicit modeling and exact calculations is the data-driven approach. Its premise is to remove the dependency of explicit representations by acquiring movement data and extracting the desired information. This thesis is based on a research project where such a data-driven approach is used to generate single task movement commands for a redundant manipulator. These are easily adaptable to dynamic environments and do not require specialized knowledge about the problem. The project uses a technique called *kinesthetic teaching* to learn environ-

mental constraints and to specify task space trajectories. However, the system lacks the ability to automatically generate trajectories for given start and goal positions wherefore physical interaction is required. Despite being very intuitive, this limits the system to static tasks only. In situations where dynamic task generation is needed or physical interaction is not possible or not desired, automated path planning is required. Therefore,

> *the overarching goal of this thesis is to integrate an autonomous path planning method into the research project's framework, that utilizes the intuitive kinesthetic teaching interface and does not require explicit modeling of information.*

To achieve this, the target of this work is to develop a system that is able to accomplish the following major tasks:

(I) Process kinesthetic teaching input data and extract information to create an internal representation of the workspace.

(II) Ensure correctness of the representation in context of collision-freeness.

(III) Autonomously create trajectories for given start and goal positions.

(IV) Generate motion commands to control the robot along the trajectories.

1.2 Outline

Chapter 2 describes the context of the thesis. This includes the research project in general, which is called FlexIRob (2.1), a description of the kinesthetic teaching mechanism (2.2) and the learning based control algorithm (2.3).

Chapter 3 provides an overview of related research in path planning and outlines the differences to the approach presented in this thesis. The requirements for the intended work are specified along with an overview of the approach described throughout the thesis.

Chapter 4 introduces the algorithm used for processing the input data. The basic algorithm and implementation details are explained (4.1), followed by modifications that have been applied to optimize the results for navigation (4.2 and 4.3).

In Chapter 5 the eventual trajectory generation based on the results of the previous chapter is shown.

Chapter 6 investigates the applicability and stability of the path planning method based on evaluation of recorded training data from users with different levels of experience. The evaluation setup and means of data acquisition are described (6.1).

Afterwards, Sect. 6.2 and 6.3 present the results regarding different demands to the method.

Chapter 7 demonstrates the extension of the method to learn training samples in an inhomogeneous feature space to test the scalability of the system.

Chapter 8 shows a simple modification for the data acquisition phase that aims to improve the usefulness of the training data by giving haptic feedback to the user during training.

Chapter 9 summarizes the work of the thesis and discusses current limitations, as well as upcoming research questions and possible future work.

Chapter 2

The Human-Robot Co-Worker Scenario

The scenario of this thesis is located in the context of the FlexIRob system. To provide the context of this thesis, a description of the system in general is given, followed by individual components, that are important for the understanding of this thesis's methods. The whole system is thoroughly described in [4] and [5], from which the following descriptions are summarized.

2.1 FlexIRob System

FlexIRob – Flexible Interactive Robot – is a showcase robotic system used in research at Bielefeld University's Cognition and Robotics Lab. The aim is to improve human-robot interaction in a way that efficient and safe co-working of humans and robots becomes possible (Fig. 2.1). The projects utilizes machine learning algorithms and interaction technology in combination with a compliant robot platform to realize intuitive human-robot interaction.

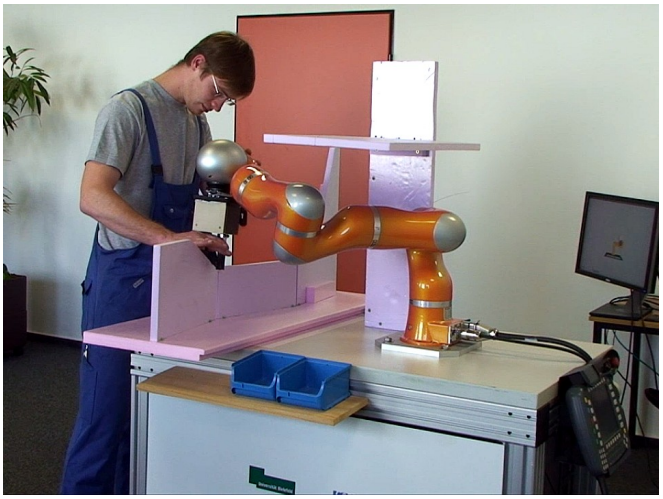


Fig. 2.1: A company worker using the FlexIRob system in a user study [4].

The system consists of a recent version of the KUKA Lightweight Robot (LWR IV) [6], which is a redundant robot manipulator with seven degrees of freedom (DoF). As described earlier, configuration and controlling of such redundant robots is a challenging task. The FlexIRob system abstracts this configuration effort to a level that allows robotics experts and lay people alike to reconfigure the robot to a new environment or a new task. This is accomplished by having the user directly teach the robot information about the environment and motions required for a task via kinesthetic teaching rather than by regular means, such as direct programming or using a computer device as the interaction interface. The gathered data from kinesthetic teaching is then processed to produce movement commands for the arm [7].

2.2 Kinesthetic Teaching

The kinesthetic teaching process of the FlexIRob system is divided into two separate but consecutive interaction phases: the configuration phase and the programming phase. These are shortly described to give an overview of the whole system. A complete description explaining all components in detail is found in [5].

In the first phase the robot is in a gravity compensation mode that allows users to freely move joints and to position the robot. The human teacher can move the robot to a desired Cartesian position with a desired joint configuration. Thereon the system switches into a recording mode with increased joint stiffnesses in which data pairs of corresponding endeffector locations and joint angles are recorded (Fig. 2.2). In this

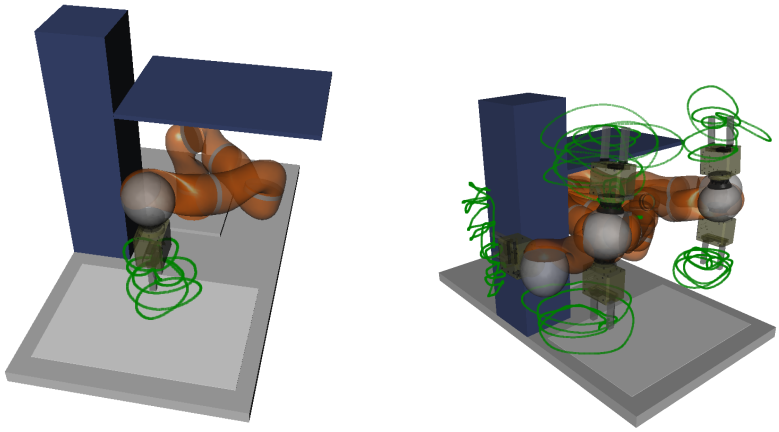


Fig. 2.2: Kinesthetic teaching with multiple training areas and different redundancy resolutions [5].

mode the movements are limited to a small area so that the teacher cannot change the joint configuration significantly. Afterwards, the robot switches back to gravity compensation mode and the user can move the robot to another area. The training is finished if data has been recorded in all areas of the workspace required for the intended task. In this way the constraints of the environment are implicitly modeled by the user within the training data. The constraint knowledge is then extracted through application of a machine learning algorithm to gain an inverse kinematics mapping.

In the subsequent programming phase, the robot uses the so called *assisted gravity compensation* mode [5]. The learned inverse kinematic is used to assist the teacher in defining a task space trajectory, which is called a *task*. The user only has to move the endeffector and thereby specify the pathway of the trajectory. The system maps these positions to joint angles and controls the robot accordingly. However, the programming phase is not important in context of this thesis, because the goal is to provide an autonomous way of generating trajectories. The work of this thesis provides an alternative to the programming phase (Fig. 2.3). The configuration phase remains unaffected.

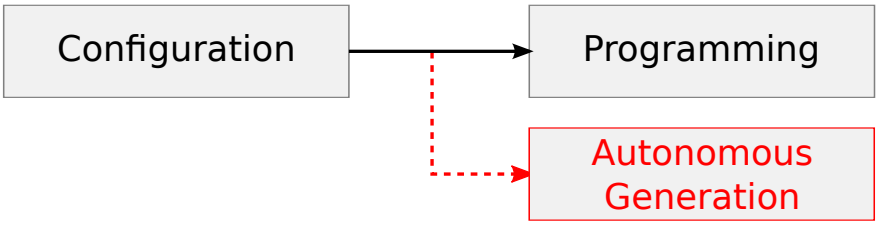


Fig. 2.3: Process for defining a task.

2.3 Learning of Inverse Kinematics

During the configuration phase the endeffector positions and the associated joint angles are recorded for supervised training of an artificial neural network [5]. The algorithm used is an extreme learning machine (ELM), which is a random-projection based single hidden layer feed-forward neural network with efficient read-out learning [8]. The implementation provides a static mapping from endeffector positions $x \in \mathbb{R}^3$ or $\mathbb{R}^6$ in task space to joint angles $q \in \mathbb{R}^d$, where d=7 applies for this scenario. For each position in the workspace the neural network learns a specific solution of the inverse kinematics from the training data. While different redundancy resolutions in different parts of the

workspace are possible, only one solution per task space position is provided by the mapping. The ELM and the mapping it represents are defined by input and output dimensions themselves without any knowledge of the configuration of the robot or kinematic properties.

Once learning is finished, the trained network is embedded into a hybrid controller as illustrated in Fig. 2.4. This allows to control the robot using task space positions only. The ELM provides a redundancy resolution q_c as constraint for an analytic controller, called "CBF Hierarchical Controller" in the illustration. This controller tries to satisfy the constraint as good as possible with the primary goal to accurately approach the target position using a feedback control loop. This control mechanism is used in Chapter 5 to move the robot along generated trajectories.

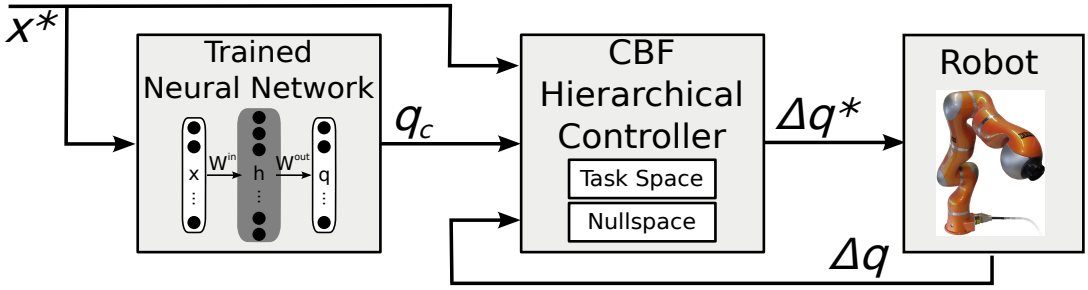


Fig. 2.4: Hierarchical control scheme of the FlexIRob system [5].

Chapter 3

Application and Research of Path Planning

The task of planning motion trajectories for robot manipulators has been subject of research for several decades. With robots receiving more and more attention in a multitude of applications, their use in industry and research has become indispensable. With the growing number of applications also the complexity of robotic systems and their respective control mechanisms increased.

This chapter will provide an overview of existing path planning approaches and point out the differences to the method proposed in this thesis.

3.1 General Path Planning Approaches

The general approach to path planning expects a complete description of the geometry of a robot and environment populated with obstacles [9, 1]. A robot possesses a state, which is defined by a point q in the so-called *configuration space* $\mathcal{C}$ [10]. In case of an articulated robot manipulator, such a point q consists of a list of joint angles $(\theta_1, \theta_2, \dots)$ and joint translations $(d_1, d_2, \dots)$, depending on whether rotatory and prismatic joints are present. A configuration q is characterized as *free*, if a robot in this state neither collides with an obstacle nor with itself. The set of all free configurations forms the *free space* $\mathcal{C}_{free}$ and the *obstacle space* is defined as its complement with $\mathcal{C}_{obs} = \mathcal{C} \setminus \mathcal{C}_{free}$. Path planning is then defined as the task to move a point q within $\mathcal{C}_{free}$ from an initial position q_{init} to a goal position q_{goal} [1].

Early approaches in path planning followed mostly the same principle: discretize the configuration space and construct a connectivity graph in it that allows to find a path using graph search algorithms.

These early approaches can be represented by three distinct algorithms: *roadmaps*, *cell decomposition* and *potential fields* [11]. Roadmaps capture the connectivity of $\mathcal{C}_{free}$ in 1D curves along which the robot is allowed to move. By completely exploring the space a roadmap guarantees to find a collision-free path if and only if it exists. In cell decomposition algorithms the free space is subdivided into simple – usually convex –

regions called cells. Cell connectivity is captured the same as in the *roadmap* algorithm. A path query locates the two cells containing q_{init} and q_{goal} and looks for a path of cells connecting them. The path of cells forms a channel of free space between q_{init} and q_{goal}. Finally, potential fields do not explicitly construct a connectivity graph, but use a potential function over the free space. The robot is guided towards the goal using an attractive component and repelled from obstacles using a repulsive component. These three approaches differ in the way the graph is constructed and represented. Nonetheless, they share the fact that explicit representations of the obstacle boundaries in the configuration space are calculated. However, this complete representation is computationally expensive and scales exponentially with the number of degrees of freedom [12].

In need of ways to control more complex systems with higher degrees of freedom, the group of *sampling-based motion planning* algorithms has emerged. The idea is to avoid the complete construction of the configuration space or the obstacle boundaries, respectively, by sampling only a limited amount of information required for the task at hand. This requires only information about whether a configuration lies in $\mathcal{C}_{free}$. Modern collision detection algorithms provide exactly this information, once a complete description of the environment is available. Hence, sampling-based motion planners are defined as those whose only information about $\mathcal{C}_{obs}$ is obtained by sampling the $\mathcal{C}$-space using a collision detector [12].

Typical classification of sampling-based planners uses two categories: multi-query and single-query approaches. A popular example for the multi-query approach is the *Probabilistic Roadmap Method*, developed in a large number of variations [13]. It is proven to be very efficient and applicable for a wide range of motion planning problems. Planning is divided into two stages. The first stage is the pre-computation of a roadmap that reproduces the connectivity of $\mathcal{C}_{free}$. Using the collision detector the configuration space is sampled according to a suitable probability distribution. All free configurations are stored as nodes, or *milestones*, in the roadmap. Milestones are connected if a straight-line path completely located in $\mathcal{C}_{free}$ exists between them. In the second stage the roadmap is processed for path queries to connect q_{init} with q_{goal} [1].

The single-query approaches avoids the need for a computationally expensive pre-computation of the roadmap. Instead, the roadmap is constructed on-the-fly during a path query for a specific q_{init} with q_{goal}. The motivation is to access the configuration space only as much as is required for a task instead of constructing a complete roadmap. A frequently used technique for single-query approaches is the *Rapidly-Exploring Random Tree* (RRT) algorithm by Lavalle [14]. In this approach usually two roadmaps or

trees are initialized with their root at q_{init} and q_{goal}, respectively. The two trees are expanded by sampling $\mathcal{C}$ at random and inserting free configurations as nodes into the trees. A path is found as soon as the trees meet each other, i.e., a node of one tree is connected to a node of the other.

These sampling-based approaches are capable of efficiently solving a large number of complex problem statements in high-dimensional configuration spaces. However, in terms of computation time they still suffer from exponential scaling with the number of degrees of freedom [1]. In view of vastly increasing numbers of degrees of freedom, e.g. in complex robot hands or full-body humanoid robot platforms, other approaches with better scaling are necessary.

3.2 Alternative Approaches

In the recent years, research started to focus on alternative ways to improve the performance of path planning algorithms, e.g. by further abstracting the problem representation or extending the search to multiple spaces at the same time instead of using only the configuration space.

Diankov *et al.* [15] introduced their *BiSpace Planning* algorithm, which improves the planning performance for complex high-dimensional problems by simultaneously exploring multiple spaces. Two separate RRT searches are used and combined: one tree is constructed in the configuration space rooted at the current (initial) configuration. This tree explores the full configuration space to guarantee the fundamental requirements for a free path, i.e., feasibility, executability and collision-freeness. The second tree explores the task space backwards from the goal position and acts as an adaptive, well informed heuristic. A one-directional mapping from configuration space to task space is used to connect the forward tree with the backward tree. Afterwards, the configuration space tree attempts to follow the task space tree to the goal.

Behnisch *et al.* [3] reduce path planning efforts by shifting the planning problem to a high-level representation. Their target is to reactively and locally avoid obstacles. This is a hybrid approach in the sense that the search is divided into two steps: planning and local obstacle avoidance. For planning, a classic sampling-based random tree algorithm is used. But instead of sampling in configuration space, their approach constructs the tree in task space with corresponding configuration space positions. The reactive local planner adjusts the configuration space trajectories using distance-based potential functions to avoid obstacles, which leaves the task space trajectories unaffected. By reducing the dimension of the search space from the higher-dimensional

configuration space to the lower-dimensional task space they were able to effectively reduce computation time. However, they also limited the general-purpose capabilities of their system, because it is not applicable for arbitrary problem statements.

3.3 Data-driven Approach

The approach presented in this thesis shares a number of common features with the path planning approaches described above – yet possess fundamental differences as well. It makes use of a graph structure to represent the information required to search for paths. This graph is constructed in the task space, just as the methods described by Diankov *et al.* and Behnisch *et al.*, but will also include configuration space information during the creation process. Therefore, it can also be considered as a hybrid approach.

However, *all* of the presented approaches have in common, that they rely on explicitly modeled information about the robot and obstacles. Collision detectors and similar tools are used in order to categorize configurations as free. The approach of this thesis completely bypasses this requirement by constructing the graph structure using real-word demonstration data obtained from kinesthetic teaching.

The overall approach works the following way: the input data provides information about *free* locations in the workspace. Free is defined the same as for sampling-based methods. This information is processed to create a topological network of nodes representing these free locations. The nodes are connected by edges, which show valid transitions between nodes. An edge is considered *valid* if every location assumed by the robot during the transition is also *free*. The creation process of the network is described in Chapter 4. Afterwards, Chapter 5 describes, how such a network is used to search for paths and to generate motions.

The topological network, more specifically the nodes of the network, are defined by the input data. That is, the input data specifies the *navigation space* in which the network lives. In this chapter, the navigation space is the three-dimensional Cartesian space. A data point consists of the coordinates of the endeffector location, which is further referred to as the *translation*. However, using this navigation space is not a limitation of the algorithm and other potential spaces exists. Chapter 7 will describe the application of the method to a navigation space consisting of the *translation* and *orientation* of the endeffector. But also other sorts of spaces may be applicable, for example the joint space of the robot.

3.3.1 Criteria for Method Selection

The appropriate choice of an existing algorithm or the design of a new one to solve a specific problem is in almost every case a crucial step towards the success of a project. Although the algorithm for the graph generation in this thesis was chosen beforehand, this section will discuss the reasons for its selection and advantages/disadvantages of the algorithm.

In context of the kinesthetic teaching paradigm used to gather input data, special requirements are placed for an appropriate algorithm. The demonstration data possesses several characteristics that are important for the algorithm selection and that impose constraints on the processing. These are in detail:

Incremental Learning: The learning method has to be able to incrementally add data to the network. The network has to be build up iteratively as soon as a data sample becomes available.

Anytime evaluation: Using the graph for path planning shall be possible at any time – within the range of the provided data – without the need for any extensive global processing. The adaptation of the graph to the training data should be as fast as possible, because adding input data and using the network for path planning may be done in turns. This also excludes the repeated use of (possibly randomized) input data until the algorithm converges into a stable state, i.e. a local minimum of an error function. Therefore, it is desirable to extract as much information as possible from one training iteration of an input point.

Continuous data: The amount of data to be processed is unknown. The input has to be considered as a continuous data stream with no specific end of training. This requires the algorithm to be capable of online learning.

Unknown topology: Prior knowledge about the topology is not available. The algorithm has to be able to adapt to arbitrary topologies. The only available information is that the data is clustered into distinct clouds, which are connected mostly by single paths.

Sparse and redundant input: Data from kinesthetic teaching is usually sparse compared to generated data, e.g. from a random distribution, and it contains much redundancy. The FlexIRob training framework forces the user to fixate the endeffector to a steady position for several moments to switch modes of operation, which produces a series of repeated data points. Due to the high data sampling rate of the framework of 100 Hz the difference of two consecutive input points is usually very small, so that a subsampling has to be used to filter irrelevant data points. Users also tend to move

the endeffector in circular trajectories, which produces clusters of data points.

Correlated stimuli: The input stimuli are generated from real-world movements of the robot and are therefore highly correlated. Randomizing or shuffling of the input data is neither possible nor desired.

The Instantaneous Topological Map (ITM) algorithm is well suited to handle these requirements and characteristics. It is a topological neural network first introduced by Jockusch & Ritter in 1999 [16]. The ITM has been specially designed in need of an algorithm capable of incremental learning of correlated input stimuli, as produced by exploration movements in robotics. The algorithm is based on the Growing Neural Gas algorithm (GNG) by Fritzke [17]. The GNG's main feature is the ability to learn practically any topology. This is possible due to its adaptive number of nodes, in contrast to for example the classical Self-organizing Map described by Kohonen [18] with a predefined topology and fixed number of nodes. However, the GNG does not perform well when confronted with correlated input stimuli. This has been proven by Jockusch & Ritter, who state that GNGs, if confronted with correlated stimuli, "face severe degradation, which excludes their use for many on-line applications" [16].

In summary, the ITM fulfills most of the stated requirements. It is capable of learning correlated stimuli. Learning is done only locally, which allows to add nodes at any location without influencing the rest of the network. The algorithm does not need to iteratively converge to correctly represent the input data, but produces reliable results as soon as sufficient data has been provided. The ITM is by construction an online learning algorithm. It can learn arbitrary topologies and no prior knowledge is required for initialization. With these features, the algorithm is very qualified for the intended purpose and therefore has been chosen as the network generation method.

Chapter 4

Learning the Free Workspace in a Topological Map

The main element of the path planning mechanism proposed in this thesis is a topological network, which contains the information about accessible and restricted areas of the workspace. This chapter will explain in detail the network generation algorithm, which is based on the Instantaneous Topological Map. At first the implementation and configuration of the basic ITM is described. The algorithm had to be extended to resolve disadvantages that surfaced when used in combination with input data generated from kinesthetic teaching. These extensions are motivated and described in Sect. 4.2 and Sect. 4.3.

4.1 Instantaneous Topological Map

The basic ITM algorithm works on a set of nodes i, represented by a corresponding weight vector w_i. The nodes are connected by a set of undirected edges, which define a local neighborhood $\mathcal{N}(i)$ for each node i. The algorithm starts with two connected nodes, for example the first two input stimuli. After that, for each stimulus ξ a set of four rules is applied:

1. Matching: Find the nearest node n and the second-nearest node s of the input ξ according to a given distance metric $D(a, b)$, e.g. the Euclidean distance.

$$n = \arg\min_{i} D(\xi, w_i)$$
$$s = \arg\min_{j, j \neq n} D(\xi, w_j)$$

2. Reference vector adaptation: Move the nearest node n towards the input ξ.

$$\Delta w_n = \eta \cdot (\xi - w_n)$$

3. Edge update:
   (i) Create a new edge (n, s) connecting the nearest and second-nearest nodes if they are not connected yet.
   (ii) For each neighbor c of n: if w_s lies within the Thales sphere defined by w_n and w_c, delete edge (n, c) due to redundancy. It is replaced by the newly created edge (n, s).

$$\forall\, c \in \mathcal{N}(n)\text{: if } (w_n - w_s) \cdot (w_c - w_s) < 0 \text{ then remove edge } (n, c)$$

   (iii) Delete unconnected nodes.
4. Node update:
   (i) If the stimulus ξ lies outside of the Thales sphere defined by w_n and w_s and its distance to w_n is greater than e_{max}:

$$(w_n - \xi) \cdot (w_s - \xi) > 0 \text{ and } D(\xi, w_n) > e_{max}$$

   - Create new node r with stimulus $w_r = \xi$.
   - Connect r and n.

   (ii) If the distance between w_n and w_s is closer than $\frac{1}{2}e_{max}$, remove s.

The creation and deletion of edges and nodes is illustrated in Fig. 4.1.

The algorithm also possesses a range of positive properties. The ITM is not subject to the fovea effect due to its creation rules. The fovea effect describes the situation, in which the node density in a network is higher in areas where more stimuli occur. For the ITM, new nodes are only created when a local area does not have a representative node. This allows it to handle training data with non-uniform stimulus density.

In terms of computation time the ITM is also very efficient. The only step directly dependent on the number of nodes is the matching step, whereas the costs for edge adaptation depend on the neurons' average number of connections.

4.1.1 Implementation

The ITM has been implemented as specified in the preceding section. The configuration of the ITM network uses two parameters that define the outcome of the algorithm and that have to be specified. These parameters are the maximum quantization error e_{max}

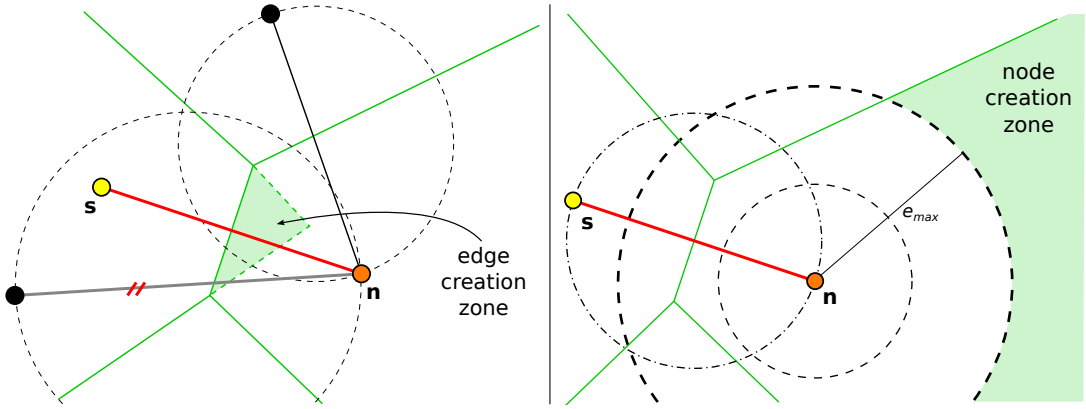


Fig. 4.1: Edge update (left): Edges are added when a stimulus lies within the region where the Voronoi cell of n intersects the Voronoi cell of s if n were not present. Edges are removed when s is inside the Thales sphere defined by n and one of its neighbors. The edge to the neighbor is then removed. Node update (right): New nodes are created when the distance between a stimulus ξ and the nearest node n is too large and the Thales sphere defined by n and s does not cover that region (based on [16]).

and the adaptation rate η. In this case, the Euclidean metric is used as distance function D for the nearest neighbor search.

The adaptation rate η is the ITM's equivalent to the learning rate of other topological networks. Whereas a learning rate is in other algorithms essential to adjust a network to the input data, Jockusch describes η as a smoothing parameter, that slowly arranges the nodes to have approximately uniform distances. According to Jockusch's experimental verification, this step may even be omitted. Due to its frequent node creation and deletion mechanism the network is still able to produce a uniform distribution of nodes. In fact, in context of this thesis, using the reference vector adaptation is even proven to be counterproductive. First, the clear goal is to explore the space as good as possible, as the nodes of the graph will be used as direct input for navigation. The adaptation however causes distortion of the nodes. Their location in the navigation space changes. This means that it is no longer guaranteed that they are *free*.

The second reason lies in the nature of the input data. The primary motivation during training is to provide input data for the ELM. For this purpose, circular movements produce useful training data. In contrast, if training the ITM with reference vector adaptation, these repeated orbital movements cause the nodes to be "attracted" to the center of a cluster, thereby reducing the overall explored area significantly. Fig. 4.2

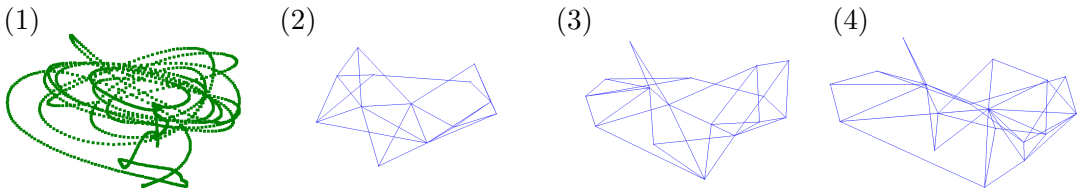


Image	η	Nodes	Edges	max ν	Ø ν	min L_e	max L_e	Ø L_e
2	0.10	12	26	7	4.17	0.084	0.186	0.109
3	0.03	17	39	7	4.47	0.0895	0.232	0.120
	0.02	18	43	12	4.67	0.0773	0.233	0.121
	0.01	18	44	12	4.78	0.0624	0.246	0.124
4	0.00	18	44	12	4.78	0.0557	0.256	0.127

Fig. 4.2: ITM training with different reference vector adaptation rates: (1) depicts the input data and (2), (3) and (4) the results for different η values.

illustrates training results for different η values. The table at the bottom displays statistical data for the different networks, where ν denotes the valence of nodes and L_e the edge length. The edge length has been chosen as a measure of "network size", meaning that a high average edge length indicates, that the space occupied by the network is also high. As the results show, with decreasing η the maximum and average valence increase along with the maximum and average edge length. If nodes are moved closer due to reference vector adaptation, then fewer nodes and edges are created or existing ones are removed. Visual evaluation of the resulting networks confirms these assumptions. In picture (2) the network is visibly "smaller" than the volume occupied by the training data and its shape is more convex. These effects are reduced in picture (3) and further in picture (4), where the shape of the network is very close to the training data. As a result, the reference vector adaptation was completely omitted in this thesis.

The second parameter, the maximum quantization error e_{max}, determines the resolution or the density of the resulting network. The choice for this parameter highly depends on the input data and the intended use case. Fig. 4.3 demonstrates the training of an ITM with different e_{max} values using a snippet of a kinesthetic teaching session.

The far left picture (1) shows the training snippet. Picture (2) shows the result for e_{max}=0.01. The ITM is constructed along the trajectory of the input stimulus with

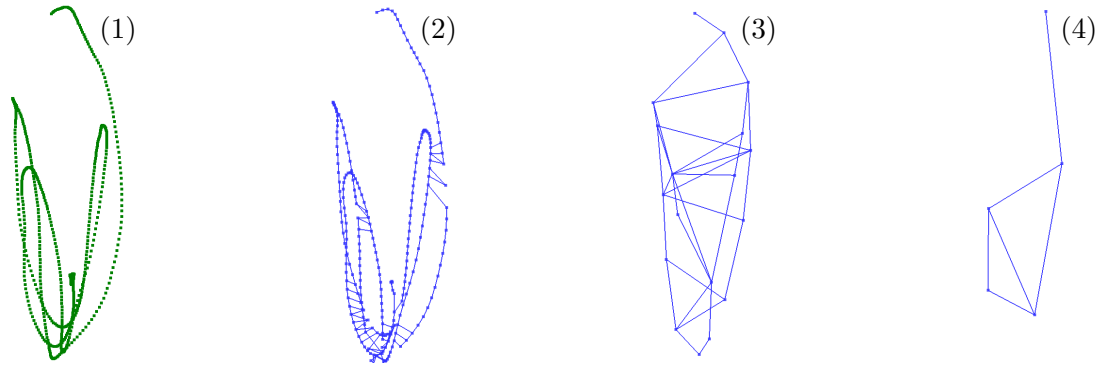


Fig. 4.3: ITM network generated from a training data snippet using different e_{max} values. From left to right: 0.01, 0.10, 0.20.

very few connections between the circular lines. This can be described as overfitting. The network learns the detailed topology of the input data, while the actual goal is to learn the space in which the stimuli occur. The third picture shows a network that is well connected without too many unnecessary nodes. Picture (4) depicts a result in stark contrast to the first one. The value of e_{max} is too big for the network to create enough nodes to sufficiently learn the input area and therefore results in an underfitting behavior. A deeper analysis for finding an e_{max} value fitting to the scenario is provided in Chapter 6.

Fig. 4.4 shows a first result when applying the basic algorithm without reference vector adaptation and e_{max}=0.10. The input data consists of about 15 training areas in a scenario with two simple cubes as obstacles. The corresponding network is shaped similar to the input data and all training areas are reachable by traversing along the edges inside the network. But there are also some holes in the network, where not enough data was available. Also, for example at the very bottom, some training areas are poorly connected to the rest of the network. These insufficiencies are addressed later in this chapter.

4.2 Joint Space Validation

Using the basic implementation as described in the previous section enables the system to learn a network in the input space which can be used for navigation purposes. This is sufficient in cases where the robot performs only simple movements during training. But in context of a redundant robot manipulator more complex movements have to be considered. The problem is, that pairs of Cartesian positions may occur, which

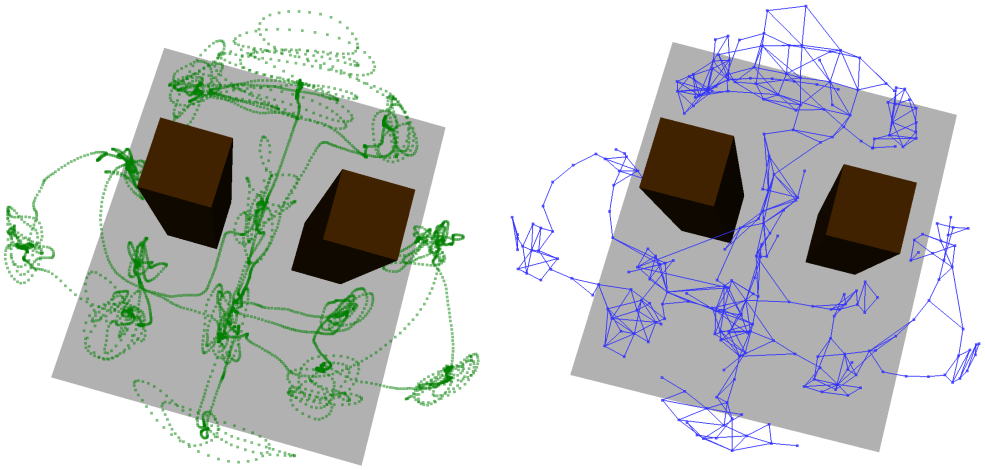


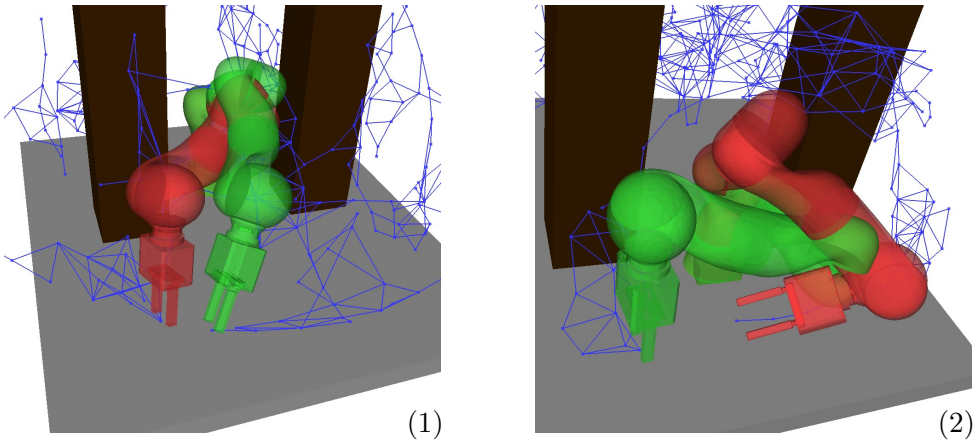
Fig. 4.4: Visualization of a kinesthetic teaching session showing the input data (left) and a corresponding generated ITM network (right).

the ITM considers as nearest neighbors due to the Euclidean distance measurement, but which correspond to large movements of the robot. That is, the postures differ significantly. A posture is here defined as the joint angle configuration of the robot. Fig. 4.5 depicts these two situations, where a simple movement is made and the two positions are compatible (1) and therefore *valid*. In contrast, picture (2) shows a complex movement for which the positions are not compatible (2), therefore *invalid*.

The reason for this problem occurring in the first place when using an ITM is that the network only covers the Cartesian space. However, in this scenario the Cartesian space is strongly connected to the joint space of the robot. To be able to produce a valid navigation graph even for complex movements, which exploit the full potential of the redundancy, the joint space has to be integrated into the ITM network generation.

This task is solved using a joint space validation for potential edges in Cartesian space. The detection of such invalid connections is done by comparing the postures in which the robot approaches the positions. For this, the distance D_θ between the joint angle vectors is calculated using the Euclidean metric. These vectors consist of the angular positions of the motors given in radians.

There are two major cases in which invalid connections may occur. The first is the classic elbow up and elbow down situation, which is also present for non-redundant manipulators. This situation can be characterized by the fact, that a transition between the two postures is not possible – while at the same time maintaining the endeffector



(1) (2)

Fig. 4.5: *Valid* and *invalid* connections between nodes: (1) The two postures are very similar and a transition between them is possible. (2) The postures originate from different redundancy resolutions (elbow up and elbow down) and a direct transition is not possible.

location (for the robot type used in this thesis). Furthermore, switching between elbow up and elbow down positions often results in movements through or close to singularities. The second case is when the robot changes the redundancy resolution while staying in the same spot or close to it. The difference to the first case is that a smooth transition is possible, while the elbow up to elbow down switch does not allow such a transition – without changing the endeffector position. In Fig. 4.6 the two cases are illustrated by example. While for case (1) no intermediate stages exist and for every occurrence edge creation must be prevented. Case (2) is different; changes in the redundancy resolution for near positions are not critical *per se* and appear very often. In most cases a smooth transition is possible, because the resolutions are still very much alike. But for those cases where postures are not similar enough, a fitting joint distance threshold $d_{\theta\,\max}$ has to be found, that separates them from valid connections. The modification in form of an ITM rule looks as follows:

3. Edge update **with validation**:

   (i) Create a new edge (n, s) connecting the nearest and second-nearest nodes if they are not connected yet **and if** $\boldsymbol{D_\theta(n, s) < d_{\theta\,\max}}$.

   (ii) For each neighbor c of n: if w_s lies within the Thales sphere defined by w_n and w_c, delete edge (n, c) due to redundancy. It is replaced by the newly

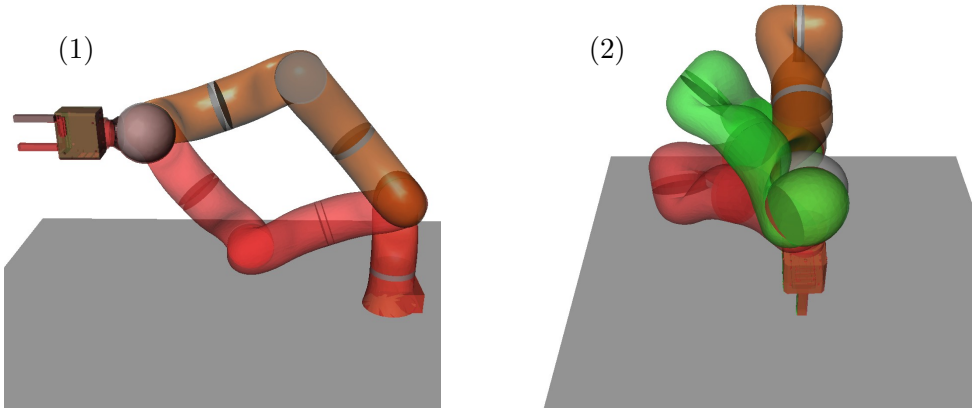


Fig. 4.6: (1) Elbow up and elbow down solution for the same endeffector position. (2) Initial position (orange) and two other possible redundancy resolutions with increasing differences.

created edge (n, s).

$\forall\, c \in \mathcal{N}(n)$: if $(w_n - w_s) \cdot (w_c - w_s) < 0$ then remove edge (n, c)

(iii) Delete unconnected nodes.

To be able to calculate D_θ for a pair of nodes, a set of joint angles has to be stored for each node. In this scenario, there are two possible sources of joint angles, which can be used. The first is to store the joint angles during training, in which the robot effectively approached the location of a node. The second is to use the mapping of the ELM instead. The major difference is that the recorded angles are accurate – in terms of closeness of the endeffector to the target location – while the ELM angles imply a generalization error. However, the recorded angles are only available for locations represented by an input sample and interpolation is not possible without further model knowledge (without losing accuracy). On the other hand, the ELM provides a continuous mapping between task space and joint space. Also, the ELM angles are eventually used for the motion generation. Both ways have been tested and are presented below.

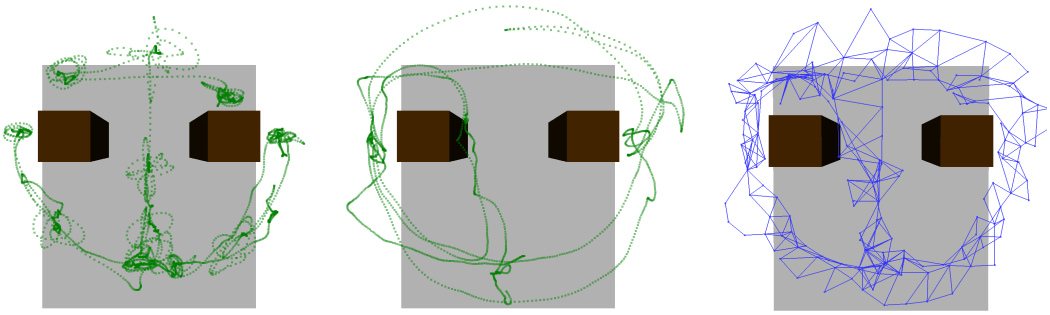
4.2.1 Validation Using ELM Mapping

The Cartesian position information is acquired and stored the moment the robot takes up a position. The joint angles of the ELM on the other hand are calculated later on

and are – in case of incremental learning of the ELM – subject to variations during the training. They are not equivalent to the joint angles recorded during training, but rather the representation of the mapping between Cartesian positions and joint angles produced by the current state of the ELM (as explained in Sect. 2.3). Using the ELM mapping is considered for the following reason: the redundancy resolution that is chosen for the robot when executing trajectories is controlled by the mapping of the ELM, not by the joint angles the robot takes on during data acquisition. Furthermore, the recorded joint angles may be inconsistent. This is possible because of the free movement mode during the training of the ELM. In this mode, which allows to move the robot from one training area to the other and to reposition the robot as needed, arbitrary movements are possible. For the ELM training this is irrelevant, since the data from this mode is not included while learning. However, the ITM will use this data to connect the training areas. If a user moves the robot between two training areas multiple times using different redundancy resolutions, this inconsistency may influence the ITM negatively. When using ELM-produced angles, these inconsistencies are automatically avoided. For the training areas, the assumption implied by the FlexIRob scenario is that only a single redundancy resolution per workspace region is taught. The possible inconsistencies during free movement do not influence the ELM mapping.

To investigate, if this approach produces reliable results, a training session is used in which potentially invalid connections are created by training different redundancy resolutions at locations close to each other. Fig. 4.7 (left) shows the training data. Close to each obstacle are two training areas, one approaching from each side of the obstacle. The pictures on the right show the recording of longer movements in gravity compensation mode in which the robot has been moved to all places of the workspace. During this movement the robot has intentionally been used with a configuration inverse to the corresponding other training area. At that moment, the regular ITM creates connections between the two cluster pairs. Moving along these edges would force the robot to move through the obstacles, because the differences of the two posture's joint angles is big. This situation has to be prevented at all costs using this criterion.

General properties of the networks can be examined by calculating edge costs in joint space, D_θ, when training the ITM for different e_{max} values. That is, the distance in the joint space given in radian. Such a distance can be interpreted as the total angular movement required to transfer the robot from one configuration into the other. Tab. 4.1



basic training (b) test movement (t) ITM for basic+test

Fig. 4.7: Training session used for evaluating the joint distance validation.

shows the corresponding network statistics. The values given are the maximum and average edge length L_e [m], the maximum and average joint space distance D_θ per edge $[\frac{\text{rad}}{\text{edge}}]$ and the average joint space distance per task space distance $[\frac{\text{rad}}{\text{m}}]$.

The upper four rows correspond to networks trained without the additional test movements. As to be expected, the average edge length L_e increases nearly proportionally with e_{max}. A more interesting discovery is made when observing the maximum and average joint space distances per edge. The average $D_\theta/edge$ behaves similar to the average edge length while the maximum values increase insignificantly. The joint space distances per meter – D_θ divided by the edge length L_e – remains nearly constant as well. This leads to the assumption, that a static threshold $d_{\theta\,\max}$ for D_θ is possible for the given range of e_{max} values. When comparing these results with networks trained with the additional test movements seen in the bottom four rows of Tab. 4.1, an overall increase in joint distance is observable in the fifths and sixths columns. The maximum distance of an edge as well as the average distances are larger in the bottom four rows.

The free movements are especially important for the ITM. Much of the movements are not directly in or at the border of the ELM training areas, but a bit farther away. The ITM requires these movements to connect the training areas. For the corresponding joint angles, the ELM extrapolates the data from the training areas. This of course introduces some generalization errors, which causes greater joint angle distances. In consequence, the validation must also be reliable for positions not near to ELM training areas, because the test movement are important for overall network generation. They are equivalent to the movements between training areas in the normal ELM training, which appear during the whole training session.

data	e_{max}	max L_e	Ø L_e	max D_θ/edge	Ø D_θ/edge	Ø D_θ/L_e
b	0.06	0.254	0.086	1.71	0.375	4.53
b	0.08	0.358	0.115	1.83	0.503	4.50
b	0.10	0.336	0.140	1.93	0.591	4.38
b	0.12	0.364	0.165	1.99	0.696	4.39
b+t	0.06	0.254	0.086	1.86	0.406	4.65
b+t	0.08	0.358	0.115	2.75	0.539	4.65
b+t	0.10	0.336	0.140	2.76	0.640	4.53
b+t	0.12	0.364	0.165	2.75	0.722	4.35

Tab. 4.1: Joint space movement D_θ evaluated using ELM-produced joint angles. The networks compared are trained with different e_{max} values and either the basic training (b) or in combination with the test movements (b+t).

The edges are examined by testing systematically different thresholds for D_θ and observing which edges are marked as invalid. The observation started with $d_{\theta\,\max}$=2.5 rad and stepped in decrements of 0.1 rad until no more connections between the opposing training areas were created. At about 1.5 rad the first edges were detected as invalid. From 1.0 rad onwards, regular edges were also detected as invalid. The first value for which undesirable connections between the training areas was completely eradicated was 0.5 rad, which is already below the average of 0.54 rad. Nearly all edges created during the test movement were also detected as invalid. Pictures demonstrating this validation are shown in Appendix A.1. According to these result, the trade-off between preventing invalid connections and removing valid ones is very high, which led to further investigation of the ELM output.

The reason for this misbehavior of the validation, where valid and invalid edges are likewise marked invalid, is the generalization behavior of the ELM. This is especially notable between areas with opposing joint configurations. At these positions, the ELM mediates between the configurations, which decreases the joint distances. In Fig. 4.8 (1) this behavior is demonstrated for a path between two such areas. The robots visualize the joint angles, to which the ELM maps the anchor points of the path. Apart from the huge position error for the endeffector, this clearly shows that there is no hard boundary (or boundary area) at which the configuration switches. The joint angles merge seamlessly between the two configurations. In picture (2) the same path is visualized with the real joint angles from training data. Here, the configuration directly switches between the first two points. In consequence, the use of the ELM mapping for validating edges is not reliable.

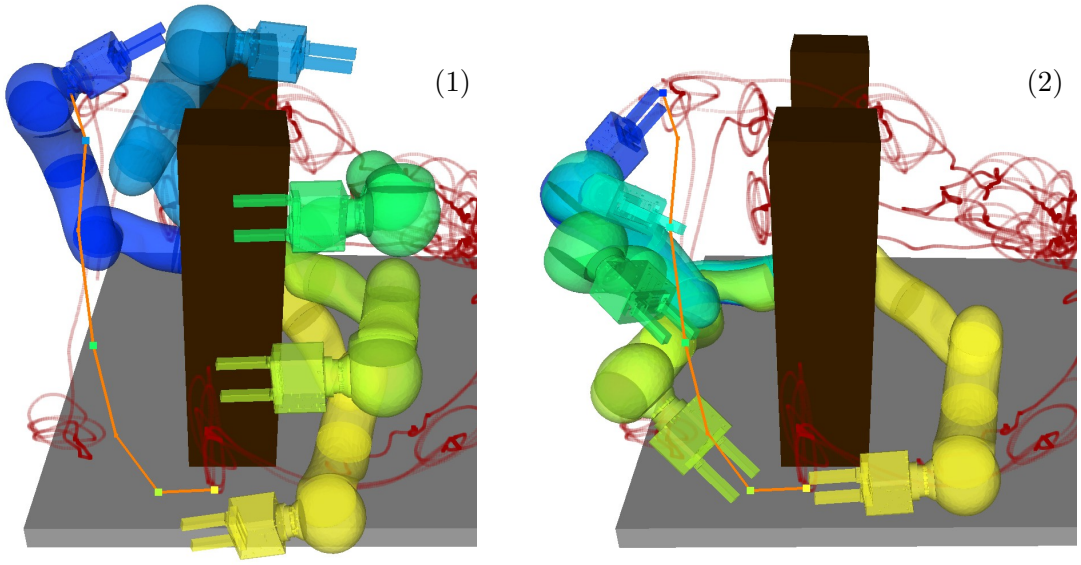


Fig. 4.8: Visualization of a path (orange) and the joint angles corresponding to the anchor points using the ELM mapping (1) and the joint angles recorded during training (2). Each depicted posture belongs to the anchor point with the same color. The training data is shown in red.

4.2.2 Validation Using Training Angles

The same procedure as with the ELM mapping is applied to test whether the joint space validation produces more reliable results using angles recorded during training. Tab. 4.2 shows the equivalent network statistics for the new set of joint angles. The general observation reveals that the joint spaces distances D_θ have significantly increased. While the maximum joint space distance per edge is nearly equal in all cases, the values for D_θ / L_e are quite different. They are not proportional to e_{max} in any way. The average distance per edge for all networks increased by 26 % when compared to ELM joint angles. This further supports the hypothesis that using the ELM angles is not suited for validation, because the mapping of the ELM is continuous in the whole workspace. This causes smoother transitions.

As in Sect. 4.2.1, calculated joint angle distances for the edges are used to find a threshold $d_{\theta\,\max}$, which prevents the creation of edges between areas with opposing joint configurations. Since the value range of D_θ for the ELM angles case and the recorded angles case is quite different from the ELM, the threshold values are not directly comparable. But in contrast to the prior evaluation, the use of recorded joint angles allows for finding a very suitable threshold. This means, that nearly no valid

data	e_{max}	max L_e	Ø L_e	max D_θ/edge	Ø D_θ/edge	Ø D_θ/L_e
b	0.06	0.250	0.080	2.50	0.390	4.69
b	0.08	0.275	0.105	2.61	0.553	4.97
b	0.10	0.258	0.125	2.51	0.683	5.26
b	0.12	0.312	0.147	2.50	0.759	5.07
b+t	0.06	0.254	0.086	3.92	0.616	6.99
b+t	0.08	0.358	0.115	3.90	0.765	6.55
b+t	0.10	0.336	0.140	3.83	0.902	6.36
b+t	0.12	0.364	0.165	3.85	0.978	5.92

Tab. 4.2: Joint space movement D_θ evaluated using recorded joint angles.

edges are marked as invalid and that connections between opposing training areas are always detected. For a threshold of 1.0 rad the result was similar to the ELM validation with 0.5 rad. Nearly all of the edges created during the additional test movements were incorrectly marked as invalid. Up to a threshold of $d_{\theta\,\max}$=2.5 rad this reduced drastically, where only a few edges were falsely detected as negative. At a value of 3.0 rad none of the regular edges were invalidated and all edges connecting the critical areas were successfully prevented. The result of the validation is shown exemplarily in Fig. 4.9, further pictures for different thresholds are given in Appendix A.1. The same values were confirmed using other sets of training data. As a result, the joint

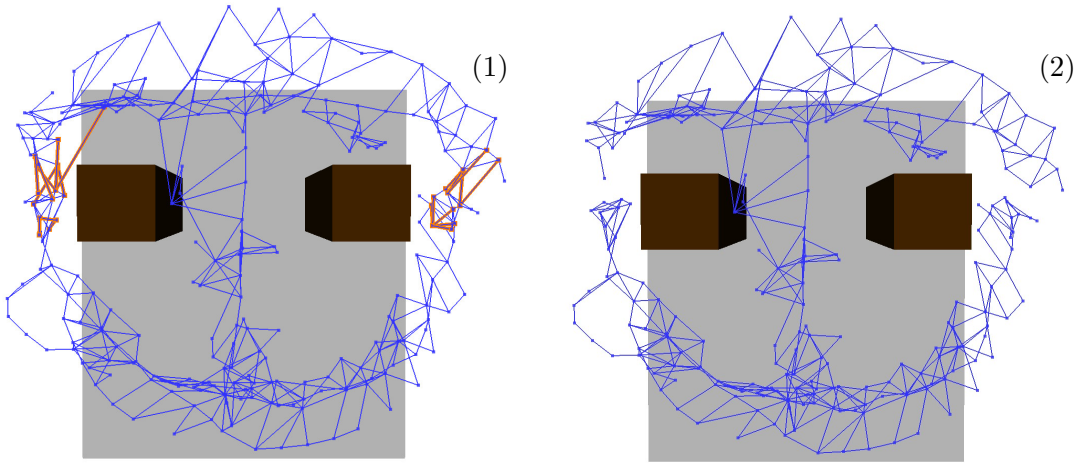


Fig. 4.9: Joint distance validation with a threshold of 3.0 rad. (1) shows a subsequent validation with invalid edges marked in orange and (2) depicts the network with validation directly integrated into the training phase.

distance criterion allows to reliably detect invalid edges when using joint angles directly recorded during training and proves to be stable for multiple test cases.

4.3 Network Optimization

Sparseness of the input data acquired from kinesthetic teaching proved to be a major obstacle for the ITM learning algorithm. When learning a plain ITM network as described in Sect. 4.1, the resulting network is allocated in several clusters with only sparse connections between them just as the robot has been moved between the training areas. This is depicted in Fig. 4.4. Such a network is not suited well for use as a navigation graph. The paths found by a shortest path algorithm may be unnecessary long and detoured in many cases due to holes in the network and the low interconnection of clusters.

A better suiting network may be obtained by enhancing the ITM algorithm to correct such deficits. Algorithms for detecting and resolving holes and other undesired properties in a graph are well known and understood in the research field of graph theory, e.g. the hole detection algorithm described by Nikolopoulos *et al.* [19]. Holes found by such an algorithm could then be closed by creating edges between the corresponding nodes. However, such an approach is computationally expensive and even more importantly, requires global processing on the graph. This violates the prerequisite of an online-training capable method. In order to satisfy this constraint, two heuristics that do not depend on further model knowledge to augment the network have been developed. The idea is to preserve the *natural structure* of the ITM by generating additional training samples from the set of acquired input data as it is processed. Applying postprocessing on top of the regular training would alter the network not according to the creation rules and may cause inconsistencies. The process of increasing the amount of training data by generating new samples from the available data is called bootstrapping (BS). As these heuristics are very specific to the scenario, their description and verification is done using actual training data instead of more general, artificially generated data.

4.3.1 δ-Bootstrapping for Local Connectivity

The target of the δ-bootstrapping is to close small holes in the network and increase local connectivity. A naïve approach is to apply noise to the training data to generate

new samples:

$$\xi_c = \begin{pmatrix} w_x \\ w_y \\ w_z \end{pmatrix} + \begin{pmatrix} \gamma_x \\ \gamma_y \\ \gamma_z \end{pmatrix}, \gamma_i \in \mathcal{N}(\mu_i, \sigma_i^2) \tag{4.1}$$

This entails several disadvantages: first and foremost, the addition of noise has to produce samples with a distance greater than e_{max} in order to have any effect. This raises concern for the maximum distance that may be applied by the noise. If chosen too high, samples within space occupied by obstacles could be produced. Since no model information like endeffector size is available, an appropriate selection for this parameter proves difficult.

An approach that utilizes information modeled *implicitly* by nodes' local neighborhoods is chosen to avoid this problem. This heuristic adds a fifth rule to the set of training rules, which applies additional training for every original input sample:

5. Sample additional data from neighborhood for each newly created node r (Fig. 4.10):

   (i) Construct the δ-neighborhood $\mathcal{N}_\delta(r)$ for $\delta > 1$:

   $\mathcal{N}_\delta(r)$: every node c connected to node r through at least 2 and at most δ edges.

   (ii) For each $c \in \mathcal{N}_\delta(r)$: train net with input stimulus ξ_c (omitting step 5):

$$\xi_c = w_c + \frac{1}{2} \cdot (w_r - w_c). \tag{4.2}$$

This also introduces a new parameter of neighborhood depth, δ. In contrast to the noise added to the x-, y and z-coordinates in Eq. (4.1), this parameter is much simpler to determine. It is an integer number with a very limited range. The minimum value is two, since a value of one would consider only neighbors directly connected to the node and therefore have no effect. The theoretical maximum value is defined by the longest path possible in the network. However, since this heuristic increases computation time exponentially as a function of δ (assuming an approximately uniform valence for all nodes), the practical range of δ values is very small.

Furthermore, the parameter does not scale with the range of values or number of components of the input data, but rather with its intrinsic dimension. As evaluated in [20], the intrinsic dimension of the input data manifests for the ITM in the average number of connections of a node. For example, a two-dimensional data set embedded

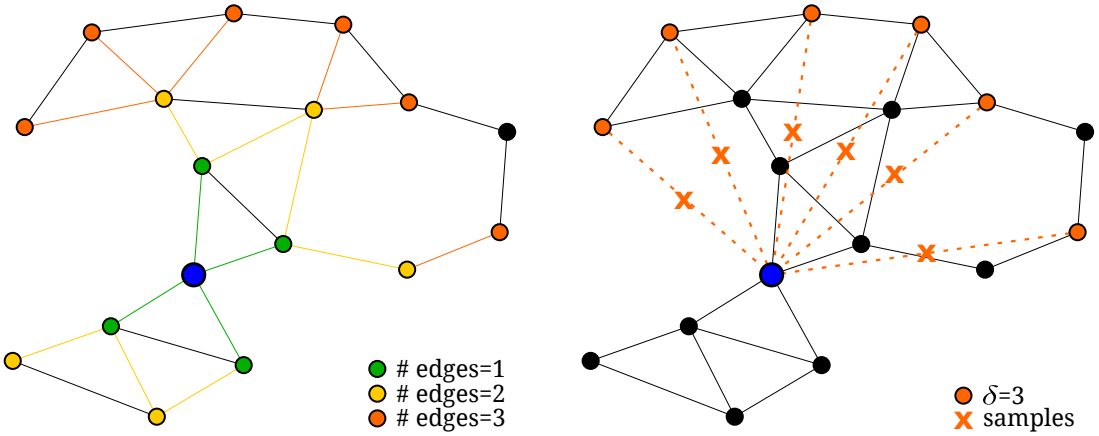


Fig. 4.10: Local neighborhood (left) and sample generation (right). For brevity, the picture only shows samples generated for δ=3 while the algorithm creates samples for *all* nodes included in # edges=2...n.

in a feature space of a higher dimension will produce four to six emanating edges per node on average, regardless of the feature vector size [17]. As a result, δ can be chosen relative to this average. A high average valence will cause the δ-neighborhood to grow faster with increasing δ than a low average valence. To compensate for this effect, a high average valence should be accompanied by a low δ value and vice versa.

In addition to exponentially increasing computation time, higher δ values will cause the network to converge to its convex hull. Fig. 4.11 demonstrates this behavior. The picture series shows new samples being generated in a concave network structure. Whereas for δ=2 the samples are very close to existing nodes, for δ>5 the samples create a convex structure. However, this behavior is limited by the magnitude of e_{max}. If chosen higher, partially concave networks with a depth smaller than this value will not be filled with nodes and therefore remain.

For this reason, a high δ value will produce nodes far away from explored areas, which may be harmful in most scenarios. For the same reason, the fifth rule is only applied for a newly created node r and not each iteration for the winning node n. Such an approach would result in repeated exploration of the same nodes every time a node is nearest to the input stimulus. Each time the δ-neighborhood of a node grows, it further increasing the impact of the δ-bootstrapping. The consequence is a similar convergence to the network's convex hull.

For this thesis' scenario both cases produce undesired behavior, expressed by nodes with potentially harmful proximity to obstacles or even collisions. To avoid this, δ must

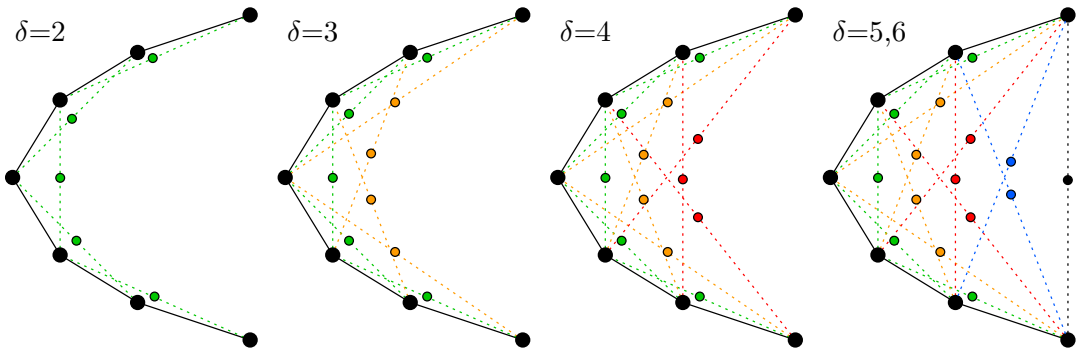


Fig. 4.11: Convergence of the network for increasing δ values: colored points depict the new samples for each δ.

not be chosen too large. Tab. 4.3 shows an overview of the networks characteristics if constructed with different δ values. The table shows the number of nodes and edges of the network in the second and third column. The fourth column lists the maximum and average valence of the nodes, denoted as ν. The last three columns give an approximation of the error ϵ in meters caused by the heuristic. The correlation between this error and the parameter e_{max} is the following: e_{max} is lower limit to distance between any two nodes in the network. But at the same time it is the *maximum* distance a stimulus can be apart from any node, before a new node is created to reduce the quantization error for the stimulus. However, the nodes created by the heuristic do not originate from regular training data, but from additionally created samples. These samples may not be located within the area of the regular training data. To measure this error, the minimum distance to any sample of the training data is calculated for each node created during the bootstrapping. From those values the minimum, maximum and average distances are calculated. The ITM network itself is generated from the same training data as shown in Fig. 4.4 (left) with e_{max}=0.10.

δ	Nodes	Edges	max ν	Ø ν	min ϵ	max ϵ	Ø ϵ
0	175	330	10	3.76	n.a.	n.a.	n.a.
2	177	388	13	4.37	0.0154	0.026	0.0208
3	206	586	14	5.68	0.0053	0.115	0.0341
4	270	988	15	7.31	0.0089	0.140	0.0538
5	367	1602	17	8.72	0.0038	0.220	0.0714

Tab. 4.3: Network statistics of training with different δ values. Networks are trained with e_{max}=0.10.

For δ=2, the additional fifth rule has practically no effect on node generation. The

generated training samples are too close to existing nodes and therefore the creation of nodes is prevented due to e_{max}. Despite this, there is a noticeable increase in the number of edges. If the ITM network is trained with δ=3, then the number of nodes increases by 18 % with respect to training with δ=0, whilst the number of edges significantly increases by 78 %. The connectivity increases from δ=2 to δ=3 in a similar manner as from δ=0 to δ=2. However, the estimated error is also significantly greater. The maximum error produced by the heuristic is a measure for obstacle collisions originating from node creation in unexplored areas. Even if the average error is small, a single node with a big error may cause a collision. The maximum error has quadrupled to a total of 11.5 cm when compared to δ=2. By increasing δ to 4 and 5, the error reaches more than 20 cm. Despite the highly increased connectivity, this order of magnitude increment in maximum error renders the resulting network unsafe and therefore useless for navigation purposes.

However, there is a simple way to gain the increased connectivity of a higher δ value without increasing the error too much. High errors are produced when a sample ξ_c is generated between nodes with a large distance, since this directly influences the maximum distance a node may be apart from the network. Filtering out such potentially high-error samples before consolidating them with the training data prevents their negative effects. For this, limiting the maximum distance allows to drastically lower the error in exchange for a moderate loss of connectivity gain. Choosing a threshold for the maximum distance is fairly easy when taking a look at the node creation rules of the ITM. A stimuli has to have a distance of at least e_{max} to any node, aside from the Thales sphere criterion. When creating a sample between two existing nodes, their distance to it in consequence has to be at least two times e_{max}. Choosing three times e_{max} as the threshold provides a span of exactly e_{max} for new nodes to be created, which proved to be a suitable value. Tab. 4.4 shows the results for the training procedure with this limit turned on. For δ=3, the maximum error is reduced by more than

δ	Nodes	Edges	max ν	Ø ν	min ϵ	max ϵ	Ø ϵ
0	175	330	10	3.76	n.a.	n.a.	n.a.
2	177	388	13	4.37	0.0154	0.026	0.0208
3	191	509	14	5.32	0.0151	0.053	0.0266
4	203	596	14	5.86	0.0146	0.111	0.0399
5	211	653	15	6.18	0.0053	0.128	0.0445

Tab. 4.4: Network statistics of training with different δ values and distance limit for node pairs.

half compared to the previous heuristic while the gain in average valence is decreased by only 10 %. Even for δ =5, the maximum error is only slightly higher than for δ=3 without limitation, which was the first value having a significant effect. Exemplary results for δ=4 are shown in Fig. 4.12.

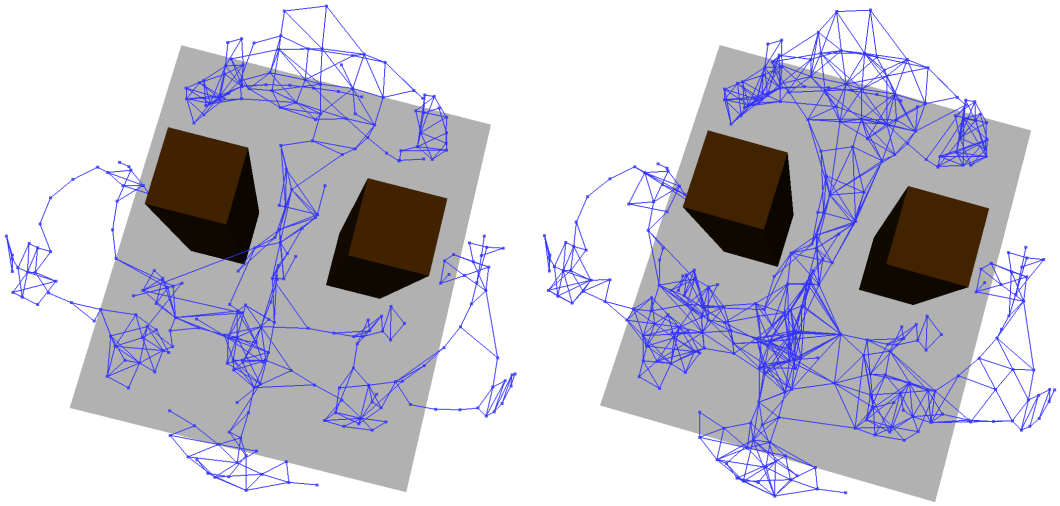


Fig. 4.12: Comparison of the ITM training without δ-bootstrapping (left) and with δ-bootstrapping for δ=4 (right). The network density is significantly increased and most of the smaller holes are closed.

4.3.2 Bootstrapping of Global Interconnectivity

While the δ-bootstrapping is able to close holes and to increase nodes' local connectivity, the problem of weakly connected clusters remains. Even if the distance between clusters is small, they are only connected if the robot has been moved between them during training. In many situations, this forces the path planner to traverse along other clusters to reach an adjacent one, even though a direct movement should be possible. Therefore, a second heuristic called *Global Interconnectivity Bootstrapping* (GI-bootstrapping) searches for potential cluster pairs and tries to connect them. Most of the time the mere distance between two postures is sufficient enough a criterion to connect them. But since these connections will be part of a trajectory for the robot, a transition between two postures is not always possible and a validation is necessary. Situations where the physical distance for the endeffector is small, but the corresponding postures are very different have been explained in Sect. 4.2 and an example for both valid and invalid connections has been given in Fig. 4.5. This heuristic validates po-

tential connections using exactly the same method as described in Sect. 4.2 to prevent such connections.

the heuristic operates as follows: For each newly created node r check if there are potential nodes for a connection. A connection between two nodes has to fulfill three criteria to be valid. Firstly, a Cartesian distance threshold must be fulfilled. A nearest neighbor search selects all nodes that are within a sphere with the radius d around the node r. This threshold value has been picked by evaluating several sets of training data. The average distances between clusters for different users have been observed to find a fitting value that connects adjacent clusters and prevents the creation of edges longer than the size of obstacles in the environment. A value of 0.25 m included a sufficient number of nodes to find beneficial connections and to exclude counterproductive ones. This is of course very specific to the scenario. Secondly, from all selected nodes, those that stand in close connection to r are removed. For this, every node that is contained in the δ-neighborhood $\mathcal{N}_\delta(r)$ is excluded, because this heuristic is the opposite of δ-bootstrapping. The GI-bootstrapping tries to find *global* connections while the δ-bootstrapping searches for possible *local* connections. At last and most importantly, the aforementioned joint space criterion filters out invalid posture transitions. If all three criteria for a pair of nodes are fulfilled, an input sample is generated between them like in the δ-bootstrapping. Formulated as a sixth rule for the ITM, this heuristic can be described as follows:

6. Sample additional data from global neighborhood for each newly created node r:
   (i) Construct the δ-neighborhood $\mathcal{N}_\delta(r)$ for $\delta > 1$
   (ii) Construct the neighborhood by distance $\mathcal{N}_d(r)$ with radius d:
   $\mathcal{N}_d(r)$: $\{c \mid D(r,c) < d\}$.
   (iii) Construct the set of potential node pairs: $\mathcal{N}_p(r) = \mathcal{N}_d(r) \setminus \mathcal{N}_\delta(r)$
   (iv) For each $c \in \mathcal{N}_p(r)$: if $D_\theta(r,c) < d_{\theta\,\max}$, then train net with input stimulus ξ_c:

$$\xi_c = w_c + \frac{1}{2} \cdot (w_r - w_c). \tag{4.3}$$

Results for this heuristic are shown in Fig. 4.13. In picture (1) a number of valid connections are marked. These connections create shortcuts at places where otherwise a larger movement along the network is required. For example the training area at

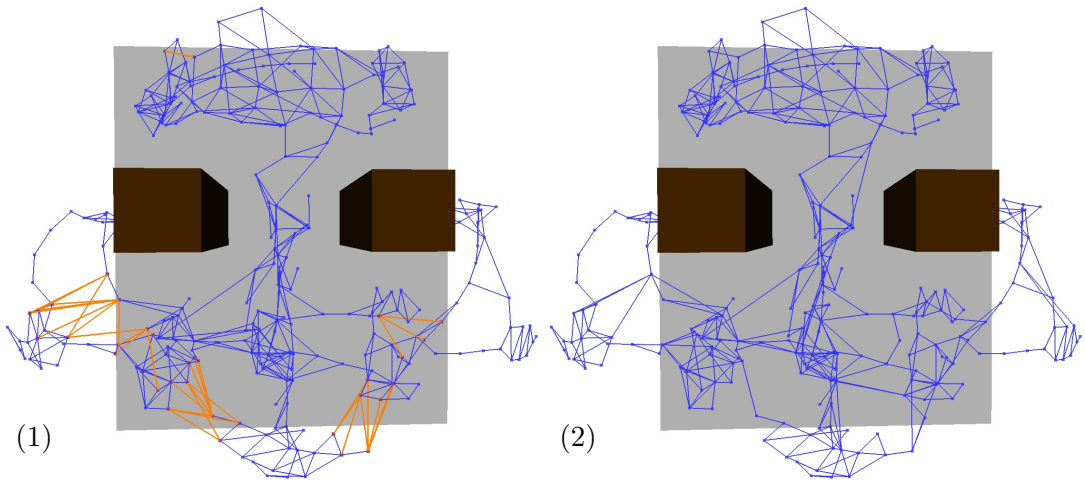


Fig. 4.13: Results of the GI-bootstrapping for d =0.25 m and max D_θ=2.5 rad. (1) shows a subsequent global search with compatible node pairs marked in orange. In (2) the heuristic is integrated into the training phase, where training samples are created between compatible nodes.

the very bottom is connected to the rest of the graph by only a single edge. The new connections add the possibility to directly move to the training areas on either side. Picture (2) shows the result when the heuristic is used during training to create training samples. For each connection marked in (1) a sample is generated and edges are created depending on the ITM edge creation rules.

4.4 Software Implementation

The presented algorithms have been implemented using C++ as programming language. They are integrated as separate components into the FlexIRob framework according to the information-driven design principles used in the project [7, 21]. The implementation utilizes the *Compliant Control Architecture* (CCA) software framework [22] to feature modularized components. CCA itself uses the *Robotics Service Bus* middle ware [23] to allow interprocess and network communication. The implementation includes a graph learning component realizing the ITM algorithm explained in this chapter and a visualization component for the graph, as well as components introduced later in the thesis. The visualization uses the *Open Robotics and Animation Virtual Environment* [24] to visualize the robot and environment. In terms of computation times, the implementation fulfills the real-time requirement without any

problems. Using an Intel® Xeon® E5530 Quadcore CPU with 2.40GHz computation times below 1 ms on average for a single input sample are achieved. This has been tested for the complete algorithm with both bootstrapping heuristics enabled (δ=4) and confirmed for large networks of 300 and more nodes. Bearing in mind the position update rate of 100 Hz – i.e., a position update every 10 ms – the algorithm is feasible even for accelerated input streams ten times faster than real-time.

4.5 Discussion

In this chapter the network learning as basis for the data processing has been shown. The implementation of the Instantaneous Topological Map has been explained along with a description of the parameters and their mode of action.

The network creation has been extended with an edge validation mechanism required for the intended scenario. This has been motivated by deficits that became apparent in context of this thesis and a solution has been provided. To achieve this, the configuration space is used to test whether a correct and safe transition between pairs of nodes is feasible. The mechanism has been tested with ELM-produced joint angles and training data joint angles. The comparison has shown that the latter outperforms the former due to the generalization behavior of the ELM.

While applying the algorithm on real-world training data, several deficits in the resulting network were discovered. In order to overcome these and improve the network, the ITM algorithm has been extended with two heuristics. The first one uses local information modeled implicitly to bootstrap the network, which increases the connectivity between the nodes without generating large quantization errors. The effect of the introduced parameter δ for neighborhood depth has been systematically investigated and the typical behavior for big and small values has been shown along with a guideline on how to choose a suitable value for δ. The GI-bootstrapping tackles the problem of low interconnection between the clusters as a result of the kinesthetic teaching approach used to generate the input data. For this, a global search, that respects the desired online-training capability of the network, is used to find potential connections, which are then tested using the joint angle criterion.

Chapter 5

Autonomous Task Generation and Execution

The next step in this thesis is to find a trajectory of consecutive postures that allow to safely move the robot from its initial position to a goal position. The previous chapter described the generation of a network in the workspace. The nodes of this network represent free locations and the edges specify allowed movements between the nodes. In the context of path planning, this network will further be referred to as the navigation graph. This chapter describes how a graph is used to create the actual path and to control the robot. In general, this is done using a search algorithm to find a path that minimizes some cost function. The method used here is known as the A* algorithm and guarantees to find an optimal path. After a path has been found, postprocessing is applied to generate a smooth trajectory of waypoints and an execution controller controls the robot to move along the path.

5.1 Path Finding Using A*

The target of finding a shortest path in a graph has been a research target in the field of artificial intelligence for decades. The A* algorithm is one of the fundamental and most popular algorithms, which has been first introduced by Hart *et al.* [25] in 1968. It is an extension of the very popular Dijkstra algorithm [26]. A* is a heuristic search algorithm that guarantees to find the shortest path from the start node to the goal node, if it exists. The scenario in this thesis is, that the robot is in the vicinity of a node in the navigation graph and that a target location in the workspace is provided. The goal is then to find a path connection these two locations.

The pseudo code of the A* algorithm is displayed in Alg. 5.1. It uses two sets: the set `closed` contains all nodes that have already been visited and the set `open` contains the nodes that are due to be explored. For each node two values are stored. The *g-score* holds the cost to travel from `start` to a node along the best known path and in `parent` the preceding node of this path is stored. For determining the next node to expand a second score is calculated, the *h-score*. Whilst the g-score is the sum of all

Algorithm 5.1 A* Algorithm

```
1: function A*(start,goal)
2:     closed = {};                                   /*set of expanded nodes*/
3:     open = {start};                                /*set of open nodes*/
4:     g(start) = 0;                                  /*cost at start node*/
5:     parent(start) = start;
6:
7:     while (open ≠ {}) do                           /*loop open set*/
8:         current = open.pop();
9:         if (current == goal) then
10:            return current                         /*path found!*/
11:        closed.insert(current);
12:        open.remove(current);
13:        for all neighbors n in 𝒩(current) do
14:            if (n ∉ closed) then
15:                if (n ∉ open) then
16:                    g(n) = ∞;
17:                    parent(n) = NULL;
18:                update_node(current,n);
19:    return "no path found"

20: function UPDATE_NODE(c,n)                         /*update cost for a neighbor*/
21:    if (g(c) + cost(c,n) < g(n)) then
22:        g(n) = g(c) + cost(c,n);
23:        parent(n) = c;
24:        if (n ∈ open) then
25:            open.remove(n);
26:        open.insert_sorted(n,g(n)+h(n));
```

travel cost up to the current node, the h-score approximates the distance from a node to the `goal` node. This is for example the linear distance, which is in this scenario the distance between the node and `goal` according to the chosen metric D. The nodes in `open` are then sorted in ascending order by their total distance from `start` to `goal`, which is the sum of the g- and h-score of a node. With respect to the optimality of A*, the estimation of the h-score has to be admissible, meaning that the costs to reach the goal must never be overestimated. Using linear distance satisfies this condition.

At first, `closed` and `open` are initialized with the empty set and the set containing only `start`, respectively. The algorithm then processes the `open` set, iteratively expanding the first node of the set and adding its neighbors to the `open` set. For

each neighbor that has not already been expanded (therefore not being an element of `closed`) the g-score and parent values are updated if the previous g-score is higher than the g-score of the current node plus the cost to reach the neighbor. The algorithm finishes if either the current node becomes the `goal` node and the shortest path is found or the `open` set is empty, which means that no path exists. The actual path is then created by recursively following the parent nodes from `goal` until reaching `start`. Fig. 5.1 illustrates an example exploration for the procedure of A* step by step.

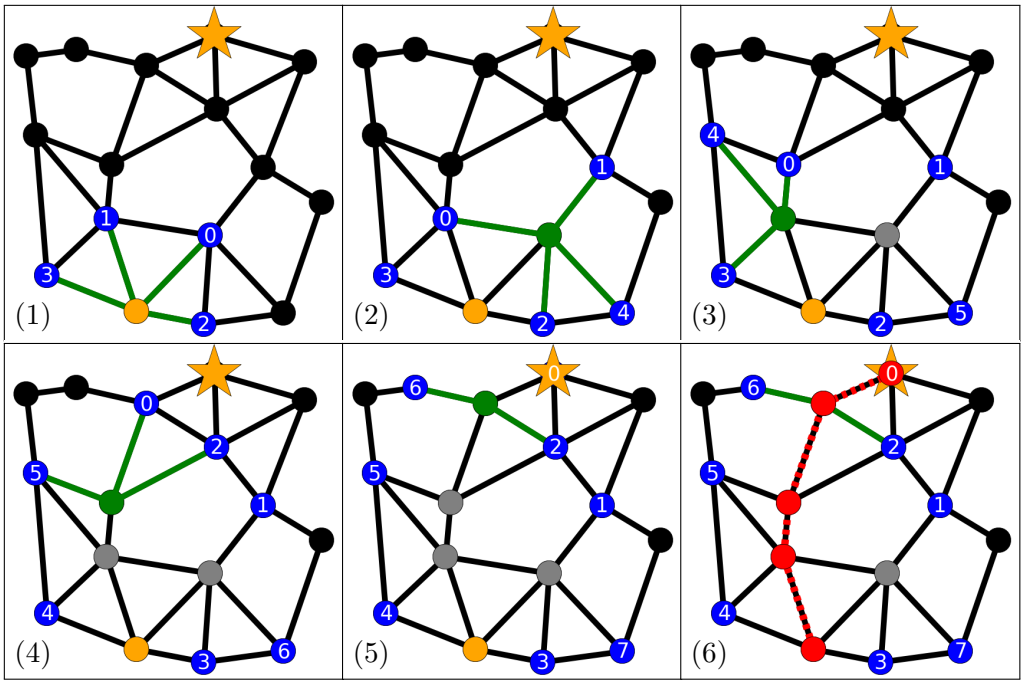


Fig. 5.1: An example for an A* shortest path search: In orange depicted are the `start` and `goal` nodes, green is the `current` node, grey are nodes in the `closed` set. The blue nodes are from the `open` set with the number showing the processing order according to the estimated path length.

5.2 Path Smoothing

The result of the A* algorithm is a sequence of nodes, which function as waypoints. This sequence basically fulfills the requirement of connecting start and goal position. However, according to the resolution of the graph a movement using only these points may be rough and suboptimal. Such a path will generally not be in straight lines and smooth curves, but in slight oscillations, as the example in Fig. 5.2 (1) depicts.

To counteract this effect, the waypoints are interpolated and smoothing is applied. This is done using cubic spline interpolation. In this method, points are interpolated using piecewise continuous polynomials (the splines) of a low degree. In contrast, in regular polynomial interpolation a single polynomial of high degree is used. The advantage of spline interpolation is that the error can be kept small even though low degree polynomials are used [27]. Furthermore, spline interpolation does not have the disadvantage of strong oscillations, that interpolation with high-order polynomials suffers from. The SciPy library – a collection of algorithms for scientific work in (the programming language) Python [28] – is used to implement spline interpolation and smoothing. The implementation of the library is described in [29]. The smoothing is also a feature of the library and not part of this thesis. Basically, smoothing uses the quadratic error of the interpolated points with respect to the input points for reducing oscillations. Instead of having an error of zero, which means no smoothing, the sum of the divergences for all points is limited to a threshold s. The actual value for s has been determined empirically, so that only a small amount of smoothing is applied, which is enough to remove the smaller oscillations caused by the graph structure. The result for the example in Fig. 5.2 is shown on the right-hand side.

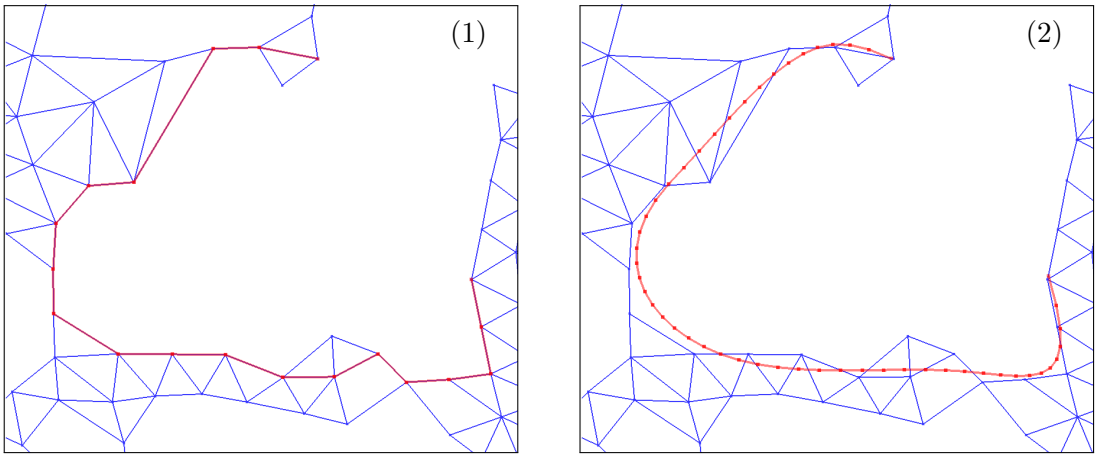


Fig. 5.2: Interpolation and smoothing of the via points using cubic spline interpolation of the path in the navigation graph found by A*.

5.3 Motion Generation

With path search and interpolation completed, the final step left is the execution of a path. The robot has to move from point to point along the edges. The FlexIRob framework allows robot manipulation fairly easily using the existing position controller. As explained in Sect. 2.3, the redundancy resolution is chosen using the ELM network. With a joint configuration provided by the ELM for a given task space position, the hierarchical controller moves the robot as close to the position as possible.

Using this position control, a simple task-space-level controller allows the robot to follow a trajectory. The robot receives the first position and starts approaching it. As soon as the distance between the current position and the target falls below a threshold, the robot is commanded to the next point of the path. A high threshold further "smoothes" the path, because the robot does not accurately approach the targets. A value too small, on the other hand, causes stuttering in the movement, because the position controller already stops the movement before the new position is received. A good compromise between both effects is found for a threshold of about 8 cm, which allows sufficiently accurate approaches while still maintaining a continuous movement. An example of this is shown in Fig. 5.3.

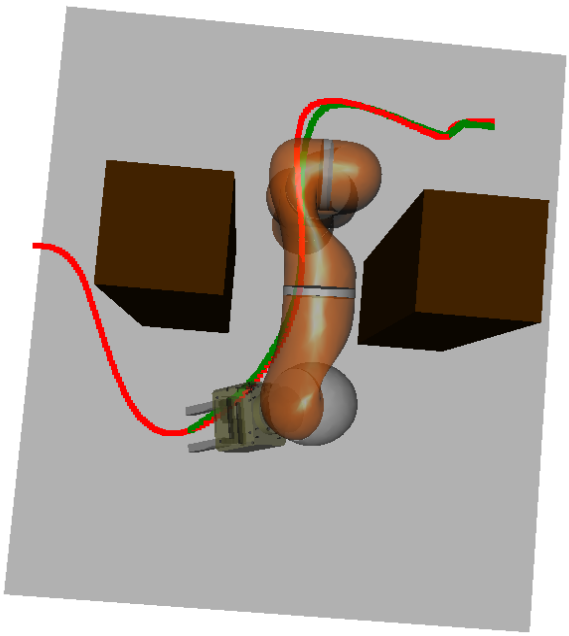


Fig. 5.3: Robot movement along a trajectory. The target path is depicted in red, while the actual movement of the robot is shown in green.

5.4 Software Implementation

The algorithms explained in this section have been implemented using the Python programming language. Information exchange with the C++ graph learning component is achieved through the Robotics Service Bus interface, as mentioned in Sect. 4.4. The average computation time for a path has been investigated by generating paths for random start and goal positions using a graph with approximately 200 nodes. The time was measured starting in the moment when target and goal positions are received until calculations are finished and the first movement command is send. The computations therefore include finding of the nearest nodes for the positions, searching a path with A* and interpolating the points with splines. This resulted in computation times of about 70 ms on average for 1000 generated paths using the same hardware as described in Sect. 4.4.

5.5 Discussion

This chapter explained the utilization of the ITM network to generate waypoint trajectories for the robot. Using the A* algorithm, a shortest path with respect to the Euclidean distance is calculated, which consists of a list of consecutive nodes. The positions connected to these nodes are utilized to control the robot. To generate more fluent trajectories, the points are interpolated and smoothed using spline interpolation. Finally, the rudimentary execution controller is described, which has been used to test trajectories on real robot hardware.

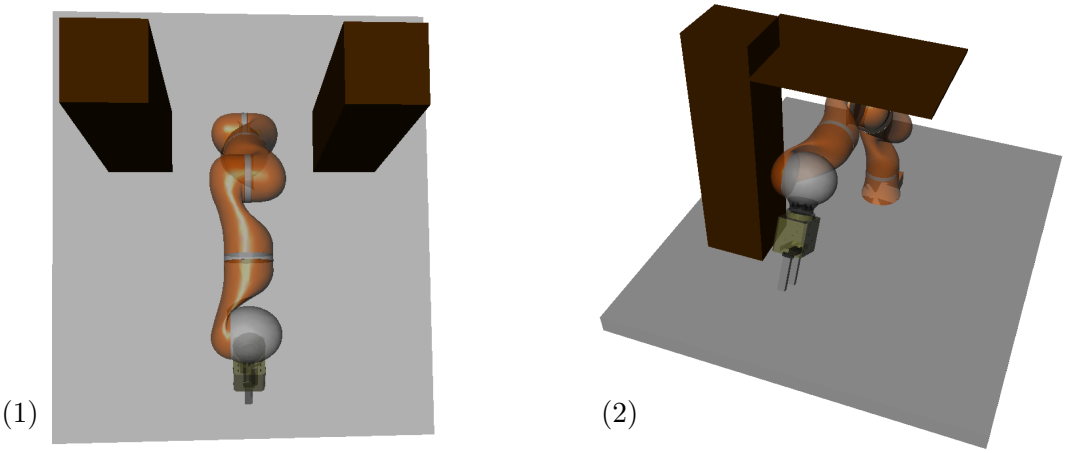
Chapter 6

User Experiments on Method Applicability

The previous chapters demonstrated the mechanics and results of the path planning method for simple test cases and constructed training data. This has proven the concept as working and that the data-driven approach has been successfully implemented. This chapter will evaluate the applicability of the method when deployed in experiments with users. For this, at first the testing conditions and evaluation methods are described and subsequently the gathered data is presented and interpreted. At first, the ITM parameter e_{max} is analyzed using the basic ITM algorithm. After that, the bootstrapping heuristics are tested for effectiveness and the error they lead to.

6.1 Setup and Data Acquisition

The general evaluation consists of two scenarios, which are shown in Fig. 6.1. The first one consists of two cube obstacles parallel to the robot and the second one of a single obstacle that limits the movement to one side and above the robot. Data has been



(1) (2)

Fig. 6.1: Obstacle scenarios in the evaluation.

recorded for three users with different knowledge about the particular robot system and robotics in general. User A possesses no prior knowledge of robotics at all and used the FlexIRob system for the first time. User B is actively involved in robotics research and is associated with the FlexIRob development, but also had no prior knowledge of this thesis. At last, User C is very knowledgeable about the kinesthetic teaching environment and mechanics of the FlexIRob system as well as this thesis' method.

During the training phase, all users are advised to train areas in the workspace exhaustively. They stopped the training on their own as soon as they deemed it sufficient to correctly move the robot at every location of the workspace. Fig. 6.2 conveys an impression of what a training data set looks like for each of the scenarios.

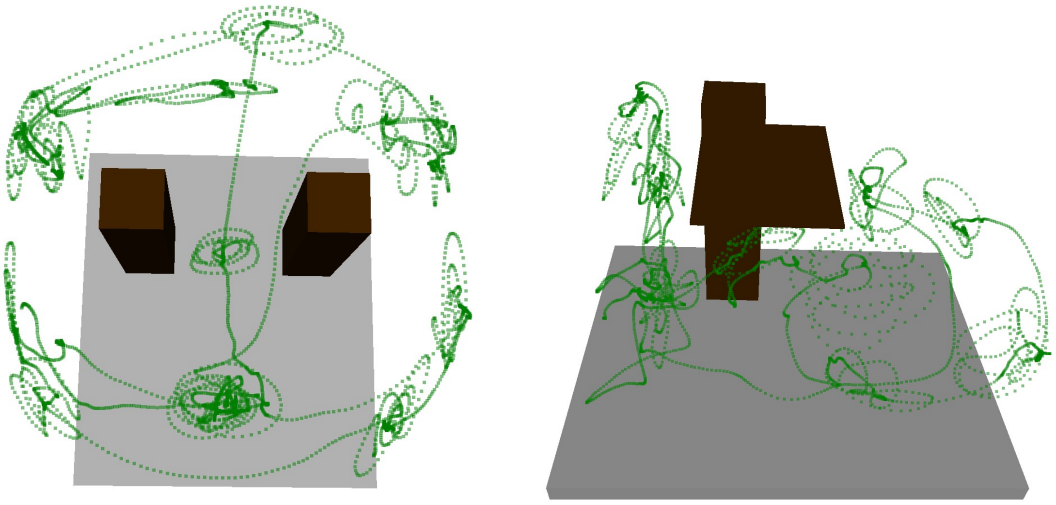


Fig. 6.2: Example training data set for each evaluation scenario. Left: Scenario 1 with data from user B. Right: Scenario 2 with data from user C.

6.2 Influence of the Maximum Quantization Error

The first analyses are regarding the qualities of the basic ITM network depending on parameter e_{max}. Optimizations are completely disabled. For each test user (A, B, C) and each scenario (1, 2) a number of networks are learned for different e_{max}, from 0.06 to 0.20. This range has been chosen, because it covers a wide range of practical values. If chosen below 0.06, strong overfitting occurs, whereas values bigger then 0.20 result in equally strong underfitting (both effects have been explained in Sect. 4.1.1).

6.2.1 Nodes, Edges and Valence

In general, the number of nodes and edges created are in all cases not influenced by the user behavior, but only by the value of e_{max}. This is illustrated in Fig. 6.3 (left), where the relative number of nodes is shown in relation to e_{max}. The number of nodes follows a negative exponential curve. That is, it is very high for small values and starts to stagnate at larger values. The illustration for the number of edges progresses very similarly and is omitted.

This shows the strong impact of e_{max} on the number of nodes, as already mentioned in Sect. 4.1.1. The result for small values is overfitting, because too many nodes with few connections are created. On the other hand, a big value causes underfitting because of the low node generation rate.

A good network for navigation has as many nodes as needed to appropriately represent the training data, but not as many as to cause overfitting. An indicator for both overfitting and underfitting is a low average valence in the network. In the former case the nodes follow the input stimulus and are too far apart to create edges to adjacent movements. In the latter case there are simply not enough neighboring nodes created in a training. To observe this, Fig. 6.3 (right) illustrates the relation between e_{max} and the average valence. Even though the differences are not large, the general trend of the valence confirms the assumption, that low and high values for e_{max} are not beneficial for overall connectivity. While the maximum valences are found for e_{max} values around 0.10, for smaller and larger values the valences tend to decrease.

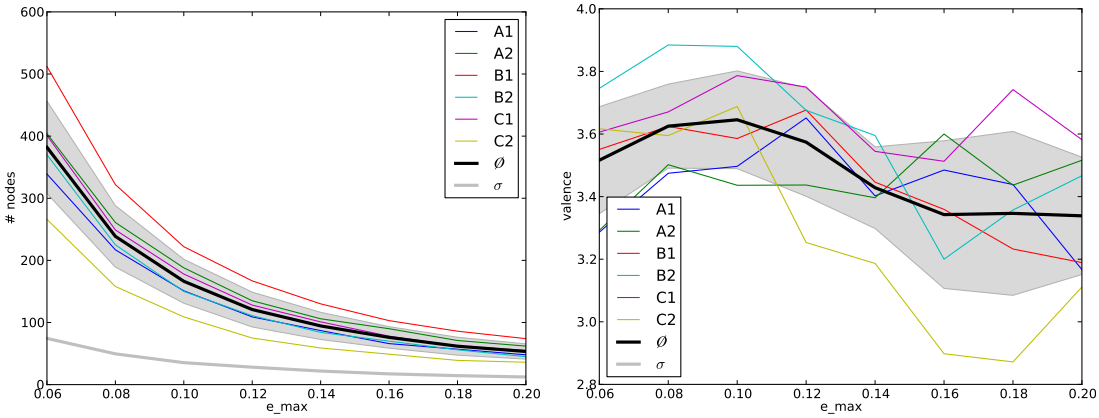


Fig. 6.3: Influence of e_{max}. Left: Number of nodes. Right: Average valence, the standard deviation is not shown due to the reduced value range.

6.2.2 Training Data Representation

A deeper analysis of the over- and underfitting behavior is provided by observing, how well the training data is represented by the network. That is, the effective quantization error. The following measurement is used:

Quantization Error: The distance of a training sample t to its nearest node n:

$$E_t = \arg\min_k \|t_j - n_k\| \tag{6.1}$$

Fig. 6.4 (left) illustrates a graph (e_{max}=0.12) as well as the training data used to generate it. Each node occupies the location of a training sample, however, most of the training samples are not represented by a node. The higher the distances of training samples to their nearest nodes are, the more trained areas are not represented by the graph. For training samples outside of the convex hull of the graph this implies, that they are counted as unknown area. The diagram on the right depicts the results of the error calculations for all combinations of users and scenarios. It clearly shows a linear relation. Therefore, no threshold exists at which a significant change occurs. If a small effective error is desired, then e_{max} has to be chosen accordingly.

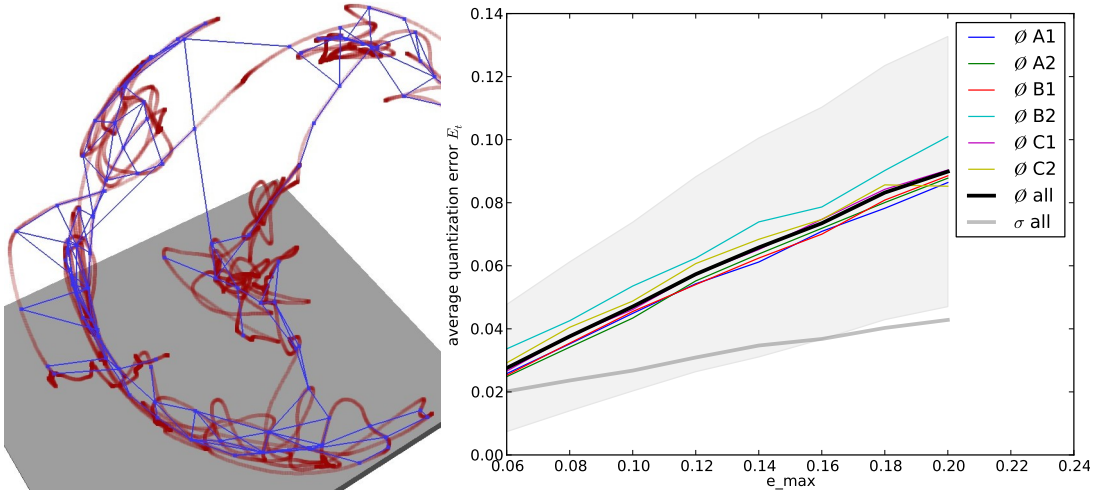


Fig. 6.4: Quantization error for training samples. Left: Network with corresponding training data (red) for e_{max}=0.12. Right: Effective Quantization error in relation to e_{max}. The graphic shows the average error for all training samples for networks learned with different e_{max} values.

6.2.3 Path Lengths and Curvature

To evaluate the influence of the numbers of nodes on actual paths – for each user in each scenario and for each e_{max} value – a set of paths is created and analyzed. The paths are generated as follows:

- The bounding box of the training data is computed.
- Horizontal paths are created for two directions: along the x-axis (from negative to positive x values) as well as in y-direction. The intention is to make use of as much nodes as possible. Paths in z-direction are disregarded, because in most cases the training data is located on a single plane.
- A grid of 10 $\times$ 10 test points is computed on each corresponding plane of the bounding box.
- For each pairing of a point on the positive boundary with a point on the negative boundary a path is created, see Fig. 6.5.

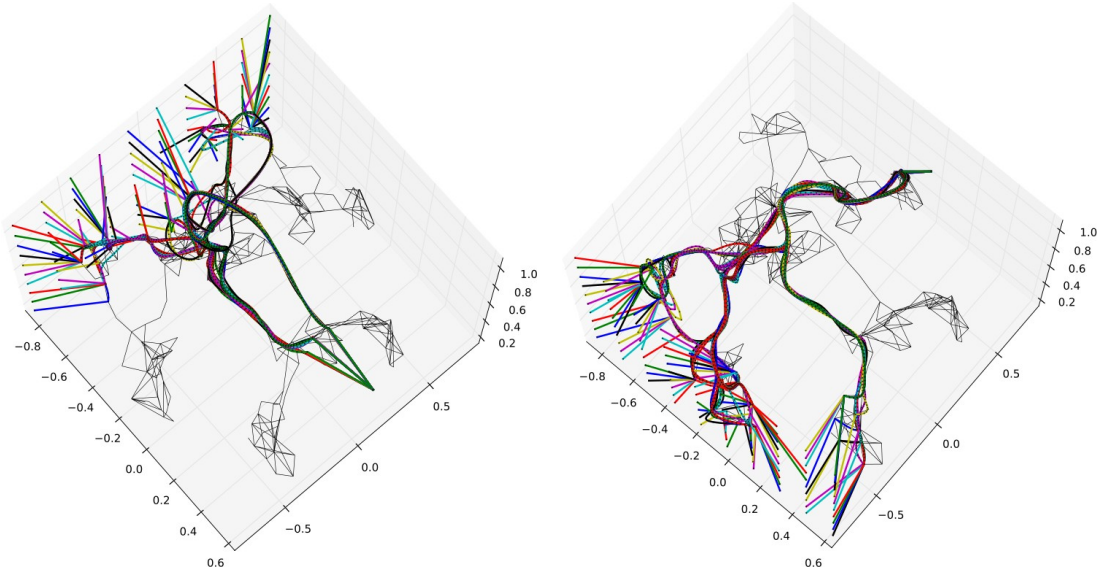


Fig. 6.5: Excerpt of paths created during evaluation of a single user-scenario combination. Both pictures depict the paths created for a single goal point, but with different start points. The left picture shows the paths from points on the $-x$ plane to a point on the $+x$ plane. The picture on the right-hand side shows the same for y values.

- The connections from the boundary points to the first and last point of a path are not interpolated. This is because the same behavior is used in real-world navigation. Outside of the area the network was trained in, no information is available. If a target or start point lies outside, the programmed behavior is to move up to the network on the shortest path available. Therefore, all movement outside of the network follows a straight line, because interpolation causes additional movement in unknown area.
- Each permutation is tested with $100 \cdot 100 \cdot 2 = 20000$ paths.

The resulting paths are analyzed regarding the following two properties:

Path Lengths: How long is an eventual path calculated by A* after smoothing and interpolation:

$$L_p = \Sigma_{i=0}^{N-1} \|p_{i+1} - p_i\| \tag{6.2}$$

Path Curvature: The amount of bending in a path. This is measured using a curvature index proposed by Petreska & Billard [30]. The curvature index is defined as the ratio between the total arc length of a path and the distance between the start and end point. The distance metric is the Euclidean distance:

$$C_p = \frac{L_p}{\|p_N - p_0\|} \tag{6.3}$$

This value provides a rough estimation about the overall shape of a path. For example a curvature index of 1.0 indicates a straight path, whereas $\frac{\pi}{2}$ corresponds to a semicircular path.

The overall result of the path length and curvature analysis is visualized in Fig. 6.6. The diagram shows the normalized average path length and curvature for all evaluations. Separate statistics for length and curvature of the different scenario and user combinations can be found in Appendix A.2 and A.3.

As can be seen from Eq. (6.3), the curvature is strongly related to the path length. The lines shown in the diagram confirm this relation, because both follow nearly the same progression. This is explained by the creation of start and end points using an uniform grid. The distances between start and end points vary only little. Thus, the curvatures are approximately the length values scaled by some factor. The progression of the lines shows that the average path length and the average path curvature are

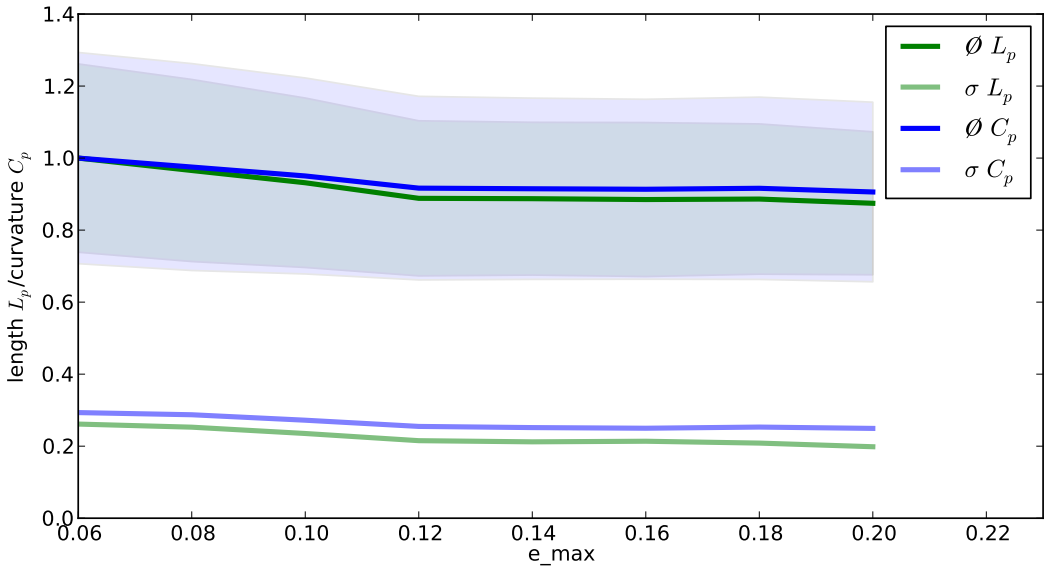


Fig. 6.6: Overall path length and path curvature as function of e_{max}.

highest for e_{max}=0.06. They decrease by up to 11 % (in length) and 9 % (in curvature) at e_{max}=0.12, from which on they start to flatten out. Furthermore, the standard deviation also follows a similar progression up to e_{max}=0.12.

This behavior can be interpreted as follows: the paths lengths and curvatures are higher for smaller values for two reasons. First, overfitting causes paths to contain more detours, because of missing connections. Second, the lower e_{max} is, the more waypoints (=nodes) are respected as constraints in the interpolation. The algorithm tries to fit a curve to these points as good as possible with respect to smoothing. This inclusion of more nodes for the same path length cause small oscillations. An illustration of this is given in Fig. 6.7. In general, this behavior can be interpreted as overfitting in the output function of the path generation. The reason for this is the chosen method. The path is generated using interpolation with slight smoothing. An approximation method instead of an interpolation is required to fully compensate for this erratic behavior. However, this is avoided for a reason. First, this further raises the complexity, because an appropriate approximation method has to be found and configured. The approximation may produce greater errors with respect to proximity to the training data than interpolation. Furthermore, using approximation to counter a misconfiguration of the ITM is only treating the symptoms of the problem. The cure is to resolve the problem at the root by finding a better suiting e_{max} value for interpolation.

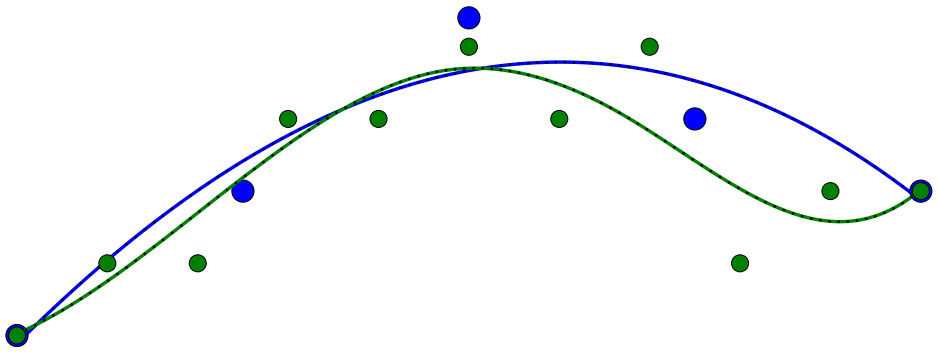


Fig. 6.7: Example for path oscillations caused by interpolation. The blue and green points depict nodes of two graphs with big and small e_{max} values, the lines are the respective interpolations.

In contrast to the disadvantages of overfitting due to small values for e_{max}, large values cause the opposite effect: adjacent training areas and movements between these may not create nodes and be connected, because they are *too* close to each other. In some tests of the evaluation, this even caused the average paths lengths to start increasing again (see Fig. A3).

In conclusion, the best fitting value for e_{max} seems to be 0.12. It provides a good balance between overfitting and underfitting in the network. Higher values do not impair the length and curvature, but provoke undesired effects. The effective quantization error increases linearly with e_{max}, for which a small value is desired. Therefore, the evaluations in the next chapters will use e_{max}=0.12 for all test cases.

6.3 Evaluation of the Bootstrapping Heuristics

The evaluation of the heuristics will be in the following manner: at first the effectiveness of the δ-bootstrapping is examined. This follows the same procedure as before. The heuristic is used with δ values from 2 to 5 and a constant e_{max} of 0.12 to train four networks for each combination of user and scenario. The results are then compared with the basic network without optimization and each other. Afterwards, the GI-bootstrapping is used *in addition* to the δ-bootstrapping. A separate observation of the GI-bootstrapping is omitted, because it is not intended for standalone use.

6.3.1 δ-Bootstrapping

For each learned network the same set of paths is generated and examined for path length and curvature. Fig. 6.8 illustrates the acquired evaluation data. The first two groups display the general network properties. Noticeable is the absent change in number of nodes. Compared to the short analysis in Sect. 4.3.1, this evaluation shows nearly no increase in the number of nodes. This can be explained by the following: The general network topology stays similar – but of course with fewer nodes. The distance required for a sample to trigger the creation of a new node is increased by 50 % (from e_{max}=0.08 to 0.12). However, the samples are created in the middle of the virtual edges between pairs of nodes. As it seems, the topology of the network does not allow these points to exceed a distance of 0.12 to any node. Though the number of nodes does not increase, the heuristic still has a significant effect on the number of edges. The second group in Fig. 6.8 shows the average valence. Even for δ=2 the valence increases by nearly 20 % compared to the reference network. A δ value of 3 produces an increase of 30 %. Only starting from δ=4 the valence increase is not as high as from 0 to 2 and from 2 to 3. As a result, the heuristic effectively counters the problem of the ITM creating only one edge per sample even when the rules would permit further edges. In sparse data sets, these edges are not created due to the low number of samples per region. This heuristic produces the samples that trigger the creation of such edges. The evaluation of the error produced by the heuristic at this point is pointless, because there are almost no new nodes to observe.

The latter two groups in Fig. 6.8 display the path lengths and curvature. Both show a similar progress. They slightly drop with increasing δ by a maximum of 4 % and 3 %,

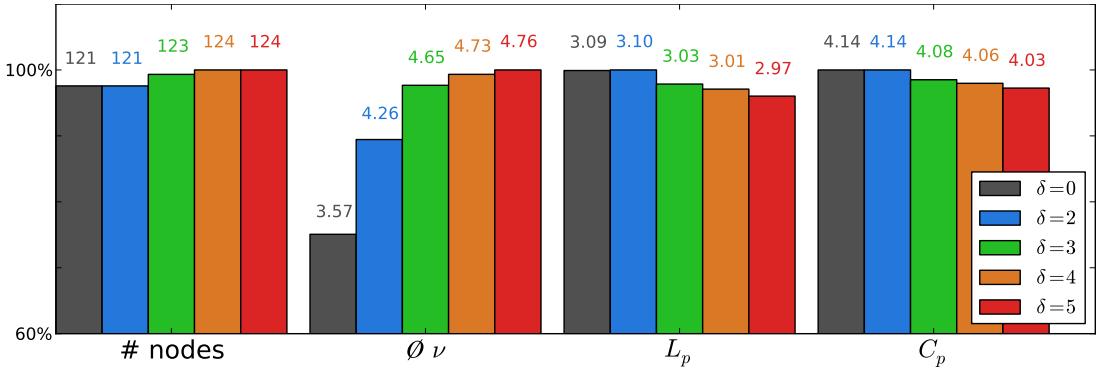


Fig. 6.8: Evaluation data for the δ-bootstrapping. All data displayed is the average over all evaluations.

respectively. As a result, even though the connectivity is significantly increased, the impact on path length and curvature is small but noticeable. In general, an appropriate choice for δ seems to be 4. The positive influence up to this value is noticeable, while difference to δ=5 is small. Bearing in mind, that the computation cost increases exponentially with δ, choosing a higher value is not reasonable.

6.3.2 δ- and GI-bootstrapping

Fig. 6.9 illustrates the results, when the GI-bootstrapping is used in addition to the δ-bootstrapping. As to be expected, the number of nodes and the average valence are not noticeably influenced. However, only by creating very few edges in the network, the heuristic is able to further decrease the path length and curvature. This validates the idea of the heuristic to improve the path generation by finding shortcuts in the network.

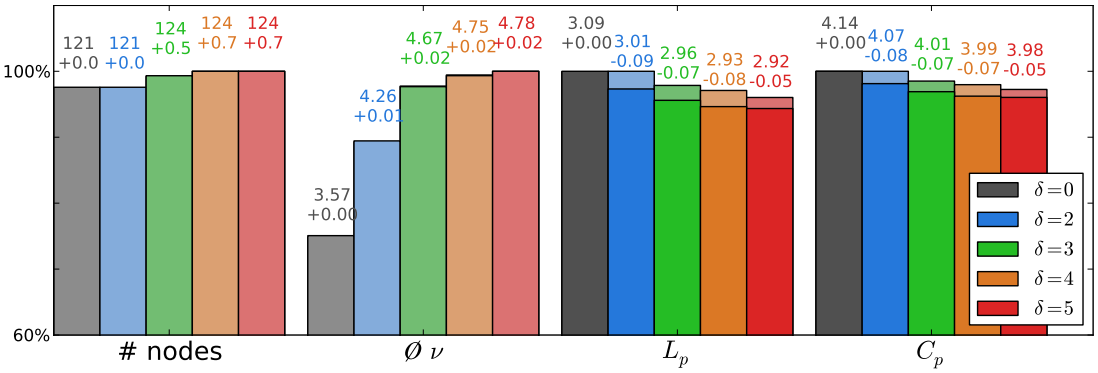


Fig. 6.9: Evaluation data for the δ-bootstrapping and GI-bootstrapping. Pale bars correspond to δ-bootstrapping only. The bottom numbers display the difference to the corresponding network without GI-bootstrapping.

6.4 Discussion

This chapter has provided a thorough examination of the implemented methods and algorithms on the basis of recorded real-world user data. The used testbed has been described to provide an overview of the acquired data. At first the evaluation was concentrated on the basic ITM algorithm, for which a fitting value for e_{max} had to be found. The general impact of the parameter on the resulting network has been demonstrated by observing the number of nodes and the average valence. The number of nodes decreases exponentially with increasing e_{max}, whilst the average valence peaks

at around e_{max}=0.10 and decreases for smaller as well as larger values of e_{max}. The quality of the training data representation decreases linearly with increasing e_{max}. Additionally the fluctuation for single training data samples increases along with the error. The conclusion regarding these network qualities is to choose a rather smaller value for e_{max} to keep the quantization error and underfitting small.

After that, the networks have been evaluated by generating a large number of paths for every possible combination of recorded training data set and e_{max} value. The resulting path lengths and curvatures were calculated and compared. Both properties decrease for values up to e_{max}=0.12 . Higher values do not have a positive effect and are therefore not recommended, bearing in mind that the quantization error increases linearly. In consequence, e_{max}=0.12 is considered as optimal according to the acquired training data.

At last, the heuristics added to the ITM algorithm have been examined by creating networks and paths with and without the heuristics.

Compared to the analysis in Chapter 4, a bigger e_{max} value reduces the impact of the δ-bootstrapping, because almost no new nodes are created. The heuristic still triggers the creation of new edges, boosting the valence by up to 20 %. The impact on the length and curvature of actual paths is lower, though. They decrease only by 3–4 %. The evaluation of the GI-bootstrapping has shown, that it is able to noticeably influence the graph generation by only adding a few edges as shortcuts. Together both heuristics decrease the path length and curvature by up to 6 %. On one hand, these relatively small influence of the heuristics (despite the much higher increase in connectivity) suggests that there is not much potential to improve the path qualities. On the other hand, it also shows, that networks generated with the basic ITM algorithm are already very suitable for generating paths. However, one has to bear in mind, that the evaluations of the path qualities are only theoretically, since the paths have not been used on the actual robot or in the simulation. Such an evaluation is much more time-intensive, but might provide further insights.

Chapter 7

Navigation in Advanced Spaces: Orientation Learning

The previous chapters described the learning of the ITM for the three-dimensional Cartesian space. That is, the ITM learns only the translation of the endeffector position, but not its orientation. The orientation of the endeffector during navigation is then implicitly defined in the redundancy resolution. To test out the general scalability of this thesis' approach, this chapter describes as a proof of concept the modification of the ITM algorithm to allow learning the combination of translation and orientation at the same time.

7.1 Problem Statement

The problem introduced by this task is to combine two inhomogeneous spaces into one single input space. The construction of the ITM fundamentally depends on the nearest neighbor search in the input space. If two input spaces are simply combined by concatenating the feature vectors to a 3+3D vector, the nearest neighbor search weights one of them more important than the other, depending on the ratio of the value ranges. Translation coordinates are specified in meters, where the value range is approximately [-1,1]. On the other hand, the range for orientation angles in radian is [-2π,2π] – more than six times larger. Furthermore, the input spaces have different importance factors. If the orientation receives too much weight, changes in the translation have no impact and only orientation changes trigger the creation of new nodes. The same applies vice versa. In consequence, a good compromise is necessary between both spaces.

7.2 Algorithm Modification

The first aspect is the representation of the orientation. Orientation in this context can be described as a set of parameters, that define the angular position of the endeffector relative to another reference position. There are multiple ways to describe this relation. Some of them are easier to interpret for humans, like Euler angles,

where an orientation is specified by rotation angles around the coordinate system axes. This is easy to visualize, but incurs certain disadvantages, like multiple ambiguities in expressing single orientations or the gimbal lock. The gimbal lock is a geometric problem, where after an unfavorable combination of rotations two axes are aligned. In consequence, both rotate around the same axis and a degree of freedom is lost [31]. A commonly used way to avoid these limitations is the representation of orientations using quaternions. They are an extension of complex numbers, that allow to describe three-dimensional geometrical problem statements with complex algebra. A detailed description of quaternions is omitted at this point and can be found in [32, 33]. The main reason for using quaternions in this context is the need for an unambiguous and non-redundant representation. The goal is to learn prototypes in the orientation space. If input samples with the same orientation were to occur during training the ITM cannot recognize them as equal if their representation differs from one another and two distinct nodes would be created. Hence, the nearest neighbor search must be able to identify similar orientations. Quaternions provide an easy way fulfill this requirement. In the quaternion space, each orientation has exactly two representations, q and $-q$. The following paragraph shows a metric, that measures the closeness of two quaternions independently of the possible two representations, so that it is not required to consider them.

The problem that needs to be solved is to combine the translation and orientation spaces to allow the ITM to learn both at the same time. The first step in the ITM training is the nearest neighbor search. For this purpose, a metric is required that provides a single distance value, so that nodes can be sorted according to their distance to a stimulus. Having two separate distance values for each space would not allow to sort them. The distance metric is adapted in a way that allows to specify a weighting between the translation and orientation spaces. The following metric combines them using a weighting factor ω:

$$D_{TO} = (1 - \omega) \cdot D_T + \omega \cdot D_O \tag{7.1}$$

This allows to easily mediate the weighting of translation and orientation. For ω=0 orientation is completely disregarded and for ω=1 the converse is true. The metric used in the translation space D_T remains the Euclidean distance. To measure the closeness of two orientations q_1 and q_2, the angle of rotation required to get from one

orientation to the other is computed [33]:

$$D_O = cos^{-1}(2 \cdot \langle q_1, q_2 \rangle^2 - 1) \tag{7.2}$$

where $\langle \cdot, \cdot \rangle$ is the inner product. If imagining the quaternions as points on a unit sphere, this formula gives the angle between the lines connecting the origin with the respective points. The ambiguity of quaternions is resolved by taking the square of the inner product.

The second step, the reference vector adaptation, does not need to be changed, because it is omitted anyway. The Thales sphere criterion in the edge adaptation step can be employed for a seven-dimensional vector with translation and orientation values as well, so that no adaptation is required.

In the fourth step, the e_{max} becomes important for the node adaptation. Since the edge length is no longer only the translational distance, but also consists of an orientation part. It may be possible to split this mechanism to use two separate thresholds e^T_{max} and e^O_{max}, but this complicates the decision of when to create a new node. One has to decide if a new node is created if either both thresholds are exceeded or only one of them. The first case is not applicable, because situations can occur, where one of them is greatly exceeded, while the other may never be exceeded. Consequently, no nodes would be created. If only one of them needs to be exceeded, node creation may become unbalanced. If there is much movement in only one of the spaces, the number of nodes shall be defined as *normal*. But if movement in both spaces is close to the threshold before either of them is exceeded, then only one node is created for the double amount of movement than in the *normal* case. Finally, choosing to have a weighting between the e_{max} thresholds amounts to weighting the distances. As a result, the approach that has been chosen is to use only one threshold, which limits the combined distance.

7.3 Estimation of the Weighting Parameter

The weighting parameter ω in Eq. (7.1) was determined empirically. For this purpose, a special training session was recorded, as depicted in Fig. 7.1. The training proceeded as follows:

1. Start on the right-hand side with endeffector oriented downwards (1st training area)

2. Movement to the center (2nd training area)
3. Movement to the left-hand side (3rd training area)
4. Orientation change to leftwards, same endeffector position (4th training area)
5. Orientation change to upwards, same endeffector position (5th training area)
6. Movement to the center (6th training area)
7. Movement to the right-hand side (7th training area)

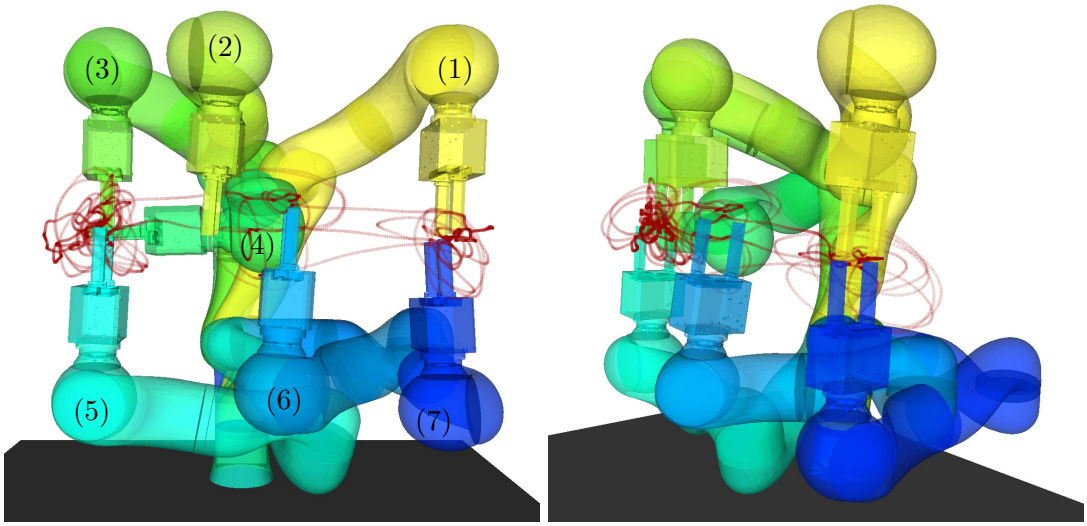


Fig. 7.1: Training areas of the orientation learning session.

The intention of this training session is to train two orientations at the first training area without changing the orientation at this area. The task is then to find a path connecting the downwards position with the upwards position. If the ITM correctly learns this orientation, this path has to be indirect by first traversing to the left side, changing orientation and then moving back.

For empirical determination of the weighting and an according new value for e_{max} the following two-step strategy is deployed: the target is to find e_{max} and ω so that the translation space is influenced as little as possible (w.r.t. training without orientation) and that the weighting of the orientation space allows to successfully learn the orientation *in addition* to the translation.

There are seven training areas of which theoretically only three are respected when training with ω=0. This is because the translations are redundant. The result is a network with n nodes. Assuming that by including orientations all seven areas become distinct, the number of nodes shall increase to $\frac{7}{3}n$, because nodes are created in seven areas instead of three. Therefore, at first an appropriate ω is determined for which the number of nodes roughly meets this assumption. But since the error threshold e_{max} is shared for both translation and orientation, the edge length in translation space L_T will become unavoidably shorter. To counter this, the second step is to increased e_{max} until L_T remains approximately the same as before. This way, the translation space receives the same importance as before and the orientation is included with enough weight to allow learning of different orientations at same training areas. The resulting values produce a network, in which fewer nodes are created relative to the size of the space, because e_{max} is higher after the number of nodes has been doubled by increasing ω. However this is tolerated in favor of the importance of the consistency in translation space. The resulting network is tested by calculating the aforementioned task with the orientation switch at the right training area.

In Tab. 7.1 the network statistics for determining the value is shown. The top row shows the reference network without orientation. For ω values smaller than 0.05 the network does not correctly learn the translation space. Due to the influence of the orientation, edges are removed when the robot moves back from left to right, but ω is still too low to create new nodes (Fig. 7.2, top). This is explained easily: an edge between two nodes is created if one of them does not already have a connection to another node which covers that particular area (Thales criterion). Other connections are removed, if the new connection takes over, because it is better fitting (to a new stimulus). This is strongly related to the distance between nodes. When the orientation

e_{max}	ω	Nodes	Edges	Ø L_T	Ø L_O	Ø L_{TO}
0.08	0.00	26	49	0.0944	0.643	0.0944
0.08	0.03	44	74	0.0873	0.432	0.0976
0.08	0.05	48	86	0.0878	0.317	0.0993
0.08	0.07	55	93	0.0845	0.286	0.0986
0.08	0.10	57	100	0.0810	0.260	0.0989
0.08	0.15	63	107	0.0799	0.211	0.0995
0.08	0.20	74	122	0.0755	0.201	0.1014
0.094	0.10	46	76	0.0934	0.29	0.1137

Tab. 7.1: Statistical data for the orientation learning results.

is included at this point, but the weight is too low, then no new nodes are created. But it is still possible that edges are removed, because neighboring nodes replace them. The orientation is included in the inner product of the Thales criterion and indirectly increases the importance of the translation. One can imagine that the area of a seven-dimensional Thales sphere between two nodes covers more nodes, than the three-dimensional sphere does and therefore more edges are removed. The resulting holes are only closed, if new nodes are created and the Thales spheres become smaller. The result are multiple independent networks.

For $0.05 \leq \omega \leq 0.15$ the network is able to create nodes evoked by orientation changes, as the table shows that the number of nodes is increased compared to the first line. For values of 0.20 and above the network starts overemphasizing the orientation space. By observing the network creation process, it is apparent that edges are created for similar orientations, i.e., irrespective of the nodes' translation (Fig. 7.2, bottom). Edges are created even between node pairs with one node on the left side and the other on the right side. For ω=0.1 the number of nodes is approximately $\frac{7}{3}$ times higher than for ω=0.0 and with e_{max}=0.094 the new edge length corresponds to the translation-only case.

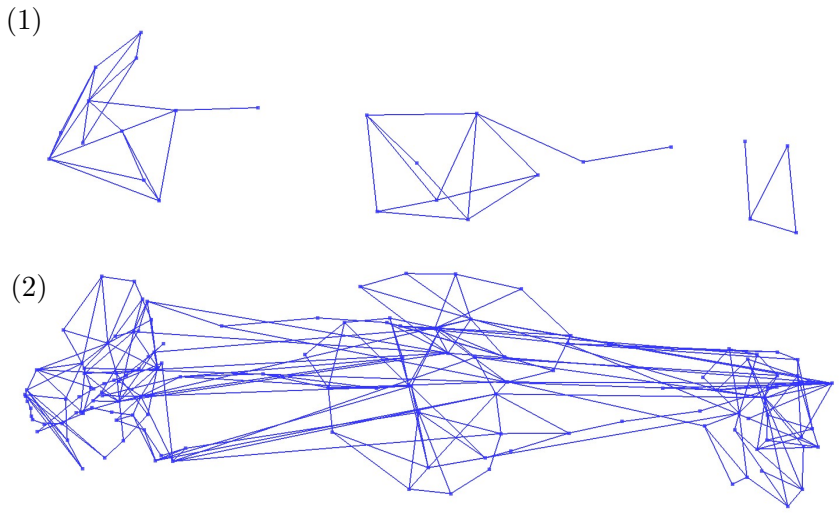


Fig. 7.2: Incorrect learning of orientation and translation spaces. Top: Orientation is weighted very low and causes removal of correct edges. Bottom: Orientation is too emphasized and edges are mostly created based on orientation similarities.

Finally, a path is generated similar to the description in Chapter 5. Fig. 7.3 depicts the start and goal position and the resulting path. The path proceeds as desired first to left-hand side and then back, instead of directly changing the orientation. Since this is only a proof of concept, the orientations are not interpolated between the nodes of the path, as opposed to the translations. The newly generated points just reuse the orientation of the next original target point. However, quaternions allow for a fairly easy interpolation and several popular approaches exist, for example Spherical Linear Quaternion Interpolation (SLERP, [34]).

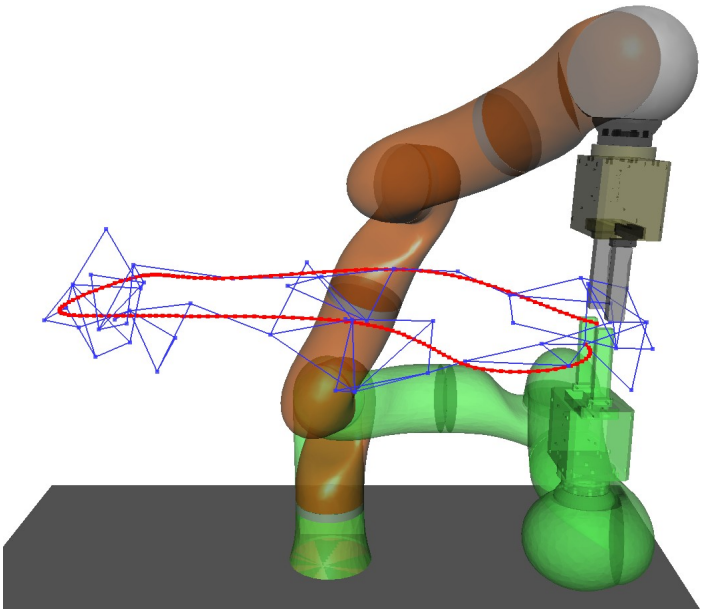


Fig. 7.3: Generated path for position with same translation and opposing orientation. The start position is orange and the goal position is green. The path (red) traverses over to the other side and back to change the orientation of the endeffector.

7.4 Discussion

This chapter proved the general scalability of the path planning approach using an ITM to learn the free workspace. The method has been extended to learn a navigation graph in a feature space that combines both translation and orientation of the endeffector. An adapted distance metric allows to specify the weighting between both spaces for the fundamental nearest neighbor searches. Using a simple test case, it was shown that navigating between different orientations while respecting the translation is possible.

Chapter 8

Haptic User Feedback

After the implementation of the method had been finished and tested on the real-world robot, the intended goal of this thesis had been proven to be functional and to fulfill the overall goal. However, during experiments with the implementation and the user study another aspect was discovered, that may be very beneficial for the applicability of the method. Users showed a strong need for feedback about the current state of their training. During recording, the only available feedback was a visualization of the emerging network on a computer monitor. Users tended to maintain frequent visual contact with the monitor to be aware of their trained areas and to plan their next motions. This problem has also been discovered in a user study of the FlexIRob system [4]. Therefore, feedback during training is identified as an important requirement to improve usability. The aim of the feedback is to let users know if the current movements of the robot are redundant or if they explore new areas. Further, it is desirable to guide users towards an unexplored area if they are moving within explored areas.

However, using visual feedback has shown to have a major drawback: the visualization projects the three-dimensional graph onto the two-dimensional surface of the computer monitor. This lacks the ability to sufficiently represent the geometry of the graph and the visualization has to be constantly rotated to focus the active region. Furthermore, the users attention was directed away from the robot, which is undesireable in a human-robot coworker scenario with respect to safety and didactics.

An optimal solution to this problem may be to use methods of Augmented Reality to directly visualize the graph in the workspace and therefore in the users' current field of view. However, this approach requires the respective equipment. Furthermore, Augmented Reality is also a very active field of research which requires profound knowledge for appropriate results [35]. The setup of this thesis allows for another way of providing feedback. Research in Augmented Reality often encounters the problem of lacking haptic feedback for projections. In current research, this deficit is compensated by using haptic feedback devices to allows users to feel virtual objects [36]. The robot itself together with the kinesthetic teaching interface provides a directly usable tool for providing haptic feedback. This chapter demonstrates how this opportunity has been used to implement a rudimentary haptic feedback component. The objective was to

test if such an approach can provide useful feedback and if further research into this type of technology is advisable.

8.1 Impedance-based Feedback

The haptic feedback is realized by utilizing two components of the FlexIRob software and the robot: the *joint impedance control* of the robot [37] and the *gravity compensation mode* [7] of the system. The impedance control works in a way that it receives position commands and – if no external forces act on the robot – holds the current positions. If external forces do occur, for example because the user pushes against a joint, it behaves in a spring-like manner. The more force the user applies, the larger the discrepancy between actual position and set position, the stronger is the force of the robot to retake the set position of the command. This impedance is specified by two parameters, *stiffness* and *damping*. The latter can be interpreted as the spring constant and the former as the resistance offered by the robot in reaction to the applied force. The gravity compensation mode works by updating the position command in a feedback loop with the current actual position of the robot. Depending on the intensity of stiffness and damping the robot can be moved easily or the stiffness can be increased up to a point where no movement is possible at all (w.r.t. the maximum force of the robot).

These two components are combined to achieve the desired haptic feedback, which works the following way: the goal is to tell the users, when they are moving the robot inside or outside of the graph *during the free movement mode*. This is accomplished by increasing the stiffness of the impedance control inversely proportional to the distance between the endeffector position and the second nearest node of the graph (see Fig. 8.1). The graph is constantly growing, as the robot is moved through unexplored areas. Hence, one node is always within the sphere with a radius of e_{max} around the endeffector. The presence of this node does not allow to make assumptions, if the robot is in the graph or moves towards it, thus the second nearest node is used as a beacon. The stiffness is increased as soon as the distance to the second node is smaller then e_{max}. This indicates, that the endeffector has been moved towards the graph. If the users move the robot even nearer to the second node, the stiffness increases further and they are informed, that they move the robot towards explored space. However, the stiffness is never increased to a point, that users cannot move the robot or struggle with it. The stiffness calculation is shown in Alg. 8.2.

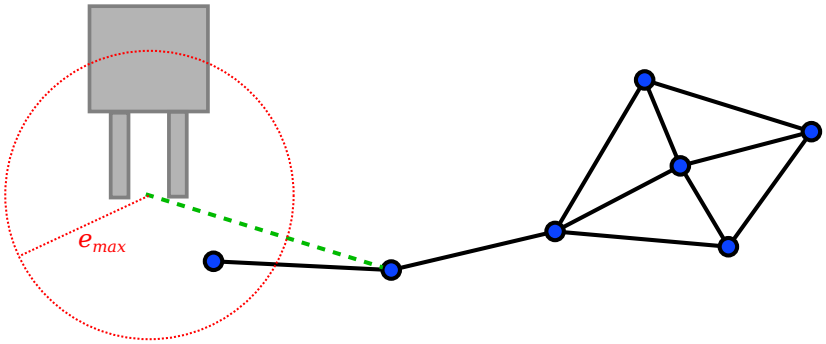


Fig. 8.1: Distance between endeffector and graph. Depicted in red is the sphere around the endeffector with radius e_{max} and in green the distance to the second nearest node.

On top of that, a second modification is added to support the user. If the robot is inside of the graph and if the user stops holding the robot, it shall move out of the graph on its own. This behavior is realized in a simplified manner: the robot only moves upwards in z-direction and towards zero x and y coordinates, that is towards the z-axis. By this the robot is able to avoid the graph in two directions. However, it does not guarantee to move outside of the graph, e.g. if the area around the z-axis is fully explored. This movement is easily accomplished my moving only those joints, whose rotation axis is not the z-axis. Specifically, these are the second, fourth and sixth joint. As long as the endeffector distance to the second nodes is lower than e_{max}, a small offset is added to the position update for the gravitation compensation component of these joints. The pseudo code for this behavior is found in Alg. 8.3

This implementation realizes a simple graph avoidance, but with limited useful-

Algorithm 8.2 Haptic Feedback Stiffness

```
1:  defaultStiffness = 20;                                  /*default stiffness value*/
2:  maximumStiffness = 150;                                 /*maximum stiffness value*/

3:  function CALC_STIFFNESS(pos)
4:      s = get_secondnearest_node(pos);                    /*nearest node to pos*/
5:      d_s = D(pos,s)                                      /*distance to s*/
6:      if (d_s < e_max) then
7:          factor = 1 − (d_s / e_max);
8:      else
9:          factor = 0
10:     return defaultStiffness + factor · (maxStiffness − defaultStiffness);
```

Algorithm 8.3 Haptic Feedback Perturbation

```
1: maximumPerturbation = 0.05;                          /*maximum perturbation value*/

2: function ADD_PERTURBATION(pos)
3:     s = get_secondnearest_node(pos);                 /*nearest node to pos*/
4:     d_s = D(pos,s)                                   /*distance to s*/
5:     if (d_s < e_max) then
6:         factor = 1 − (d_s / e_max);
7:     else
8:         factor = 0
9:     q = get_position_command();                      /*get joint position as float[7]*/
10:    if (q[2] > 0) then
11:        q[2] = q[2] − factor · maximumPerturbation
12:    else
13:        q[2] = q[2] + factor · maximumPerturbation
14:    if (q[4] > 0) then
15:        /*directions of joints are different*/
16:        q[4] = q[4] + factor · maximumPerturbation
17:    else
18:        q[4] = q[4] − factor · maximumPerturbation
19:    if (q[6] > 0) then
20:        q[6] = q[6] + factor · maximumPerturbation
21:    else
22:        q[6] = q[6] − factor · maximumPerturbation
23:    set_position_command(q)
```

ness. In a more correct implementation the robot should not move upwards, but in a situation-dependent direction to move out of the graph. This is a far more complex approach, because further information is required to move the robot in a dynamic direction, e.g. an inverse kinematic. Therefore, only a simple approach is used for this haptic feedback proof of concept.

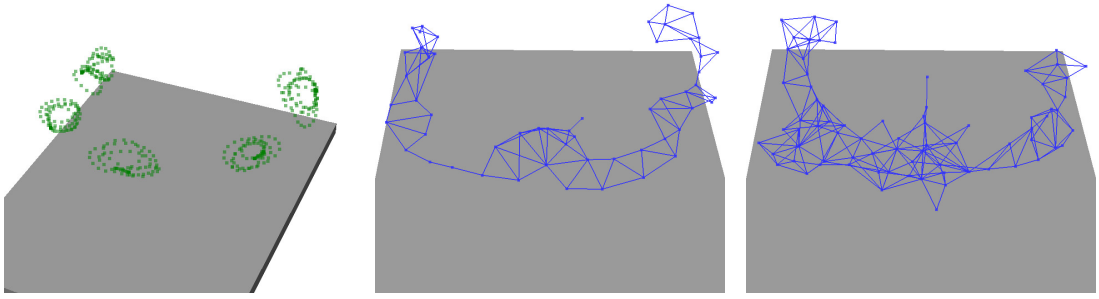
8.2 User Study

To find out if this implementation is useful and if haptic feedback in general is accepted, two users have been consulted to test the training and report back their opinion. While one of the users only trained the robot without a specific task, the other user was given a direct order on how to proceed. The user was asked to train five areas at the same height, which are positioned approximately in a semi-circle (one in front of the robot

and two at each side). The user trained this two times: The first run without the haptic feedback and second one with feedback enabled. In the second run with haptic feedback, the user was advised to try using the feedback to not move through already trained areas. The results are depicted in Fig. 8.2. A direct comparison between the two runs is not meaningful, because the task was changed in the second run. Even though, a comparison of the two graphs shows, that the user was able to train a more voluminous graph. Especially the connections between the clusters have increased. As mentioned earlier, users tend to move along the same route when moving the robot back to a training area, which they already have trained. Here, the user actively tried to avoid this. Of course, this is also possible by only telling users to do so. But using haptic feedback for this, a user does not need to keep track of his movements or rely solely on the 2D visualization.

The overall feedback of the two users was positive towards receiving haptic feedback through the robot. They liked the idea, because it seemed intuitive. However, both showed irritation at some point. On one hand, this might be due to the very rudimentary implementation. On the other hand, introducing haptic feedback to improve the ITM training during ELM training combines two different sub-tasks. Up to this point the ITM generation was solely a "byproduct" of the ELM training. By actively tapping into the ELM training the unique goal of the ELM training vanishes.

In conclusion, haptic feedback in general seems to be useful and well accepted by users. However, the concrete way of integrating it into the ELM training might not be not optimal and the concept requires additional thought. The important aspect is to find a way in which ELM and ITM training can be merged effectively into a single training procedure. This is not a trivial task and requires more research.



Training Areas Without Feedback With Feedback

Fig. 8.2: Evaluation with a user to test the haptic feedback. The left picture illustrates the "task" given to the user. The other two pictures show the resulting graph for the runs without and with feedback.

Chapter 9

Conclusion

The purpose of this thesis was to extend the FlexIRob system with an autonomous path planning component. The premise for this was to go without explicit modeling of the robot and environments in a simulation. The method was supposed to utilize the kinesthetic teaching interface of the system to accomplish this goal. After motivating the need for this type of path planning the Instantaneous Topological Map was introduced in Chapter 4, which embodies the main aspect of the component. The ITM is used to represent the information about the environment obtained from kinesthetic teaching and therefore addresses goal (I) of Sect. 1.2. Sect. 4.2 showed how the algorithm was extended to ensure the correctness of the ITM in context of the redundant robot system. By utilizing configuration space information the system validates edges in terms of feasibility for being performed by the real robot before adding them to the navigation graph. It has been shown that this approach produces reliable results when correct input data in the configuration space is present, thereby fulfilling goal (II). Sect. 4.3 introduced further extensions to the algorithms which bootstrap the input data to improve the shape of the navigation graph.

The goals (III) and (IV) can be considered as the second major part of this thesis' work and are addressed in Chapter 5. It has been shown that a specific tasks is easily solved once a valid navigation graph has been constructed. The graph is processed to connect arbitrary start and goal positions through a list of nodes and edges. Afterwards, the path in the graph is transformed into smooth movement commands for the robot. The software implementation satisfies the goal of real-time online learning and processing. The learning of the graph is feasible even for accelerated input data streams of more than ten times real-time. Computation times for path search and motion generation can be considered as virtually real-time with delays less than one tenth of a second on average between receiving start and goal inputs and the first movement of the robot. In conclusion, the target of the thesis has been successfully accomplished and it has been proven that *path planning for a redundant robot manipulator using sparse demonstration data* is feasible.

The implementation has been tested in a small user study with participants of different expertise in robotics and knowledge of the system. The evaluation aimed towards

finding suitable parameters for all parts of the method. Each parameter has been systematically investigated and its influence has been shown. On this basis a choice for each parameter based on the gathered user data has been presented. Afterwards, chapters 7 and 8 investigated further aspects of such a data-driven path planning approach. In Chapter 7 the ITM algorithm has been adapted to an higher-dimensional navigation space. It has been proven that the algorithm is scalable and allows for operation in spaces other than the three-dimensional Cartesian space. Finally, Chapter 8 evaluated the idea of giving haptic feedback about the training status to the user. A basic approach to this showed promising results, though there is much more effort required for a productive application of such feedback.

9.1 Outlook

Further work may address the concept of orientation learning. As the possibility to include orientations in the ELM learning has been introduced to the FlexIRob system during the work of this thesis, more effort is required to efficiently use this feature. The extension has been tested with an artificial test scenario. Application in real-world scenarios still shows significant deficits which have to be resolved by further research.

Another aspect for future work is to omit the need for two learning components in the system, i.e., the ELM and ITM. By extending the ITM with generalization capabilities, it may be possible to generate joint angles for arbitrary locations in the workspace instead of using the ELM. This avoids the somewhat redundant learning of the same input data and possible inconsistencies caused by this.

The major aspect of this thesis was to process kinesthetic teaching data to construct a navigation graph and autonomously generate trajectories. A rudimentary execution controller has been implemented to generate movement commands for the robot and test the system. However, this controller is not very accurate and stuttering movements of the robot can occur. Hence, a better controller for generated trajectories, e.g. a PID controller [38], might be called for and may be addressed in future work.

Appendix A

Evaluation Data

A.1 Joint Space Validation

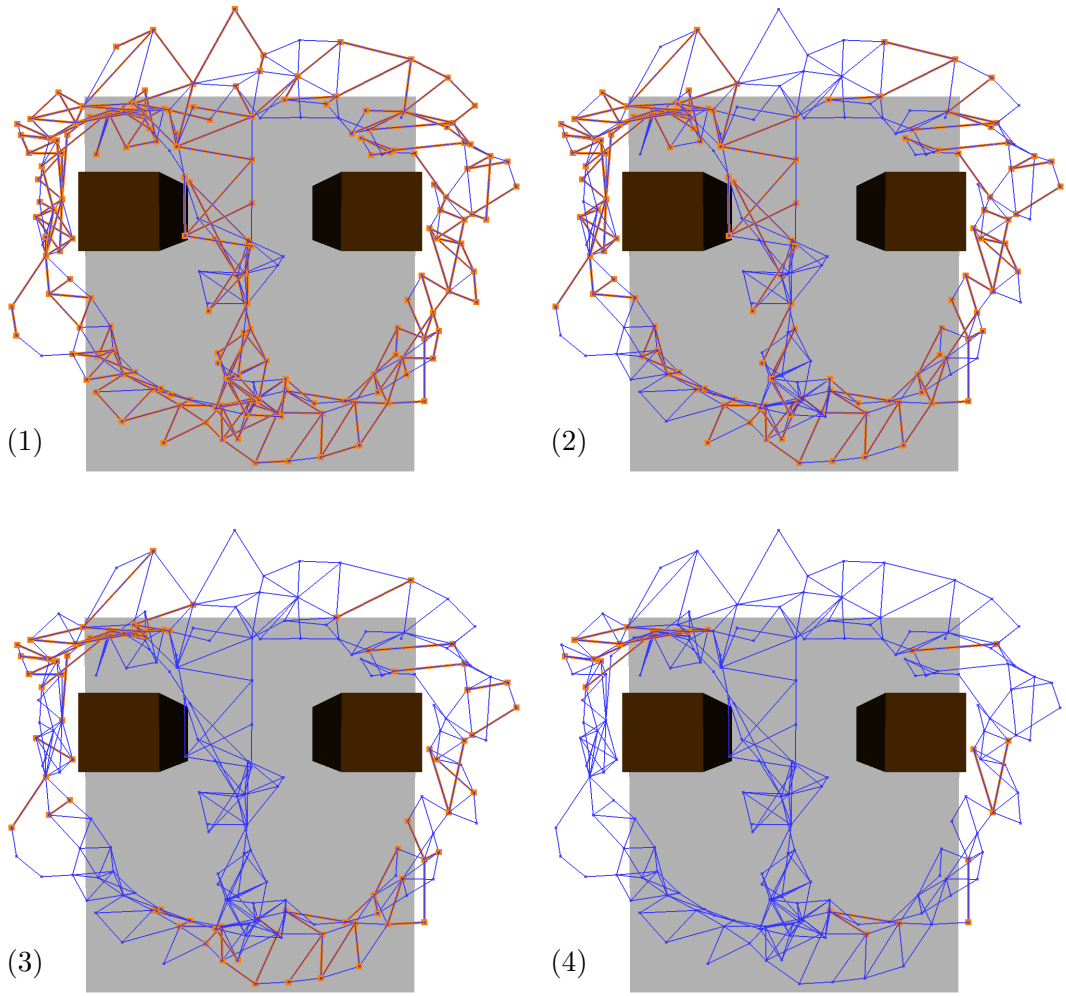


Fig. A1: Joint space validation using ELM-produced angles with threshold $D_{\theta\,\max}$ of 0.5 rad (1), 0.75 rad (2), 1.0 rad (3) and 1.5 rad (4).

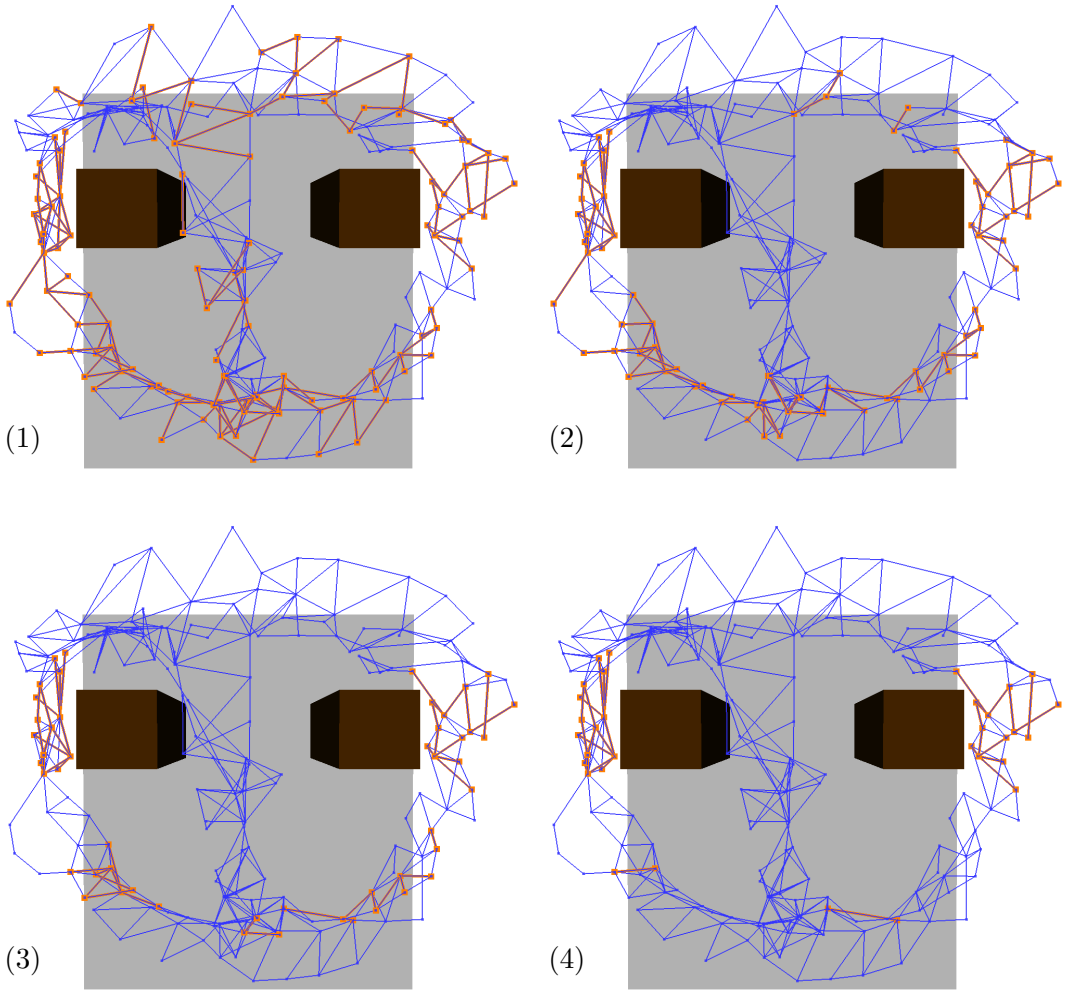


Fig. A2: Joint space validation using recorded angles with threshold $D_{\theta\,\max}$ of 1.0 rad (1), 1.5 rad (2), 2.0 rad (3) and 2.5 rad (4).

A.2 Path Lengths

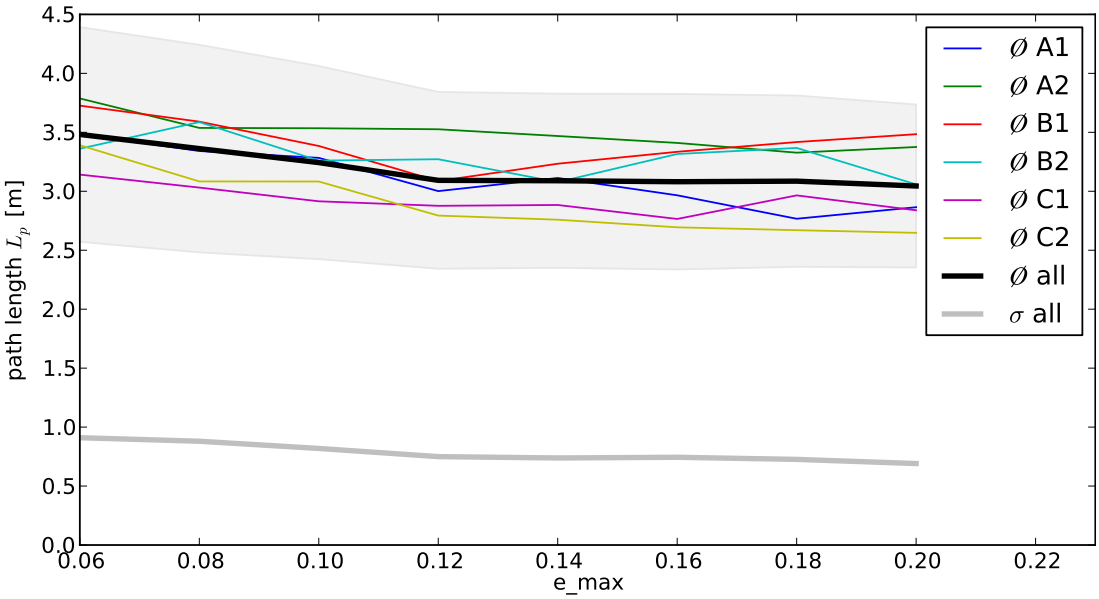


Fig. A3: Average path lengths for each evaluation constellation.

Con \ e_{max}	0.06	0.08	0.10	0.12	0.14	0.16	0.18	0.20
A1	3.48	3.34	3.28	3.00	3.11	2.97	2.77	2.86
A2	3.79	3.54	3.53	3.53	3.47	3.41	3.33	3.38
B1	3.73	3.59	3.38	3.09	3.23	3.33	3.42	3.48
B2	3.36	3.59	3.26	3.27	3.08	3.32	3.37	3.06
C1	3.14	3.03	2.92	2.88	2.88	2.77	2.97	2.84
C2	3.39	3.08	3.08	2.79	2.76	2.69	2.67	2.65
Ø	3.48	3.36	3.24	3.09	3.09	3.08	3.09	3.04
σ	0.91	0.88	0.82	0.75	0.74	0.74	0.73	0.69

Tab. A1: Average path lengths for each evaluation constellation.

A.3 Path Curvatures

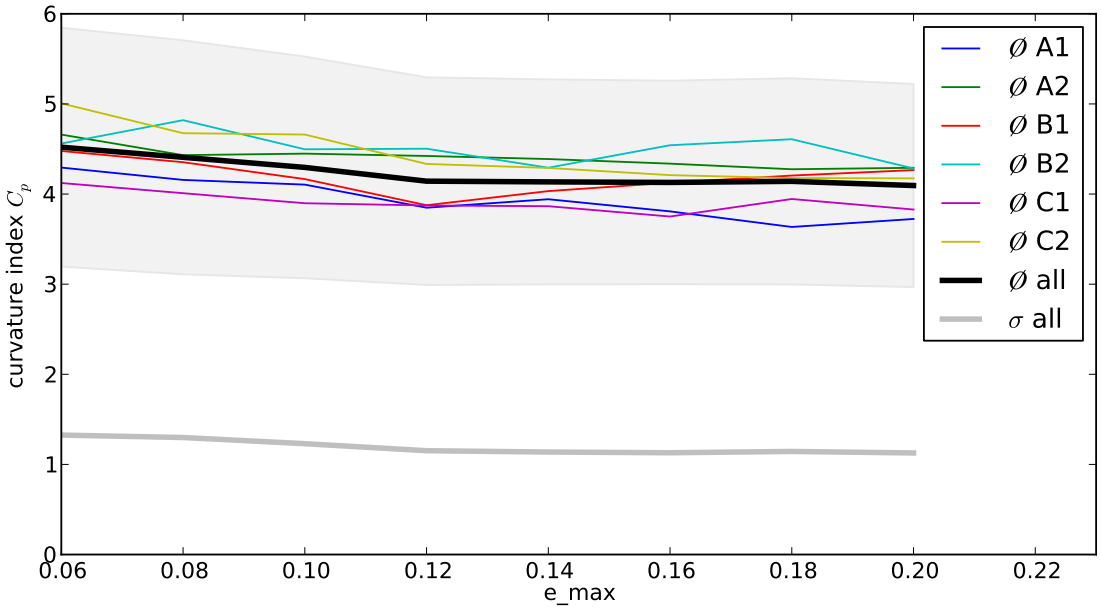


Fig. A4: Average path curvatures for each evaluation constellation.

Con \ e_{max}	0.06	0.08	0.10	0.12	0.14	0.16	0.18	0.20
A1	4.29	4.16	4.10	3.85	3.94	3.81	3.63	3.72
A2	4.66	4.43	4.45	4.42	4.39	4.34	4.28	4.29
B1	4.48	4.35	4.17	3.88	4.03	4.13	4.21	4.27
B2	4.56	4.82	4.50	4.50	4.29	4.54	4.61	4.28
C1	4.12	4.01	3.90	3.87	3.87	3.75	3.95	3.83
C2	5.01	4.68	4.66	4.33	4.29	4.21	4.18	4.17
Ø	4.52	4.41	4.30	4.14	4.14	4.13	4.14	4.09
σ	1.33	1.30	1.23	1.15	1.14	1.13	1.14	1.13

Tab. A2: Average path curvature for each evaluation constellation.

List of Figures

List of Tables

Bibliography

[1] B. Siciliano and O. Khatib. *Springer Handbook of Robotics*. Gale virtual reference library. Springer, 2008.

[2] Bruno Siciliano. Kinematic control of redundant robot manipulators: A tutorial. *Journal of Intelligent and Robotic Systems*, 3(3):201–212, 1990.

[3] Matthias Behnisch, Robert Haschke, and Michael Gienger. Task space motion planning using reactive control. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2010.

[4] Sebastian Wrede, Christian Emmerich, Ricarda Grünberg, Arne Nordmann, Agnes Swadzba, and Jochen J Steil. A user study on kinesthetic teaching of redundant robots in task and configuration space. *Journal of Human-Robot Interaction*, 2(Special Issue: HRI System Studies):56–81, 2013.

[5] Jochen J Steil, Christian Emmerich, Agnes Swadzba, Ricarda Grünberg, Arne Nordmann, and Sebastian Wrede. *Kinesthetic teaching using assisted gravity compensation for model-free trajectory generation in confined spaces*, volume 94 of *Springer Tracts in Robotics*, pages 107–127. Editor: F. röhrbein et al. edition, 2013.

[6] Rainer Bischoff, Johannes Kurth, Guenter Schreiber, Ralf Koeppe, Alin Albu-Schaeffer, Alexander Beyer, Oliver Eiberger, Sami Haddadin, Andreas Stemmer, Gerhard Grunwald, and Gerhard Hirzinger. The kuka-dlr lightweight robot arm – a new reference platform for robotics research and manufacturing. *Joint 41st International Symposium on Robotics and 6th German Conference on Robotics*, pages 741–748, 2010.

[7] Arne Nordmann, Christian Emmerich, Stefan Rüther, Andre Lemme, Sebastian Wrede, and Jochen J Steil. Teaching nullspace constraints in physical human-robot interaction using reservoir computing. In *International Conference on Robotics and Automation*, pages 1868 – 1875, St. Paul, 2012.

[8] Guang-Bin Huang, Dian Hui Wang, and Yuan Lan. Extreme learning machines: a survey. *International Journal of Machine Learning and Cybernetics*, 2(2):107–122, 2011.

[9] Héctor H González-Banos, David Hsu, and Jean-Claude Latombe. Motion planning: Recent developments. *Autonomous Mobile Robots: Sensing, Control, Decision-Making and Applications*, 2006.

[10] Tomás Lozano-Pérez and Michael A Wesley. An algorithm for planning collision-free paths among polyhedral obstacles. *Communications of the ACM*, 22(10):560–570, 1979.

[11] Jean-Claude Latombe. *ROBOT MOTION PLANNING.: Edition en anglais.* Springer, 1990.

[12] Stephen R Lindemann and Steven M LaValle. Current issues in sampling-based motion planning. In *Robotics Research*, pages 36–54. Springer, 2005.

[13] Roland Geraerts and Mark H Overmars. Sampling and node adding in probabilistic roadmap planners. *Robotics and Autonomous Systems*, 54(2):165–173, 2006.

[14] Steven M. Lavalle. Rapidly-exploring random trees: A new tool for path planning. Technical report, 1998.

[15] Rosen Diankov, Nathan Ratliff, Dave Ferguson, Siddhartha Srinivasa, and James Kuffner. Bispace planning: Concurrent multi-space exploration. *Proceedings of Robotics: Science and Systems IV*, 63, 2008.

[16] Jan Jockusch and Helge Ritter. An instantaneous topological mapping model for correlated stimuli. In *Proc. Int. Joint Conf. on Neural Netw. (IJCNN 99)*, page 445, 1999.

[17] Bernd Fritzke. A growing neural gas network learns topologies. In *Advances in Neural Information Processing Systems 7*, pages 625–632. MIT Press, 1995.

[18] Teuvo Kohonen. The self-organizing map. *Proceedings of the IEEE*, 78(9):1464–1480, 1990.

[19] Stavros D. Nikolopoulos and Leonidas Palios. Hole and antihole detection in graphs. In *Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms*, SODA '04, pages 850–859, Philadelphia, PA, USA, 2004. Society for Industrial and Applied Mathematics.

[20] Ján Jockusch. *Exploration based on neural networks with applications in manipulator control.* PhD thesis, Bielefeld University, 2000.

[21] Sebastian Wrede. *An information-driven architecture for cognitive systems research*. PhD thesis, Bielefeld University, 2008.

[22] *Software Abstractions for Simulation and Control of a Continuum Robot*, volume 7628, Tsukuba, Japan, 11/2012 2012. Springer.

[23] Johannes Wienke and Sebastian Wrede. A middleware for collaborative research in experimental robotics. In *2011 IEEE/SICE International Symposium on System Integration (SII)*, pages 1183–1190. IEEE, 2011.

[24] Rosen Diankov and James Kuffner. Openrave: A planning architecture for autonomous robotics. *Robotics Institute, Pittsburgh, PA, Tech. Rep. CMU-RI-TR-08-34*, page 79, 2008.

[25] P.E. Hart, N.J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[26] E.W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271, 1959.

[27] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes in C (2Nd Ed.): The Art of Scientific Computing*. Cambridge University Press, New York, NY, USA, 1992.

[28] Eric Jones, Travis Oliphant, Pearu Peterson, et al. SciPy: Open source scientific tools for Python. `"http://www.scipy.org/"`, 2001–.

[29] Paul Dierckx. Algorithms for smoothing data with periodic and parametric splines. *Computer Graphics and Image Processing*, 20(2):171–184, 1982.

[30] Biljana Petreska and Aude Billard. Movement curvature planning through force field internal models. *Biological cybernetics*, 100(5):331–350, 2009.

[31] David Hoag. Apollo guidance and navigation: Considerations of apollo imu gimbal lock. *Canbridge: MIT Instrumentation Laboratory*, pages 1–64, 1963.

[32] Jack B Kuipers. *Quaternions and rotation sequences*. Princeton university press, 1999.

[33] Berthold KP Horn. Some notes on unit quaternions and rotation. 2001.

[34] Erik B Dam, Martin Koch, and Martin Lillholm. *Quaternions, interpolation and animation*. Datalogisk Institut, Københavns Universitet, 1998.

[35] DWF Van Krevelen and R Poelman. A survey of augmented reality technologies, applications and limitations. *International Journal of Virtual Reality*, 9(2):1, 2010.

[36] K. Murakami, R. Kiyama, T. Narumi, T. Tanikawa, and M. Hirose. Poster: A wearable augmented reality system with haptic feedback and its performance in virtual assembly tasks. In *2013 IEEE Symposium on 3D User Interfaces (3DUI)*, pages 161–162, 2013.

[37] Alin Albu-Schäffer, Christian Ott, and Gerd Hirzinger. A unified passivity-based control framework for position, torque and impedance control of flexible joint robots. *The International Journal of Robotics Research*, 26(1):23–39, 2007.

[38] Aidan O'Dwyer. *Handbook of PI and PID controller tuning rules*, volume 2. Imperial College Press London, 2009.

Declaration of Authorship

I declare that the work presented here is, to the best of my knowledge and belief, original and the result of my own investigations, except as acknowledged. Formulations and ideas taken from other sources are cited as such. It has not been submitted, either in part or whole, for a degree at this or any other university.

Ich versichere, dass ich die vorliegende wissenschaftliche Arbeit selbständig verfasst und keine anderen als die angegebenen Hilfsmittel verwendet habe. Die Stellen der Arbeit, die anderen Werken dem Wortlaut oder dem Sinn nach entnommen sind, wurden unter Angabe der Quelle als Entlehnung deutlich gemacht. Diese Arbeit hat in gleicher oder ähnlicher Form meines Wissens nach noch keiner Prüfungsbehörde vorgelegen.

Bielefeld, January 16, 2014

Daniel Seidel