
THE UNIVERSITY
*of* EDINBURGH

# Motion Synthesis for High Degree-of-Freedom Robots in Complex and Changing Environments

Yiming Yang


*Doctor of Philosophy*

School of Informatics
College of Science and Engineering
University of Edinburgh

2017

Yiming Yang:

*Motion Synthesis for High Degree-of-Freedom Robots in Complex and Changing Environments*

Doctor of Philosophy, 2017 ©

SUPERVISOR:

Sethu Vijayakumar

LOCATION:

Edinburgh, United Kingdom

## ABSTRACT

The use of robotics has recently seen significant growth in various domains such as unmanned ground/underwater/aerial vehicles, smart manufacturing, and humanoid robots. However, one of the most important and essential capabilities required for long term autonomy, which is the ability to operate robustly and safely in real-world environments, in contrast to industrial and laboratory setup is largely missing. Designing robots that can operate reliably and efficiently in cluttered and changing environments is non-trivial, especially for high degree-of-freedom (DoF) systems, i.e. robots with multiple actuators. On one hand, the dexterity offered by the kinematic redundancy allows the robot to perform dexterous manipulation tasks in complex environments, whereas on the other hand, such complex system also makes controlling and planning very challenging. To address such two interrelated problems, we exploit robot motion synthesis from three perspectives that feed into each other: end-pose planning, motion planning and motion adaptation. We propose several novel ideas in each of the three phases, using which we can efficiently synthesise dexterous manipulation motion for fixed-base robotic arms, mobile manipulators, as well as humanoid robots in cluttered and potentially changing environments.

Collision-free inverse kinematics (IK), or so-called end-pose planning, a key prerequisite for other modules such as motion planning, is an important and yet unsolved problem in robotics. Such information is often assumed given, or manually provided in practice, which significantly limiting high-level autonomy. In our research, by using novel data pre-processing and encoding techniques, we are able to efficiently search for collision-free end-poses in challenging scenarios in the presence of uneven terrains.

After having found the end-poses, the motion planning module can proceed. Although motion planning has been claimed as well studied, we find that existing algorithms are still unreliable for robust and safe operations in real-world applications,

especially when the environment is cluttered and changing. We propose a novel resolution complete motion planning algorithm, namely the Hierarchical Dynamic Roadmap, that is able to generate collision-free motion trajectories for redundant robotic arms in extremely complicated environments where other methods would fail. While planning for fixed-base robotic arms is relatively less challenging, we also investigate into efficient motion planning algorithms for high DoF ($30 - 40$) humanoid robots, where an extra balance constraint needs to be taken into account. The result shows that our method is able to efficiently generate collision-free whole-body trajectories for different humanoid robots in complex environments, where other methods would require a much longer planning time.

Both end-pose and motion planning algorithms compute solutions in static environments, and assume the environments stay static during execution. While human and most animals are incredibly good at handling environmental changes, the state-of-the-art robotics technology is far from being able to achieve such an ability. To address this issue, we propose a novel state space representation, the Distance Mesh space, in which the robot is able to remap the pre-planned motion in real-time and adapt to environmental changes during execution.

By utilizing the proposed end-pose planning, motion planning and motion adaptation techniques, we obtain a robotic framework that significantly improves the level of autonomy. The proposed methods have been validated on various state-of-the-art robot platforms, such as UR5 (6-DoF fixed-base robotic arm), KUKA LWR (7-DoF fixed-base robotic arm), Baxter (14-DoF fixed-base bi-manual manipulator), Husky with Dual UR5 (15-DoF mobile bi-manual manipulator), PR2 (20-DoF mobile bi-manual manipulator), NASA Valkyrie (38-DoF humanoid) and many others, showing that our methods are truly applicable to solve high dimensional motion planning for practical problems.

# 摘要

在过去的几年，尤其是在无人车、无人机、智能制造以及人形机器人等领域，关于机器人的使用和研究受到了广泛的关注。然而，一个关系到能否实现机器人长时间自治的关键问题仍然没有得到解决，即如何让机器人能够在复杂、动态的环境下安全稳定的运作，而非像工业和实验室机器人那样。设计能够在复杂、动态的环境下稳定工作的机器人并非易事，尤其是对于较高自由度的冗余系统，即由多个电机控制的机器人。一方面，冗余的机械结构提升了系统的灵活性，使其能够在复杂环境下完成灵巧的操作；但另一方面，系统的冗余性也带来了规划和控制上的困难。为了解决这两个内在关联的问题，此论文从三个依次递进的角度来研究机器人运动合成：终点位姿规划 (End-Pose Planning)，运动规划 (Motion Planning)，以及在线运动适应 (Motion Adaptation)。通过在每一个领域提出一系列新的观点并进行整合，我们能够在复杂多变的环境中快速的为机械臂及人形机器人规划出安全有效的运动轨迹。

无碰撞逆运动学，即终点位姿规划，是调用运动规划或步态规划等其他模块的重要前提条件，同时也是一个尚未解决的问题。此类信息通常都是假设已知，并在实际应用中由操作人员手动输入，这显然严重的限制了高度自治系统的发展。在此论文的研究中，通过提出独特的数据预处理和编码技术，我们能够在包含崎岖路面等富有挑战的复杂环境中快速计算出无碰撞的有效终点位姿。

运动规划模块将在成功获得有效终点位姿之后得以实施。尽管有学者认为运动规划问题已经被较好的解决，我们仍然发现在很多情况下，尤其当环境非常复杂且变化时，已有的运动规划算法无法解决实际问题。此论文提出了全新的分辨率完整的运动规划算法，即层级动态路线图 (Hierarchical Dynamic Roadmap)，其能够在极度复杂的环境中实时的为多关节冗余机械臂计算无碰撞轨迹，而这一功能是其他现有算法所无法实现的。解决固定基座机械臂的运动规划问题难度相对较低，我们同时也对高自由度（30 – 40维度）的人形机器人运动规划问题进行了研究，这需要将双足平衡时刻考虑在内。实验结果表明，我们提出的算法能够在复杂的环境中更快速的为不同的人形机器人规划出平衡且无碰撞的全身运动轨迹。

终点位姿规划和运动规划都在静态的环境中进行计算，并假设环境在机器人运动过程中保持静止。人类和大多数动物拥有令人难以置信的应对环境变化的能力，然而

目前最先进的机器人系统也远不具备这种能力。为了解决这一问题，此论文提出了独特的状态空间表达方式，即距离网孔 (Distance Mesh)，其能够在机器人运动过程中实时的改变预先规划的轨迹以适应环境的变化。

通过整合此论文所提出的终点位姿规划算法、运动规划算法以及在线运动适应算法，我们构建了一个能显著提高机器人自主程度的系统框架。我们所提出的方法在多种不同的机器人平台上得到了验证，例如UR5（6自由度固定基座机械臂）、KUKA LWR（7自由度固定基座冗余机械臂）、Baxter（14自由度固定基座双臂机器人）、Husky with Dual UR5（15自由度移动双臂机器人），PR2 （20自由度移动双臂机器人）以及NASA Valkyrie（38自由度人形机器人）等，显示了我们的算法能够被应用于解决多自由度机器人系统的运动规划问题。

# DECLARATION

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

*Edinburgh, United Kingdom, 2017*

Yiming Yang

March 28, 2018

# PUBLICATIONS

Some ideas and figures in this thesis have been published in the following articles:

Y. Yang, W. Merkt, V. Ivan, Z. Li, and S. Vijayakumar. HDRM: A Resolution Complete Dynamic Roadmap for Realtime Motion Planning in Complex Environments. In *IEEE Robotics and Automation Letter* (RA-L), 3(1):551–558, 2018.

V. Ivan, Y. Yang, W. Merkt, M. Camilleri, and S. Vijayakumar. EXOTica: a library for easy creation of tools for optimisation and planning. In *Robot Operating System (ROS) Volume 3*, Springer, 2018.

Y. Yang, W. Merkt, H. Ferrolho, V. Ivan, and S.Vijayakumar. Efficient Humanoid Motion Planning on Uneven Terrain Using Paired Forward-Inverse Dynamic Reachability Maps. In *IEEE Robotics and Automation Letter* (RA-L), 2(4):2279–2286, 2017.

W. Merkt, Y. Yang, T. Stouraitis, C. E. Mower, M. Fallon, S. Vijayakumar. Robust Shared Autonomy for Mobile Manipulation with Continuous Scene Monitoring. In *Proceeding of IEEE Conference on Automation Science and Engineering (CASE)*, pages 130-137, Xi'an, China, 2017.

Q. Li, I. Chatzinikolaidis, Y. Yang, S. Vijayakumar and Z. Li, Robust foot placement control for dynamic walking using online parameter estimation. In *Proceeding of IEEE/RAS International Conference on Humanoid Robotics (Humanoids)*, pages 165-170, Birmingham, UK, 2017.

Y. Yang, V. Ivan, Z. Li, M. Fallon and S. Vijayakumar. iDRM: Humanoid motion planning with realtime end-pose selection in complex environments. In *Proceeding of IEEE/RAS International Conference on Humanoid Robots (Humanoids)*, pages 271–278, Cancun, Mexico, 2016.

Y. Yang, V. Ivan, W. Merkt and S. Vijayakumar. Scaling sampling-based motion planning to humanoid robots. In *Proceeding of IEEE International Conference on Robotics and Biomimetics (ROBIO)*, pages 1448–1454, Qingdao, China, 2016.

Y. Yang, V. Ivan and S. Vijayakumar. Real-time motion adaptation using relative distance space representation. In *Proceeding of IEEE International Conference on Advanced Robotics (ICAR)*, pages 21–27, Istanbul, Turkey, 2015.

A. Orthey, V. Ivan, M. Naveau, Y. Yang, O. Stasse, and S. Vijayakumar. Homotopic particle motion planning for humanoid robotics. Rapport LAAS n° 15107, 2015.

# VIDEOS

Some videos highlighting notable outcomes of the thesis are available at following links:

Title: *Real-Time Motion Planning in Complex Scenes using Hierarchical Dynamic Roadmap*
URL: https://youtu.be/4AzbmiTI1iE

Title: *Efficient Humanoid Motion Planning on Uneven Terrain*
URL: https://youtu.be/o-05EHf-gg8

Title: *Robust Shared Autonomy for Mobile Manipulation with Continuous Scene Monitoring*
URL: https://youtu.be/5jFU7oCP4vk

Title: *Humanoid Motion Planning with Real-Time End-Pose Selection in Complex Scenes*
URL: https://youtu.be/e09B-hAVKNs

Title: *Scaling Sampling–based Motion Planning to Humanoid Robots*
URL: https://youtu.be/AZQY_QOX0Pw

Title: *Real-Time Motion Adaptation using Relative Distance Space Representation*
URL: https://youtu.be/A1dhiLyLo5U

Title: *Real-Time Motion Transfer Using Interaction Mesh*
URL: https://youtu.be/bFXhAILI91c

# ACKNOWLEDGEMENTS

Firstly, I would like to express my sincere gratitude and thanks to my supervisor Prof. Sethu Vijayakumar for providing this opportunity and the continuous support, for his patience, motivation, and immense knowledge. His guidance helped me in all the time of research and writing of this thesis. I would also like to express my appreciation to Dr. Taku Komura, Dr. Michael Herrmann, Dr. Maurice Fallon, Dr. Subramanian Ramamoorthy, and Dr. Zhibin Li, for their valuable insight and help in various domains in robotics.

Secondly, I would like to thank all the brilliant roboticists who I have worked with over the past four years, namely Dr. Vladimir Ivan, Dr. Jose Cano Reyes, Wolfgang Merkt, Henrique Ferrolho, Theodoros Stouraitis, Chris Mower, Qingbiao Li and all other members from the Statistical Machine Learning and Motor Control Group (SLMC) for their support and friendship. I would also like to thank the Institute of Perception, Action and Behaviour (IPAB), School of Informatics and Edinburgh Centre for Robotics (ECR) for offering me the great opportunity to work with cutting-edge robot platforms and other equipments, and of course, special thanks to all the robots for their cooperation and dedication.

I would like to thank Dr. Kiyoto Ito, Dr. Nobutaka Kimura, and other members from the robotics group at Hitachi Central Research Laboratory, for providing me the opportunity to better understand the difference and tight connection between academia and industry.

I would like to express my special thanks to Rui Song, Lijun Teng, Xinnuo Xu and Rui Li for their priceless friendship and making the past four years of my PhD a pleasant memory.

Finally, but by no means the least, I would like to thank my parents. I would not be able to come this far without their faith in me and the unfailing support and encouragement.

致吾妻。

# CONTENTS

# LIST OF FIGURES

# LIST OF TABLES

# 1 INTRODUCTION

The vision of advanced robots, such as dexterous manipulators, bipedal or mobile-based[1] humanoids, derives from the dream that one day they can accomplish universal tasks or even co-exist with mankind, to release humans from heavy and dangerous works. However, the state-of-the-art technology is still far from being able to fulfil such vision. Existing robot systems mostly operate in industrial settings under the assumptions that — the robots only have six or less Degree-of-Freedom (DoF) with the base bolted to the ground; the environment is simple and static which the robots have full knowledge of; finally and in fact most importantly, the robots are deployed to positions require repeatable but rudimentary motions with minimal dexterous capabilities.

Such restricted industrial setup is most imposed by systematic and environmental complexities. The systematic complexity refers to the high-dimensional kinematic structure, potentially with a wheeled or bipedal base, which offers excellent dexterity but in turn, its complexity also makes the motion synthesis extremely challenging, in particular for safe and reactive tasks in complex environments and in close proximity to people. The environmental complexity refers to the cluttered and changing environment around the robot. We human, and most animals are incredibly good at handling complex environmental objects and unexpected perturbations during actions, but most of the current robotic systems are incapable of achieving anything close to such capability. The systematic and environmental complexities are different but related, the complex system that makes planning difficult sometimes is urgently needed to overcome problems such as avoiding obstacles in complex environments.

How can we efficiently generate reliable motion plans for high DoF robots, and how can we give them the ability to adapt to run-time perturbations? To address

1 Traditionally, humanoid refers to a human-like robot, i.e. with two arms and two legs. However, we sometimes call a system with two arms but a mobile base also a humanoid robot.

these questions and toward better robot autonomy, the solutions fall into two areas — efficient high-dimensional motion planning and real-time online motion adaptation.

## 1.1 MOTION PLANNING

When designing a robot system, one of the core functionality we should consider is for the robot to know how to move, which is called *motion planning* (LaValle [2006]), a fundamental problem in robotics which involves automatically finding a sequence of actions that takes the robot from a start to a goal state. There are two key features in motion planning, optimality and completeness. Optimality describes the quality of the plan with respect to defined cost metrics, i.e. can the system find the best and optimal way to move to a goal state. Completeness reflects the robustness of the system, i.e. is the system able to solve all possible problems in different scenarios. Generally speaking, motion planning algorithms can be grouped into optimization-based and search-based approaches. Optimization-based algorithms (Ratliff et al. [2009], Ivan et al. [2013], Schulman et al. [2014]) solve mostly convex optimisation problems to find locally optimised solutions, whereas search-based algorithms (Lavalle [1998], Kuffner and LaValle [2000], Kavraki et al. [1996], Elbanhawi and Simic [2014]) search through the entire configuration space to find globally valid solutions. It is clear that optimization-based methods aim for optimality and search-based methods are designed for achieving completeness. Optimization-based approaches could get stuck in local minima and fail to produce valid solution when the problem is non-convex or ill-defined, e.g. in complex and cluttered environments Koren and Borenstein [1991]. On the other hand, search-based approaches, asymptotically, promise to find valid solutions for complex problems. Thus, in this thesis we mainly explore search-based algorithms for solving complex problems in difficult scenarios, where the globally valid solution can be further optimized afterwards — as a warm start for optimization-based methods.

Search-based algorithms are typically used in path planning problems for mobile robots (Karaman et al. [2011], Nasir et al. [2013]) and in motion planning problems for fixed-base low DoF robotic arms (Murray et al. [2016]). Redundant robotic arms,

mobile manipulators and humanoid robots, on the other hand, have complex structures that makes searching much more challenging and time consuming (Kuffner et al. [2005]). Also, traditional search-based methods are incapable of efficiently providing balanced configurations for legged systems. Thus, optimization-based rather than sampling-based algorithms, are commonly used on humanoids to generate feasible balanced motion (Dai et al. [2014]). Such approach works fine in simple and static environments, however, optimization-based methods face great challenges providing valid motion in complex environments with cluttered and moving obstacles.

## 1.2 END-POSE PLANNING

Apart from classical motion planning, another challenging but interesting problem arises with high-DoF redundant robots — *End-Pose Planning*, or equivalently, collision-free inverse kinematics (IK). Normally, the task is defined in the workspace, e.g. a desired pose for the end-effector to reach, but the actual planning is mostly carried out in joint configuration space. The start state is known a priori, which normally is the current state, but the goal state that is constrained by the workspace task is unknown. Thus, as the very literal interpretation, end-pose planning is a procedure for finding the goal state, or so-called end-pose, in the joint configuration space that satisfies the task's workspace constraints. The end-pose will be then used as the input goal state for motion planning.

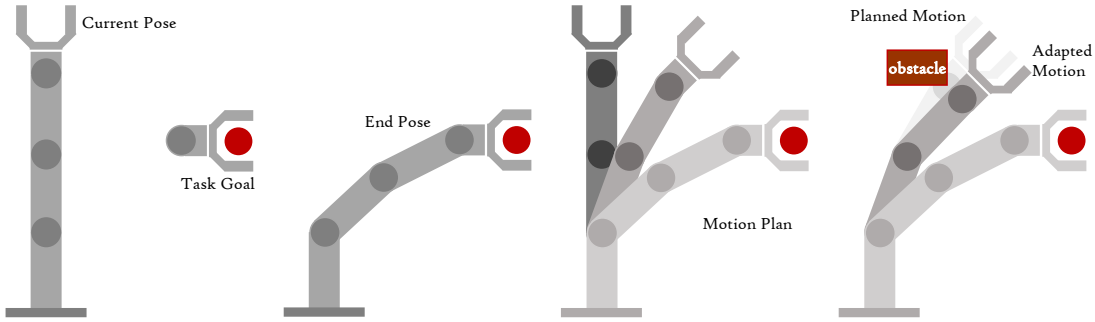
While motion planning is a well known area that has been extensively studied, we find that the research on end-pose planning has been lagged behind. One possible explanation could be that, in classical problems with non-redundant robotic systems, a given workspace goal can have at most one configuration space solution, which makes considering collision-free constraint unnecessary. However, for redundant systems (typically with seven or more actuators), there exist multiple end-poses that satisfy an identical workspace pose constraint (Siciliano [1990], Henten et al. [2010]), some of which might be invalid due to self-collision or collision with the environmental objects. Hence, how to efficiently find a sufficient valid end-pose becomes critical for robot operating in complex environments. Such problem is exacerbated

on humanoid robots with the extra redundancy introduced by the moveable base. For example, when reaching a close object, we would consider walk directly to the target; however, in cases where the object is placed at a distance or behind an obstacle, we would need to walk around the obstacle first before taking the reaching action. In complex environments with multiple collision objects, the most practical solution is to let a human operator to hand-craft such information (Koolen et al. [2013], Tedrake et al. [2014], Oh et al. [2017]), which becomes the limiting factor for high-level and long-term autonomy. Thus, how to find appropriate and potentially optimal standing locations becomes an important and interesting problem for humanoids or mobile manipulators.

## 1.3 MOTION ADAPTATION

After having planned the motion, the robot starts to execute the trajectory and hopes to accomplish the task without violating any constraints, e.g. collision-free, balance, end-effector pose, etc. This is true only under the assumption that the environment and target stay static, and the robot has perfect motor control and sensing systems, which is not the case in real-world scenarios. Various types of uncertainties exist in practice, such as inaccurate motor execution, sensory noise, perturbations and changing environments, that make the goal unachievable by naïvely following the planned trajectory. Online adaptation techniques must be applied to compensate any potential uncertainties in the system and environment. Replanning is the most trivial approach for handling during-execution changes (Karaman et al. [2011]), however, online replanning is too expensive to be applied on high-DoF robots.

A particular joint configuration corresponds to unique robot rigid body posture in the world and vice versa, any environmental changes that collide with the robot posture also invalidate the configuration space state, thus configuration space trajectory by nature can not adapt to changes. By using surjective but not injective alternative spaces to configuration space mapping (Ho and Komura [2009], Al-Asqhar et al. [2013], Ho et al. [2014]), one obtains so-called alternative space trajectory that has multiple corresponding configuration space and workspace trajectories. An alternate




(a) Problem setup (b) End-pose planning (c) Motion planning (d) Motion adaptation

Figure 1.1: Motion synthesis overview. (a) Problem setup, a target location is given in workspace; (b) the robot needs to first find a configuration space end-pose that satisfy the goal constraint; (c) then the robot needs to plan a motion that supposedly to reach the target; (d) finally, during execution, the robot needs to actively modify the planned motion to adapt to any unforeseen perturbations.

space state may still be valid even if some of the corresponding configurations become invalid, thus the whole alternate space trajectory is still valid. Work has been done applying such property on online motion adaptation (Ho et al. [2010a], Ivan et al. [2013]). However, a majority of the proposed alternative spaces only deal with internal changes and pre-known environmental objects. Very few ones, if any, are capable of handling unexpected objects.

## 1.4 PROBLEM STATEMENT

We have so far presented the problems and limitations in motion synthesis for high-DoF robots: efficient motion planning, end-pose planning and real-time online adaptation. A more formal definition is given as

$$EndPose = \textbf{EndPosePlanning}(CurrentState, TaskGoal, Environment), \tag{1.1}$$

$$MotionPlan = \textbf{MotionPlanning}(CurrentState, EndPose, Environment), \tag{1.2}$$

and

$$MotorCommand = \textbf{MotionAdaptation}(MotionPlan, CurrentEnvironment). \tag{1.3}$$

An overview of the whole process is highlighted in Figure 1.1. Given a workspace task, i.e. reach and grasp the red target, as shown in Figure 1.1a. The first step to find the end-pose that is collision-free and reaches the target, as shown in Figure 1.1b. After having the end-pose found, the next step is to plan a motion that can potentially transit the robot from current state to the end-pose, as shown in Figure 1.1c. The last step is to execute the planned motion, however, the robot needs to adjust the original plan to adapt to any runtime changes, which is the online motion adaptation phase, as shown in Figure 1.1d.

For legged or wheeled robots, the MotionPlanning() can be further divided into firstly

$$\mathit{BaseMovementPlan} = \textbf{BasePlanning}(\mathit{CurrentBase}, \mathit{DesiredBase}, \mathit{Environment}), \quad (1.4)$$

where the BasePlanning() refers to walking planning for legged system and navigation planning for wheeled system, the *DesiredBase* can be extracted from *EndPose*. After having arrived at the desired base location, a fixed-base motion planning is then invoked

$$\mathit{MotionPlan} = \textbf{MotionPlanning}(\mathit{StateAfterBaseMovement}, \mathit{EndPose}, \mathit{Environment}). \quad (1.5)$$

## 1.5 THESIS STRUCTURE

Different approaches have been proposed attempting to address these problems, e.g. Kuffner et al. [2005], Yoshida [2005], Park et al. [2014], Dai et al. [2014]. However, most of the existing work has only demonstrated a proof of concept in simulation or simple environments, which is not yet ready to be applied to complex scenarios. In this thesis, we will try to address each of these problems while considering very complex, cluttered and changing environments. Figure 1.2 layouts the thesis structure. The main body is organized in three parts — Part I, End-Pose Planning; Part II, Motion Planning; and Part III, Motion Adaptation. In each part, we first discuss preliminary and related work, followed by a series of novel contributions, respectively.

While the goal is assumed given when working with classical motion planning algorithms such as Rapidly-exploring Random Tree (RRT, Lavalle [1998]) and Probabilistic Roadmap (PRM, Kavraki et al. [1996]), we start with arguing how to firstly find the goal, i.e. the end-pose planning. A valid end-pose is a feasible robot state[2] that satisfy necessary task constraints, e.g. the end-effector reaches the target pose. While existing work mainly focuses on solving the kinematic reachability problem in simple environments (Zacharias et al. [2013], Vahrenkamp et al. [2013]), in Chapter 3, we introduce a novel dynamic reachability map method that enables real-time end-pose planning capability in complex environments. By using a novel configuration-to-workspace encoding and indexing technique, we are able to store a sufficient number of high dimensional configurations and their collision information, which can be quickly updated during run-time. In Chapter 4, we further extend the method to plan end-poses for bi-manual manipulation problems with uneven terrains by utilizing the forward and inverse dynamic reachability maps.

After having the end-pose found, motion planning algorithms can then take the end-pose as input to generate a motion plan. Although motion planning has been claimed as well studied (Elbanhawi and Simic [2014]), we find existing methods are inefficient for complex environments, especially with high dimension systems and complex environments. In Chapter 6, we introduce a novel resolution complete planner, namely the hierarchical dynamic roadmap, that is able to plan valid motion for high DoF robots in complex and cluttered environments in real-time. The motion planning problem is exacerbated on legged systems with the extra balance constraint, where the average planning would be significantly increased. In Chapter 7, we proposed a generic method that scales up existing methods for solving motion planning problems for humanoids in complex environments. Result shows that the proposed method is much more efficient than existing approaches and easily applicable onto different humanoid platforms.

While end-pose and motion planning algorithms compute solutions in static environments[3], the environment might change during execution and the robot needs to

2 A feasible robot state normally refers to a collision-free state, and balanced state for legged robots.
3 The environments are different between planning requests, but assumed to be static during each request.

adjust the pre-planned motion to adapt to the changes. In contrast to exiting methods that only deals with only internal model discrepancies (Ivan et al. [2013]) or relies on heavy computation (Pan and Manocha [2011]), in Chapter 9, we introduce a novel distance mesh representation is that enables real-time motion adaptation in unstructured and changing environments.

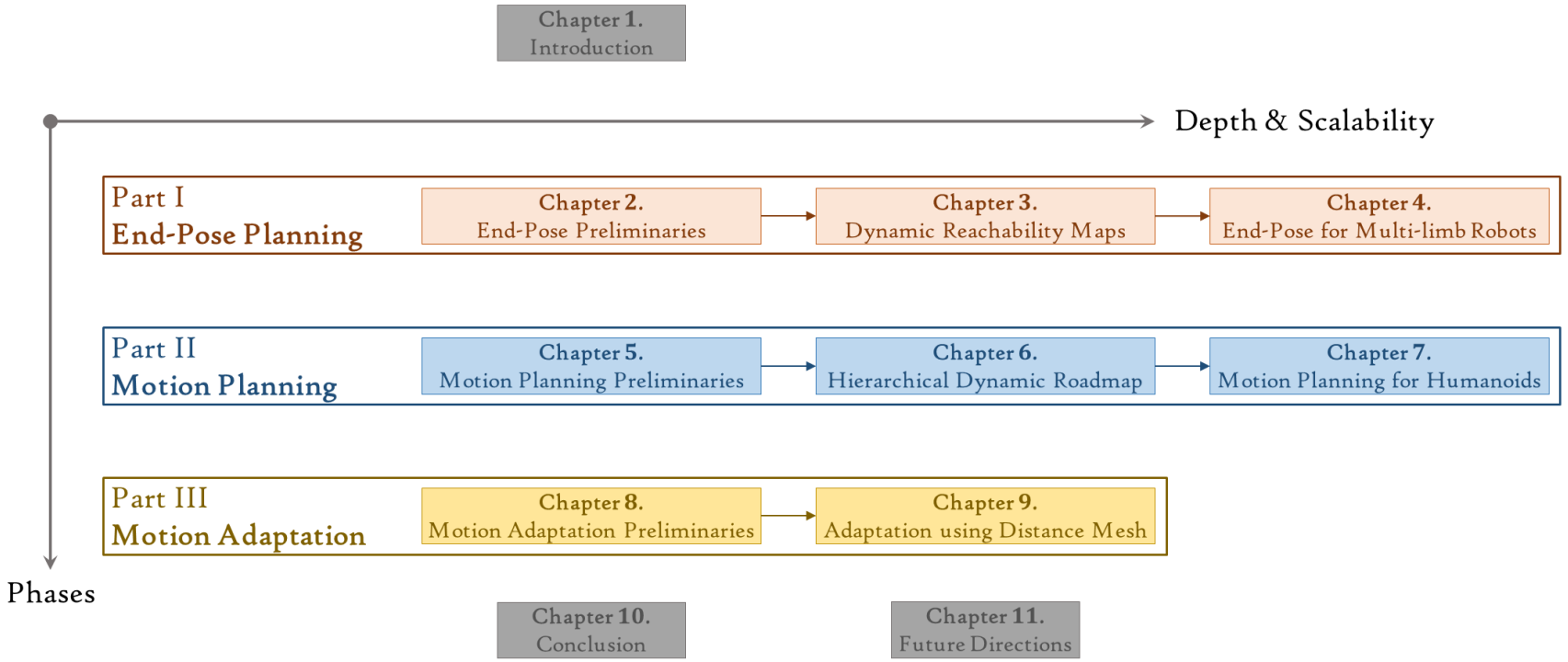



Figure 1.2: Thesis structure layout. The vertical components, e.g. Part I-III, advance to different phases in the motion synthesis framework (Figure 1.1). The horizontal components advance to more challenging problems in the particular phase respectively.

## 1.6 CONTRIBUTIONS

The main contributions of this thesis include:

- The introduction and generalization of reachability map (RM), inverse reachability map (IRM) and their dynamic versions, i.e. **dynamic reachability map** (DRM) and **inverse dynamic reachability map** (IDRM).
- A novel end-pose planning algorithm using IDRM that allows the robot to, in real-time, select sufficient standing location and full-body configuration in complex and cluttered environments. Yang et al. [2016a]
- An extension of the original IDRM end-pose planning method (only works on flat ground) that utilizes forward and inverse reachability maps, which enables bi-manual end-pose planning for humanoid robots in complex environments with uneven terrains. Yang et al. [2017]
- A novel motion planning method, the **Hierarchical Dynamic Roadmap** (HDRM), which is a resolution complete planning algorithm that is able to plan valid motion for high dimensional robotic arms in real-time. Yang et al. [2018]
- A deeply customized sampling-based planning algorithm that efficiently generates balanced and collision-free full-body motion for humanoid robots in complex environments. Yang et al. [2016b]
- A novel alternate space representation namely the **Distance Mesh** (dMesh), that allows the robot to adapt to unexpected changes during execution. Yang et al. [2015]
- A software framework, the **EXtensible Optimization Toolset** (EXOTica), that provides easy access for using/benchmarking/developing different algorithms and applications for robotic planning and control. All of the aforementioned contributions are prototyped, implemented and tested using the EXOTica framework. Ivan et al. [2018]

# Part I

# END-POSE PLANNING

# 2

# END-POSE PLANNING PRELIMINARIES

In classical robot motion planning, we often say "*given the start and goal states*". However, in most cases, we only know the start state, e.g. current state, but the goal state is usually unknown or indirectly defined. So, before diving into motion planning (Part II), in this part of the thesis, we would like to first discuss how to find the appropriate goal state, which is referred to as end-pose planning.

## 2.1 PROBLEM FORMULATION

Let $\mathbf{q} \in \mathbb{R}^N$ be a $N$ dimensional vector denote a configuration, i.e. the position values of all joints, of a robot with $N$ joints. Let $\mathcal{C}$ denote the set of all possible configurations, which is called the *configuration space* where

$$\mathbf{q} \in \mathcal{C} \subseteq \mathbb{R}^N. \tag{2.1}$$

Motion planning algorithms (Equation 1.2) normally require a start state $\mathbf{q}_{\text{start}}$ and a goal state $\mathbf{q}_{\text{goal}}$ as inputs, where the former is the current state which is known, but the latter is often unknown.

Let $\mathcal{W} \subseteq \mathbb{R}^W$ denote the *workspace* that the robot is operated in where normally $W = 6$, i.e. 3 for position and 3 for orientation. Let $\mathcal{A}(\mathbf{q}) \subset \mathcal{W}$ denote the robot rigid body posture in the workspace at configuration $\mathbf{q}$ and $\mathcal{O} \subset \mathcal{W}$ be the workspace region that is occupied by environmental objects. The *obstacle region* in configuration space,

$$\mathcal{C}_{\text{obs}} = \{\mathbf{q} \in \mathcal{C} \mid \mathcal{A}(\mathbf{q}) \cap \mathcal{O} \neq \emptyset\}, \tag{2.2}$$

is the set of all configurations, $\mathbf{q}$, at which $\mathcal{A}(\mathbf{q})$ collides with obstacles $\mathcal{O}$. The remaining region of configuration space is called the *free region*,

$$\mathcal{C}_{\text{free}} = \mathcal{C} \setminus \mathcal{C}_{\text{obs}}. \tag{2.3}$$

It is obvious that all valid and feasible robot configurations must be in the free region, otherwise the robot will be in collision with the environment. However, not all configurations in the free region are necessarily valid since it only guarantees collision-free constraint between the robot and environment, whereas the validity of a configuration also depends on other constraints, such as self-collision-free, balance, etc. It is convenient to denote the set of all valid configurations as $\mathcal{C}_{\text{valid}} \subseteq \mathcal{C}_{\text{free}}$.

In robotics, kinematics is used to describe the motion of a systems composed of joined parts. The *forward kinematics* (FK) is defined as

$$\begin{aligned} \mathbf{y} = \text{FK}(\mathbf{q}) : \mathcal{C} &\longmapsto \mathcal{W} \\ \mathbb{R}^N &\longmapsto \mathbb{R}^W \end{aligned} \tag{2.4}$$

where $\mathbf{y} \in \mathcal{W}$ is the end-effector pose, e.g. tool pose of a robotic arm, hand or foot pose of a humanoid robot, at configuration $\mathbf{q}$. The inverse problem, *inverse kinematics* (IK),

$$\begin{aligned} \mathbf{q} = \text{IK}(\mathbf{y}) : \mathcal{W} &\longmapsto \mathcal{C} \\ \mathbb{R}^W &\longmapsto \mathbb{R}^N \end{aligned} \tag{2.5}$$

is a mapping from workspace to configuration space, where the output $\mathbf{q}$ is a configuration at which the end-effector reaches $\mathbf{y}$.

The robot system is called *non-redundant* if $N \leq W$ and *redundant* if $N > W$. When $N < W$, for any input $\mathbf{y}$, there may or may not exist a valid output $\mathbf{q}$; and when $N = W$, for every input $\mathbf{y}$, there exists one and only one unique output $\mathbf{q}$. However, when the robot is redundant, i.e. $N > W$, for an input $\mathbf{y}$, there exists infinite valid solutions. More intuitively, this means the robot has more DoFs than strictly necessary to perform the task, e.g. a 7-DoF robotic arm is considered redundant for reaching desired end-effector pose $\mathbf{y}^* \in SE(3) \subset \mathbb{R}^6$ since $7 > 6$. In such case, the robot can exploit the extra/redundant DoFs to achieve secondary tasks, such as avoid obstacles,

increase manipulability, and reduce power consumption. In most practical situations, avoiding collision is one of the top priority tasks. However, to find an IK solution that satisfy end-effector pose constraint $\mathbf{y}^*$ and also collision-free is non-trivial, especially in complex and cluttered environments. Hence, the process of finding valid IK solution, or so-called end-pose planning,

$$\mathbf{q}_{\text{goal}} = \textbf{EndPosePlanning}(\mathbf{y}^*) : \quad \mathcal{W} \longmapsto \mathcal{C}_{\text{valid}}, \tag{2.6}$$

becomes a key prerequisite, where other actions such as motion planning can only proceed after a valid end-pose, $\mathbf{q}_{\text{goal}}$, has been found. Note that the problem is exacerbated as the type of the robot base considered changes. For example, a mobile manipulator has an extra 3 DoFs ($SE(2)$) introduced by the planner base; similarly, a humanoid robot with two legs has extra 12 DoFs ($SE(3)$ for each foot).

## 2.2 COLLISION-FREE INVERSE KINEMATICS

Collision-free IK is an old, yet unsolved problem in robotics. For redundant robots in particular, there exist an infinite number of solutions, which makes it very difficult to get a closed form solution. Buss [2004] showed that for algorithms such as Jacobian Transpose, Pseudo-Inverse, Damped Least Squares, etc., one common fact is that different initial states lead to different IK solutions. Hence, a naïve collision-free IK approach is to randomly restart with different initial seeds until a collision-free solution is found. Obviously, such method is inefficient, especially in cluttered environments. Numerical methods are normally used by applying dexterity measures and Jacobian inverse (Guo and Hsia [1990], Sciavicco and Siciliano [1988]). Pre-computation is also heavily used in collision-free IK (Zacharias et al. [2013], Torres et al. [2014]) for speeding up online computation by providing good initial hint of the seed pose. Machine learning schemes such as Locally Weighted Projection Regression (D'Souza et al. [2001]) and Neural Network (Mao and Hsia [1997]) are also used for learning inverse kinematics, with or without considering obstacles in the environment. Leibrandt et al. [2015] showed that, with advanced computer hardware, real-time online collision-free IK computation can be achieved for a tube robot. However, most of

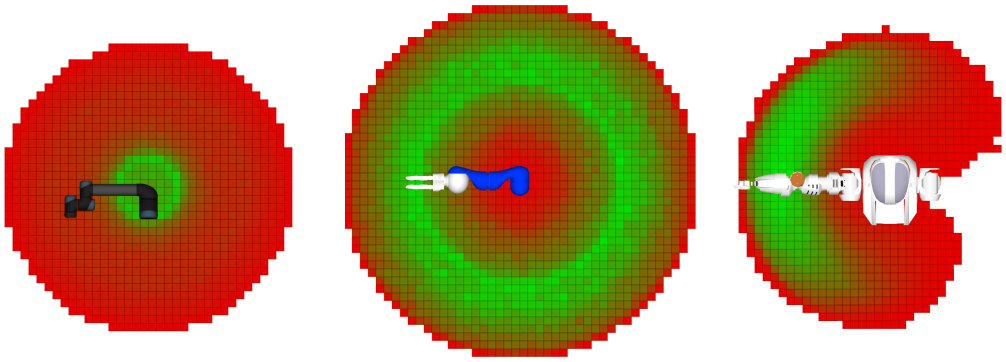
these methods worked for robots with simple or special kinematic structures. Moreover, these methods assume the base of the robot is fixed, whereas wheeled and legged bases were not considered.

## 2.3 ROBOT BASE PLACEMENT

Although the base of robotic arm is normally fixed, where to fix the base is an interesting problem in itself. Positioning robots at appropriate locations is an important step in system design for industrial applications. Spensieri et al. [2016] showed that the productivity can be significantly improved by placing the robot wisely. Abolghasemi et al. [2016a] proposed an Easy-of-Reach-Score (ERS) that describes how an end-effector pose can be reached with a 6-DoF robotic arm from different base locations. They have also introduced a learning method to estimate the ERS, which allows the robot to efficiently calculate the ERS in new environments within one second. However, only grounded obstacles (or obstacles on a horizontal table Abolghasemi et al. [2016b]) were considered, where floating objects that could potentially blocking elbow and wrist links were not taken into account. Romay et al. [2014] introduced a so-called template-based manipulation method, where a valid robot pelvis pose is associated to each grasp pose. The collision environment was not taken into account in their approach, instead, a human-in-the-loop scheme was used to evaluate the collision status. Such human-in-the-loop and semi-autonomous approach is widely used in practice to manually select robot base location, such as in the DARPA Robotics Challenge (Oh et al. [2017], Tedrake et al. [2014], Koolen et al. [2013], Kohlbrecher et al. [2015]).

## 2.4 REACHABILITY MAPS

As we have discussed, pre-computation is commonly used in inverse kinematics. This section presents a particular type of pre-computation method, which will be extended in Chapter 3, to solve more complex problems in cluttered environments for high dimensional systems.



(a) UR5 (b) LWR (c) Valkyrie left arm

Figure 2.1: Top-down view of forward reachability maps of (a) 6-DoF Universal Robotics UR5; (b) 7-DoF KUKA LWR robotic arm; (c) 10-DoF Valkyrie humanoid left upper body, i.e. 3-DoF torso and 7-DoF arm.

To find a valid end-pose, one important property to consider is the reachability of the robot. In other words, given a desired end-effector reach pose, we need to first check if the robot is able to reach that pose, and how to reach that pose if it is reachable. A robot's kinematic reachability is fixed once it has been designed and built, because such ability purely depends on the kinematic model of the robot, e.g. number of joints, the length of each link, and the range of motion of each joint. This means the reachability of a particular robot can be analysed off-line and accessed directly during runtime. Such information is often referred as the *Reachability Map*. There exist two types of reachability maps — the *Forward Reachability Map* (FRM/RM) and the *Inverse Reachability Map* (IRM). As we will discuss later, the FRM is more suitable for fixed-base robotic arms, while IRM is designed for mobile manipulators and humanoids.

### 2.4.1 *Forward Reachability Map*

The FRM introduced by Zacharias et al. [2007, 2013][1] describes how a robotic arm can reach certain workspace poses by its end-effector. For a discretized workspace, each voxelized volume is associated with a score that shows how many robot configurations can reach this particular region. By sampling a large number of configurations, e.g. in billions, the recorded scores can approximate the physical reachability of the

1 The method is called *the capability map* in the original work.

robot. Furthermore, each workspace volume ($(x, y, z) \in \mathbb{R}^3$) can be divided into many different orientation sectors ($SO(3)$) to enable more accurate reachability queries in $SE(3)$. Since the classic FRM only records numerical scores, i.e one `integer` or `double` type data for each workspace voxel, the storage is not an issue, where one can sample tens or even hundreds of billions configurations to build a very accurate FRM that densely covers the entire configuration space. During online phase, given a desired end-effector pose $\mathbf{y}^* \in SE(3)$, the FRM can check if the pose is reachable or not, and show the quality of the pose, e.g. the reach pose has better quality if the score of corresponding workspace cell is higher. Examples of forward reachability maps of different robots are highlighted in Figure 2.1. The coloured voxels are regions where the robot can reach, and greener ones have higher reachability scores. We can see that the LWR robot, in general, has "better" reachability than UR5, where the latter has very poor reachability in medium and distant regions. Thus, the reachability maps sometimes can be used as a tool to analysis the goodness or optimality of the kinematic design of certain robot platforms.

By recording also the actual joint values of all the configurations, the FRM can be used directly as IK solver similar to approximate table lookup approaches (Morris and Mansor [1997], Halfar [2013]). In theory, if one generate an infinite number of configurations with an infinitely fine workspace grid, the FRM should return one, or multiple if redundant, valid configurations that satisfy the desired end-effector pose $\mathbf{y}^*$. In practice without infinite number of configurations, we first find a configuration that reaches close to $\mathbf{y}^*$, which will be used as the initial seed configuration for classical IK solvers as warm start.

The FRM assumes that the robot base is fixed since it requires a bounded workspace volume to constructed a finite number of workspace cells. However, the base movement of a mobile manipulator or humanoid is unbounded, which leads to an infinite number of workspace cells, meaning that FRM can not be directly applied to robots with floating bases. Although one can randomly or systematically search for possible base locations (Leidner et al. [2014]), such procedure can be trapped in cluttered environments where the selected base locations are occupied by obstacles.

### 2.4.2 *Inverse Reachability Map*

As an improvement upon the original FRM, the *Inverse Reachability Map* (IRM) proposed by Vahrenkamp et al. [2013] is capable of finding feet/base poses and configurations for mobile manipulators or humanoids, by storing a map calculated from the end-effector pose to infer where to put the robot's feet or base. In contrast to FRM, IRM is constructed in the end-effector's frame and transforms all the configurations to poses with respect to the end-effectors as the origin. By doing so, all the configurations with unbounded base poses are transformed to a bounded volume originated at the end-effector, which makes it feasible to store in theory any number of full-body configurations and base poses using a finite number of workspace cells. Vahrenkamp et al. [2013] first applied the IRM method on a mobile robot and Burget and Bennewitz [2015] extended the work to humanoids.

## 2.5 LIMITATIONS IN COMPLEX ENVIRONMENTS

Both the FRM and IRM consider only the kinematic reachability without taking into account the collision between robot and environment, which works fine in free or simple environments. However, such artefact leads to planning failures when facing complex and cluttered environments, where many poses that are kinematically reachable but in collision. For example, in Figure 2.2, the shadowed robot posture is the output of regular IK planner, which is invalid due to the collision between robot and environment. The humanoid robot needs to take advantage of its dexterous kinematic structure and floating base for reaching distant target and avoiding obstacles.

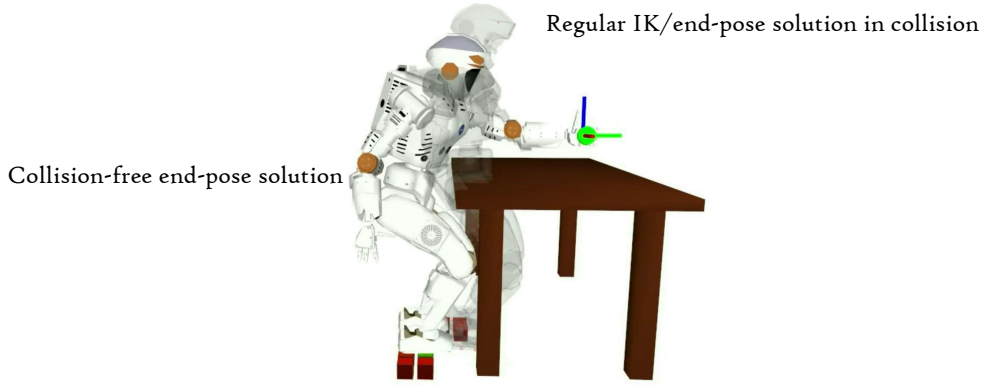



Figure 2.2: Illustration of end-pose planning in presence of obstacles. Shadowed robot posture: IK/end-pose computed without considering collision; solid robot posture: different end-pose is found when taking collision environment into account.

# 3

# END-POSE PLANNING USING DYNAMIC REACHABILITY MAPS

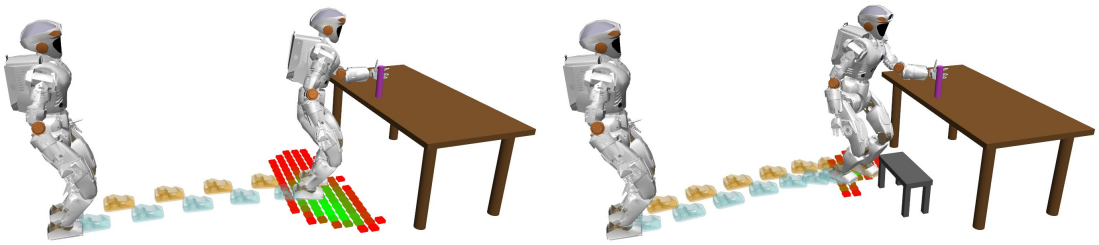


Figure 3.1: Illustration of end-pose planning in different environments using inverse dynamic reachability map. Left: a variety of feasible stances (coloured grids) and postures are available for the humanoid robot to reach the target; right: solutions are reduced due to the obstacle on the ground. A valid and sufficient end-pose is a key pre-requisite to other tasks, such as footstep planning and motion planning.

To address the problem of end-pose planning in complex environments (Section 2.5), in this chapter, we introduce a novel approach that allows the FRM and IRM to dynamically update the collision status of all configurations stored in the map, so that only collision-free ones will be selected during planning. The new maps are called the *forward dynamic reachability map* (FDRM/DRM) and *inverse dynamic reachability map* (IDRM) respectively.

As we will show later, although the applications of forward and inverse maps are very different, they are very similar from an algorithmic point of view. Thus, rather than trying to describe FDRM and IDRM separately, we explain the two maps jointly in two phases — offline map construction (Section 3.1) and online planning (Section 3.2).

## 3.1 OFFLINE MAP CONSTRUCTION

The number, as well as names, of end-effectors vary across different robots, for consistence, we define two types of end-effectors — *root* and *tip* links. For example, the

fixed-base of a robotic arm is called the root and the gripper is called the tip. Note that, a kinematic structure can only have one root but potentially multiple tips, for instance, the base of a mobile manipulator is the root, while the end-effectors of any upper-body limbs are the tips. More intuitively, a reachability map, either forward or inverse, is about fix one end of the kinematic chain and figure out where the other end can reach. However, the forward and inverse indeed have physical meanings. In general, a map is called forward if the defined root and tips correspond to the "true" root and tips. For example, for a robotic arm, the true root is the base and tip is the gripper. So, a map will be called DRM if the base is defined as root and gripper is defined as tip; and the map will be called IDRM if the gripper is defined as root and base defined as tip. The "true" root is normally the parent links in the kinematic tree, and tip is normally the child or end-effector. For the lower-body of humanoids, our ultimate goal is to place the feet, rather than the pelvis, to desired locations. Also, feet are normally the child/leaf links in the kinematic tree, which are considered as the "true" tips. Thus, a map with pelvis as root and feet as tips is considered as DRM, while a map with foot as root will be considered as iDRM.

### 3.1.1 *Generate valid samples*

To construct a reachability map, we first need to generate valid robot configuration samples which will be analysed and stored. In general, the samples should satisfy joint limits, balance and self-collision-free constraints. There exist many different approaches for generating valid robot configurations, such as MoveIt! (Şucan and Chitta [2013]) and OMPL (Şucan et al. [2012]) for generating self-collision-free configurations, and Drake (Tedrake [2014]) for generating quasi-statically balanced configurations. We generate a large number, $M$, of valid samples and transform them to postures where the root link is at the origin of the map for further processing. Examples of valid full-body configurations of a humanoid robot are highlighted in Figure 3.2, where the mid-point of the two feet and the left palm are selected as the root and tip links. More specifically, in FDRM, the mid-point of the feet is the root

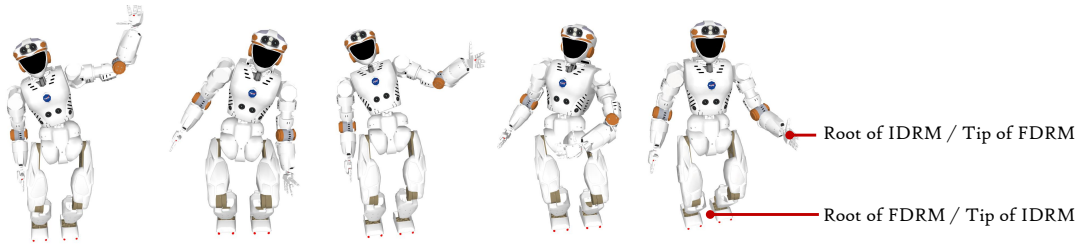



Figure 3.2: Balanced full-body configurations generated for FDRM/IDRM. The mid-point of the two feet and the left palm are selected as the root and tip links.

link and left palm is the tip link; and in IDRM, the left palm is the root link and mid-point of the feet is the tip link.

### 3.1.2 *Space Discretization*

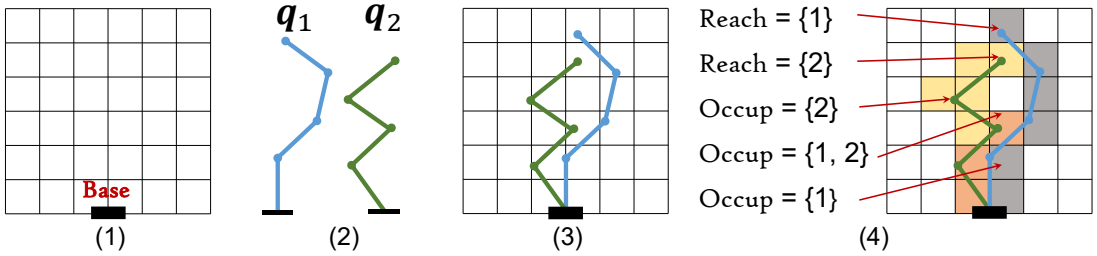
Let $x_{\min}, x_{\max}, y_{\min}, y_{\max}, z_{\min}, z_{\max} \in \mathbb{R}$ be the bounds of a workspace volume in the root frame. Given a discretization resolution $s_x, s_y, s_z \in \mathbb{R}_+$ in each axis, a voxelized workspace $\mathbb{V}$ including

$$V = \lceil \frac{x_{\max} - x_{\min}}{s_x} \rceil \times \lceil \frac{y_{\max} - y_{\min}}{s_y} \rceil \times \lceil \frac{z_{\max} - z_{\min}}{s_z} \rceil \in \mathbb{N}_+ \tag{3.1}$$

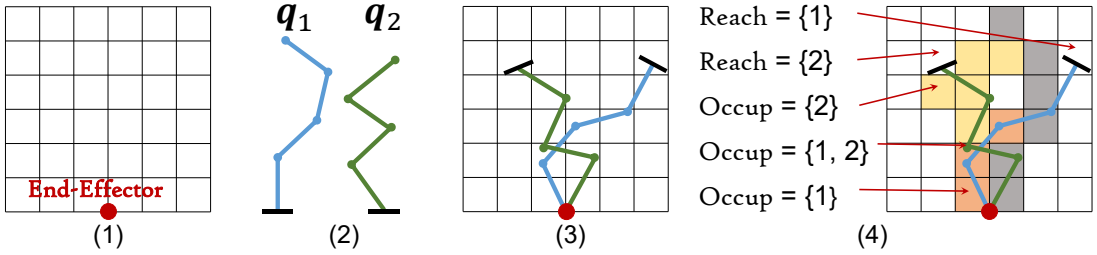
voxels $v \in \mathbb{V}$ can be created. Murray et al. [2016] showed that by choosing different grid resolution in different workspace regions, they can obtain dense and accurate space representation in interested area and sparse representation in less important area. For simplicity, one can also set $s_x, s_y, s_z$ to identical values to generate a uniform grid, as highlighted in Figure 3.3. Each voxel $v$ is associate with two lists — a reach list $R_v$ and an occupation list $O_v$.

### 3.1.3 *Generate reach list*

The reach list $R_v$ of voxel $v$ stores the information of how this voxel can be reached by the tip link(s) while keeping the root link at the origin. In the present section, we assume the robot has only one tip link, where the reach list is a vector of integers storing the indices of samples whose tip link reach this voxel. Scenarios with multi-limb




(a) Forward Dynamic Reachability Map (DRM)




(b) Inverse Dynamic Reachability Map (IDRM)

Figure 3.3: Examples of DRM and IDRM offline map construction. From left to right: 1) discretized space; 2) generate valid samples; 3) transform samples to map origin; and 4) generate reach and occupation lists.

robots will be discussed in Chapter 4. We can further divide the voxel into sectors with different orientations to obtain more accurate reachability map (Kuffner [2004], Zacharias et al. [2013]). In general, this is a trade-off between storage and online computation, where one extreme is to not store any reachability and calculate the forward kinematics for all samples on-the-fly; and the other extreme is to store an infinite number of voxels and orientation sectors so that every sample has its own reachability voxel and no forward kinematic calculation is required. Between the two extremes is the area where we can store the reachability voxels up to certain resolution and leave some finest reach pose calculation to online process. In practice, we should make the decision wisely based on hardware availability, e.g. computational power and storage resource.

An example of DRM/IDRM offline construction is given in Figure 3.3. In the forward map scenario (Figure 3.3a), the fixed base is the root link and end-effector is the tip link; the root and tip links are swapped in the inverse map (Figure 3.3b). The configurations need to be transformed to postures where the root link aligns with the origin before processing the reach lists. As shown in the third column, the configurations are transformed to correct postures where the fixed base and end-effector are

aligned with the map origin in DRM and IDRM respectively. Then the reach list of each voxel can be found by calculating the forward kinematics of the tip link with respect to the root link, as highlighted in the fourth column in Figure 3.3.

### 3.1.4 *Generate occupation list*

The reason behind reach list is clear, which is to indicate whether a workspace region is reachable or not. However, such information is correct only if the environment is empty or at least static. In practice, the pose which is reachable during sampling might no longer be reachable in new environments due to collision. Thus, it is crucial to have a method to efficiently check the collision status of all the samples and remove invalid ones before querying the reachability. In a workspace $\mathcal{W}$ with obstacle $\mathcal{O} \subset \mathcal{W}$, we need to identify all invalid configurations that are in collision with the present environment,

$$Q_{\text{invalid}} = \{\mathbf{q} \mid \mathcal{A}(\mathbf{q}_n) \cap \mathcal{O} \neq \emptyset, n \in M\}, \tag{3.2}$$

and remove them from the map. However, such process, which may involve millions of conventional collision checking between robot $\mathcal{A}(q_n)$ and environment $\mathcal{O}$, is too expensive to be deployed during runtime.

In complex and changing environments, since the obstacle $\mathcal{O}$ is unknown until the actual planning query arises, there is very limited information for us to pre-process. Fortunately, after the workspace voxelization, we can pre-analyse the collision status up to a certain resolution to reduce runtime computation. The pre-processed information for collision checking is stored in the so-called occupation list,

$$O_v = \{n \mid \mathcal{A}(\mathbf{q}_n) \cap v \neq \emptyset, n \in M\}, \tag{3.3}$$

for each voxel $v \in \mathbb{V}$. The occupation list of voxel $v$ stores the indices of samples that occupy the voxel $v$ when the root link is at the origin. In actual implementation, to generate the occupation lists for all voxels, we first create a static environment with $V$ cube obstacles, each of which represents a workspace voxel. Then for each

Table 3.1: Size and limits of different data types on X64 architecture system.

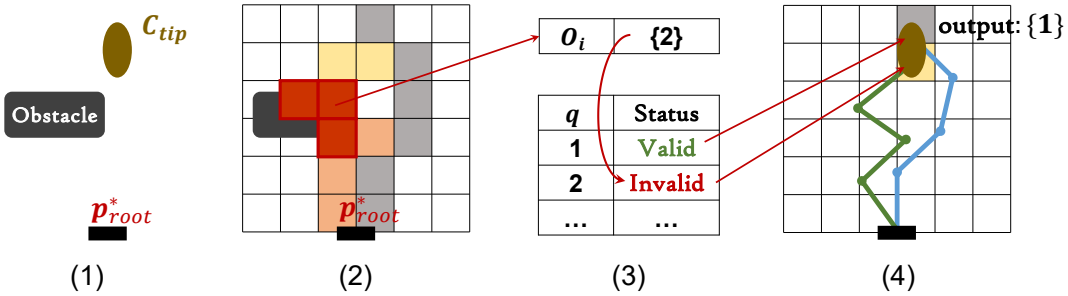
| size (Byte) | numerical limit | data type example (C++) |
|---|---|---|
| 1 | 255 | `unsigned char` |
| 2 | 65,535 | `unsigned short` |
| 3 | 16,777,215 | struct (`unsigned char`, `unsigned short`) |
| 4 | 4,294,967,295 | `unsigned int` |
| 8 | 9,223,372,036,854,775,807 | `unsigned long int` |

sample $\mathbf{q}_n, n \in M$, we apply conventional collision checking to find the voxels that the present sample occupies,

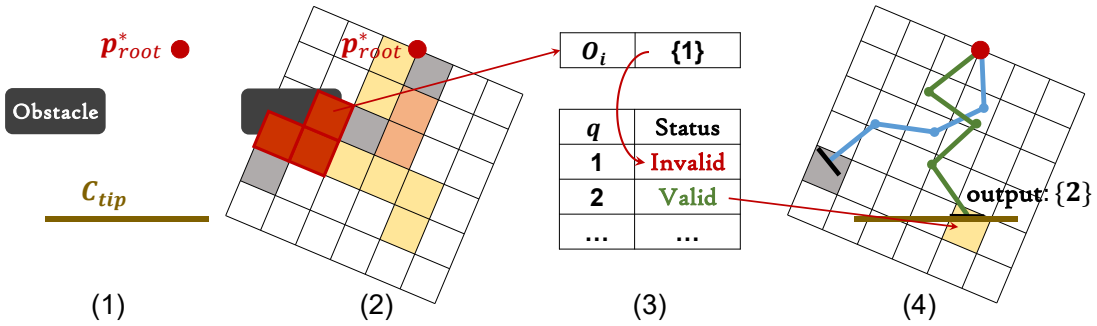
$$V_n = \{v \mid \mathcal{A}(\mathbf{q}_n) \cap v \neq \emptyset, v \in \mathbb{V}\}. \tag{3.4}$$

Finally, we append index $n$ to the occupation list of each voxel $v \in V_n$. For example, in the fourth column of Figure 3.3, considering only samples $q_1, q_2$, white voxels have empty occupation lists $O_v = \{\ \}$, grey voxels have occupation lists $O_v = \{1\}$ that means these voxels intersect with sample $q_1$, yellow voxels have occupation lists $O_v = \{2\}$ that means these voxels intersect with sample $q_2$, and orange voxels have occupation lists $O_v = \{1, 2\}$ which means these voxels intersect with both sample $q_1$ and $q_2$.

After having generated the reach and occupation lists, such information need to be stored on offline storage devices, e.g. a hard drive, and will be loaded into memory later during runtime. Depending on the hardware and system architecture used, one should carefully design the data structure and encoding/decoding methods to efficiently store and load these information and avoid unnecessary storage and computation. For example, the sizes and limits of different data types on X64 architecture system is listed in Table 3.1. Depending on the number of samples stored, different data types should be used for storing the reach and occupation lists. In cases where $65,535 < M \leq 16,777,215$, instead of using the built in 4 Bytes `unsigned int`, we can create a customized 3 Bytes data structure. By doing so, additional (very minimum) computation is required to retrieve the actual index $n \in M$ from the structure, but we can save up to 25% storage space.




(a) Forward Dynamic Reachability Map (DRM)




(b) Inverse Dynamic Reachability Map (IDRM)

Figure 3.4: Examples of DRM and IDRM online update. From left to right: 1) problem setup; 2) transform map to root pose; 3) validate collision status; and 4) check tip pose constraints and find valid samples.

## 3.2 ONLINE END-POSE PLANNING

During online planning phase, given desired tip link pose, $\mathbf{y}^*$, we need to find a valid end-pose $\mathbf{q} \in \mathcal{C}_{\text{valid}}$ whose tip link reaches $\mathbf{y}^*$. In this section, we discuss how to use the DRM/IDRM as an end-pose planner (Equation 2.6) to provide valid end-poses. An example of DRM/IDRM online process steps is illustrated in Figure 3.4.

### 3.2.1 *Map relocation*

First, we transform the maps to desired root link location. For DRM, the root is normally the base link which is fixed, where no transformation is required. In scenarios where the desired target pose is too far from the current base location, one can move the base as well if it is not strictly "fixed". However, where to move the base is not always trivial, which eventually also becomes an end-pose planning problem for

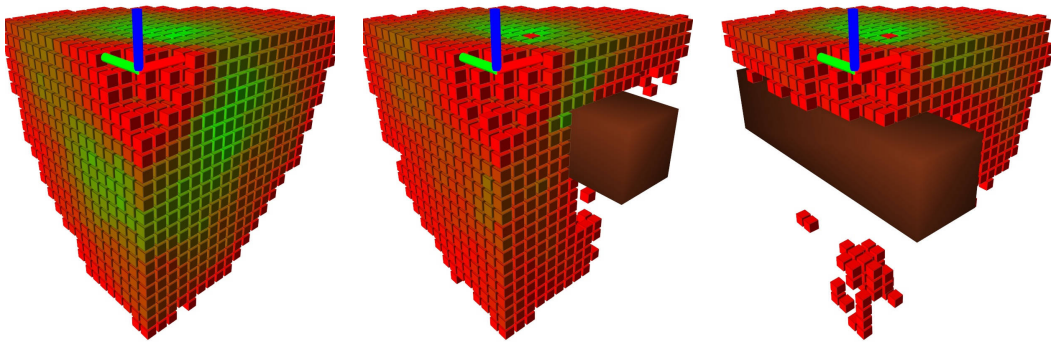


Figure 3.5: Octant view of DRM collision update. The coloured voxels indicate the capability of reaching these voxels with the tip link. The left figure shows the full map in free space, followed by two updated maps in different environments respectively.

moveable base robot. For IDRM where the end-effector is the root link, the whole map needs to be transformed to $\mathbf{p}^*_{\text{root}} = \mathbf{y}^*$, as shown in Figure 3.4b. Note that each voxel in the workspace will be used as a collision object, meaning that transform the map involves calculating $V$ (number of voxels) times of geometry transformations, which might be expensive when the workspace is large or finely discretized. On the other hand, the environment often contains less number of collision objects, in which case, instead of transforming the IDRM to $\mathbf{p}^*_{\text{root}}$ in the world frame, one can also transform all environmental objects to the IDRM frame.

### 3.2.2 *Collision update*

Next, we describe how to remove those samples that are in collision with current environment. In actual implementation, it is difficult to "remove" configurations, i.e. removing an element from a vector structure, which is unnecessary and time consuming. Instead, we keep tracking the validity of all samples and "remove" collision samples by setting its status to invalid, as illustrated in third column in Figure 3.4. We first deploy a conventional collision checking between the discretized workspace $\mathbb{V}$ and current environment $\mathcal{O} \subset \mathcal{W}$, to find a set of voxels that is occupied by environmental obstacles,

$$V_{\text{occup}} = \{v \mid v \cap \mathcal{O} \neq \emptyset, v \in \mathbb{V}\}. \tag{3.5}$$

From the occupation list $O_v$ of each voxel $v \in V_{\text{occup}}$, we can find the configurations that intersect with these voxels. The status of these configurations are set to invalid since they are now in collision with the environment. An example is illustrated in Figure 3.5, for better visualization, only an octant view is shown in the figure while the whole map is in the shape of sphere. The left figure shows the whole map in free space, the middle and right figures are updated maps in different environments.

### 3.2.3 *Constraint update*

After having the collision status updated, remaining configurations are all collision-free, but not all of these are valid for the task, which should also be removed. The constraint of the tip link, $C_{\text{tip}}$, varies across different tasks. For example, for a fixed-base reaching problem, as shown in Figure 3.4a, we want the tip link (end-effector) to be close to the desired target pose $\mathbf{y}^*$; for a moveable base reaching problem, as shown in Figure 3.4b, the tip (moveable base) pose is valid as long as it is on the ground, so $C_{\text{tip}}$ should only constrain height and orientation, while allowing free horizontal movement.

To check if a configuration satisfy the constraint, we need to know the tip pose $T_{\text{tip}}(\mathbf{q}_n)$, which can be calculated using forward kinematics. Depending on hardware availability and planning speed requirement, we can calculate the tip pose during either pre-processing or online phase. Offline calculation requires more memory for storing these information, whereas online calculation demands less storage but requires more computation resources.

### 3.2.4 *End-pose selection and adjustment*

The remaining valid configurations after the collision and constraint update are called the *end-pose candidates*, $Q_{\text{valid}}$. We score the samples according to a Jacobian based manipulability measure (Burget and Bennewitz [2015]),

$$\text{score}_n = \sqrt{\det J(\mathbf{q}_n) J(\mathbf{q}_n)^T}, \tag{3.6}$$

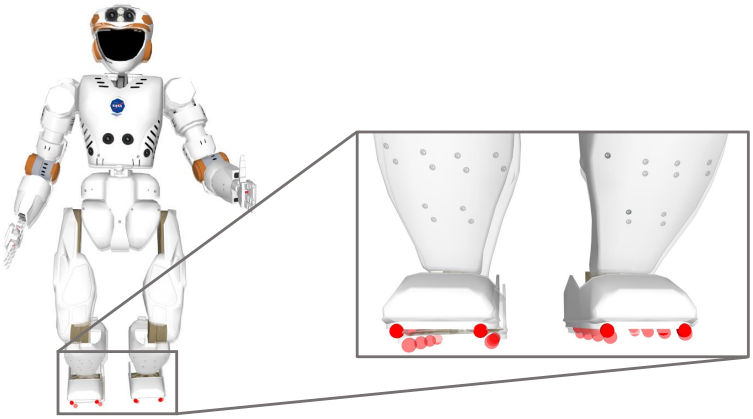


Figure 3.6: Illustration of IK adjustment. The shadowed posture is the selected candidate from an IDRM dataset of a humanoid that reaches desired left palm pose. The solid posture is the final end-pose calculated by a standard full-body IK by using the selected candidate as seed pose. The IK solver corrected the leg configurations to ensure perfect ground contact, while the workspace difference between the two posture is negligible, meaning that the final end-pose is also collision-free since the candidate is guaranteed to be collision-free.

where $J(\mathbf{q}_n)$ is the Jacobian matrix of $\mathbf{q}_n$. The scores of all samples are calculated offline and readily available during run time. In addition, we introduce another cost term $|\,\mathbf{q}_n - \mathbf{q}_{\text{curr}}\,|$ to penalize samples that are far away from the current configuration $\mathbf{q}_{\text{curr}}$. The best candidate can be found by

$$\mathbf{q}^* = \underset{\mathbf{q}_n \in Q_{\text{valid}}}{\arg\max}\ w_a \text{score}_n + |\,\mathbf{q}_n - \mathbf{q}_{\text{curr}}\,|_{w_b}, \tag{3.7}$$

where $w_a$ and $w_b$ are constant weighting factors.

The selected configurations might not strictly satisfy the desired tip pose $\mathbf{y}^*$, where standard IK solvers can be applied to finalize the configuration. Essentially, the end-pose planner finds a collision-free seed pose to increase the chance for standard IK to converge to a collision-free output. In practice where the DRM/IDRM contains millions or potentially tens of millions of configurations, the selected sample is already very close to the desired result, where only minor changes are required in the configuration space and the corresponding workspace movement is negligible, meaning that the final output is also collision-free, as shown in Fig. 3.6. In unlikely scenarios where the final end-pose is in collision, the next best candidate will be selected as a new seed pose until valid solution is found.

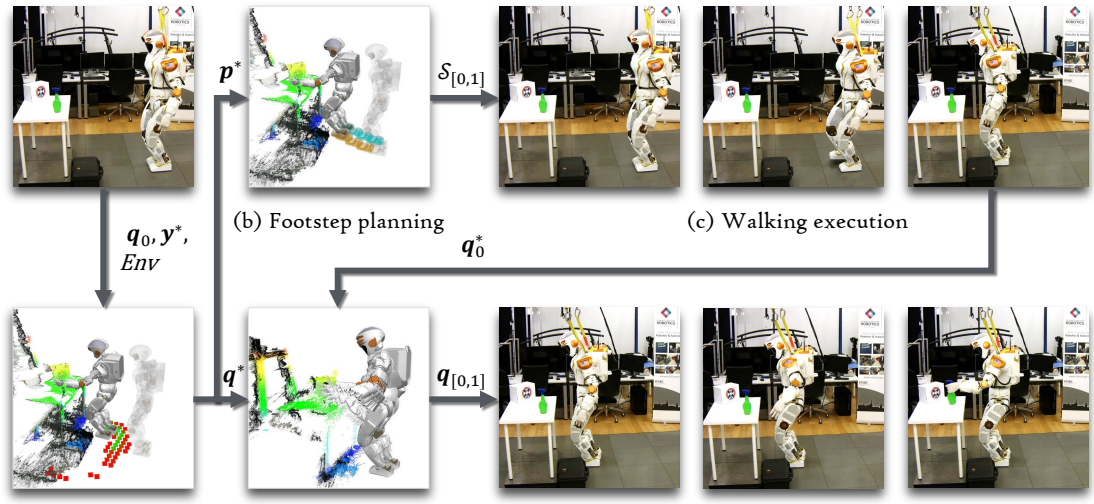



Figure 3.7: Humanoid motion synthesis system overview. The end-pose planner first searches for a valid end-pose (a), which will be used to generate a footstep plan (b). The footstep plan is then executed to bring the robot to desired standing location (c). Finally, a reaching motion is generated and executed to reach the target (d and e).

So far we have discussed the theoretical concepts of dynamic reachability maps. In next section, we highlight a practically application showing how to use the inverse map (IDRM) to plan end-poses for floating base humanoids.

## 3.3 END-POSE PLANNING FOR HUMANOIDS

Humanoid robots are designed for accomplishing a wide variety of tasks in human friendly environments but have redundant many DoFs, which makes real-time planning and control extremely challenging. In real world applications, such as in the DARPA Robotics Challenge (DRC, Pratt and Manzo [2013]), it was unreliable to directly plan whole body motions. Typically, operators manually decided where the robot stood and what the desired posture should be to execute an action. However, such end-pose is non-trivial to find, especially in complex and changing environments, where standing locations and full-body postures need to be changed for avoiding obstacles, as illustrated in Figure 3.1. In this section, we present how to address this problem by utilizing the proposed IDRM method.

### 3.3.1 *Humanoid Motion Planning Framework*

As mentioned in Section 1.4, we formulate humanoid motion planning problem as a combination of end–pose planning (Equation 1.1), footstep planning (Equation 1.4) and motion planning from a fixed stance (Equation 1.5). Interleaved with walking and full-body motion execution, the overall system flow can be summarised by Figure 3.7. The present section focuses on solving the end-pose planning problem, whereas the motion planning will be discussed later in Chapter 7.

### 3.3.2 *IDRM construction*

The general IDRM construction steps have been discussed in Section 3.1. This section explains in details how to construct an IDRM for humanoid robot, in particular how to generate balanced full-body configurations. The dimensionality of the IDRM for a $N$-DoF humanoid robot is $\mathbf{q} \in \mathbb{R}^{N+6}$, where the extra 6 DoFs represent the robot transformation in the world frame. For simplicity, in this chapter, we only consider single hand reaching problem with the relative position of the two feet fixed, i.e. the mid-point of the two feet is the tip link of IDRM and the reaching hand is the root link. More complex scenarios with bi-manual reaching problems will be discussed in Chapter 4.

#### 3.3.2.1 *Generate valid samples*

The main difference between fixed-base robot and humanoid is that the later has an extra balance constraint. We apply a full-body IK solver (Tedrake [2014]) to generate feasible quasi-statically balanced configurations

$$\mathbf{q}^* = \text{FullBodyIK}(\mathbf{q}_{\text{seed}}, \mathbf{q}_{\text{nom}}, \mathbf{C}) \tag{3.8}$$

which is a sequential quadratic programming (SQP) solver in the form of

$$\begin{aligned} \mathbf{q}^* = & \underset{\mathbf{q}\in\mathbb{R}^{N+6}}{\arg\min} \mid \mathbf{q} - \mathbf{q}_{\text{nom}} \mid^2_{Q_q} \\ \text{subject to} \quad & \mathbf{b}_l \leq \mathbf{q} \leq \mathbf{b}_u \\ & c_i(\mathbf{q}) \leq 0, c_i \in \mathbf{C} \end{aligned} \quad , \tag{3.9}$$

where $Q_q \succeq 0$ is the weighting matrix, $\mathbf{b}_l$ and $\mathbf{b}_u$ are the lower and upper joint bounds. The seed pose $\mathbf{q}_{\text{seed}}$ is used as the initial value in the first iteration of SQP solver. The output $\mathbf{q}^*$ is a configuration that satisfies all the constraints defined in $\mathbf{C}$ and is close to $\mathbf{q}_{\text{nom}}$. The constraints include quasi-static balance constraint, end-effector pose constraint, etc. We say a robot is quasi-statically balanced if the centre-of-mass (CoM) projection lies within the support polygon with no velocity and acceleration along any axis. We only store postures that are quasi-statically balanced, self-collision-free and reach an area of interest in front of the robot. Note that one can still reach targets behind by rotating the whole robot, which is the key feature of stance pose selection. We repeat the sampling process with random seed and nominal poses until $M$ number of random balanced samples are generated.

#### 3.3.2.2 *Space discretization and memory consumption*

Depending on robot size, different workspace voxel resolution should be chosen (Section 3.1.2). In our implementation for the NASA humanoid robot, Valkyrie, which is close to 2 meter tall, we have created a $4^3m^3$ workspace volume, i.e. $[-2, 2]$ for each axis, with $10cm$ voxel resolution. According to Equation 3.1, such a discrete workspace includes $V = 68921$ voxels.

The memory required for storing such IDRM varies based on different number of samples, $M$, as illustrated in Figure 3.8. The sample configuration storage is the memory required to store the full-body configuration for each sample, which is approximately equivalent to the memory required for the regular IRM (Burget and Bennewitz [2015]). We can find that the storage for occupation lists is the significant component. Ultimately, IDRM requires much more memory storage than IRM. However, as we show later, IDRM can handle online end-pose queries much faster than

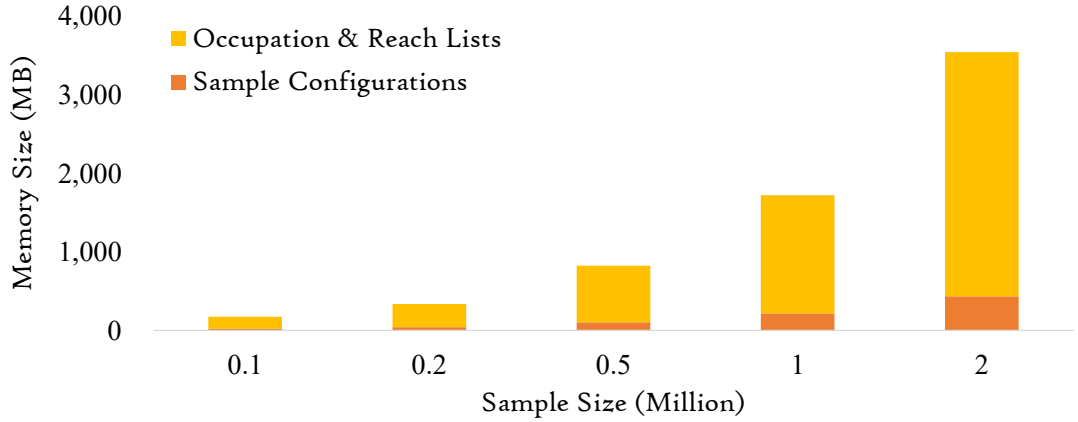
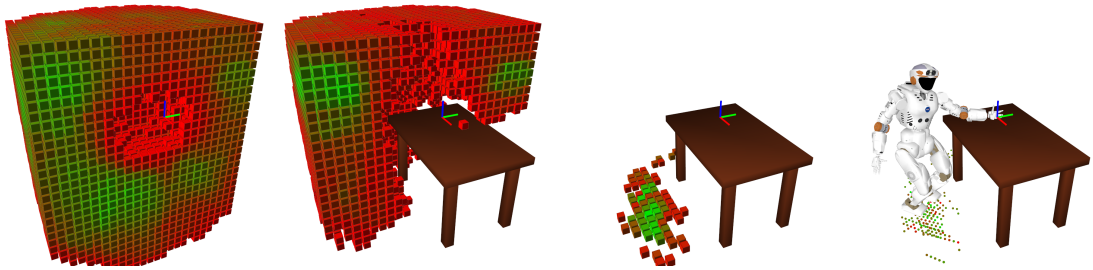



Figure 3.8: Memory consumptions of IDRM, which is approximately in a linear relationship with the number of samples.

IRM. In other words, IDRM essentially trades off storage for better efficiency of online computation. As mentioned earlier, this data needs to be store efficiently to minimize memory consumption. We store the configurations and occupation information in the most intuitive but naïve way, i.e. an integer vector. A more efficient storage technique will be discussed in Chapter 6.

### 3.3.3 *End-pose planning*

Online end-pose planning using IDRM has been detailed in Section 3.2. For humanoid reaching problem, we only need to define an appropriate tip constraint, $C_{\text{tip}}$. Assume the robot needs to stand on a flat floor with horizontal feet orientation, which means the $C_{\text{tip}}$ should constrain roll and pitch of feet transformation close to zero. To avoid checking all samples we first find the voxels $V_{\text{ground}}$ that may contain balanced samples, i.e. voxels that intersect with the floor. Figure 3.9 illustrates different phases during end-pose planning for humanoids. Note that only a cross section of the IDRM is plotted for visualization, whereas the whole IDRM should have the shape of a sphere.



(a) Original IDRM. (b) Collision update. (c) Constraint update. (d) Selected end-pose.

Figure 3.9: IDRM end-pose planning example. The iDRM is transformed into world frame, and the axis indicates desired end-effector pose in the world frame. The coloured voxels indicates the tip (feet) reachability to that voxel while having the reaching hand at the target pose. Greener voxels contain more collision-free reaching configurations, but each coloured voxel has at least one configuration.

### 3.3.4 *Evaluation*

In order to evaluate the end-pose planning performance, we compare IDRM against the following three approaches:

- *Random Placement* (RP): the robot's feet are randomly placed close to the target within a certain radius. This may be reasonable when no further information is available. Then a random configuration is passed to IK solver to obtain a result.
- *Random Placement DRM* (R-DRM): first, we create a regular DRM with mid-point of feet as the root and left palm as the tip. When we process an online query, we select stance poses randomly (similarly to RP) and transform the DRM to this location. We then select a seed configuration from the DRM.
- *Inverse Reachability Map* (IRM): by bypassing the collision update (Section 3.2.2), we obtain a regular IRM approach equivalent to Vahrenkamp et al. [2013], Burget and Bennewitz [2015].

All these three methods as well as IDRM iterate until a balanced and collision-free result is found. The IDRM dataset with 1 million samples and 10*cm* voxel resolution is used.

We set up with 3 different scenarios of grasping tasks at an increasing level of difficulties, as illustrated in Fig. 3.10. In the simple task, the target is placed on top of

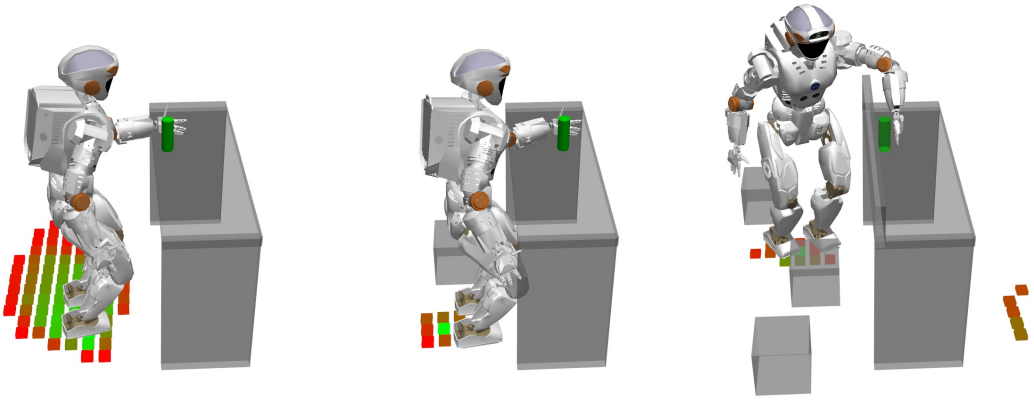


Figure 3.10: Evaluation scenarios, from left to right: easy, medium and hard tasks.

Table 3.2: Humanoid end-pose planning computation time of single-arm reaching problems (in seconds).

| Algorithms | Tasks | | |
|---|---|---|---|
| | Easy Task | Medium Task | Hard Task |
| RP | 0.1916 | 1.2322 | 2.2654 |
| R-DRM | 0.7521 | 2.2531 | 38.8050 |
| IRM | 0.0440 | 0.9560 | 2.2910 |
| **IDRM** | 0.0553 | 0.0566 | 0.0678 |

the table close to the edge. There is no other obstacles apart from the table itself. In the second scenario, the target is moved away from the edge of the table, with a new obstacle placed at the comfortable standing location. A more challenging scenario is set up where multiple obstacles are placed on the floor and close to the upper body as well. In each case, the reaching hand must achieve the full $SE(3)$ desired pose. In order to fully explore the capabilities of different approaches, each scenario has 10 sub-scenarios with slightly different target and obstacle positions. For each sub-scenario, the result of the RP and R-DRM are averaged over 100 trials (IRM and IDRM will always find same result in each sub-scenario). The sub-scenarios' results are then averaged into the 3 different scenarios.

Table 3.2 highlights the performance of the end-pose planning queries for different tasks. The result shows that RP performs relatively well due to its simplicity. R-DRM is not originally designed to work with the floating base system, so the algorithm requires extra time to transform and update the fixed-base DRM thus heavily

slows down the whole process. IRM and IDRM outperformed RP and R-DRM in simple tasks mainly because these two algorithms are originally designed for efficient end-pose planning for floating base robots. In difficult scenarios, the random base placement in R-DRM can lead to the cases where the stance location is occupied by obstacles and thus the DRM needs to iteratively invalidate all samples. This also implies one of the major limitations of regular IRM approach that IRM has no knowledge about collision information. In the cases where the samples with the highest scores are in collision, the algorithm will still select and evaluate them. The valid samples with relatively low scores can only been found after many iterations. The computational time of IDRM is approximately constant in different scenarios. Apart from the initial collision check between IDRM voxels and the environment, IDRM treats all environments equally no matter simple or complex. Since the collision samples are already removed, the selected sample is guaranteed to be collision-free. Also, the selected stance pose allocates the robot close to a balanced posture. The final IK solver can adjust the sample with a negligible amount of workspace movements such that the first candidate sample is sufficient for finding valid end-poses.

## 3.4 CONCLUSION

This chapter presents a novel approach, the forward and inverse dynamic reachability maps. In particular, we have demonstrated that IDRM is able to plan, in real-time, valid stance locations and collision-free full-body configurations for humanoid robots in complex and cluttered environments. We have implemented and validated the method using the model of a 38-DoF humanoid robot, NASA Valkyrie, and carried out evaluations to compare the performance of IDRM against other approaches. The results suggest that IDRM method is capable of searching for valid solutions in different environments in a more efficient manner than other alternatives — typically finding a valid end-pose within 0.1 seconds.

In this setup, we consider only stance poses on a flat ground where the relative positions of two feet are fixed with the same orientations on the horizontal surface. Moreover, we only solve end-pose problem for single arm reaching tasks. In next

chapter, we will discuss how to solve multi-limb end-pose planning problems for humanoids on uneven terrains.

# 4

# END-POSE PLANNING WITH MULTI-LIMB STRUCTURES

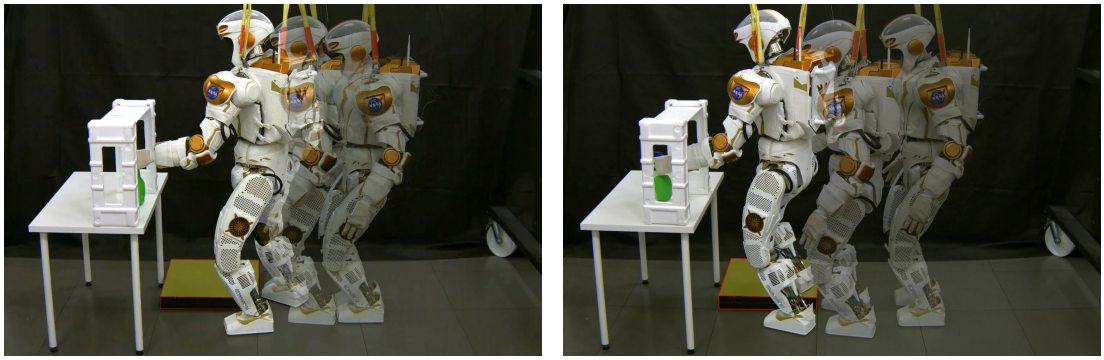


Figure 4.1: Motion planning of grasping on uneven terrains. The robot automatically chooses collision-free stance locations and grasping configurations.

To make full use of the dual-arm and bipedal nature of humanoid robots, it is essential to find appropriate end-poses for bimanual manipulation tasks in environments with uneven terrains. However, it is non-trivial to directly extend the iDRM method to include both dual-arm and bipedal features due to the curse of dimensionality, as the memory required to ensure a sufficient configuration space coverage increases exponentially making it infeasible to run on current commodity hardware.

To resolve this issue, in this chapter, we propose a hybrid approach which combines the advantages of both the Forward and Inverse Dynamic Reachability Map, i.e. DRM and IDRM, to plan end-poses for humanoid robots in complex and rugged environments. We use an upper-body iDRM to first find valid upper-body configurations and pelvis poses. We then use a lower-body DRM to find valid leg configurations on uneven floors. A valid full-body end-pose is then obtained by combining valid upper-body and lower-body configurations. We have validated this approach on the 38-DoF NASA Valkyrie humanoid robot and demonstrated that the proposed method is able to find valid, i.e. balanced and collision-free, end-poses for humanoid robots online for grasping tasks on uneven terrains, as shown in Figure 4.1.

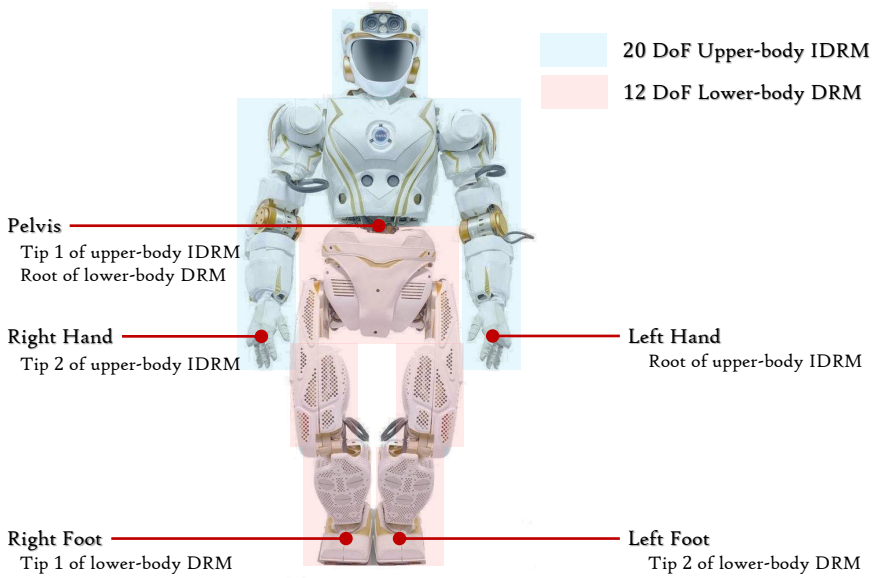



Figure 4.2: Upper-body IDRM and lower-body DRM for the 38 DoF NASA Valkyrie Robot. Each leg has 6 DoF and each arm has 7 DoF, the robot torso has 3 DoF and the neck has 3 DoF. The pelvis represents an extra 6 DoF virtual joint that connects the robot to the world.

## 4.1 REACHABILITY MAPS WITH MULTIPLE TIP LINKS

Recall that both DRM and IDRM can have only one root link but multiple tip links. For a robot with $K$ tip links, the reach list stores a list of paired values specifying both sample and tip indices, i.e. $R_v = \{(n,k)\dots\}$, where $n \in M$ is the sample index and $k \in K$ is the tip index.

As discussed in Section 3.3, an IDRM can be used directly for humanoid end-pose planning, but is limited to environments with flat ground only. As the IDRM can have multiple tip links, a direct and naïve approach is to create an IDRM with one root link and three tip links, where one hand is selected as the root and the rest three limbs are treated as tip links. However, this significantly increases the dimensionality of the problem, i.e. the number of samples has to increase exponentially with each tip link to cover the high dimensional space. Consequently, the required memory size is so large that it becomes infeasible to run on any commodity hardware.

To plan end-poses on uneven terrain while keeping a manageable number of samples and memory size, we take advantage of the robot's inherent structure to treat upper-body and lower-body separately. We separate the robot at the torso pelvis joint,

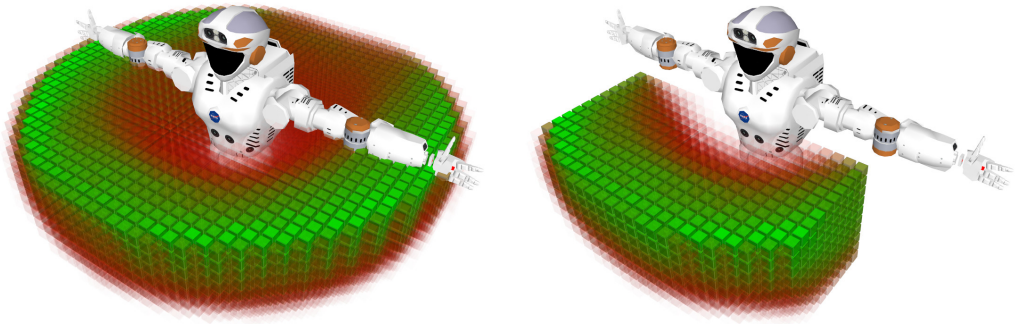


Figure 4.3: Left: the upper-body's full reachability map; right: the reachability map constrained to the front of the robot. All colored voxels are reachable by the robot and greener voxels are regions with high reachability scores. Only part of the map is plotted for clarity (the whole map is sphere shaped).

as illustrated in Figure 4.2. We create an IDRM for the upper-body and a DRM for the lower-body. One hand can be chosen as the root of the upper-body IDRM, and the other will be the second tip. In the rest of this section, we will discuss how to create the two maps, and then combine and use them to plan a bimanual tasks on uneven terrain.

### 4.1.1 *Constructions of DRM/IDRM for humanoids*

#### 4.1.1.1 *Upper-body iDRM*

In this case study, the left hand is selected as the root link of the upper-body iDRM, and the right hand and pelvis are treated as two tip links. Several iDRM datasets with different number of samples (all with 10cm workspace voxel resolution) are generated for the 20-DoF upper-body of Valkyrie. Traditionally, samples of an inverse reachability should cover the whole configuration space, i.e. for the case of a humanoid, samples of the map should reach behind the robot. However, since the robot's sensor are predominantly facing forward, we want to express a preference for stable stance locations that give us reasonable manipulability. We adopt a heuristic in our method, where we only store samples with both hands reaching comfortable manipulation poses in front of the robot, as shown in Figure 4.3. Note that the robot can still manipulate objects that are currently far away or behind the robot by walk-

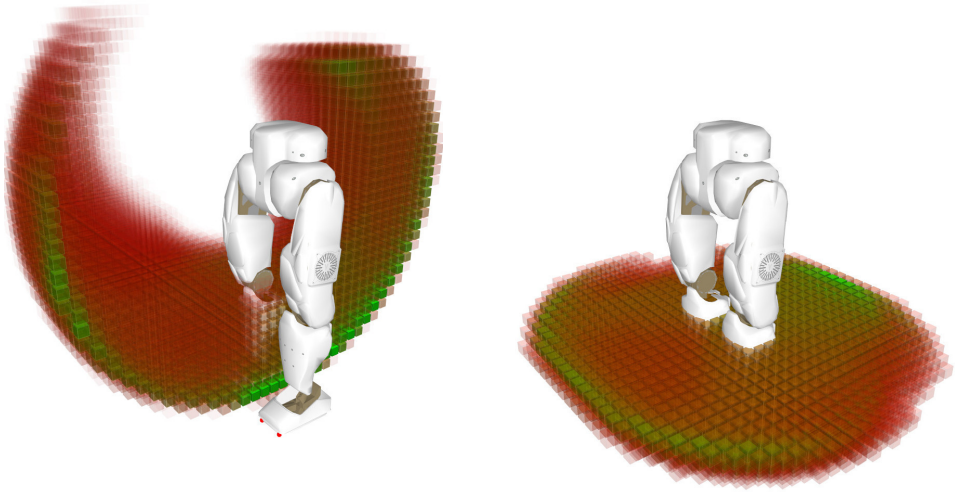


Figure 4.4: Left: the lower-body's unconstrained reachability map, only part of the map is plotted for clarity; right: the reachability map constrained to feet placed below the pelvis.

ing to an appropriate pre-action stance location, which is the key point of end-pose planning.

#### 4.1.1.2 *Lower-body DRM*

The lower-body of Valkyrie has 12-DoF (6-DoF per leg). Though the legs have a large range of motion, the manifold of balanced configurations is much smaller even on uneven terrain. Therefore, we have reduced the "*reachability*" map for the lower-body so that the legs have the range to adapt to the uneven terrain but they won't reach most unnatural poses[1]. To this end, we generate lower-body configurations with two feet placed in a region below the pelvis ($0.8 - 1.1$ meter for Valkyrie), as shown in Figure 4.4. This ensures that the lower-body DRM has sufficient samples to adapt to uneven terrain without demanding extra memory for storing poses that can't provide support for the robot, e.g. poses where the feet reach above the pelvis.

1 Though a metric of being "unnatural" appears to be subjective, it has meaningful implications for achieving such poses on a real robot in terms of joint range and sustainable power. In our work, we define the terms *natural* and *comfortable* as the distance in the configuration space from a chosen nominal configuration derived from the posture shown in Figure 4.2.

**Algorithm 1** Humanoid End-Pose Planning

**Require:** $\mathbf{y}^*_{\text{lhand}}, C, \epsilon$
**Ensure:** $\mathbf{p}^*_{\text{lfoot}}, \mathbf{p}^*_{\text{rfoot}}, \mathbf{q}^*$
1: $\mathbf{y}^*_{\text{root}} = \mathbf{y}^*_{\text{lhand}}$
2: Transform $M_{\text{upper}}$ to $\mathbf{y}^*_{\text{root}}$ //Figure 3.4b (2)
3: CollisionUpdate($M_{\text{upper}}$) //Figure 3.4b (2-3)
4: $\mathbf{Q} = \emptyset$
5: **for** $\forall \mathbf{q}_n \in$ collision-free subset of $M_{\text{upper}}$ **do**
6: $T_{\text{pelvis}}, T_{\text{rhand}}$ =TipGlobalPoses($\mathbf{q}_n, \mathbf{y}^*_{\text{root}}$)
7: **if** SatisfyConstraint($T_{\text{pelvis}}, T_{\text{rhand}}, C$) **then** //Figure 3.4b (4)
8: Transform $M_{\text{lower}}$ to $T_{\text{pelvis}}$ //Figure 3.4a (2)
9: CollisionUpdate($M_{\text{lower}}$) //Figure 3.4a (2-3)
10: **for** $\forall \mathbf{q}_m \in$ collision-free subset of $M_{\text{lower}}$ **do**
11: $\mathbf{p}_{\text{lfoot}}, \mathbf{p}_{\text{rfoot}}$ =TipPoses($T_{\text{pelvis}}(\mathbf{q}_n), \mathbf{q}_m$)
12: **if** ValidTerrainContact($\mathbf{p}_{\text{lfoot}}, \mathbf{p}_{\text{rfoot}}$) **then** //Figure 3.4a (4)
13: $\mathbf{q} = \{\mathbf{q}_n, \mathbf{q}_m\}$
14: **if** $\mathbf{q}$ is balanced **then**
15: cost $= f(\mathbf{q})$
16: **if** cost $< \epsilon$ **then**
17: **return** $\mathbf{p}_{\text{lfoot}}, \mathbf{p}_{\text{rfoot}}, \mathbf{q}$
18: **else**
19: $\mathbf{Q} = \mathbf{Q} \cup (\mathbf{p}_{\text{lfoot}}, \mathbf{p}_{\text{rfoot}}, \mathbf{q}, cost)$
20: $\mathbf{p}^*_{\text{lfoot}}, \mathbf{p}^*_{\text{rfoot}}, \mathbf{q}^*$ =LowestCost($\mathbf{Q}$)
**return** $\mathbf{p}^*_{\text{lfoot}}, \mathbf{p}^*_{\text{rfoot}}, \mathbf{q}^*$

### 4.1.2 *End-Pose Planning for Bi-manual Tasks on Uneven Terrain*

Let $M_{\text{upper}}$ be the upper-body IDRM and $M_{\text{lower}}$ be the lower-body DRM. Given a task $\mathbf{y}^* = (\mathbf{y}^*_{\text{lhand}}, \mathbf{y}^*_{\text{rhand}})$, start states $\mathbf{p}_s, \mathbf{q}_s$ and the environment $Env$, the end-pose planner needs to find an end-pose that contains $\mathbf{p}^* = (\mathbf{p}^*_{\text{lfoot}}, \mathbf{p}^*_{\text{rfoot}})$ and $\mathbf{q}^*$. Firstly, we create two tip pose constraints $C_{\text{tip}} = \{C_{\text{pelvis}}, C_{\text{rhand}}\}$ for the upper-body IDRM, where $C_{\text{pelvis}}$ constrains the pelvis link to be inside a feasible height region and approximately perpendicular to the ground (i.e. upright), and $C_{\text{rhand}}$ constrains the right hand to be near $\mathbf{y}^*_{\text{rhand}}$. Algorithm 1 highlights our proposed end-pose planning method for bimanual tasks on uneven terrain, where in lines 1-7 $M_{\text{upper}}$ is used to find collision-free upper-body configurations that satisfy the constraints $C$, such that two hands can reach the goal $\mathbf{y}^*$ with a reasonable pelvis pose $T_{\text{pelvis}}$.

It is worth emphasizing that, given an upper-body configuration $\mathbf{q}_n$, the global pose of a link can be calculated by forward kinematics. However, since the pelvis and right hand are two tips of $M_{\text{upper}}$, we can obtain the global poses of these two links from the reach poses stored in the IDRM without computing the forward kinematics. For each tip link, i.e. pelvis and right hand, the IDRM reach pose is referenced in the root (left hand) frame. Given the desired root pose $\mathbf{y}^*_{\text{lhand}}$, the global pose of a tip link is

$$T_n^{\text{tip,world}} = \mathbf{y}^* \times T_n^{\text{tip,root}}, \tag{4.1}$$

where $T_n^{\text{tip,world}}$ and $T_n^{\text{tip,root}}$ represent the tip pose of sample $n$ in global and root frames accordingly. Here $T_n^{\text{tip,root}}$ is pre-computed for each sample during offline processing and $\mathbf{y}^*$ is given for each task. Hence, computing the global poses of the pelvis and the right hand is very efficient in our approach.

After retrieving the global poses, we can then check if the configurations satisfy the pelvis and right hand constraints. For a candidate upper-body configuration $\mathbf{q}_n$, we transform $M_{\text{lower}}$ to $T_{\text{pelvis}}$ and find valid lower-body configurations, i.e. collision-free and valid contacts with the terrain, as shown in lines 8-12 of Algorithm 1. To check foot contacts, we first extract the feasible step regions from the environment. Similar to Equation4.1 with $T_{\text{pelvis}}$ as the $\mathbf{y}^*$, we can obtain the tip (foot) poses in the global frame and check if the foot is within the step regions. If the lower-body configuration has valid contacts, we then combine the candidate upper and lower body configurations to acquire the full-body configuration. A final check is necessary to ensure the combined full-body configuration is balanced. Since multiple valid end-poses may exist, we can either iterate though $M_{\text{upper}}$ and $M_{\text{lower}}$ to find the best candidate, or stop the search once a given threshold $\epsilon$ is met based on the cost function $f(\mathbf{q})$. Different cost functions can be defined for different tasks and environments. In general, for humanoid robots, it is desirable to have an end-pose with minimum travelling distance that is close to the start/nominal configuration. The following cost function is used in our implementation

$$f(\mathbf{q}) = \|T_{\text{pelvis}}(\mathbf{q}) - T_{\text{pelvis}}(\mathbf{q}_s)\|_{W_1} + \|\mathbf{q} - \mathbf{q}_s\|_{W_2}, \tag{4.2}$$

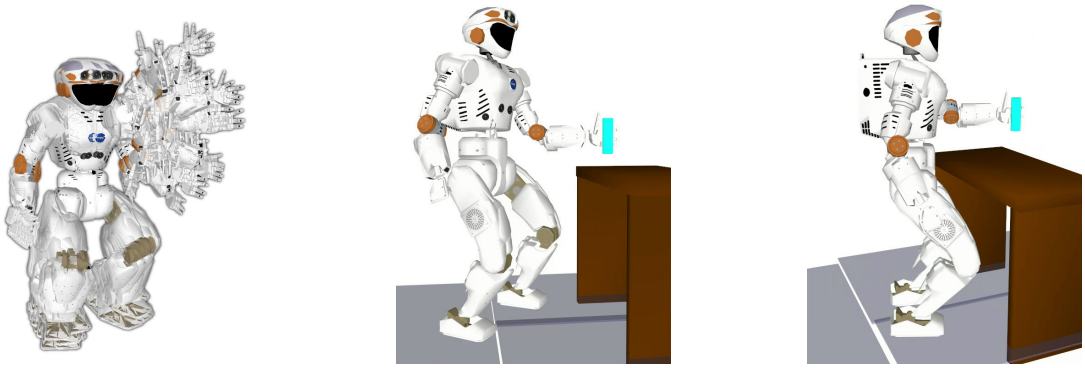


Figure 4.5: The first figure highlights the upper-body IDRM and lower-body DRM samples, followed by two examples of selected end-poses in different scenarios.

where $W_1, W_2$ are weights.

After end-pose planning, the last step is to refine the output and ensure all necessary constraints are satisfied, e.g. the hand(s) need to precisely reach the target, the feet need to be perfectly in contact with the terrain, and the pose needs to be statically balanced. A non-linear optimization-based solver Tedrake [2014] is used to adjust the candidate end-pose with respect to these constraints. The final output of the solver is an end-pose which is collision-free and satisfies all the given constraints. In cases where the solver reports failure or a solution that is in collision, one then needs to return to the next best candidate end-pose until the solver succeeds.

### 4.1.3 *Footstep and Motion Planning*

After fining the end-pose, a footstep planner is invoked to plan a set of footsteps to enable walking from current stance to $\mathbf{p}^*$, followed by a motion planner to generate a valid full-body trajectory to realize the end-pose $\mathbf{q}^*$. Footstep and motion planning are not the main focus of this work, and any suitable algorithms could be used. The footstep planner from Deits and Tedrake [2014] and the full-body motion planner from Yang et al. [2016b] are implemented here.

## 4.2 EVALUATION

### 4.2.1 *Construction of dynamic reachability maps*

We have generated maps with different root/tip links and number of samples to analyse how different splitting of the map affects the performance:

- $\Phi_1$: A upper-body iDRM with the left hand as the root, pelvis and right hand as the tips. Three datasets are generated with different number of samples: $100,000(\Phi_{1a})$, $1,000,000(\Phi_{1b})$ and $4,000,000(\Phi_{1c})$.
- $\Phi_2$: A upper-body iDRM with the left hand as the root, pelvis and right shoulder as the tips. Three datasets are generated with different number of samples: $10,000(\Phi_{2a})$, $100,000(\Phi_{2b})$ and $1,000,000(\Phi_{2c})$.
- $\Phi_3$: A right arm DRM with right shoulder as the root and right hand as the tip. Three data sets are generated with different number of samples: $10,000(\Phi_{3a})$, $100,000$ $(\Phi_{3b})$ and $1,000,000(\Phi_{3c})$.
- $\Phi_4$: A lower-body DRM with the pelvis as the root, left and right feet as the tips. Four datasets are genreated with different number of samples : $1,680(\Phi_{4a})$, $44,400(\Phi_{4b})$, $227,400(\Phi_{4c})$ and $742,560(\Phi_{4d})$.

All datasets are created with 10cm workspace grid resolution. The construction time and file size are highlighted in Table 4.1. The construction time of $\Phi_1$ maps are relatively longer because many of the samples are discarded and only these with both hands fall into the region of interest are kept. The $\Phi_1$ maps are also expensive to store since the kinematic structure includes the entire upper-body with two arms. It is worth emphasizing that the file size of $\Phi_1$ is similar to $\Phi_2$ and $\Phi_3$ combined with same number of samples, e.g. $\Phi_{1b} \approx \Phi_{2c} + \Phi_{3c}$.

The proposed end-pose planning method can be obtained by combining $\Phi_1$ and $\Phi_4$, for example, combining $\Phi_{1a}$ and $\Phi_{4a}$ gives a dataset with a theoretical $10^5 \times 1680 =$ 168 million full-body configurations; combining $\Phi_{1c}$ and $\Phi_{4c}$ gives a dataset with a theoretical 909.6 trillion full-body configurations. A further split method can be

Table 4.1: Analysis of upper- and lower-body reachability maps.

| Map | | No. samples | Construction time (min) | File size (MB) |
|---|---|---|---|---|
| Upper-body two arms | $\Phi_{1a}$ | $10^5$ | 28.8 | 108 |
| | $\Phi_{1b}$ | $10^6$ | 289.7 | 1,082 |
| | $\Phi_{1c}$ | $4 \times 10^6$ | 1090.8 | 4,352 |
| Upper-body left arm | $\Phi_{2a}$ | $10^4$ | 0.25 | 9 |
| | $\Phi_{2b}$ | $10^5$ | 2.61 | 91 |
| | $\Phi_{2c}$ | $10^6$ | 25.0 | 879 |
| Right arm | $\Phi_{3a}$ | $10^4$ | 0.05 | 2 |
| | $\Phi_{3b}$ | $10^5$ | 0.58 | 22 |
| | $\Phi_{3c}$ | $10^6$ | 6.19 | 217 |
| Lower-body two legs | $\Phi_{4a}$ | 1,680 | 0.24 | 1 |
| | $\Phi_{4b}$ | 44,400 | 6.15 | 33 |
| | $\Phi_{4c}$ | 227,400 | 30.0 | 160 |
| | $\Phi_{4d}$ | 742,560 | 103.5 | 535 |

obtained by combining $\Phi_2$, $\Phi_3$ and $\Phi_4$, for example, combining $\Phi_{2c}$, $\Phi_{3c}$ and $\Phi_{4c}$ gives a dataset with a theoretical $2.274 \times 10^{17}$ full-body configurations. It is clear that the total number of full-body configurations increases exponentially with the number of components. However, combining these maps significantly slows down the on-line planning as we are about to show.

### 4.2.2 *End-pose planning benchmarking setup*

We have crated a set of benchmark problems by passing random hands and feet pose constraints, as well as quasi-static balance constraint , into the full-body IK solver to obtain a random but balanced configuration. The configurations are filtered for self-collisions. We then populate spherical obstacles into the free environment randomly but not colliding with the robot until a required number of obstacles is reached. Finally, we can extract the height and position of each foot from the generated configuration and create terrain areas accordingly. A valid end-pose planning problem is thereby generated. We also store the desired poses for both hands, collision environments and terrain areas. Note that the robot configurations are generated to ensure the problem is solvable with at least one solution. The configuration is not known

Table 4.2: End-pose planning performance across different lower-body datasets and using the non-linear full-body IK.

| Method | Map success rate | IK success rate | Final success rate | Avg. time(s) |
|---|---|---|---|---|
| $\Phi_{1b} + \Phi_{4a}$ | 72.7% | 71.8% | 71.4% | $0.08 \pm 0.02$ |
| $\Phi_{1b} + \Phi_{4b}$ | 73.7% | 72.8% | 72.5% | $0.09 \pm 0.03$ |
| $\Phi_{1b} + \Phi_{4c}$ | 80.7% | 79.0% | 78.7% | $0.13 \pm 0.10$ |
| $\Phi_{1b} + \Phi_{4d}$ | 86.3% | 84.8% | 84.2% | $0.23 \pm 0.33$ |
| Non-Linear IK | - | 99.8% | 59.3% | $0.03 \pm 0.01$ |

to the candidate algorithm, and the algorithm is allowed to find a different but valid solutions if multiple solutions exist. In our benchmarking, we created 1000 random problems, each of which contains 20 spherical obstacles with $15 - 20cm$ radius.

### 4.2.3 *Simulation benchmarking*

#### 4.2.3.1 *Different lower-body datasets*

As we have mentioned, the lower-body is used for maintaining balance rather than for maximum reachability. Thus, we should use a dataset that contains enough samples which is sufficient for finding balanced configurations rather than having a dataset with millions of samples that consumes huge amount of memory and slows down online computation. We combine $\Phi_{1b}$ with different $\Phi_4$ maps to analyse the affects different lower-body maps might introduce and therefore select the suitable one for other experiments. We also evaluated the performance by directly applying the non-linear IK without using DRM/iDRM. Table 4.2 shows the success rate and average planning time using different methods. The map success rate is the rate of DRM/iDRM reports finding valid candidate end-poses, which is then passed to the IK adjustment function. The IK success rate is the rate of non-linear IK successfully adjusted the candidate poses and satisfy all constraints. The pose is then passed to a collision checking function, a final success is reported if the pose is collision-free.

We notice that these methods can not achieve 100% success rate, which is caused by several factors: firstly, although we have created each map with millions of con-

figurations, it is still inefficient to cover the high dimensional full-body configuration space (38 for Valkyrie); secondly, in the interest of time, we only allow the method to try the first 10 different poses from $\mathbf{Q}$, where a valid pose with relatively high cost might be discarded; lastly, some valid poses which are not in collision may get invalidated due to aliasing of the occupancy grid. Such artefacts can be reduced by using a finer workspace grid, but they can't be completely eliminated. This is a common issue with all grid-based methods.

It is interesting that the final success rate is very close to the initial map success rate, which means that once the DRM/iDRM maps find candidate end-poses, those poses are very likely to be valid. On the other hand, the direct non-linear IK method reports a 99.8% success rate, but only 59.3% is finally valid, e.g. collision-free. The result suggests that using only the non-linear IK is inefficient in cluttered environments, and the proposed method is indeed improving the success rate.

The benchmarking was done in randomized and complex environments designed to fully evaluate different approaches. Although the methods do not achieve 100% success rate in the benchmarking, as we will show later in Section 4.2.4, they are sufficient for solving practical problems. Based on the result we conclude that the success rate as well as planning time increase with the number of lower-body samples. We use the lower-body dataset $\Phi_{4c}$ for the rest of the experiments. However, other datasets with more samples might be used depending on the different demands between success rate and planning time.

#### 4.2.3.2 *Different map combinatorics*

We choose to split the humanoid robot into two parts at pelvis. However, one can further split the upper-body into smaller parts, e.g. left body part ($\Phi_2$) and right arm ($\Phi_3$). Table 4.3 shows the end-pose planning result of using different upper-body maps, where the success rate and planning time increases with the number of samples as expected. However, the further splitting ($\Phi_2 + \Phi_3 + \Phi_4$) leads to a much longer planning time while the success rate is not significantly improved compare to the proposed splitting ($\Phi_1 + \Phi_4$). Furthermore, in the case of using further split method with maps $\Phi_{2c} + \Phi_{3c} + \Phi_{4c}$, the final success rate is lower than using pro-

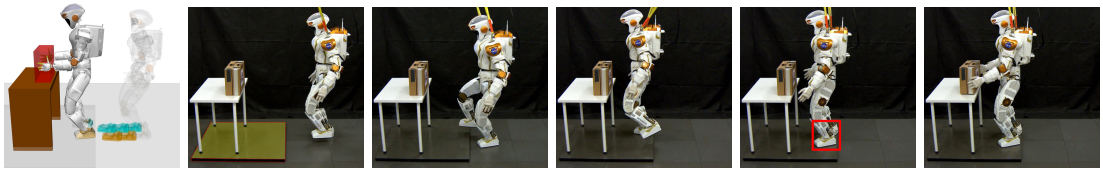
Table 4.3: End-pose planning performance analysis of using same lower-body dataset with different upper-body datasets. Considering the trade-off between success rate and planning time, the method $\Phi_{1c} + \Phi_{4c}$ is used for hardware experiments.

| Method | Total No. samples | Map success rate | IK success rate | Final success rate | Avg. time (s) |
|---|---|---|---|---|---|
| $\Phi_{1a} + \Phi_{4c}$ | $2.274 \times 10^{10}$ | 57.9% | 57.1% | 56.8% | $0.04 \pm 0.01$ |
| $\Phi_{1b} + \Phi_{4c}$ | $2.274 \times 10^{11}$ | 80.7% | 79.0% | 78.7% | $0.13 \pm 0.10$ |
| $\Phi_{1c} + \Phi_{4c}$ | $9.096 \times 10^{11}$ | 88.6% | 85.7% | **85.1**% | $0.40 \pm 0.37$ |
| $\Phi_{2a} + \Phi_{3a} + \Phi_{4c}$ | $2.274 \times 10^{13}$ | 70.0% | 65.1% | 63.7% | $0.10 \pm 0.05$ |
| $\Phi_{2b} + \Phi_{3b} + \Phi_{4c}$ | $2.274 \times 10^{15}$ | 91.3% | 83.5% | 80.4% | $0.56 \pm 0.39$ |
| $\Phi_{2c} + \Phi_{3c} + \Phi_{4c}$ | $2.274 \times 10^{17}$ | 96.9% | 85.0% | 81.2% | $8.08 \pm 4.68$ |

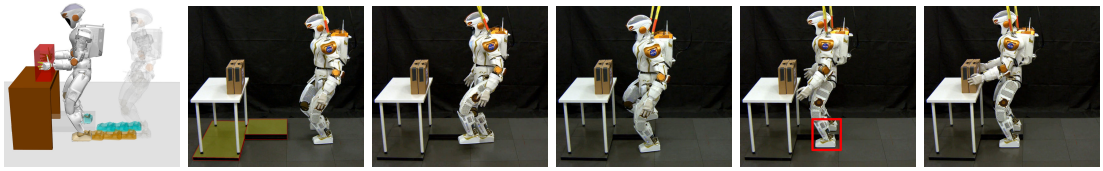
posed split method with maps $\Phi_{1c} + \Phi_{4c}$. Note that the map reports a 96.9% success rate, but dropped to 85.0% after IK adjustment, most of which were caused by fail to satisfy balance constraint. This means further splitting the body leads to higher chance of violating the balance constraint of the full-body. Splitting the upper- and lower-body at the pelvis link thereby is proved to be the most practical considering the trade-off between coverage, planning success rate, and algorithm runtime. We use the proposed split method with datasets $\Phi_{1c}$ for upper-body and $\Phi_{4c}$ for lower-body for the following experiments on robot hardware,

### 4.2.4 *Hardware experiments*

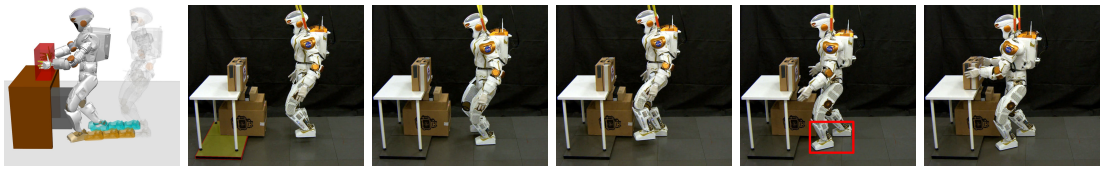
To demonstrate the capability of end-pose planning on uneven terrain, we created three bimanual box-picking tasks with different terrain types. In the first scenario $B1$ (Figure 4.6a), the robot has to walk onto a higher floor, which in theory can be found by classic iDRM as well; in the second case $B2$ (Figure 4.6b), the robot has to stand on surfaces at two different heights; in the last scenario $B3$ (Figure 4.6c), the robot needs to avoid a collision between its right leg and a large obstacle during the picking task. Our method is capable of finding different collision-free end-poses in these environments. We found that the possible pelvis poses are quite limited in practice for bimanual tasks, i.e. the robot has to stand directly in front facing the box in order to pick it up with two hands. Nevertheless, our DRM/iDRM hybrid method



(a) $B1$: Pick up a box from a higher terrain.



(b) $B2$: Pick up a box by placing the right on a higher terrain.



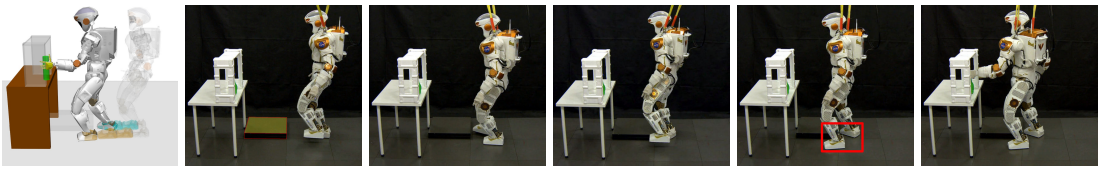
(c) $B3$: Pick up a box while the right leg position is restricted by a large obstacle.

Figure 4.6: Bimanual box-picking tasks on the terrains of different heights. The robot is able to automatically find appropriate standing locations and full-body configurations.

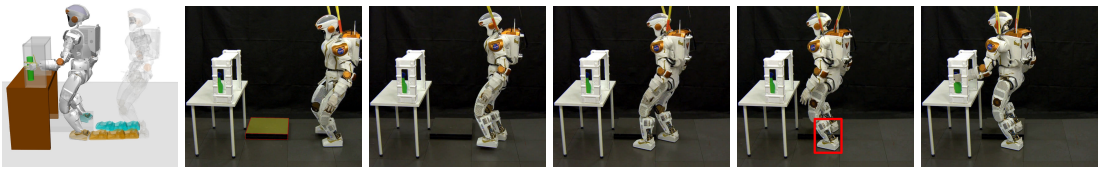
provides a valid solution for the robot to perform bimanual picking tasks in presence of uneven terrain.

We further validated two single-arm grasping tasks where the target was placed at different locations, as shown in Figure 4.7. A upper-body iDRM is created with the left hand as root link and pelvis as tip link. The right arm joints are set to a pre-defined nominal configuration for all samples, as shown in Figure 4.5. The constrain set $C$ then contains pose constraints only for the pelvis but not for the right hand. In the first scenario $S1$ (Figure 4.7a), the target was placed at the edge of the table, where the robot could easily grasp without being too close. So, the robot could stay away from the high surface, while keeping the target at a reachable distance. Whereas in the second task $S2$ (Figure 4.7b), the target was placed further away from the edge of the table and enclosed by the obstacle. The end-pose planner found a feasible configuration to place two feet on different surfaces so the robot was close enough for grasping the target.

We would like to highlight that with the modular and combined forward inverse dynamic reachability maps presented in this work, we are able to find end poses which include lunging body or taking a sidestep (in scenarios $B3$ and $S1$) for increas-



(a) *S1*: Grasp a target placed at the edge of the table.



(b) *S2*: Grasp a target placed deeper on the table.

Figure 4.7: Single-handed grasping tasks on the terrains of different heights. Case I: the target is easily reachable, so the robot does not need to be too close to the table; Case II, the robot needs to be closer to the table by placing the right foot on the uneven terrain.

ing the reachable workspace by leveraging the advantage of the legged system. This is in contrast with the scenarios demonstrated in Chapter 3 where we limited the foot poses to a constant distance and planning for the mid-feet point. A supplementary video can be found at https://youtu.be/o-05EHf-gg8.

## 4.3 CONCLUSION

We presented a novel end-pose planning algorithm that combines the Forward and Inverse Dynamic Reachability Map (DRM/IDRM) for humanoid robots to automatically find appropriate end-poses in presence of uneven terrain. Using NASA's Valkyrie humanoid as a testbed, we demonstrated the effectiveness of the proposed method in planning end-poses for both single-arm and bimanual tasks on uneven terrains.

A current limitation of our method is the amount of memory required for storing the maps, e.g. 4.5GB for Valkyrie using the datasets $\Phi_{1c}$ and $\Phi_{4c}$. Techniques such as interpolation can be used to reduce the required number of samples, thus reducing the storage size. However, those approaches are normally used for low dimensional problems and do not scale. More importantly, collision information can not be encoded with the interpolation, meaning that online collision checking needs to be invoked for each interpolation which significantly increases the planning time.

Our future work involves investigating new methods of encoding the configuration-to-workspace mapping for better memory efficiency. This will allow us to increase the resolution of the voxel grid and improve the success rate of our method.

In this part of the thesis, we have discussed how to find the "goal state", for either fixed-base robotic arms or humanoid robots, single- or multi-limb structures. The assumption we have mentioned in the beginning of this part, i.e. *given the start and goal states*, has now been satisfied and the actual motion planning can proceed.

# Part II

# MOTION PLANNING

# 5

# MOTION PLANNING PRELIMINARIES

A fundamental need in robotics is to have algorithms that convert high-level specifications of tasks from humans into low-level descriptions of how to move, which often refers to the term *motion planning*. It involves automatically finding a sequence of configurations that takes the robot from a start to a goal. In Part I, we have discussed how to efficiently find the goal, i.e. end-pose planning (Equation 1.1). We now move forward to the next phase — motion planning (Equation 1.2).

Generally speaking, motion planning can be grouped into optimization-based and search-based algorithms. Optimization-based methods (Ivan et al. [2013], Ratliff et al. [2009], Schulman et al. [2014]) generate optimal trajectories with respect to some cost function, but may get stuck in local minima and fail to produce valid solution when the problem is non-convex or ill-defined. On the other hand, search-based approaches (Lavalle [1998], Kuffner and LaValle [2000], Kavraki et al. [1996], Elbanhawi and Simic [2014], Yang et al. [2016b]) promise to find valid solutions for complex problems. As titled, the aim of this thesis is to solve high dimensional planning problems in complex and cluttered environments, which fits nicely to the type of problems promised by search-based methods. Thus, we will focus on solving the motion planning problems by efficiently finding a valid trajectory using search-based method. The trajectory then can be executed directly, or further improved by other optimization-based methods.

## 5.1 COMPLETENESS IN ROBOT MOTION PLANNING

The notion of *completeness*, a key property for motion planning algorithms, becomes very important in this thesis since we are considering planning problems in very complex environments. An algorithm is considered *complete* if for any input it cor-

rectly reports whether there is a solution or not. If a solution exists, it must return one in finite time. Optimization-based methods are incomplete due to local optima, meaning that in many cases the algorithms are unable to find a solution when one or even multiple solutions exist. Unfortunately, search-based algorithms are also incomplete. A weaker notion of completeness called *probabilistic completeness* is used to describe random sampling-based algorithms. This means that with enough samples, asymptotically, infinite, the probability that it finds an existing solution converges to one. Another term, *resolution completeness*, is used if an algorithm guarantees to find a solution in finite time; however, if a solution does not exist, the algorithm may run forever by incrementally increasing the sampling resolution; or, the algorithm may terminate in finite time by reporting no solution at certain resolution, though a solution may exist at a finer resolution.

## 5.2 SEARCH-BASED MOTION PLANNING

This section presents two main categories of search-based motion planning algorithms, *discrete search planning* and *sampling-based planning*. We first generalize each approach and then discuss some legacy as well as state-of-the-art algorithms in each category.

### 5.2.1 *Discrete Search Planning*

The discrete search planning provides introductory concepts for better understanding of the search-based planning, which will be directly used or extended by many other more advanced algorithms.

Let $\mathbf{x}$ be a *state* of the robot or/and world, and $\mathcal{X}$ be the set of all possible states called the *state space*[1]. It is important that, for discrete planning, the state space set $\mathcal{X}$ must be countable, i.e. finite or countably infinite. Let $\mathbf{u}$ be an *action*, which can

1 The state space is equivalent to configuration space if $\mathbf{x} = \mathbf{q}$.

**Algorithm 2** General discrete search algorithm

**Require:** $\mathbf{x}_{\text{start}}, \mathbf{x}_{\text{goal}}, f$
**Ensure:** $\mathbf{x}_{[0:T]}, T \in \mathbb{N}$
$Q_{\text{open}}$.Insert($\mathbf{x}_{\text{start}}$) and $Q_{closed}$.Insert($\mathbf{x}_{\text{start}}$)
$p(\mathbf{x}_{\text{start}}) = (\text{Null}, \text{Null})$
**while** $Q_{\text{open}}$ is not empty **do**
    $\mathbf{x} \leftarrow Q_{\text{open}}$.GetFirst()
    $Q_{\text{open}}$.Remove($\mathbf{x}$)
    **if** $\mathbf{x} = \mathbf{x}_{\text{goal}}$ **then**
        Success = True
        Break
    **for all** $\mathbf{x} \in N_{\mathcal{X}}(x)$ **do**
        **if** $\mathbf{x}' \notin Q_{\text{closed}}$ **then**
            $Q_{\text{open}}$.Insert($\mathbf{x}'$) and $Q_{\text{closed}}$.Insert($\mathbf{x}'$)
            $p(\mathbf{x}') = (\mathbf{x}, \mathbf{u})$
        **else**
            $p(\mathbf{x}') \leftarrow \text{Resolve}(\mathbf{x}, \mathbf{x}', p(\mathbf{x}'))$
**if** Success = True **then** $\mathbf{x}_{[0:T]} = \emptyset$
    **while** $p(x) \neq \text{Null}$ **do**
        $\mathbf{x}_{[0:T]}$.PushFront($\mathbf{x}$)
        $\mathbf{x} = p(\mathbf{x}')$

be applied to a state $\mathbf{x}$, to produce a new state $\mathbf{x}'$. We define $f$ as the *state transition function*,

$$\mathbf{x}' \leftarrow f(\mathbf{x}, \mathbf{u}). \tag{5.1}$$

Let $U(\mathbf{x})$ denote all the actions that a state $\mathbf{x}$ can take. The set of all neighbour states of $\mathbf{x}$, i.e. states that can be reached from $\mathbf{x}$ by applying an action $\mathbf{u} \in U(\mathbf{x})$,

$$N_{\mathcal{X}}(\mathbf{x}) = \{\mathbf{x}' \mid \mathbf{x}' \leftarrow f(\mathbf{x}, \mathbf{u}), \mathbf{u} \in U(\mathbf{x})\}, \tag{5.2}$$

is called the neighbourhood of $\mathbf{x}$.

Given a start state $\mathbf{x}_{\text{start}} \in \mathcal{X}$ and a goal state $\mathbf{x}_{\text{goal}} \in \mathcal{X}$, the planning problem is to find a sequence of actions that when applied, transforms the start state to goal state. A general search template is highlighted in Algorithm 2. There are two different status for each state: *open* or *closed*. A state is marked as open if it has not been visited yet, whereas initially all states except $\mathbf{x}_{\text{start}}$ are open. A state is marked as closed if it has been visited at least once. The transitions are recorded in $p(\mathbf{x}') = (\mathbf{x}, \mathbf{u})$, where $\mathbf{x}$

is often called the *parent* of $\mathbf{x}'$. This information is used to retrieve the plan in correct order. When looping through the open set $Q_{\text{open}}$, the choice of which state to check first varies across different algorithms. Also, when a state gets visited more than once, how to resolve the parent state is handled differently in $\text{Resolve}(\mathbf{x}, \mathbf{x}', \text{p}(\mathbf{x}'))$. Different algorithms have been proposed to solve the search problem, we now presents several classical search algorithms. It is worth emphasising that, since the state space is finite and all states are known, these discrete planning algorithms are *complete*.

#### 5.2.1.1 *Breadth First Search*

The breadth first search method (Lee [1961]) maintains $Q_{\text{open}}$ as a *first-in first-out* queue, which returns the state that is firstly added. This property causes the frontier to grow uniformly, meaning all possible trajectories with $T$ steps are exhausted before trajectories with $T + 1$ steps are investigated. Therefore, breadth first method guarantees to find a path with minimum steps $T$. Note that the trajectory with minimum steps is not necessarily the optimal one. There is no work to do in the $\text{Resolve}()$ function for breadth first method.

#### 5.2.1.2 *Depth First Search*

Opposite to breath first search, the open set $Q_{\text{open}}$ in depth first search method is a *last-in first-out* queue, where the last added state will be returned first. As a result, depth first search explores faster in the space and can find longer trajectories very early. Similar to breadth first, there is no work to do in $\text{Resolve}()$ function for depth first method as well.

#### 5.2.1.3 *Dijkstra's Algorithm*

The aforementioned two algorithms have no preference of one action over any others during the search, meaning that they only find a feasible path without reasoning about the quality of it. The Dijkstra's algorithm (Dijkstra [1959]), however, finds optimal plans based on given criteria.

Let $c(\mathbf{x}, \mathbf{u}) \in \mathbb{R}_{\geq 0}$ be a non-negative value denote the cost to apply action $\mathbf{u}$ on state $\mathbf{x}$, and $C(\mathbf{x})$ denote the *cost-to-come* from the initial state $\mathbf{x}_{\text{start}}$ to $\mathbf{x}$. The cost-to-come

can be computed incrementally during the search process. Initially, $C(\mathbf{x}_{\text{start}}) = 0$, and the cost-to-come is computed as $C(\mathbf{x}') = C(\mathbf{x}) + c(\mathbf{x}, \mathbf{u})$ each time a new state $\mathbf{x}'$ is generated. In Dijkstra's algorithm, the open set $Q_{\text{open}}$ is sorted based on the cost-to-come, and the state with lowest $C(\mathbf{x})$ is returned when calling $Q_{\text{open}}$.GetFirst(). This, however, means that if a state $\mathbf{x}'$ is visited again, the new path to $\mathbf{x}'$ might have lower cost than the currently stored path. So, extra process is required in Resolve() to ensure the optimality, where the path with smaller cost is selected and the $p(\mathbf{x}')$ is reassigned accordingly.

#### 5.2.1.4 *A* Search*

The A* (A-star, Hart et al. [1968]) is an extension of Dijkstra's algorithm that tries to reduce the total number of explored states and improve search efficiency by introducing a heuristic estimate of the cost to get to the goal from a given state. Let $G(\mathbf{x})$ denote the *cost-to-go* from state $\mathbf{x}$ to $\mathbf{x}_{\text{goal}}$. The A* search algorithm works in exactly the same way as Dijkstra's algorithm. The only difference is the function used to sort $Q_{\text{open}}$, where the sum $C(\mathbf{x}) + G(\mathbf{x})$ is used, implying that the priority queue is sorted by estimates of the optimal cost from $\mathbf{x}_{\text{start}}$ to $\mathbf{x}_{\text{goal}}$. The A* algorithm guarantees to find optimal plans based on the metric defined in $G(\mathbf{x})$.

### 5.2.2 *Sampling-based Planning*

Let $b_{n,l}$ and $b_{n,u}$ denote the lower and upper joint limits of the robot's $n$-th actuator, $n \in N$, where the joint position can be set to any real values $q_n \in [b_{n,l}, b_{n,u}]$. Thus, the configuration space $\mathcal{C}$ is *uncountably infinite*, which means all these aforementioned discrete planning algorithms are inapplicable since these algorithms require the state space $\mathcal{X}$ be finite or countably infinite. Fortunately, it is not always necessary to know all states, one can still find valid plans by only knowing or searching a subset of the entire configuration space. In the rest of this section, we will discuss some so-called sampling-based planning algorithms that are able to plan complex motion in a uncountably infinite configuration space. However, note that these algorithms are incomplete, instead, they are probabilistically complete. We first explain some

**Algorithm 3** General single-query sampling-based algorithm

**Require:** $\mathbf{q}_{\text{start}}, \mathbf{q}_{\text{goal}}$
**Ensure:** $\mathbf{q}_{[0:T]}$
Initialise search graph $\mathcal{G}(V, E)$ //$V$: vertex/configuration/state, $E$: edge/local-path
$\mathcal{G}$.AddVertex($\mathbf{q}_{\text{start}}$)
$\mathcal{G}$.AddVertex($\mathbf{q}_{\text{goal}}$)
**while** Terminate = False **do**
    hasNewSample = False
    **while** hasNewSample = False **do**
        $\mathbf{q}_{\text{new}}$ = NewSample()
        hasNewSample = SatisfyAllConstraints($\mathbf{q}_{\text{new}}$)
    $\mathcal{G}$.Expand($\mathbf{q}_{\text{new}}$)
    **if** $\mathbf{q}_{\text{new}} = \mathbf{q}_{\text{goal}}$ **then**
        $\mathbf{q}_{[0:T]}$ = ConstructPath($\mathcal{G}$)
        **return** Success
**return** Failure

preliminaries in sampling-based methods, followed by details of several well-known algorithms.

Given a start state $\mathbf{q}_{\text{start}} \in \mathcal{C}_{\text{free}}$ and a goal state $\mathbf{q}_{\text{goal}} \in \mathcal{C}_{\text{free}}$, the planning problem can be defined as finding a sequence of configurations

$$\begin{aligned} \mathbf{q}_{[0:T]} \leftarrow & \text{ MotionPlanning}(\mathbf{q}_{\text{start}}, \mathbf{q}_{\text{goal}}) \\ \text{subject to} \quad & \text{Edge}(\mathbf{q}_t, \mathbf{q}_{t+1}) \subset \mathcal{C}_{\text{free}} \\ & \mathbf{q}_0 = \mathbf{q}_{\text{start}} \\ & \mathbf{q}_T = \mathbf{q}_{\text{goal}} \end{aligned} \tag{5.3}$$

However, the collision-free configuration space region $\mathcal{C}_{\text{free}}$ in a particular environment is difficult, or even impossible, to compute for multi-DoF robots. Instead, it is relatively easy to check whether a configuration $\mathbf{q}$ is in $\mathcal{C}_{\text{free}}$, which is normally called collision detection or collision checking. Traditionally, the main constraint in sampling-based planners is collision-free constraint (including both self-collision-free and robot-environment collision-free constraints). However, some other constraints might also be required for other types of robots. For example, a humanoid robot

**Algorithm 4** General multiple-query sampling-based algorithm

// Off-line pre-processing

**Ensure:** $\mathcal{G}(V, E)$

Initialise search graph $\mathcal{G}(V, E)$
**while** Terminate = False **do**
    hasNewSample = False
    **while** hasNewSample = False **do**
        $\mathbf{q}_{\text{new}}$ = NewSample()
        hasNewSample = SatisfyAllConstraints($\mathbf{q}_{\text{new}}$)
    $\mathcal{G}$.Expand($\mathbf{q}_{\text{new}}$)
    Terminate = HaveEnoughSamples() or TimeUp()

// On-line planning

**Require:** $\mathcal{G}(V, E)$, $\mathbf{q}_{\text{start}}$, $\mathbf{q}_{\text{goal}}$
**Ensure:** $\mathbf{q}_{[0:T]}$

$V_{\text{start}}, V_{\text{goal}}$ = FindStartAndGoalVertices($\mathbf{q}_{\text{start}}, \mathbf{q}_{\text{goal}}$)
FoundSolution, $\mathbf{x}_{[0:T]}$ = DiscreteSearch($V_{\text{start}}, V_{\text{goal}}, \mathcal{X} \equiv \mathcal{G}$)
**if** FoundSolution = True **then**
    $\mathbf{q}_{[0:T]}$.PushFront($\mathbf{q}_{\text{start}}$)
    $\mathbf{q}_{[0:T]}$.PushBack($\mathbf{q}_{\text{goal}}$)
    **return** Success
**else**
    **return** Failure

state needs to satisfy also balance constraint. Thus, in general, a trajectory should contain only states in the *valid region*,

$$\mathbf{q}_{[0:T]} \rightarrow \mathcal{C}_{\text{valid}}, \tag{5.4}$$

where a valid state $\mathbf{q} \in \mathcal{C}_{\text{valid}} \subseteq \mathcal{C}_{\text{free}}$ satisfies all necessary constraints, such as collision-free, within joint limits, balanced, etc.

Sampling-based algorithms treat collision check or validity check as a black-box function,

$$\phi(\mathbf{q}) : \mathcal{C} \rightarrow \{\text{True, Flase}\}, \tag{5.5}$$

where $\phi(\mathbf{q}) = \text{True}$ if $\mathbf{q} \in \mathcal{C}_{\text{valid}}$, and $\phi(\mathbf{q}) = \text{False}$ if $\mathbf{q} \notin \mathcal{C}_{\text{valid}}$. To obtain a valid state, one can keep generating random states $\mathbf{q} \in \mathcal{C}$ and pass them into the validity check module until a valid state $\mathbf{q} \in \mathcal{C}_{\text{valid}}$ is found. After having the *state sampler*

and *validity checker* modules ready, different algorithms can be developed. In general, there are two different types of problems in sampling-based algorithms, *single-query* and *multiple-query*. For single-query problem, the start and goal states are given only once per query, which means there is no need for pre-computation. In contrast, for multiple-query problems, different start and goal states are given for same environment, where pre-processing could make the planning more efficient. Many different sampling-based methods have been proposed over the last two decades, most of which can be generalized by Algorithm 3 and 4. The main differences across those algorithms are the ways of how new samples are generated, i.e. NewSample() and how the graph is expanded, i.e. $\mathcal{G}$.Expand().

In particular, we explain two algorithms and their notable extensions in details — a single-query planner *Rapidly Exploring Random Tree* (RRT, Kavraki et al. [1996]) and a multiple-query planner *Probabilistic Roadmap* (PRM, Lavalle [1998]).

#### 5.2.2.1 *Rapidly Exploring Random Tree (RRT)*

The process of Algorithm 3 can be seen as incrementally constructing a search tree that gradually increases the size of $\mathcal{C}_{\text{free}}$ without considering the whole set $\mathcal{C}$, which is uncountably infinite. In general, this family of trees is called *rapidly exploring dense tree* (RDT). In a special case where the new sample is generated randomly when calling the NewSample(), the tree is referred as a *rapidly exploring random tree* (RRT, Lavalle [1998]). Algorithm 5 highlights the graph expansion process in RRT. Given a new state $\mathbf{q}_{\text{new}}$, the closest state $\mathbf{q}_{\text{near}}$ in the current tree is found. An interpolation function will be invoked, if the new state is too far from the closest one, to find the state along the same direction with the maximum allowed distance for one step. This is because the probability of having collision increases with the distance. A well chosen maximum distance for one step, $d_{\max}$, should reduce the chance of running into collision while keeping relatively faster exploration speed. Finally, a CheckMotion$(\mathbf{q}_{\text{near}}, \mathbf{q}_{\text{new}})$ function is called to evaluate if the motion from $\mathbf{q}_{\text{near}}$ to $\mathbf{q}_{\text{new}}$ is valid, e.g. collision-free. A typical CheckMotion$(\mathbf{q}_a, \mathbf{q}_b)$ function assumes the robot can move from $\mathbf{q}_a$ to $\mathbf{q}_b$ linearly, where a densely interpolated states between $\mathbf{q}_a$ to $\mathbf{q}_b$ will be checked. Different motion checking and interpolation functions can be applied in special cases, such

**Algorithm 5** RRT implementation of $\mathcal{G}$.Expand($\mathbf{q}_{\text{new}}$) in Algorithm 3

$\mathbf{q}_{\text{near}} = \mathcal{G}.\text{GetNearestNeighbor}(\mathbf{q}_{\text{new}})$
$d = \text{Distance}(\mathbf{q}_{\text{new}}, \mathbf{q}_{\text{near}})$
**if** $d > d_{\max}$ **then**
    $\mathbf{q}_{\text{new}} = \text{Interpolate}(\mathbf{q}_{\text{near}}, \mathbf{q}_{\text{new}}, d_{\max})$
**if** CheckMotion($\mathbf{q}_{\text{near}}, \mathbf{q}_{\text{new}}$) = True **then**
    $\mathcal{G}.\text{AddVertex}(\mathbf{q}_{\text{new}})$
    $\mathcal{G}.\text{AddEdge}(\mathbf{q}_{\text{near}}, \mathbf{q}_{\text{new}})$

as Kinodynamic (Hsu et al. [2002]) or Dubins-Car planning (Karaman and Frazzoli [2013]). New state and edge will be appended to the existing tree if the motion is valid. The search procedure keeps repeating until the new sample is equal or close to the goal state.

There exist many different RRT variations (Elbanhawi and Simic [2014]), where the two most famous extensions are RRT* (RRT-Star) (Karaman and Frazzoli [2011]) and RRT-Connect (Kuffner and LaValle [2000]). The RRT* method performs a re-wire step, after new vertex and edges are added, to find better paths from start to the new state. Given infinite run time, RRT* converges to an asymptomatically optimal solution. However, such re-wire significantly slows down the exploring particularly for solving high dimensional problems. On the other hand, by introducing a bi-directional search scheme, RRT-Connect shows great improvement in terms of efficient valid motion planning in high dimensions.

#### 5.2.2.2 *Probabilistic Roadmap (PRM)*

While RRT-like approaches create new explore trees from scratch every time for different start and goal states. The well known multi-query planning method PRM creates a roadmap off-line and reuses the map for different planning queries. During the off-line construction phase, instead of finding one closest vertex in the existing tree, a set of near vertices will be selected. The motion between new sample and each near vertex is checked, and the valid ones will be added into the map. During on-line phase, given a start and goal states, $\mathbf{q}_{\text{start}}$ and $\mathbf{q}_{\text{goal}}$, we first find the closest vertices in the existing roadmap, $V_{\text{start}}, V_{\text{goal}}$. A discrete search, such as A* or Dijkstra's algorithm, is then invoked to find a path in the roadmap that connects $V_{\text{start}}$ and $V_{\text{goal}}$. Finally,

**Algorithm 6** PRM implementation of $\mathcal{G}$.Expand($\mathbf{q}_{\text{new}}$) in Algorithm 4

1: $\mathcal{G}$.AddVertex($\mathbf{q}_{\text{new}}$)
2: $N_{\mathcal{G}}(\mathbf{q}_{\text{new}}) = \mathcal{G}$.GetNearStates($\mathbf{q}_{\text{new}}$)
3: **for all** $\mathbf{q}_n \in N_{\mathcal{G}}(\mathbf{q}_{\text{new}})$ **do**
4: **if** CheckMotion($\mathbf{q}_{\text{n}}, \mathbf{q}_{\text{new}}$) = True **then**
5: $\mathcal{G}$.AddVertex($\mathbf{q}_{\text{new}}$)
6: $\mathcal{G}$.AddEdge($\mathbf{q}_{\text{n}}, \mathbf{q}_{\text{new}}$)

the start and goal states, $\mathbf{q}_{\text{start}}$ and $\mathbf{q}_{\text{goal}}$, are appended to the path accordingly. An interesting fact is that, although the discrete search algorithm guarantees completeness on the roadmap, but the roadmap itself is a finite subset of the infinite configuration space states. Thus, the discrete search algorithm is complete, however, a PRM algorithm, that internally uses discrete search, is incomplete. Normally, PRM is allowed to draw new samples only, which makes it probabilistically complete. We can also completely remove off-line phase and generate new roadmap on-line, which will make PRM a single-query planner (Sánchez and Latombe [2003]). Similar to RRT*, there also exists an optimal version of PRM, namely the PRM* (PRM-Star, Karaman and Frazzoli [2011]). In complex environment, both on-line sampling and re-wiring are very expensive processes due to multiple collision checking function calls.

## 5.3 DYNAMIC ROADMAPS

In sampling-based algorithms, collision checking is normally the most expensive operation and reportedly consumes up to $90 - 95\%$ of the planning time Hsu and Sun [2004]. Lazy collision checking is used to delay the collision checking until it is needed Bohlin and Kavraki [2000]. One can also define possible collision regions and limit collision checking to these regions Sánchez and Latombe [2003]. However, these techniques only reduce the collision checking time indirectly by reducing the number of calls, but not the actual computation time of collision checking function. Parallel implementations have been proposed to speed up collision checking and motion planning Bialkowski et al. [2011], Pan and Manocha [2012], but these approaches focus on parallelization and system implementation based on existing algorithms.

In contrast, the Dynamic Roadmap (DRM[2]) algorithm Leven and Hutchinson [2002], an extension to PRM, algorithmically reduces the collision checking time by encoding configuration-to-workspace occupation information. Given different environments, the DRM can efficiently remove invalid edges and form a valid subset of the full roadmap. Subsequently, search algorithms can proceed without considering collision checking since the remaining vertices and edges are all collision-free.

However, encoding the occupation information requires to store a significant amount of data, which needs to be loaded into memory during run-time. In the early work Kallman and Mataric [2004], Leven and Hutchinson [2002], the low amount of available memory allowed storing only small roadmaps with limited number of vertices and edges. Without enough vertices and edges to densely cover the whole configuration space,the DRM algorithm was *incomplete* with very low planning success rates Kallman and Mataric [2004], Voelz and Graichen [2016].

### 5.3.1 *Dynamic Roadmap Preliminaries*

A classical PRM contains a connected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} \subset \mathcal{C}_{\text{free}}$ are the vertices and $\mathcal{E}$ are the edges that connect two neighbouring vertices, as highlighted in Figure 5.1 (left). However, these vertices and edges are generated during off-line pre-processing, which may not be valid in unknown and non-static environments. The validity of pre-stored vertices and edges must be checked, and in many cases we need to sample new collision-free configurations during the on-line phase which is very time consuming.

The dynamic roadmap (DRM) is a variation of the PRM proposed by Leven and Hutchinson [2002]. The DRM is *dynamic* in the sense that the graph $\mathcal{G}$ can be dynamically updated in different environments. The invalid vertices and edges can be efficiently identified and removed, with the remaining ones forming a new graph of only valid vertices and edges. This reduced graph is ready for path searching algorithms without considering collision checking. The key feature of DRM is a configuration-to-workspace mapping, as highlighted in Figure 5.1 (right). One can find the list

2 It is worth emphasising that, in Part I, the term DRM refers to **Dynamic Reachability Map** for end-pose planning, while in the present Part, DRM refers to **Dynamic Roadmap** for motion planning.

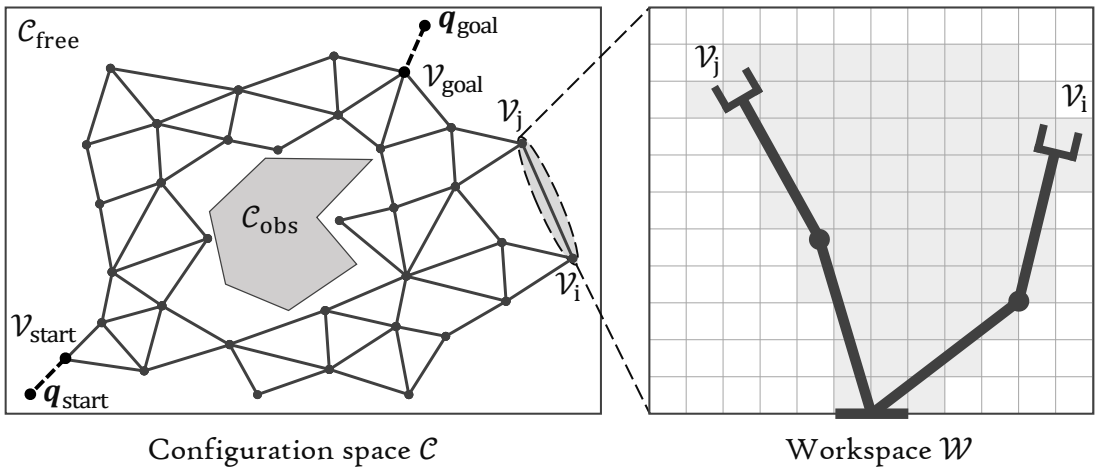



Figure 5.1: Left: probabilistic roadmap in configuration space; right: workspace swept volume of an edge.

of discretized workspace voxels which an edge occupies, referred to as the swept volume. If one or more of the voxels in the swept volume are in collision with the environment, then the corresponding edge becomes invalid. In practice, it is inefficient to check the swept volume of all edges when the roadmap contains too many vertices and edges. Instead, the occupation information is stored per each workspace voxel, i.e. each voxel stores a list of edges that sweep through this voxel. In a new environment, we first find all the voxels that are occupied by the environmental obstacles. Then by iterating through the occupation lists of these invalid voxels, all the invalid edges can be found and removed accordingly.

### 5.3.2 *Limitations: Curse-of-Dimensionality*

The main observed limitation of the existing DRM method is its low success rate, meaning that in many cases, the DRM cannot find a valid plan even when a solution exists. The success rate could be as low as $20-30\%$ Kallman and Mataric [2004], Voelz and Graichen [2016]. This is due to inadequate edges in the roadmap. However, the number of edges cannot be arbitrarily increased due to the *curse-of-dimensionality* and limited memory for storing the swept volume information.

For a 6-DoF robotic manipulator, where $K = 15$ different discretization values are chosen for each joint, e.g. evenly from $[-\pi, \pi]$, we have $15^6 \approx 11.4$ million configu-

rations for the entire arm—corresponding to the number of vertices in the DRM. A DRM with this many vertices can easily form hundreds of millions of edges. If each edge sweeps around 500 workspace voxels, that means the swept volume information may contain tens of billions of indices (`unsigned int`[3]), which is infeasible to store on commodity hardware. Note that this is only a 6-DoF case, the problem is undoubtedly exacerbated as the dimensionality of the robot further increases.

To make the problem tractable, compression techniques were introduced to reduce the number of vertices and edges to a manageable level Leven and Hutchinson [2002], Murray et al. [2016]. However, it is clear that the roadmap loses completeness once any standard compression method gets introduced, which will then result in a low success rate. A method of hit-matrix was proposed to compress the dataset without reducing the total number of vertices and edges Schumann-Olsen et al. [2014], which allows the DRM to store up to one million vertices. However, for robots with more actuators, a roadmap with only one million configurations is far from being complete.

It has been proven that path search algorithms such as A* is complete on the given graph Coppin [2004], but the roadmap/graph is itself incomplete as it does not cover the whole configuration space, i.e. $\mathcal{V} \subsetneq \mathcal{C}$. Fortunately, as we will formally explain in next Chapter, resolution completeness could be achieved if the roadmap contains a sufficient number of samples densely covering the configuration space at a certain grid resolution.

3 The size of one `unsigned int` is 4 Byte on 64-bit operating systems. Hence storing indices for the occupation information in this example requires $4 \times 500 \times 10^8$ Byte $\approx$ 186.5 GB of memory.

# 6

# REAL-TIME MOTION PLANNING FOR ROBOTIC ARMS


Figure 6.1: The 7-DoF KUKA LWR robot with Dexterous Hand operating inside a cage. Left: grasping the target from upright posture; right: dropping the object to the side.

As we have mentioned in Section 5.3.2, a complete or resolution complete DRM requires a tremendous number of vertices and edges, yet storing all the information for these is infeasible on commodity computers with current technology. In this chapter, we propose a new algorithm, the *Hierarchical Dynamic Roadmap* (HDRM), which is resolution complete. A novel hierarchical structure that utilizes the kinematic hierarchy and symmetry is introduced to store DRMs with tens of millions of vertices and billions of edges in a memory efficient manner. Extensive benchmarking has been carried out that proves the HDRM is indeed resolution complete and is able to find valid solutions under extremely constrained conditions within a few milliseconds or less. We further demonstrate on a 7-DoF KUKA LWR robot showing that the HDRM can generate valid motion plans for solving real-world problems in extremely constrained environments, as highlighted in Figure 6.1.

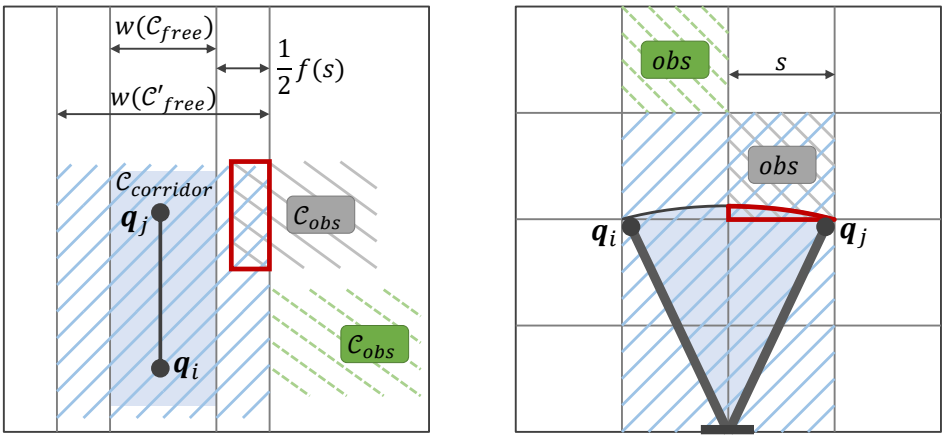
## 6.1 RESOLUTION COMPLETENESS OF A DETERMINISTIC ROADMAP WITH A DISCRETIZED WORKSPACE

As we have stated in Section 5.1, completeness is a key property in robot motion planning. In this section, we provide theoretical proof of the conditions and boundaries of resolution completeness for deterministic DRM methods with a discretized workspace, e.g. the HDRM.

The work in LaValle et al. [2004] has proven that a deterministic roadmap is resolution complete. Here, *deterministic* refers to the property of the sampling distribution. The authors proved that uniform sampling (e.g. a Sukharev grid) results in a resolution complete planning algorithm. To show this, let $\Psi$ be the subset of the power set of $\mathcal{C}$ corresponding to all open subsets that can be constructed with algebraic constraints as defined in Latombe [1991], and $\Psi(x)$ for $x \in (0, \infty)$ be the set of all $\mathcal{C}_{free}$ with the *width* of $\mathcal{C}_{free}$, $w(\mathcal{C}_{free}) \geq x$ (see LaValle et al. [2004]). The width $x$ can be viewed as the minimum width of a passable corridor in the collision free portion of the configuration space. Fig. 6.2a illustrates the corridor (with solid color background). All queries lie within this corridor, therefore, if all the queries of the deterministic roadmap also lie within the same corridor, the roadmap is hence resolution complete. The minimal with of the corridor $x$ required for completeness is defined in LaValle et al. [2004]. We extend their proof to roadmaps with discretized work space.

**Lemma 1.** *After $M$ iterations, a deterministic DRM is resolution complete for all $\mathcal{C}_{free} \in \Psi(4b(N)M^{-\frac{1}{N}} + f(s))$, where $M$ is the number of samples, $N$ is the dimension of the configuration space, $s$ is the resolution of the workspace, $b(N)$ is a factor that depends on the sampling method ($b(N) = 1$ for HDRM) and $f(s)$ is a robot-dependent function.*

***Proof**:* It has been proven in LaValle et al. [2004] that, after $M$ iterations, a deterministic roadmap planner is resolution complete for all $\mathcal{C}_{free} \in \Psi(4b(N)M^{-\frac{1}{N}})$, without workspace discretization. However, as shown in Fig. 6.2, with a discretized workspace with voxel size $s > 0$, the corresponding $\mathcal{C}_{corridor}$ and $\mathcal{C}_{obs}$ are both inflated due to the workspace discretization, where $\mathcal{C}_{corridor}$ is the narrowest corridor in the configuration space. Let $\mathcal{C}'_{corridor}$ and $\mathcal{C}'_{obs}$ denote the inflated $\mathcal{C}_{corridor}$ and $\mathcal{C}_{obs}$, respectively, and they must not intersect. After discretizing the workspace, the algorithm is




(a) Configuration Space (b) Workspace

Figure 6.2: Illustration of additional volume an obstacle in a discretized workspace occupies in the configuration space. An algorithm is resolution complete if it accounts for the additional increase corridor width $f(s)$ due to discretization.

able to solve problems for $\mathcal{C}'_{free}$ where $w(\mathcal{C}'_{free}) \geq w(\mathcal{C}_{free}) + f(s)$. Thus, the algorithm is resolution complete for all $\mathcal{C}_{free} \in \Psi(4b(N)M^{-\frac{1}{N}} + f(s))$. ■

To calculate $f(s)$, let $V(e)$ denote the voxelized swept volume of an edge $e$, and $\mathcal{C}_{V(e)}$ be the $\mathcal{C}$ space region occupied by $V(e)$, then the *width* of $\mathcal{C}'_{free}$ can be defined as

$$w(\mathcal{C}'_{free}) = 4b(N)M^{-\frac{1}{N}} + f(s) = \sup_{e \in \mathcal{E}}\{w(\mathcal{C}_{V(e)})\}, \tag{6.1}$$

which yields

$$f(s) = \sup_{e \in \mathcal{E}}\{w(\mathcal{C}_{V(e)})\} - 4b(N)M^{-\frac{1}{N}}. \tag{6.2}$$

It is practically difficult to pre-determine $f(s)$ before sampling as it depends not only on the number of samples $M$ and resolution $s$, but also on the robot's geometric shape.

## 6.2 HIERARCHICAL CONFIGURATIONS

While storing the swept volumes for hundreds of millions of edges is the most memory consuming part of DRM, storing vertices is also very expensive. This problem becomes evident only when we consider large, e.g. tens of millions, of vertices. For instance, a vertex in the six-dimensional roadmap contains six `float` values corre-

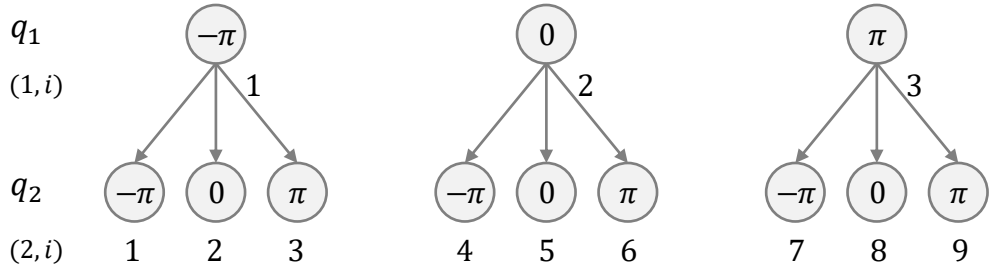



Figure 6.3: HDRM 2-DoF example with $K_1 = K_2 = 3$, the number of roadmap vertices stored in the structure is $K_1 \times K_2 = 9$.

sponding to a configuration of a 6-DoF robot. This means 10 million vertices in total contain 60 million `float` values, which require $60 \times 10^6 \times 4$ Byte $= 229$ MB of memory. Though the robot configurations are definitely required, we argue that these millions of configurations can be managed much more efficiently rather than storing them explicitly, as described next.

Let $[b_{n,l}, b_{n,u}]$ be the lower and upper bounds of joint $n \in N$ of a $N$-DoF robot. An even discretization of the $n$-th joint to $K_n \in \mathbb{N}$ values results in configurations

$$q_n(k_n) = b_{n,l} + (k_n - 1) \times \frac{b_{n,u} - b_{n,l}}{K_n - 1} \tag{6.3}$$

where $k_n \in K_n$. Let

$$\mathbf{k}(n) = [k_1, \ldots, k_n] \tag{6.4}$$

be a $n$-dimensional vector containing the joint value indices for the first $n$ joints, and

$$\mathbf{q}(\mathbf{k}(n)) = [q_1(k_1), \ldots, q_n(k_n)] \tag{6.5}$$

be a $n$-dimensional vector contains the actual joint values corresponding to $\mathbf{k}(n)$. The full $N$-dimensional robot configuration can be retrieved given $\mathbf{k}(N)$ for all joints. The hierarchical configurations can be arranged in a tree-like structure. For example, as shown in Figure 6.3, consider a 2-DoF robot, where the range of motion of each joint is $[-\pi, \pi]$. Given $K_1 = K_2 = 3$, we have $q_1(1) = q_2(1) = -\pi$, $q_1(2) = q_2(2) = 0$, and $q_1(3) = q_2(3) = \pi$. Then, $\mathbf{k}(2) = [1, 1]$ gives the robot configuration $\mathbf{q} = [-\pi, -\pi]$, $\mathbf{k}(2) = [2, 3]$ gives another robot configuration $\mathbf{q} = [0, \pi]$ and $\mathbf{k}(1) = [2]$ gives the first joint value $q_1 = 0$.

**Algorithm 7** Generate hierarchical indices from integer index

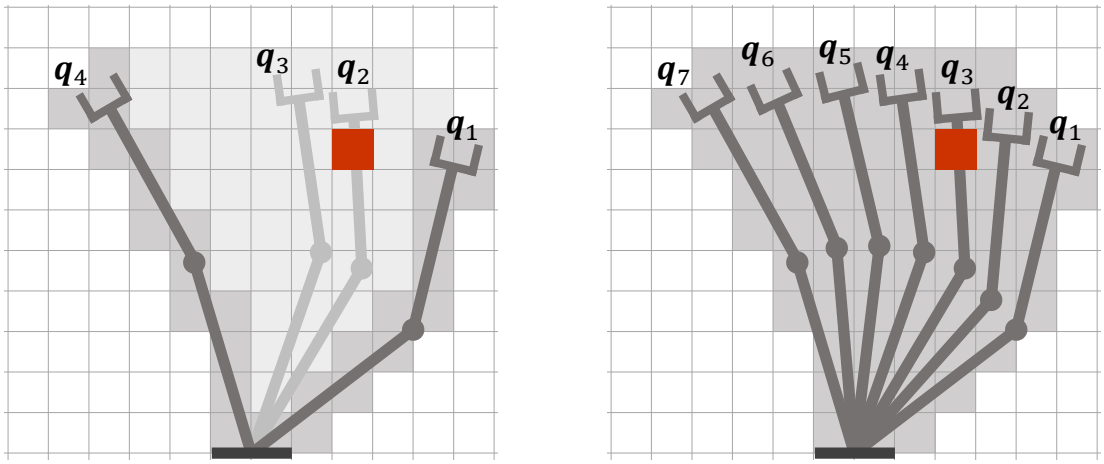
**Require:** Dimension level $n$, vertex index $i$
**Ensure:** Hierarchical indices $\mathbf{k}(n)$
1: Quotient$= i$
2: **while** $n > 1$ **do**
3: Quotient, Remainder=Division(Quotient,$\prod_1^n K_n$)
4: $k_n$ =Remainder
5: $k_1$ =Quotient
**return** $\mathbf{k}(n) = [k_1, \ldots, k_n]$

**Algorithm 8** Generate integer index from hierarchical indices

**Require:** Hierarchical indices $\mathbf{k}(n) = [k_1, \ldots, k_n]$
**Ensure:** Dimension level $n$, vertex index $i$
1: $i = 0$
2: **for** $l \in \{1, \ldots, n-1\}$ **do**
3: counter $= 1$
4: **for** $j \in \{l+1, \ldots, n-1\}$ **do**
5: counter $=$ counter $\times K_j$
6: $i = i +$ counter $\times k_l$
7: $i = i + k_n$
Return $n, i$

There exist $M = \prod_1^N K_n$ different combinations, which is the number of configurations (vertices) that can be described by this structure. So we only need to store $\sum_1^N K_n$ rather than $N \times \prod_1^N K_n$ `float` values for all the configurations. For instance, assuming $N = 6$ and $K_n = 15, n \in N$, we can store $M \approx 11.4$ million configurations with $6 \times 15 \times 4 = 360$ Byte instead of 229 MB of memory as in the previous example.

The robot configurations can be accessed with $\mathbf{k}(n)$, however, the vertices in the roadmap are indexed with one integer index $i \in M$. Let $\mathcal{H} : (n, i) \mapsto \mathbf{k}(n)$ be the map from pair $(n, i)$ to $\mathbf{k}(n)$, and $\mathcal{H}^{-1} : \mathbf{k}(n) \mapsto (n, i)$ be the corresponding inverse map. Given an index $i$ and level $n$, the first $n$ indices $\mathbf{k}(n) = \mathcal{H}(n, i)$ can be efficiently calculated using Algorithm 7. Similarly, given hierarchical indices $\mathbf{k}(n)$, the corresponding $(n, i)$ can also be found by Algorithm 8. The hierarchical configuration structure dramatically reduces the memory consumption for storing the vertices in the roadmap. As we will show in 6.4, another advantage of this structure is that the occupation lists can also be stored hierarchically.




(a) Classical DRM (b) Hierarchical DRM

Figure 6.4: (a) A long edge $\mathcal{E}(\mathbf{q}_1, \mathbf{q}_4)$ in classical DRM sweeps through a large number of workspace voxels. (b) Dense vertices and short edges in HDRM.

## 6.3 REMOVING SWEPT VOLUMES

There are two types of occupation information: the occupation voxels of a vertex (dark grey voxels in Figure 6.4) and the swept volume of an edge (light grey voxels in Figure 6.4). In the classical DRM algorithm, the edge is invalidated if one or more of the voxels in the swept volume are in collision. However, there will be many sub-edges still valid in the cases where only very few of the voxels are in collision. For example, in Figure 6.4a, if only the red voxel is in collision, the long edge $\mathcal{E}(\mathbf{q}_1, \mathbf{q}_4)$ is invalid while the sub-edge $\mathcal{E}(\mathbf{q}_3, \mathbf{q}_4)$ is still valid. Yet, the whole edge is considered invalid as these sub-edges are not stored in the roadmap. This is the underlying reason for planning failures and the low success rate of the classical DRM method. In fact, we think that storing the swept volumes does not utilize the resources because this information is used only for collision checking but not for the actual path planning. If possible, we should discard the swept volume information and use the memory for storing more critical information, such as more vertices and edges.

In our method, we store only the occupation voxels of the vertices but not the swept volumes of the edges, while still being able to check the collision status of both vertices and edges. Let $O_a$ be the occupation voxels of a vertex $a$, and $O_{a,b}$ be the

swept volume of edge $\mathcal{E}(a,b)$. If the two vertices $a,b$ are very close and the edge is so short that

$$O_{a,b} = O_a \cup O_b, \tag{6.6}$$

then we do not need to store the swept volume of the edge since it can be represented by the occupation voxels of the two end-point vertices, as illustrated in Figure 6.4b. The edge $\mathcal{E}(a,b)$ is collision-free if vertices $a,b$ are collision free, and vice versa. This ensures that a colliding workspace voxel only affects those corresponding short edges without invalidating other ones. A lower bound of $K_n$ needs to be met in order to achieve such roadmap density.

Let $\theta_n = b_{n,u} - b_{n,l}$ be the range of motion of joint $n$, $s$ be the workspace voxel size, and $r$ be the approximate radius of the robot's tip link, as illustrated in Figure 6.5. For joint $n$, set all the subsequent/child joints to zero positions, so that the rest of the robot kinematic chain is fully extended to a maximum length $R_n$. In order to satisfy (6.6), the distance between the end-effectors of two neighboring configurations must not be greater than $s + \sqrt{2}r$, so that the two end-effector links occupy same or neighboring workspace voxels. So, the following inequality constraint must be met,

$$\frac{s + \sqrt{2}r}{2\pi R_n} \geq \frac{\Delta_n}{2\pi}, \tag{6.7}$$

rearranging terms yields

$$\Delta_n \leq \frac{s + \sqrt{2}r}{R_n}. \tag{6.8}$$

This means that joint $n$ should have at least

$$K_n \geq \frac{\theta_n}{\Delta_n} + 1 = \frac{\theta_n R_n}{s + \sqrt{2}r} + 1 \tag{6.9}$$

evenly distributed values within the range of motion. We choose the minimum valid value for each $K_n$ as the minimum value already guarantees *resolution completeness* for certain workspace voxel resolution $s$. Greater $K_n$ only introduces more vertices and edges that slows down the searching process.

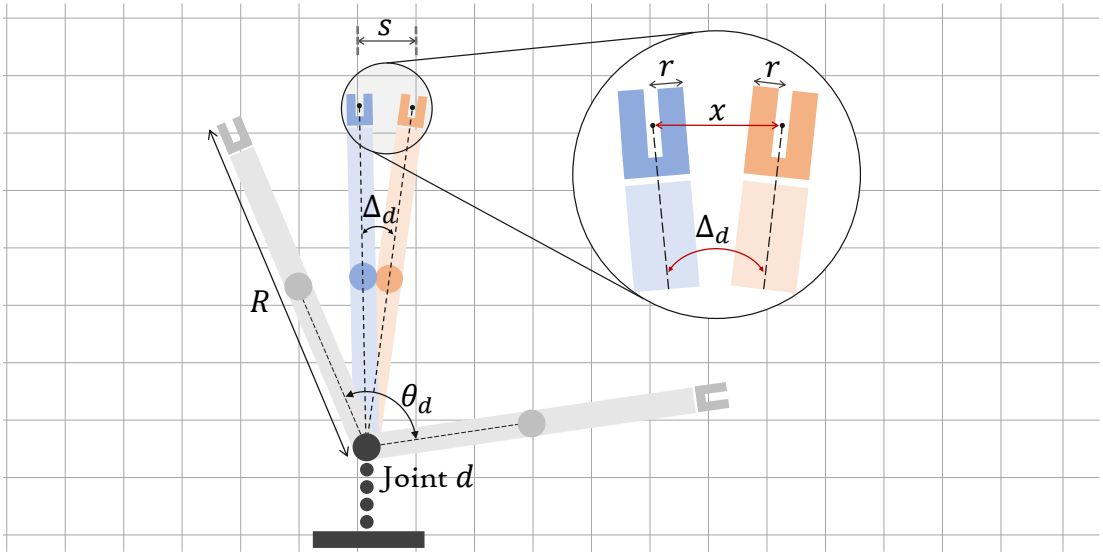



Figure 6.5: Illustration of the maximum discretization step $\Delta_n$ the $n$-th joint can take without violating (6.6).

The swept volumes can be removed if (6.9) is true for all joints. This will significantly reduce the required memory for storing the DRM. Furthermore, the information of the hundreds of millions of edges itself can be removed as well, because all the edges can be calculated analytically from the hierarchical structure. A $N$ dimensional configuration $\mathbf{k}(N)$ has $2 \times N$ neighbors (apart from the ones on the border of the range of motion), each of which forms an edge with the vertex $\mathbf{k}(N)$. Since the edges have no direction, a $N$ dimensional HDRM with $M$ vertices contains roughly $N \times M$ edges.

## 6.4 HIERARCHICAL OCCUPATION LISTS

We described how to create a hierarchical structure to efficiently store tens of millions of configurations (6.2), and explained why and how to remove the swept volumes as well as the edges (6.3). The final step involves processing and storing the occupation lists of all the vertices. When the roadmap contains tens or potentially hundreds of millions of vertices, their occupation lists are too expensive to store using classical methods. Next, we discuss how to take the advantage of the hierarchical structure to alleviate this problem.

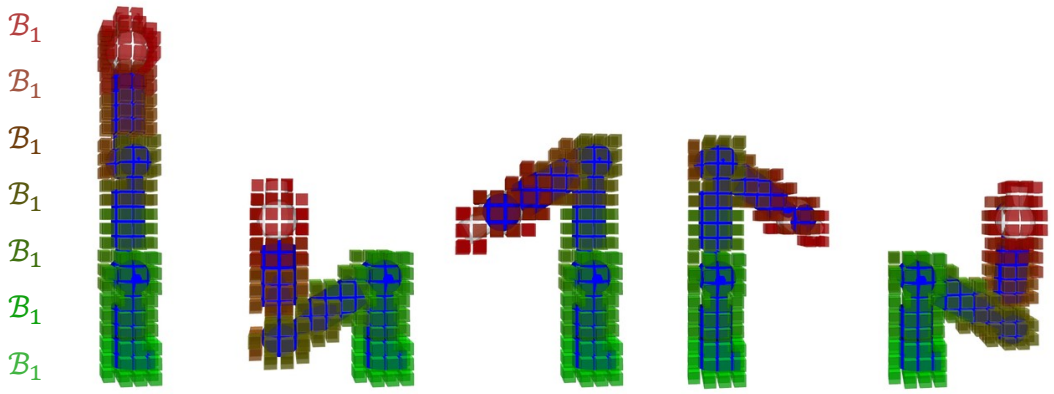



Figure 6.6: Illustration of different collision bodies of the 7-DoF LWR arm. The greener voxels represent root bodies and more red voxels represent tip bodies

Let $\mathcal{B}_n$ be a collision body between joint $n$ and $n+1$. An example of different collision bodies of the 7-DoF LWR arm is illustrated in Figure 6.6 where the greener voxels represent root bodies and more red voxels represent tip bodies. Consider $K_N$ configurations with identical values for the first $N-1$ joints but only differing at the last joint, as illustrated in Figure 6.7. These $K_N$ configurations are invalid if $\mathcal{B}_{N-1}$ is in collision at the red voxel. In the classical DRM method, the red voxel's occupation list needs to store $K_N$ indices to encode this information, where each index corresponds to a particular configuration, which is very inefficient.

Instead of storing integer indices $i \in M$ for each configuration, we store a list of pairs $(n, i)$, where $i \in \prod_1^n K_n$ and $n \in N$. A pair $(n, i)$ is added to a workspace voxel $v$'s occupation list if $\mathcal{B}_n$ of configuration $\mathbf{k}(n) = \mathcal{H}(n, i)$ occupies this voxel. In Figure 6.7, when the red voxel is in collision with the environment, based on the pair $(N-1, i)$, we can invalidate the $i$-th vertex of level $N-1$ of the hierarchical structure. It is clear that that all these $K_N$ configurations are invalid since the first $N-1$ joints already caused body $\mathcal{B}_{N-1}$ to be in collision. This means we can encode the occupation information of $K_N$ configurations using only two, rather than $K_N$ indices. Consider another case with $K_2 \cdots \times K_N$ vertices, which could be millions, that have same value $k_1$ for the first joint but differ at all other joints. If $k_1$ puts $\mathcal{B}_1$ to a colliding position with the environment, then the millions of vertices with same $k_1$ are all invalid. In such case, we can more efficiently use only a pair $(1, k_1)$ instead of millions of indices to encode the occupation information of all these vertices. As we

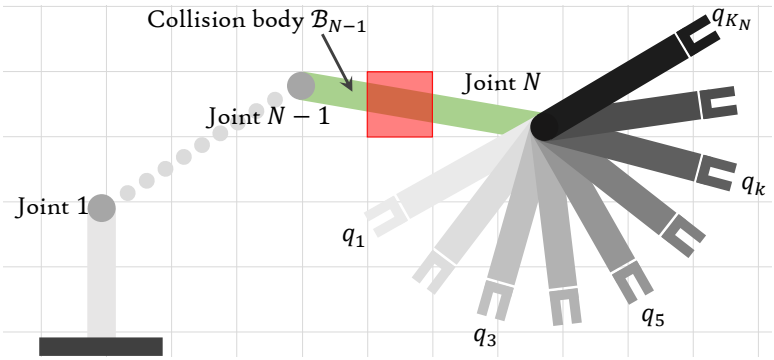



Figure 6.7: Illustration of hierarchical occupation lists.

will show later, using the hierarchical structure and this novel indexing technique, we can dramatically reduce the memory required for storing the occupation information.

Algorithm 9 shows the details of generating the full occupation lists for all workspace voxels. First, given the size of the workspace and grid resolution $s$, a set of workspace voxels $\mathbb{V}$ can be generated. Each voxel $v \in \mathbb{V}$ is associated with an empty occupation list $\mathcal{O}_v$. Lines 3-9 generate the initial hierarchical occupation lists, but we can compress the lists to further reduce memory storage (line 10-20). The compression idea is based on the fact that some robots, or part of the robots, are rotational symmetric, which means that rotating the last joint does not change the occupation list of the last link at all. More generally, if the collision body $\mathcal{B}_n$ of $K_n$ vertices ($xK_n + 1$ to $xK_n + K_n$, $x \geq 0$) from the same sub-tree of level $n$ occupies a voxel $v$, then the occupation list of $v$ needs to store only one pair of $(n-1, \cdot)$ rather than $K_n$ pairs of $(n, \cdot)$, because the first $n-1$ joints already make $\mathcal{B}_n$ unavoidably occupy voxel $v$. We *"promote"* the occupation list from level $n$ to $n-1$ if such rotational symmetry occurs.

## 6.5 MOTION PLANNING USING HDRM

With the HDRM created and loaded, our goal is to efficiently solve motion planning queries online with different start and goal states in changing environments. There are three main steps as follows.

**Algorithm 9** Generate hierarchical occupation lists

**Require:** Robot model $\mathcal{R}$, voxelized workspace $\mathbb{V}$
**Ensure:** Hierarchical occupation lists $\mathcal{O}_v, v \in \mathbb{V}$

**for** $v \in \mathbb{V}$ **do**
    Occupation list $\mathcal{O}_v = \emptyset$
**for** $n \in N$ **do**
    **for** $i \in \prod_1^n K_n$ **do**
        $\mathbf{k}(n) = \mathcal{H}(n, i)$
        Set first $n$ joints of $\mathcal{R}$ to $\mathbf{q}(\mathbf{k}(n))$
        $V = \text{findBodyOccupiedVoxels}(\mathbb{V}, \mathcal{R}, \mathcal{B}_n)$
        **for** $v \in V$ **do**
            $\mathcal{O}_v = \mathcal{O}_v \cup \{(n, i)\}$
**for** $v \in \mathbb{V}$ **do**
    **for** $n = N$ to 1 **do**
        $\mathbf{O} = \text{extractListOfDimension}(\mathcal{O}_v, n)$
        Remove duplicated indices and sort $\mathbf{O}$
        **for** $O_i \in \mathbf{O}$ **do**
            **if** $O_i \bmod K_n = 0$ & $O_{i+K_n} = O_i + K_n$ **then**
                $\mathcal{O}_v = \mathcal{O}_v \backslash \{(n, p) | p \in [O_i, \ldots, O_{i+K_n}]\}$
                **if** $n > 1$ **then**
                    $\mathbf{k}(n) = [k_1, \ldots, k_n] = \mathcal{H}(n, O_i)$
                    $\mathcal{O}_v = \mathcal{O}_v \cup \{(n-1, \prod_{p=1}^{p=n-1} k_p)\}$
                $i = i + K_n - 1$
Return $\mathcal{O}_v, v \in \mathbb{V}$

### 6.5.1 *Collision update*

First, we create a voxelized environment to represent the discretized workspace. Then given the current environment, we apply conventional collision checking on the two environments to find the list of voxels that are occupied by the obstacles. For each occupied voxel, we iterate though its occupation lists and invalidate vertices in the hierarchical structure accordingly.

### 6.5.2 *Connecting start/goal to roadmap*

The start and goal vertices $\mathcal{V}_{start}, \mathcal{V}_{\text{goal}}$ are required for graph search algorithm, which are the closest valid vertices to the start and goal configurations $\mathbf{q}_{\text{start}}, \mathbf{q}_{\text{goal}}$ . Traditionally, this involves comparing the distance between a given configuration $\mathbf{q}$ and all ver-

tices in the roadmap and finding the one with shortest distance. Such process could be very slow for a roadmap with a large number of vertices. In our approach, instead of searching though all vertices, we can analytically compute the closest one. Given a configuration $\mathbf{q}$, we can easily get the closest hierarchical configuration $\mathbf{k}_{\text{closest}}(N)$. Then, the corresponding closest vertex $i_{\text{closest}}$ can be found using Algorithm 8.

### 6.5.3 *Shortest path searching*

The last step is to find a valid path connecting $\mathcal{V}_{\text{start}}$ and $\mathcal{V}_{\text{goal}}$. The A* shortest path searching algorithm is used. We implemented the sequential version of A* using a single thread on the CPU. Parallelization is not the main focus of this thesis, however, we acknowledge that parallel version of Dijkstra or A* algorithms would make the search even more efficient Rios and Chaimowicz [2011], Schumann-Olsen et al. [2014].

## 6.6 EXPERIMENTS

The proposed HDRM method is benchmarked against classical DRM approach and standard sampling-based planners (SBP) in various scenarios with two different robot models – a 6-DoF Universal Robot UR5 and a 7-DoF KUKA LWR robotic arm. The evaluation was carried out on an Intel Core i7 – 6700K 4.0GHz CPU with 32GB 2133MHz RAM with the hardware experiments performed on the LWR.

### 6.6.1 *HDRM Construction*

In order to create the HDRM, we need to first define $K_n$ for each joint. Given the robot model, $K_n$ can be calculated by (6.7 - 6.9), as listed in Table 6.1. Two different workspace voxel resolutions are used, $s = 0.1m$ and $s = 0.05m$. Smaller $s$ leads to greater $K_n$, which means more samples are required to densely cover the space. Tail joints have smaller $K_n$, for instance the last joint of UR5 and LWR has $K_n = 1$. This is because the last joints have very short or zero extend length $R_n$ and the last body link is rotational symmetric, where very few of vertices are required to satisfy (6.9).

Table 6.1: Robot kinematic analysis for creating HDRM.

| Robot | Range of motion $\theta$ (rad), extend length $R$ (m) | $s$ (m) | $K_n$ |
|---|---|---|---|
| UR5 | $\theta_n = \{6.28, 6.28, 6.28, 6.28, 6.28, 6.28\}$ | 0.1 | $\{37, 36, 21, 9, 7, 1\}$ |
| | $R_n = \{0.98.0.97, 0.57, 0.23, 0.17, 0.0\}$ | 0.05 | $\{52, 51, 30, 12, 9, 1\}$ |
| LWR | $\theta_n = \{5.86, 4.12, 5.86, 4.12, 5.86, 4.12, 5.86\}$ | 0.1 | $\{35, 20, 21, 10, 7, 2, 1\}$ |
| | $R_n = \{0.99, 0.79, 0.59, 0.39, 0.19, 0.05, 0\}$ | 0.05 | $\{49, 27, 29, 14, 10, 2, 1\}$ |

We have also implemented classical DRM methods for comparison. For achieving completeness, we generate the vertices by uniformly sampling in the configuration space and apply no roadmap compression technique. Three classical DRM datasets are created with different number of vertices: $1,000$ ($\text{DRM}_a$), $10,000$ ($\text{DRM}_b$) and $200,000$ ($\text{DRM}_c$). A K-nearest neighbor search is then applied to find the edges in the roadmap.

### 6.6.2 *Memory Consumption*

As highlighted in Table 6.2, the HDRM scales exponentially with roadmap size compared to classical DRM methods. Meanwhile, the required memory size is much less by using the hierarchical structure. In the case of UR5 robot with 10cm voxel size, HDRM can store over 1.7 million vertices and 10 million edges using only 8.5MB of memory, which is even less than the memory required for classical DRM to store only $10,000$ vertices. In the scenario of LWR robot with 5cm voxel size, the HDRM stores over 10 million vertices and up to 75 million edges with only 266MB of memory, where the estimated memory size for classical DRM to store the same number of vertices is over 250GB.

### 6.6.3 *Motion Planning Evaluation*

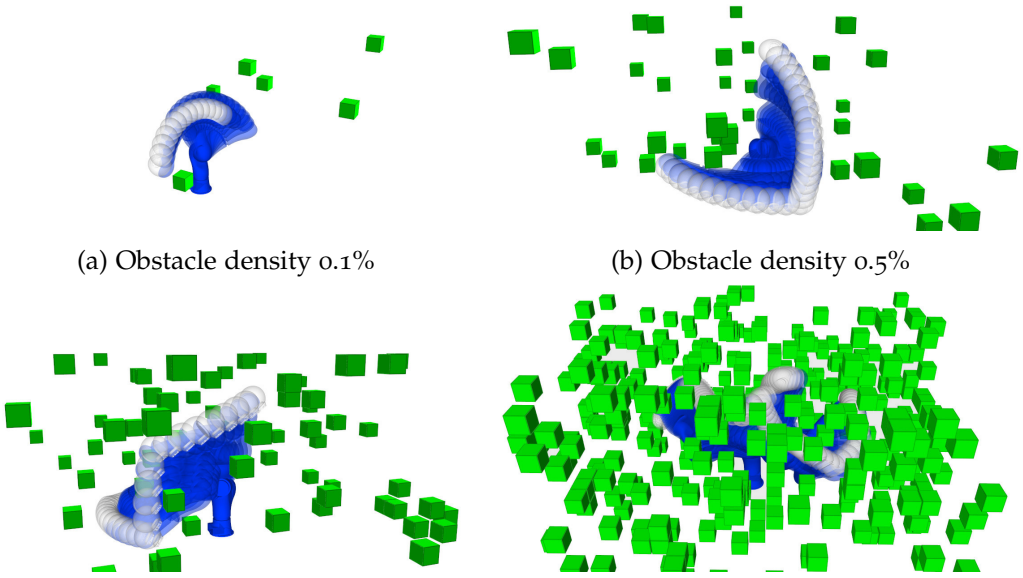
Extensive benchmarking fully analyzes the performances of eight candidate methods, which include three classical DRM methods $\text{DRM}_a$, $\text{DRM}_b$, $\text{DRM}_c$; four standard sampling-based planners (SBP), i.e. RRT, PRM, SBL Sánchez and Latombe [2003],

Table 6.2: Comparison of roadmap and memory size between classical DRM and HDRM.

| Robot | Method | No. Vertices | No. Edges | $s$ (m) | Memory size (MB) |
|---|---|---|---|---|---|
| UR5 | DRM | 1,000 | 6,336 | 0.1 | 2.8 |
| | | | | 0.05 | 13.6 |
| | | 10,000 | 61,274 | 0.1 | 22.3 |
| | | | | 0.05 | 104 |
| | | 200,000 | 1,200,956 | 0.1 | 356 |
| | | | | 0.05 | 1593 |
| | DRM Estimate | 1,762,236 | ∼10,573,416 | 0.1 | ∼3136 |
| | | 8,592,480 | ∼51,554,880 | 0.05 | ∼68439 |
| | **HDRM** | 1,762,236 | 10,573,416 | 0.1 | 8.5 |
| | | 8,592,480 | 51,554,880 | 0.05 | 145 |
| LWR | DRM | 1,000 | 6,369 | 0.1 | 7.9 |
| | | | | 0.05 | 33.4 |
| | | 10,000 | 62,031 | 0.1 | 70 |
| | | | | 0.05 | 280 |
| | | 200,000 | 1,216,755 | 0.1 | 1239 |
| | | | | 0.05 | 4793 |
| | DRM Estimate | 2,058,000 | ∼12,348,000 | 0.1 | ∼12390 |
| | | 10,742,760 | ∼64,456,560 | 0.05 | ∼256425 |
| | **HDRM** | 2,058,000 | 14,406,000 | 0.1 | 16.7 |
| | | 10,742,760 | 75,199,320 | 0.05 | 266 |

and RRTConnect; and finally the proposed HDRM. For DRM/HDRM methods, the datasets of LWR robot with 10cm voxel resolution are used. For SBP methods, we use the standard implementations from OMPL library Şucan et al. [2012] and the FCL library Pan et al. [2012] for explicit online collision checking.

To thoroughly evaluate the performance, we created five different categories of arbitrary environments with random obstacles. From simple to hard, these environments have 0%, 0.1%, 0.5%, 1% and 5% of the whole workspace occupied by obstacles, where the latter four are illustrated in Figure 6.8. The environments with 1% and 5% obstacle densities are extremely complicated for any kind of motion planning algorithm. We created 1000 random problems for each category, i.e. 1000 random environments with valid start and goal states. Note that the problem needs to be solvable, i.e. there should exist at least one valid trajectory from start to goal. To guarantee this, a random self-collision-free trajectory is first generated in free space. Obstacles



(a) Obstacle density 0.1% (b) Obstacle density 0.5%

(c) Obstacle density 1% (d) Obstacle density 5%

Figure 6.8: Random problems in environments with different workspace obstacle densities. The highlighted trajectories are valid solutions found by HDRM.

are populated into the space randomly without colliding with the trajectory. We keep populating obstacles until the required obstacle density is reached. All algorithms are given 10 seconds to solve each problem. Since all problems are solvable, the solving is only considered a success if a valid solution (which could be different from the original one) is found. Reporting no solution or exceeding the time limit will be considered as failed.

The evaluation result is highlighted in Table 6.3. Firstly, as a baseline, all algorithms achieved 100% success rate in free space. The success rate of the classical DRMs falls below 100% when the environment is populated with only a few obstacles (0.1% obstacle density). The SBP methods are in general slower because these approaches explicitly check collision for every sample, which is very time consuming. In more complicated environments (0.5% and 1% obstacle densities), the success rate of classical DRM methods dropped dramatically. SBP methods still achieved reasonable success rate, but the planning time increased considerably. The HDRM method began to surpass all other methods in complicated scenarios in terms of both success rate and planning time. In the extreme cases with 5% obstacle density, we do not

Table 6.3: Evaluation results of solving motion planning problems using different approaches. The result shows the success rate of solving 1000 problems with random environments and valid start/goal states, followed by the average solving time over the success cases (in milliseconds). All algorithms are given 10 seconds to solve each problem.

| Method | | Obstacle 0% | Obstacle 0.1% | Obstacle 0.5% | Obstacle 1% | Obstacle 5% |
|---|---|---|---|---|---|---|
| Standard SBP | RRT | 100% | 99% | 92% | 82% | 36% |
| | | 13.392 | 22.708 | 325.21 | 1036.1 | 1893.6 |
| | PRM | 100% | 100% | 100% | 99% | 34% |
| | | 5.7416 | 4.5041 | 322.86 | 656.09 | 3386.4 |
| | SBL | 100% | 100% | 100% | 100% | 31% |
| | | 6.8909 | 14.775 | 82.473 | 273.96 | 4439.4 |
| | RRTConnect | 100% | 100% | 100% | 100% | 74% |
| | | 1.1930 | 2.1220 | 10.117 | 48.926 | 1723.7 |
| Classical DRM | $DRM_a$ | 100% | 92.2% | 65.6% | 39.0% | 1.6% |
| | | 0.1564 | 0.7108 | 0.7911 | 1.2403 | - |
| | $DRM_b$ | 100% | 93.9% | 69.5% | 48.9% | 3.6% |
| | | 0.3123 | 0.8779 | 1.0203 | 1.5221 | - |
| | $DRM_c$ | 100% | 95.7% | 74.7% | 53.0% | 3.3% |
| | | 3.4152 | 4.3431 | 6.5206 | 9.2316 | - |
| **Hierarchical DRM** | | 100% | 100% | 100% | 100% | 100% |
| | | 0.2759 | 0.8574 | 1.5813 | 3.7152 | 15.506 |

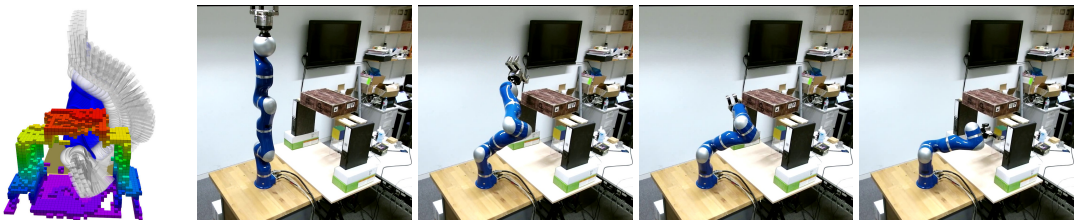
show the average planning time for the classical DRM methods as the success rate is too low. All SBP methods also reported lower success rate and much longer planning time. On the contrary, the HDRM method constantly achieved 100% success rate in these most complicated cases. The planning time is much longer than that in simpler scenarios, but still reasonably fast in these extremely constrained environments. All these results prove that HDRM is indeed resolution complete.

It is interesting that $DRM_c$ has a much smaller roadmap than HDRM, but takes longer time to find a solution even in free space. We break down the DRM/HDRM planning time into different components, as highlighted in Table 6.4. Note that the time is given in microseconds for better comparison. The collision update takes 141.6 microseconds in free space, which is basically the communication and function calls overhead. We use FCL for explicit collision checking where the time increases as

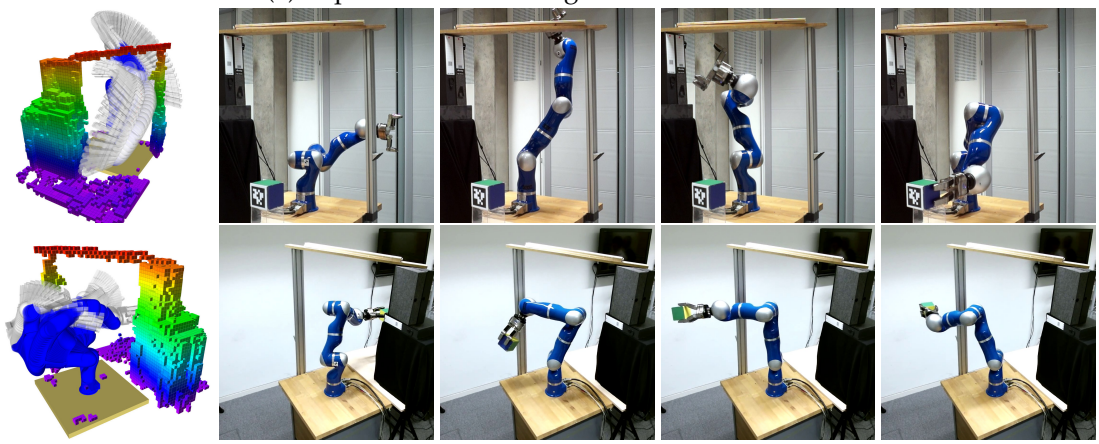
Table 6.4: Computation time of different components in DRM and HDRM (in **microseconds**).

| Method | Roadmap update | | Planning | | Total |
|---|---|---|---|---|---|
| | Find coll. voxels | Remove invalids | Connect to roadmap | A* search | |
| Obstacle density 0% | | | | | |
| $DRM_a$ | 141.6 | 0 | 14.17 | 0.581 | 156.4 |
| $DRM_b$ | | | 170.1 | 0.626 | 312.3 |
| $DRM_c$ | | | 3273 | 0.698 | 3415 |
| **HDRM** | | | 0.229 | 134.1 | 275.9 |
| Obstacle density 0.1% | | | | | |
| $DRM_a$ | 694.8 | 1.665 | 13.774 | 0.557 | 710.8 |
| $DRM_b$ | | 18.71 | 163.8 | 0.601 | 877.9 |
| $DRM_c$ | | 435.0 | 3212 | 0.711 | 4343 |
| **HDRM** | | 17.48 | 0.267 | 144.9 | 857.4 |
| Obstacle density 0.5% | | | | | |
| $DRM_a$ | 769.5 | 7.541 | 13.48 | 0.616 | 791.1 |
| $DRM_b$ | | 86.19 | 163.4 | 0.632 | 1020 |
| $DRM_c$ | | 2450 | 3300 | 0.758 | 6520 |
| **HDRM** | | 94.33 | 0.257 | 717.4 | 1581 |
| Obstacle density 1% | | | | | |
| $DRM_a$ | 1212 | 13.92 | 13.32 | 0.607 | 1240 |
| $DRM_b$ | | 145.5 | 163.5 | 0.745 | 1522 |
| $DRM_c$ | | 4728 | 3290 | 1.021 | 9231 |
| **HDRM** | | 177.9 | 0.233 | 2325 | 3715 |

expected in more complicated environments. Classical DRMs with more vertices and edges require much longer time to remove invalid roadmap parts, whereas the HDRM is able to do so relatively faster, considering the enormous number of vertices and edges. Another expensive step of classical DRMs is connecting to the roadmap, which increases exponentially with the number of vertices. After connecting to the roadmap, running A* search is actually very fast since the roadmap size is relatively small. On the other hand, the time for connecting to the roadmap is negligible for HDRM since we can analytically compute the closest vertices (6.5.2). However, the searching takes longer due to the enormous roadmap size.



(a) Experiment 1: reaching into a confined shelf



(b) Experiment 2: fetching an object in distance. First row, reaching target object; second row, retrieving object.

Figure 6.9: Experiments on a 7-DoF KUKA LWR robotic arm fitted with SCHUNK Dexterous Hand 2.0. First column is the octomap representation and planned trajectory, followed by snapshots of motion execution.

### 6.6.4 *Experimental Validation on Robot Hardware*

We further validated the HDRM method on a 7-DoF KUKA LWR manipulator fitted with the SCHUNK Dexterous Hand. We consider one fixed end-effector during a task compared to the work in Liu et al. [2006], i.e. different HDRM datasets are required for different end-effectors. Our experiments used a model of the LWR with the SCHUNK hand to generate the HDRM dataset with $s = 5$ cm voxel resolution. Four Microsoft Kinect One RGB-D sensors are fused to sense the environment and create an octomap representation Hornung et al. [2013] for collision checking. In our supplementary video (`https://youtu.be/2G5uSTCk4UY`), we demonstrate challenging motions in three different, highly constrained environments that emulate real-world tasks: reaching into a confined shelf space and grasping a target object (Figure 6.9a); retrieving distant object through a frame (Figure 6.9b); and moving an object with the robot workspace severely confined by a cage (Figure 7.1).

## 6.7 CONCLUSION

In this chapter we presented a novel method, the Hierarchical Dynamic Roadmap (HDRM), for real-time motion planning in high dimensions. The HDRM, through a sophisticated indexing scheme, is able to encode large numbers of vertices and edges (up to tens of millions) in a memory efficient manner that allows the approach to be resolution complete. An extensive benchmarking has been carried out showing that the HDRM is able to find valid motion plans in extremely complicated environments in real-time. Hardware experiments on the KUKA LWR robot show that our method is capable of incorporating live sensing information and provides collision-free and smooth trajectories that can be executed robustly for solving practical problems. Since HDRM guarantees resolution completeness and is able to plan in real-time (few milliseconds or less), the future work involves implementing a closed-loop online planning/re-planning framework for applications such as real-time interaction between human and robot in shared workspace.

For the fixed-base robots considered in this chapter, the collision-free region $\mathcal{C}_{\text{free}}$ and valid region $\mathcal{C}_{\text{valid}}$ are equal, i.e. $\mathcal{C}_{\text{free}} = \mathcal{C}_{\text{valid}}$. However, such assumption does not hold for more complex humanoid robots, where not all of the collision-free configurations are valid due to balance constraint. The valid region is only a small subset of the collision-free region, i.e. $\mathcal{C}_{\text{free}} \subsetneq \mathcal{C}_{\text{valid}}$, which makes sampling valid configurations very challenging. In next chapter, we will discuss how to efficiently scaling sampling-based planning algorithms to bipedal humanoids.

# 7

# EFFICIENT MOTION PLANNING FOR HUMANOIDS

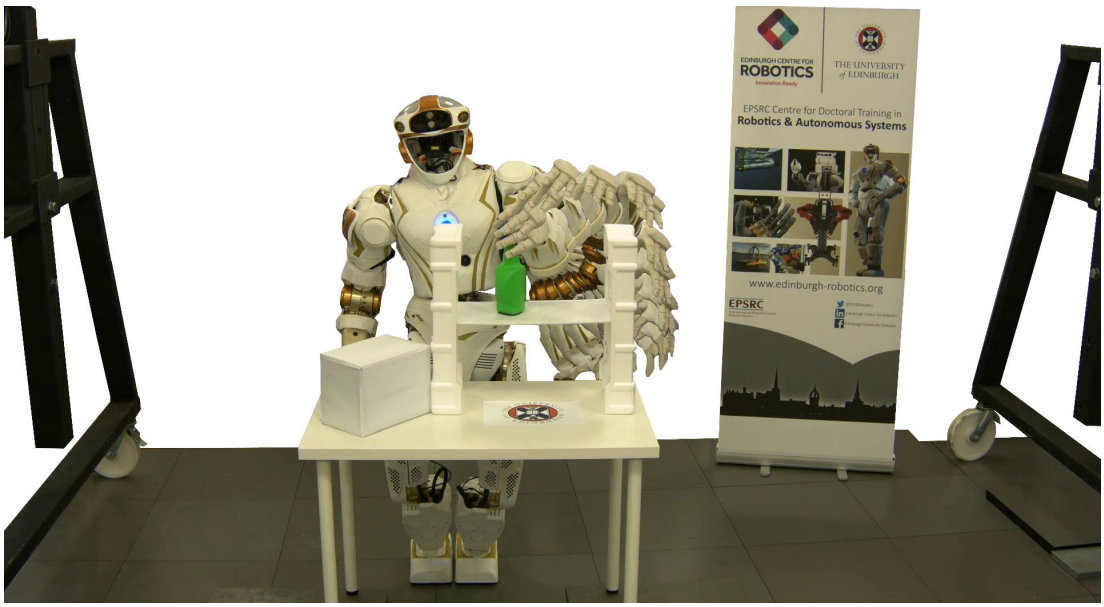



Figure 7.1: Collision-free and balanced full-body motion executed on the 38-DoF NASA Valkyrie robot. Lower body movement is not shown for clarity.

Humanoid robots are highly redundant systems that are designed for accomplishing a variety of tasks in environments designed for people. However, in contrast to the fixed-based robotic arm discussed in Chapter 6, humanoids have a large number of degrees-of-freedom which makes motion planning extremely challenging. In general, optimization-based algorithms are suitable for searching for optimal solutions even in high dimensional systems Rawlik et al. [2012] Ratliff et al. [2009], but it is non-trivial to generate optimal collision-free trajectories for humanoids using optimization approaches within timeframes acceptable for online planning, especially in complex environments. This is mainly due to the highly non-linear map between the robot and the collision environment. This mapping can be learned Howard et al. [2009] Nakanishi et al. [2013] Lin et al. [2015] or modelled in abstract spaces Yang et al. [2015] Ivan et al. [2013] for low dimensional problems, but is too difficult for high DoF humanoids due to the curse of dimensionality and it often causes local

minima problems. Additionally, solving locomotion and full-body manipulation in complex environments as one combined problem requires searching through a large space of possible actions. Instead, as illustrated in Figure 3.7, it is more effective to first generate robust walking plans to move the robot to a desired standing location, and then generate collision-free motion with stationary feet. Although assuming fixed feet position may be viewed as restrictive, we argue that a large variety of full-body manipulation tasks can still be executed as a series of locomotion and manipulation subtasks. Approaches for finding appropriate stances and goal states has been discussed in Section 3.3 and Chapter 4, in this chapter, we propose an extension to a family of sampling-based motion planning algorithms that will allow us to plan collision-free full-body motions for bipedal humanoids.

Sampling-based planning algorithms, such as RRT Lavalle [1998] and PRM Kavraki et al. [1996], are capable of efficiently generating globally valid collision-free trajectories due to their simplicity. In the past two decades, SBP algorithms have been applied to countless problems with a variety of derivatives, such as RRT-Connect Kuffner and LaValle [2000], Expansive Space Trees (EST) Hsu et al. [1997], RRT*/PRM* Karaman and Frazzoli [2011], Kinematic Planning by Interior-Exterior Cell Exploration (KPIECE) Şucan and Kavraki [2009], and many others Elbanhawi and Simic [2014]. However, since the SBP algorithms were originally designed for mobile robots and low DoF robotic arms, using them on high DoF systems requiring active balancing is still challenging. The subset of valid robot configurations forms a low dimensional manifold defined by the balance constraint. In practice, the rejection rate of random samples is prohibitively high without the explicit or implicit knowledge of the manifold.

Approaches have been proposed to address this particular problem of using SBP algorithms for humanoid robots. Kuffner et al. [2005] use a customized RRT-Connect algorithm to plan full body motion for humanoids, where they only sample from a pre-calculated pool of postures for which the robot is in balance. Hauser et al. [2008] introduce motion primitives into SBP algorithms where the sampler only samples states around a set of pre-stored motion primitives. These approaches share the common idea of using an off-line generated sample set to bootstrap online processes,

thus allowing algorithms to bypass the expensive online generation of balanced samples Kuffner et al. [2005] Hauser et al. [2008]. By storing sample configurations with different lower body postures, one can also generate full body motion that consists of coordinated locomotion and upper body movement Kuffner et al. [2002]. However, this leads to the problem where one has to store a significant number of samples to densely cover the balance manifold, otherwise the algorithms may fail while valid solutions exist but were not stored in the dataset. Another direction for solving humanoid motion planning problem is constraint sampling. Dalibard et al. [2009] replace the steering/interpolation component in RRT-Connect with a constraint connect function to ensure the new nodes added to the tree are balanced and collision free. Kanehiro et al. [2014] split the full body into several kinematic chains by fixing the base height. Such separation reduces the processing time for full body kinematics, however, one loses the redundancy of the full kinematic structure. Most of the existing approaches are normally platform specific, which makes it difficult and time consuming to transfer the work to other robot platforms for generic humanoid motion planning problems. Sampling-based planning methods for humanoids are by no means new concepts in robotics. However, these methods are often customized from basic algorithms such as RRT-Connect for particular robots and environments, it is non-trivial to reuse or apply these methods on generic humanoid robots for solving generic problems.

To this end, instead of developing new SBP algorithms specifically for humanoids, we focus on enabling the standard SBP algorithms to solve humanoids motion planning problems by modifying the underlying key components of generic SBP approaches, such as *space representation*, *sampling strategies* and *interpolation functions*. In order to make the method generic for any humanoid platforms, rather than store balanced samples during offline processing, we use a non-linear optimization based Tedrake [2014] full-body IK solver to generate balanced samples on-the-fly. Thus, the proposed method can be easily applied to different humanoid robot platforms without extensive pre-processing and setup. We evaluate the proposed method on a 36 DoF Boston Dynamics Atlas and a 38 DoF NASA Valkyrie humanoid robots, to show that our method is capable of generating reliable collision-free full-body motion for a

generic humanoid. We also evaluate the difference between sampling in end-effector and configuration spaces for different scenarios, and compare the planning time and trajectory length to find an optimal trade off between efficiency and optimality. In particular, we apply our work to solve practical reaching tasks on the Valkyrie robot, as highlighted in Figure 7.1, showing that the proposed method can generate reliable full-body motion that can be executed on full-size humanoid robots.

## 7.1 PROBLEM FORMULATION

Let $\mathcal{C} \in \mathbb{R}^{N+6}$ be the robot's configuration space, where $N$ is the number of articulated joints and the additional 6 DoF represents the unactuated virtual joint ($SE3$) that connects the robot's pelvis and the world. The valid configuration manifold is given as

$$\mathcal{C}_{valid} = \mathcal{C}_{balance} \cap \mathcal{C}_{free}, \tag{7.1}$$

where $\mathcal{C}_{balance} \subset \mathcal{C}$ is the manifold of statically balanced configurations.

For humanoid robots, valid trajectories can only contain states from valid configuration manifold, i.e. $\mathbf{q}_{[0:T]} \subset \mathcal{C}_{valid}$. Generating collision free samples is straightforward by using random sample generators and standard collision checking libraries. However, generating balanced samples is non-trivial, where a random sampling technique is incapable of efficiently finding balanced samples on the low dimensional manifold $\mathcal{C}_{balance}$ by sampling in high dimensional configuration space $\mathcal{C}$. Guided sampling or pre-sampling process is required for efficient valid sample generation. In our approach, a full-body inverse kinematic solver is employed to produce statically balanced samples. The static balance constraint is a combination of constraints on feet and CoM poses, i.e. the static balance constraint is considered as satisfied when the robot's feet have stable contact with the ground and the CoM ground projection stays within the support polygon spanned by the foot contact points. In our case, we only consider scenarios where both feet are in contact with the ground, however, as long as the contact information is given, the method stays the same for whether only one or multiple end-effectors are in contact with the environment.

### 7.1.1 *Whole-body Inverse Kinematics*

Given a seed configuration $\mathbf{q}_{\text{seed}}$ and nominal configuration $\mathbf{q}_{\text{nom}}$ and a set of constraints $\mathbf{C}$, an output configuration that satisfies all the constraints (Equation 3.8, 3.9). The set of constraints for a full-body humanoid robot may include single joint constraints, such as position and velocity limits for articulated joints, it may also include workspace pose constraints, e.g. end-effector poses, centre-of-mass position, etc. In the rest of this chapter, unless specified otherwise, we assume the quasi-static balance constraint and joint limits constraints are included in $\mathbf{C}$ by default. We use a randomly sampled state as the initial seed pose $\mathbf{q}_{\text{seed}}$ for the SQP solver Gill et al. [2005]. Depending on the implementation of the SBP algorithm, we either choose $\mathbf{q}_{\text{nom}}$ to be the current robot state or one of the neighbouring poses drawn from the pool of candidate poses already explored by the SBP algorithm.

## 7.2 SAMPLING-BASED PLANNING FOR HUMANOIDS

Let $\mathbf{x} \in \mathcal{X}$ be the state space where the sampling is carried out. The planning problem can be formulated as

$$\mathbf{q}_{[0:T]} = \text{HuamnoidSBP}(\mathbf{x}_0, \mathbf{x}_T, \mathbf{Env}) \tag{7.2}$$

where $\mathbf{x}_0$ and $\mathbf{x}_T$ are the initial and desired states, and $\mathbf{Env}$ is the environment instance in which this planning problem is defined. In order for SBP algorithms to plan motions for humanoid robots, we need to modify the following components that are involved in most algorithms as shown in Figure 7.2: the space $\mathcal{X}$ where the sampling is carried out; the strategies to draw random samples; and the interpolation function which is normally used in steering and motion evaluation steps. In the next section, we will discuss the details of modifications we applied on those components for scaling standard SBP algorithms to humanoids.

We separate the work into two parts, configuration space sampling and end-effector space sampling. In configuration space sampling approach, the state is represented in

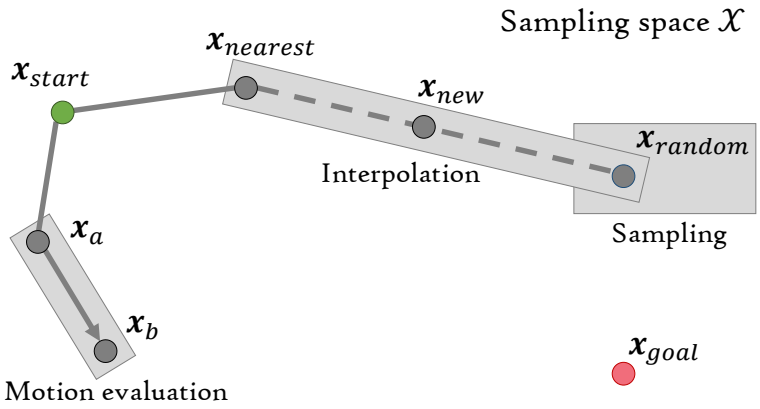



Figure 7.2: Instead of developing new algorithms, we modify those underlying components in SBP solvers to make standard algorithms be capable of solving motion planning problems for humanoid robots.

$\mathbb{R}^{N+6}$ space with joint limits and maximum allowed base movement as the bounds, the sampling state is identical to robot configuration, i.e. $\mathbf{x} = \mathbf{q} \in \mathcal{C}$. For reaching and grasping problems, one might be interested in biasing the sampling in the end-effector related constraints, e.g. to encourage shorter end-effector traverse distance. The end-effector space approach samples in $SE(3)$ space with a region of interests around the robot as the bounds, the state is equivalent to the end-effector's forward kinematics,

$$\mathbf{x} = \mathrm{FK}(\mathbf{q}) : \ \mathcal{W} \rightarrow \mathcal{C} \tag{7.3}$$

However, the final trajectories are represented in configuration space, thus we associate a corresponding configuration for each end-effector space state to avoid ambiguity and duplicated calls to the IK solver.

### 7.2.1 *Configuration Space Sampling*

Algorithm 10 highlights the components' modifications required for sampling in configuration space:

**Algorithm 10** Humanoid Configuration Space SBP

**sampleUniform**()

succeed = False
**while not** succeed **do**
    $\bar{\mathbf{q}}_{\text{rand}} = \text{RandomConfiguration}()$
    $\mathbf{q}_{\text{rand}}$, succeed = $\text{IK}(\bar{\mathbf{q}}_{\text{rand}}, \bar{\mathbf{q}}_{\text{rand}}, \mathbf{C})$
**return** $\mathbf{q}_{\text{rand}}$

**sampleUniformNear**($\mathbf{q}_{\text{near}}, d$)

succeed = False
**while not** succeed **do**
    $A \leftarrow \text{Zeros}(N + 6)$
    **while not** succeed **do**
        $\bar{\mathbf{q}}_{\text{rand}} = \text{RandomNear}(\mathbf{q}_{\text{near}}, d)$
        Set constraint $\|\mathbf{q}_{\text{rand}} - \bar{\mathbf{q}}_{\text{rand}}\|_W < A$
        $\mathbf{q}_{\text{rand}}$, succeed = $\text{IK}(\bar{\mathbf{q}}_{\text{rand}}, \mathbf{q}_{\text{near}}, \mathbf{C})$
        Increase $A$
    **if** $\text{distance}(\mathbf{q}_{\text{rand}}, \mathbf{q}_{\text{near}}) > d$ **then**
        succeed = False
**return** $\mathbf{q}_{\text{rand}}$

**interpolate**($\mathbf{q}_a, \mathbf{q}_b, d$)

$\bar{\mathbf{q}}_{\text{int}} = \text{InterpolateConfigurationSpace}(\mathbf{q}_a, \mathbf{q}_b, d)$
succeed = False
$A \leftarrow \text{Zeros}(N + 6)$
**while not** succeed **do**
    Set constraint $\|\mathbf{q}_{\text{int}} - \bar{\mathbf{q}}_{\text{int}}\|_W < A$
    $\mathbf{q}_{\text{int}}$, succeed = $\text{IK}(\bar{\mathbf{q}}_{\text{int}}, \mathbf{q}_a, \mathbf{C})$
    Increase $A$
**return** $\mathbf{q}_{\text{int}}$

#### 7.2.1.1 *Sampling Strategies*

For sampleUniform(), we first generate random samples from $\mathcal{X}$ and then use full-body IK solver to process the random samples to generate samples from the balanced manifold $\mathcal{X}_{\text{balance}}$

$$\mathbf{q}_{\text{rand}} = \text{IK}(\bar{\mathbf{q}}_{\text{rand}}, \bar{\mathbf{q}}_{\text{rand}}, \mathbf{C}) \tag{7.4}$$

where $\bar{\mathbf{q}}_{\text{rand}} \in \mathcal{X}$ is a uniform random configuration and $\mathbf{q}_{\text{rand}} \in \mathcal{X}_{\text{balance}}$ is random sample from the balanced manifold. We use $\bar{\mathbf{q}}_{\text{rand}}$ as nominal pose since we want

to generate random postures rather than postures close to other already existing samples. This is to indirectly encourage exploration of the null-space of the task. The constraint set $\mathbf{C}$ contains the static balance and joint limit constraints. When sampling around a given state (sampleUniformNear($\mathbf{q}_{\text{near}}, d$)), we first get a random state $\bar{\mathbf{q}}_{\text{rand}}$ that is close to $\mathbf{q}_{\text{near}}$ within distance $d$. The IK solver is invoked with $\bar{\mathbf{q}}_{\text{rand}}$ as the seed pose, and $\mathbf{q}_{\text{near}}$ as the nominal pose. An additional configuration space constraint is added to the constraint set

$$| \mathbf{q}_{\text{rand}} - \bar{\mathbf{q}}_{\text{rand}} |_W \leq A \tag{7.5}$$

where $A \in \mathbb{R}^{N+6}$ is a tolerance vector initially set to zero. In most cases the system will be over constrained, in which case we need to increase the tolerance to ensure balance. Normally, the lower-body joints are neglected first, i.e. increasing corresponding $w_i$, meaning that we allow the lower-body joints to deviate from $\bar{\mathbf{q}}_{\text{rand}}$ in order to keep the feet on the ground and maintain balance. We use $\mathbf{x}_{\text{near}}$ as the nominal pose since later on the random state is likely to be appended to $\mathbf{q}_{\text{near}}$, and one wants the random state be close to the near state. The new sample is discarded if the distance between $\mathbf{q}_{\text{near}}$ and $\mathbf{q}_{\text{rand}}$ exceeds the limit $d$.

#### 7.2.1.2 *Interpolation*

In order to find a balanced state interpolated along two balanced end-point states, we first find the interpolated, likely to be unbalanced state

$$\bar{\mathbf{q}}_{\text{int}} = \mathbf{q}_a + d \, | \, \mathbf{q}_b - \mathbf{q}_a \, | \, . \tag{7.6}$$

A similar configuration space constraint to Equation 7.5 is applied to constrain the balanced interpolated state $\mathbf{q}_{\text{int}}$ close to $\bar{\mathbf{q}}_{\text{int}}$

$$| \mathbf{q}_{\text{int}} - \bar{\mathbf{q}}_{\text{int}} | \leq A \tag{7.7}$$

The two end-point states $\mathbf{q}_a$ and $\mathbf{q}_b$ are valid samples generated using our sampling strategies. Due to the convex formulation of the balance constraint, a valid interpolated state is guaranteed to be found. It is worth mentioning that in some cases

**Algorithm 11** Humanoid End-Effector Space SBP

**sampleUniform**()

succeed = False
**while not** succeed **do**
    $\bar{\mathbf{x}}_{\text{rand}} = \text{RandomSE3}()$
    Set constraint $\|\bar{\mathbf{x}}_{\text{rand}} - \Phi(\mathbf{q}_{\text{rand}})\| \leq 0$
    $\mathbf{q}_{\text{rand}}$, succeed $= \text{IK}(\bar{\mathbf{q}}_{\text{rand}}, \bar{\mathbf{q}}_{\text{rand}}, \mathbf{C})$
$\mathbf{x}_{\text{rand}} = \text{FK}(\mathbf{q}_{\text{rand}})$
**return** $\mathbf{x}_{\text{rand}}, \mathbf{q}_{\text{rand}}$

**sampleUniformNear**($\mathbf{x}_{\text{near}}, d$)

succeed = False
**while not** succeed **do**
    $\bar{\mathbf{x}}_{\text{rand}} = \text{RandomNearSE3}(\mathbf{x}_{\text{near}}, d)$
    Set constraint $\|\bar{\mathbf{x}}_{\text{rand}} - \text{FK}(\mathbf{q}_{\text{rand}})\| \leq 0$
    $\mathbf{q}_{\text{rand}}$, succeed $= \text{IK}(\mathbf{q}_{\text{rand}}, \mathbf{q}_{\text{near}}, \mathbf{C})$
$\mathbf{x}_{\text{rand}} = \bar{\mathbf{x}}_{\text{rand}}$
**return** $\mathbf{x}_{\text{rand}}, \mathbf{q}_{\text{rand}}$

**interpolate**($\mathbf{x}_a, \mathbf{x}_b, d$)

$\bar{\mathbf{x}}_{\text{int}} = \text{InterpolateSE3}(\mathbf{x}_a, \mathbf{x}_b, d)$
succeed = False
$B \leftarrow \text{Zeros}(SE3)$
**while not** succeed **do**
    Set constraint $\|\bar{\mathbf{x}}_{\text{int}} - \text{FK}(\mathbf{q}_{\text{int}})\| < B$
    $\mathbf{q}_{\text{int}}$, succeed $= \text{IK}(\mathbf{q}_a, \mathbf{q}_a, \mathbf{C})$
    Increase $B$
$\mathbf{x}_{\text{int}} = \text{FK}(\mathbf{q}_{\text{int}})$
**return** $\mathbf{x}_{\text{int}}, \mathbf{q}_{\text{int}}$

the interpolation distance equation no longer holds after increasing the tolerance, i.e. $\frac{|\mathbf{x}_{\text{int}} - \mathbf{x}_a|}{|\mathbf{x}_b - \mathbf{x}_a|} \neq d$. However, this is a necessary step to ensure that the balance constraint are satisfied.

### 7.2.2 *End-Effector Space Sampling*

Algorithm 11 highlights the components' modifications required for sampling in end-effector space:

#### 7.2.2.1 *Sampling Strategies*

It is straight forward to sample in $SE(3)$ space, however, it is non-trivial to sample balanced samples from the $\mathcal{X}_{\text{balance}}$ manifold. For sampleUniform(), we first randomly generate $SE(3)$ state $\bar{\mathbf{x}}_{\text{rand}}$ within a region of interest in front of the robot. The full-body IK is invoked with an additional end-effector pose constraint

$$\|\bar{\mathbf{x}}_{\text{rand}} - \text{FK}(\mathbf{q}_{\text{rand}})\| \leq 0 \tag{7.8}$$

The sampler keeps drawing new random states $\bar{\mathbf{x}}_{\text{rand}}$ until the SQP solver returns a valid output $\mathbf{q}^*$. The valid random state $\mathbf{x}_{\text{rand}}$ can be calculated using forward kinematics, e.g. $\mathbf{x}_{\text{rand}} = \text{FK}(\mathbf{q}^*)$. The same procedure applies to sampleNear$(\mathbf{x}_{\text{near}}, d)$, but using $\mathbf{x}_{\text{near}}$ as the seed configuration.

#### 7.2.2.2 *Interpolation*

Similar to sampling near a given state, for interpolation in end-effector space, we first find the interpolated state $\bar{\mathbf{x}}_{\text{int}} \in SE(3)$ and add the following term into constraint set

$$|\, \bar{\mathbf{x}}_{\text{int}} - \text{FK}(\mathbf{q}) \,| \leq B \tag{7.9}$$

where $B \in \mathbb{R}^6$ is a tolerance vector initially set to zero. If the system is over constrained after adding end-effector pose constraint, we need to selectively relax the tolerance for different dimensions $(x, y, z, roll, pitch, yaw)$ until the IK solver succeeds. Then we reassign the interpolated state using forward kinematics, $\mathbf{x}_{\text{int}} = \text{FK}(\mathbf{q}_{\text{int}})$.

#### 7.2.2.3 *Multi-Endeffector Motion Planning*

Some tasks require coordinated motion involving multiple end-effectors, e.g. bi-manual manipulation and multi-contact motion. It is obvious that, from a configuration space point of view, there is no difference as long as the desired configuration is specified. It is also possible for the end-effector space sampling approach to plan motion with multiple end-effector constraints. Let $\mathbf{y}_k^* \in SE(3)$ be the desired pose constraints for end-effector $k \in \{1, \ldots, K\}$. A meta end-effector space $\mathcal{X} \in \mathbb{R}^{6\times K}$ can be con-

Table 7.1: Planning time (in seconds) of empty space reaching problem utilising different algorithms. The result is averaged over 100 trails.

| Algorithms | | Sampling Space | |
|---|---|---|---|
| | | End-Effector Space | Configuration Space |
| Unidirectional | RRT | $25.863 \pm 22.894$ | $1.4129 \pm 1.4466$ |
| | PRM | $4.2606 \pm 3.0322$ | $0.5912 \pm 0.5912$ |
| | EST | $28.055 \pm 18.270$ | $0.3112 \pm 0.3112$ |
| Bidirectional | BKPIECE | $5.3989 \pm 5.9470$ | $0.1781 \pm 0.0332$ |
| | SBL | $3.0602 \pm 0.9859$ | $0.2804 \pm 0.0480$ |
| | RRT-Connect | $2.8228 \pm 0.3412$ | $0.1853 \pm 0.0450$ |

structed to represent the sampling space for all end-effectors. Similar sampling and interpolation functions can be implemented by constructing extra constraints for each end-effector $k$.

## 7.3 EVALUATION

We aim to generalize the common components of sampling-based motion planning algorithms for humanoid robots so that existing algorithms can be used without extra modification. We implemented our approach in the EXOTica motion planning and optimization framework Ivan et al. [2018] as a humanoid motion planning solver, which internally invokes the SBP planners from the Open Motion Planning Library (OMPL, Şucan et al. [2012]). We have set up the system with our customized components, and evaluated our approach on the following six representative algorithms: RRT Lavalle [1998], RRT-Connect Kuffner and LaValle [2000], PRM Kavraki et al. [1996], BKPIECE Şucan and Kavraki [2009], EST Hsu et al. [1997] and SBL Sánchez and Latombe [2003]. The evaluations were performed in a single thread with a 4.0GHz Intel Core i7-6700K CPU.

Table 7.2: Evaluation of full-body collision-free motion planning. RRT-Connect$_e$ sampling in end-effector space, all other methods sampling in configuration space. No. evaluation shows the number of state evaluation calls, i.e. evaluate if a sampled/interpolated state is valid. No. IK indicates the number of online full-body IK calls, and IK time is the total time required for solving those IK, which is the most time consuming element. The result is averaged over 100 trails.

| Tasks | Algorithms | Planning time (s) | No. evaluation | No. IK | IK time (s) |
|---|---|---|---|---|---|
| Task 1 | BKPIECE$_c$ | $42.5 \pm 26.4$ | $1946 \pm 1207$ | $2598 \pm 1582$ | $41.4 \pm 25.7$ |
| | SBL$_c$ | $27.8 \pm 8.59$ | $1313 \pm 418$ | $1508 \pm 445$ | $27.0 \pm 8.33$ |
| | RRT-Connect$_e$ | $9.91 \pm 4.80$ | $597 \pm 354$ | $727 \pm 387$ | $9.51 \pm 4.58$ |
| | RRT-Connect$_c$ | $1.53 \pm 0.80$ | $95 \pm 54$ | $118 \pm 64$ | $1.48 \pm 0.77$ |
| task 2 | BKPIECE$_c$ | $40.5 \pm 21.7$ | $1911 \pm 970$ | $2473 \pm 1254$ | $39.4 \pm 20.1$ |
| | SBL$_c$ | $22.2 \pm 9.51$ | $1089 \pm 472$ | $1259 \pm 547$ | $21.5 \pm 9.23$ |
| | RRT-Connect$_e$ | $12.4 \pm 6.65$ | $656 \pm 405$ | $826 \pm 458$ | $11.9 \pm 6.41$ |
| | RRT-Connect$_c$ | $2.25 \pm 0.85$ | $106 \pm 42$ | $166 \pm 59$ | $2.19 \pm 0.83$ |
| task 3 | BKPIECE$_c$ | $45.7 \pm 19.8$ | $2057 \pm 949$ | $2758 \pm 1166$ | $44.5 \pm 19.3$ |
| | SBL$_c$ | $33.8 \pm 22.2$ | $1414 \pm 950$ | $1756 \pm 1151$ | $33.0 \pm 21.6$ |
| | RRT-Connect$_e$ | $25.3 \pm 13.9$ | $1031 \pm 532$ | $1436 \pm 720$ | $24.6 \pm 13.7$ |
| | RRT-Connect$_c$ | $3.45 \pm 0.77$ | $165 \pm 49$ | $200 \pm 53$ | $3.36 \pm 0.75$ |

### 7.3.1 *Empty Space Reaching*

In the first experiment, we have the robot reach a target pose in front of the robot in free space, where only self-collision and balance constraints are considered. This is a sanity check to show that the proposed method can be used robustly across different planning algorithms to generate trajectories for humanoid robots. We solve the reaching problem using the six testing algorithms in two different sampling spaces, each across 100 trials. The results are shown in Table 7.1. Although the planning time varies across different algorithms and sampling spaces, the result shows that standard planning algorithms are able to generate motion plans for humanoid robots using our method. As expected, bi-directional algorithms are more efficient than their unidirectional variants. Also, sampling in configuration space is much more efficient than in end-effector space due to the higher number of IK calls for the later case.

Table 7.3: Evaluation of full-body collision-free motion planning. RRT-Connect$_e$ sampling in end-effector space, all other methods sampling in configuration space. $\mathcal{C}$ cost is the configuration space trajectory length, $\mathcal{W}$ cost is the end-effector traverse distance in workspace, CoM cost is the CoM traverse distance in workspace. The result is averaged over 100 trails.

| Tasks | Algorithms | $\mathcal{C}$ cost (rad.) | $\mathcal{W}$ cost (m) | CoM cost (m) |
|---|---|---|---|---|
| task 1 | BKPIECE$_c$ | $7.37 \pm 2.43$ | $2.10 \pm 0.80$ | $0.24 \pm 0.10$ |
| | SBL$_c$ | $6.25 \pm 1.06$ | $2.14 \pm 0.71$ | $0.23 \pm 0.06$ |
| | RRT-Connect$_e$ | $2.93 \pm 0.96$ | $0.58 \pm 0.11$ | $0.07 \pm 0.02$ |
| | RRT-Connect$_c$ | $2.71 \pm 0.68$ | $0.99 \pm 0.23$ | $0.11 \pm 0.03$ |
| task 2 | BKPIECE$_c$ | $6.59 \pm 2.43$ | $1.95 \pm 0.59$ | $0.27 \pm 0.09$ |
| | SBL$_c$ | $5.34 \pm 2.00$ | $1.79 \pm 0.80$ | $0.24 \pm 0.09$ |
| | RRT-Connect$_e$ | $4.12 \pm 2.02$ | $0.77 \pm 0.08$ | $0.09 \pm 0.04$ |
| | RRT-Connect$_c$ | $3.29 \pm 1.14$ | $1.20 \pm 0.33$ | $0.14 \pm 0.05$ |
| task 3 | BKPIECE$_c$ | $7.49 \pm 2.52$ | $1.96 \pm 0.73$ | $0.25 \pm 0.08$ |
| | SBL$_c$ | $8.68 \pm 2.26$ | $2.10 \pm 0.44$ | $0.28 \pm 0.11$ |
| | RRT-Connect$_e$ | $7.19 \pm 4.93$ | $0.92 \pm 0.13$ | $0.16 \pm 0.05$ |
| | RRT-Connect$_c$ | $4.68 \pm 0.59$ | $1.38 \pm 0.12$ | $0.14 \pm 0.03$ |

### 7.3.2 *Collision-free Reaching*

We setup three different scenarios, from easy to hard, as illustrated in Figure 7.3, to evaluate the performance of different algorithms in different sampling spaces. Unfortunately, the evaluation suggests that standard unidirectional algorithms are unable to solve these problems (within a time limit of 100 seconds). Without bi-directional search, the high dimensional humanoid configuration space is too complex for sampling-based methods to explore. Table 7.3 highlights the results using four different bidirectional approaches. Note that when sampling in end-effector space, only RRT-Connect is able to find a valid solution in the given time, other bidirectional search algorithms like BKPIECE and SBL are also unable to find valid trajectories. The result indicates that RRT-Connect sampling in configuration space is the most efficient and the most robust approach for solving humanoid full-body motion planning problems. It requires the least exploration, thus bypassing expensive online IK queries. Algorithms like BKPIECE and SBL use low-dimensional pro-

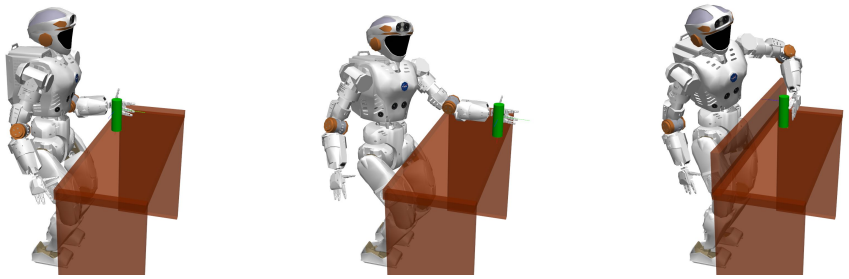
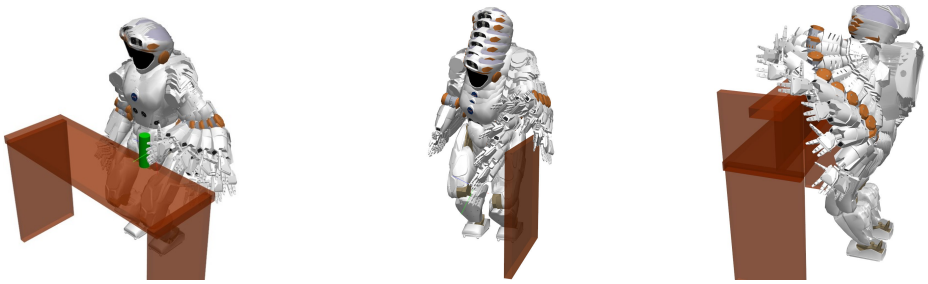
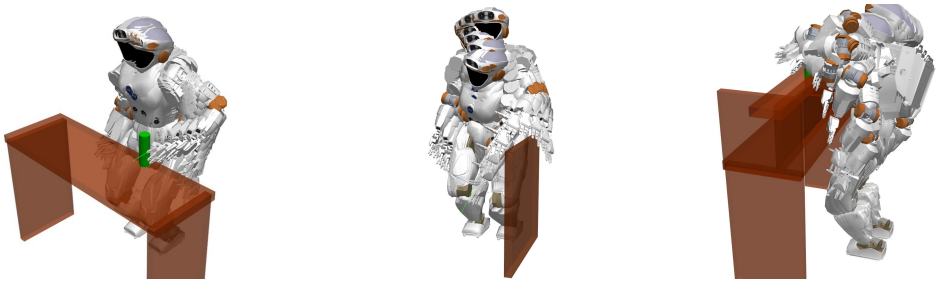


Figure 7.3: Evaluation tasks, from left to right: task 1, target close to robot; task 2, target far away from robot; and task 3, target behind bar obstacle.

jections to bias the sampling, however, the default projections which are tuned for mobile robots and robotic arms do not scale up to high DoF humanoid robots, which leads to long planning time and trajectories with high costs. This can be improved by better projection bias, but it is non-trivial to find a suitable bias without fine tuning. Also, the trajectories generated using RRT-Connect are shorter, meaning that the motion is more stable and robust. It is worth mentioning that RRT-Connect takes longer time to plan when sampling in the end-effector space than it does in the configuration space, but the planned trajectories have shorter end-effector and CoM traverse distances. In some scenarios where planning time is not critical, one can choose to use RRT-Connect in end-effector space to generate trajectories with shorter end-effector traverse distance. These results also suggest that the full-body IK computation dominates the planning time. This is in contrast with classical SBP problems where collision-detection is the most time consuming component. However, the IK solver is necessary for maintaining balance, as shown in Figure 7.5, where the trajectories' CoM projections are within the support polygon. In more complex scenarios, such as reaching through narrow passages and bi-manual tasks, most algorithms fail to generate valid trajectories apart from RRT-Connect. As mentioned, some algorithms' performance depends on the biasing methods, e.g. projection bias and sampling bias. However, it is non-trivial to find the appropriate bias for humanoids that would generalize across different tasks. Figure 7.5 highlights some examples of reaching motion in more complex scenarios with different robot models. As stated earlier, this work focuses on generalising SBP algorithms for humanoids, where as one can easily setup the system on new platforms as long as the robot model is given. For



(a) Trajectories generated using configuration space sampling.



(b) Trajectories generated using end-effector space sampling.

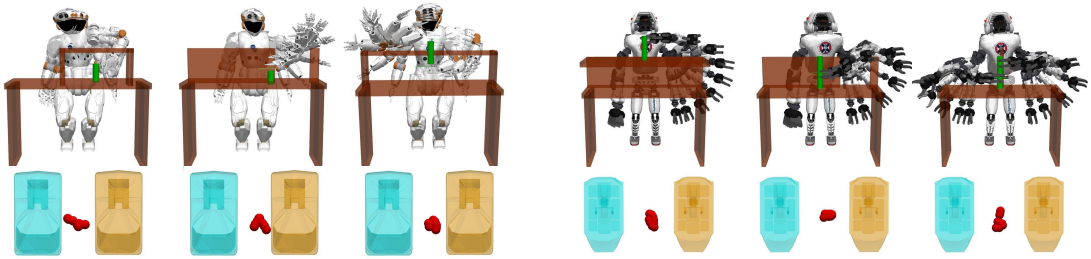
Figure 7.4: Whole-body motion plans generated using different sampling spaces. The task is identical for each column. In general, configuration space sampling leads to shorter trajectory length; end-effector space sampling leads to shorter end-effector traverse distance.

instance, one can easily switch from Valkyrie (Figure 7.5a) to Atlas (Figure 7.5b) in minutes without extensive pre-processing and setup procedures.

In order to test the reliability and robustness of the proposed method, we applied our work on the Valkyrie robot accomplishing reaching and grasping tasks in different scenarios, as highlighted in Fig 7.6. During practical experiments, the collision environment is sensed by the on–board sensor and represented as an octomap Hornung et al. [2013]. The experiment results show that our method is able to generate collision-free full-body motion plans that can be executed on full-size humanoid robot to realise practical tasks such as reaching and grasping. A supplementary video of the experiment results can be found at `https://youtu.be/AZQY_QOX0Pw`.

## 7.4 CONCLUSION

In this chapter we generalise the key components required by sampling-based algorithms for generating collision-free and balanced full-body trajectories for humanoid



(a) Planned motion for the 38-DoF NASA Valkyrie robot.

(b) Planned motion for the 36-DoF Boston Dynamics Atlas robot.

Figure 7.5: Collision-free full-body motion generated in different scenarios with different robot models. The corresponding CoM trajectories are illustrated in the second row (red dots). The framework is setup so that one can easily switch to new robot platforms without extensive preparing procedures.

robots. We show that by using the proposed methods, standard SBP algorithms can be invoked to directly plan for humanoid robots. We also evaluate the performance of different algorithms on solving planning problems for humanoids, and point out the limitations of some algorithms. A variety of different scenarios are tested showing that the proposed method can generate reliable motion for humanoid robots in different environments. This work can be transferred to different humanoid robot models with easy setup procedure that can be done in very a short period of time, without extensive pre-processing for adapting the existing algorithms to different robots, as we have tested on the 36 DoF Boston Dynamics Atlas and the 38-DoF NASA Valkyrie robots. In particular, we applied this work on the Valkyrie robot accomplishing different tasks, showing that the proposed method can generate robust full-body motion that can be executed on real robots.

The full-body inverse kinematics is crucial in terms of guaranteeing balance and smoothness, however, the result in Table 7.3 shows that the IK solver dominates over 95% of the online computation time. Although it depends on the implementation and underlying algorithms of the IK solver which is not the focus of this paper, we intend to investigate faster IK implementation to bootstrap sampling and interpolation. This will make the state space exploration more efficient, so that other standard algorithms may be able to find valid solutions within the same time window.

In this part of the thesis, we have discussed how to plan valid trajectories in complex environments, for either fixed-base robotic arms or humanoid robots. The low

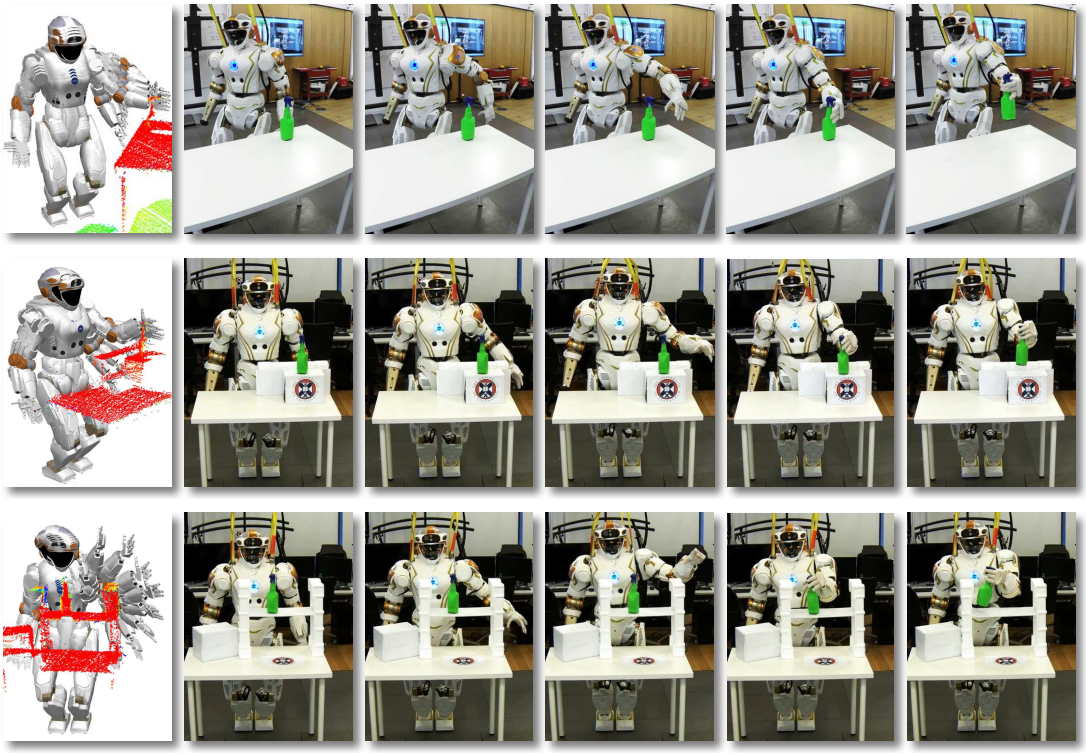


Figure 7.6: Collision-free full-body motion execution on the NASA Valkyrie humanoid robot. In each row, the first figure highlights the motion plan, followed by execution snapshots.

level controller can now follow these plans and drive the actuators, however, the task can only be correctly accomplished if the environment stays static and all tracking is perfect. In practice, any environmental changes and tracking inaccuracy might lead the execution to a failure. In order to improve robustness and success rate, online re-planning/adaptation is needed for compensating run time perturbations.

# Part III

# MOTION ADAPTATION

# 8

# MOTION ADAPTATION PRELIMINARIES

After having planned the motion (Equation 1.2), along which the robot can reach to the desired end-pose (Equation 1.1), we can then directly apply the plan in an open loop manner. However, the robot can only correctly accomplish the task in static environments with the assumption of perfect sensing and motor execution, which is not true in many practical scenarios. Controlling robots in changing environments with uncertainties is one of the most difficult problems in robotics. It arises in tasks such as manipulating moving objects, or interacting with people and other robots. The trajectory could become invalid due to various reasons, e.g. the trajectory is blocked by obstacles, the target moves outside of the working envelope of the robot, inaccurate motor execution, etc. In such scenarios, replanning is typically required to calculate a new feasible plan. However, replanning, as a naïve online adaptation approach, is an expensive process that causes delay, which makes real-time implementation of fast, dynamic motion a significant challenge. More advanced motion adaptation methods are required for addressing such challenge (Equation 1.3). In this chapter, we present related work in online adaptation and discuss the limitations of current approaches.

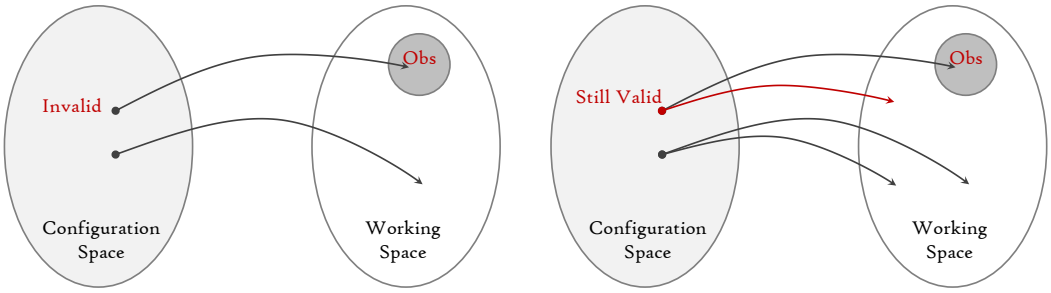
## 8.1 CLASSICAL ADAPTATION APPROACHES

There are different ways for allowing robots to operate in dynamic environments. One naïve approach is to keep replanning during execution based on the sensory information (Karaman et al. [2011]), which is a robust but also expensive approach that normally can not be used for tasks that require both accuracy and efficiency. Recently, Pan and Manocha [2011] showed that real time replanning can be achieved by using many-core GPUs, where multiple processors are created simultaneously to speed up the computation. Park et al. [2014] showed that online replanning cost can

be reduced by interleaving planning with execution, where they split a whole trajectory into multiple sub-trajectories and only plan for one of them at each step. One can also apply motion adaptation methods such as Dynamic Movement Primitives (DMP, Schaal [2006]) when one has access to a demonstrated trajectory, where any captured motion gets encoded into a set of differential equations. These methods can be used to handle perturbations during execution (Park et al. [2008], Englert and Toussaint [2014]). The Artificial Potential Field (APF, Khatib [1985]) method and its derivatives have gained popularity in the field of mobile robots to solve problems involving online collision avoidance. APFs use the idea of imaginary forces acting on the robot, where the obstacles have repulsive forces and target has an attractive force. The robot is driven by the sum of all these forces, calculated based on the minimum distance between robot and obstacles/target. From this point of view, APFs can be considered as a relative distance based approach. Park et al. Park et al. [2008] introduced a dynamic potential field where the potential field takes obstacles' velocities into account to provide more robust plans. Similar to global methods, the performance of local planners such as APF can also be improved with the aid of parallel computing (Kaldestad et al. [2014]). Khansari-Zadeh and Billard [2012] introduced a dynamical system (DS) based approach where an original motion can be modified on-the-fly to avoid convex obstacles. However, it only considers the end-effector trajectory while there are situations where the trajectories of other links are in collision as well.

## 8.2 MOTION ADAPTATION IN ALTERNATE SPACES

Classic approaches aim to find a configuration space plan $\mathbf{q}_{[0:T]}$ that satisfy all the constraints, then a unique working space trajectory is given by the configuration space plan, as shown in Figure 8.1a. The problem of using configuration space representation is that the plan is not flexible, i.e. the configuration space state will be invalidated if the corresponding working space state is in collision. Since the plan in configuration space is a continuous function (or discrete function with $\Delta t$), it is difficult to compute another feasible plan by just modifying a few configurations, and this is where replanning is required.




(a) One-to-one mapping from configuration space to working space

(b) One-to-many mapping from alternate space to working space

Figure 8.1: (a): the configuration space plan will be invalidated if the working space state is in collision. (b): a state in alternate space is still valid even if some of the working space states are in collision.

While these aforementioned methods aim to find valid configuration space plans, some approaches encode the plans in alternate spaces (Figure 8.1b). Relationship based representations have been studied in computer graphics (Ho and Komura [2009], Ho et al. [2010b], Al-Asqhar et al. [2013], Ho et al. [2010a, 2014]) for motion re-targeting problems and they have been applied to robotics in Ho and Shum [2013], Ivan et al. [2013] and Nierhoff et al. [2014]. Rather than using configuration space, these methods represent the problems in some alternate spaces in which the relationships between robot and environment are encoded, generating executable plans by capturing relational invariances. The alternate space states typically have multiple corresponding robot poses, as shown in Figure 8.1b, such that the state can be still valid if some of the corresponding poses are not.

Ho and Komura [2009] introduced a topological space *writhe* to solve character motion animation problem, where *writhe* indicates how much of the two characters, or one character and an environmental object, twist around each other. For two curves $\mathcal{C}_1$ and $\mathcal{C}_2$, the writhe $w$ can be calculated using Gauss Linking Integral (GLI)

$$w = GLI(\mathcal{C}_1, \mathcal{C}_2) = \frac{1}{4\pi} \int_{\mathcal{C}_1} \int_{\mathcal{C}_2} d\mathcal{C}_1 \times d\mathcal{C}_2 \cdot \frac{\mathcal{C}_1 - \mathcal{C}_2}{|\mathcal{C}_1 - \mathcal{C}_2|^3}, \tag{8.1}$$

where $\times$ indicates cross product and $\cdot$ is dot product. Ho et al. [2010b] also introduced their motion adaptation method using another topological space representation, namely *interaction mesh*. The key concepts involved in interaction mesh is the Laplace coordinate representation and the Laplacian deformation energy minimisa-

Figure 8.2: Remapping human motion to a robot model with different kinematic structures using interaction mesh. By continuously remapping human motion to robot model, we obtain a teleoperation method that works between agents with different kinematic structures.

tion. For instance, let a set of key points, $P = \{p_i\}, 1 \leq i \leq m$, represent robot links and other objects, then the Laplace coordinate for each point is

$$L(p) = p - \sum_{r \in N_p} \frac{r w_{pr}}{\sum_{s \in N_p} w_{ps}}, \tag{8.2}$$

$$w_{pr} = \frac{W_{pr}}{|r - p|}, \; w_{ps} = \frac{W_{ps}}{|s - p|}, \tag{8.3}$$

where $N_p$ are the neighbourhoods of $p$, $w_{pr}$ is the weighting factor that is inversely proportional to the distance between points $p$ and $r$, multiplied by constant $W_{pr}$. The interaction mesh method which is initially introduced to solve graphic problems, can be applied to solve robot problems as well, as highlighted in Figure 8.2. By choosing key points on both human and robot's upper-bodies, e.g. shoulder, elbow and wrist, we can remap configurations and continuous motion between human and humanoid robots with different kinematic structures. Similarly, Nierhoff et al. [2014] used a task-space distance mesh to represent the robot and target in a unified model to imitate human full body motion to a humanoid robot. In summary, the alternate space adaptation methods map reference motion to a new configuration space motion, which can be seen as a teleoperation technique if the reference motion is generated online. Nevertheless, these methods can be used to adjust pre-planed reference motion to correct run time errors.

These methods typically need to know the relationship in advance and encode them into the alternate space. However, in real world scenarios, there are many situations in which one encounters unexpected and unmodelled objects (e.g. moving obstacles, people) which is non-trivial to handle online with these existing alternate space methods. In next chapter, we will introduce a new alternate space representation, the *Distance Mesh*, which is able to handle unexpected objects on-the-fly.

# 9

# MOTION ADAPTATION IN CHANGING ENVIRONMENTS

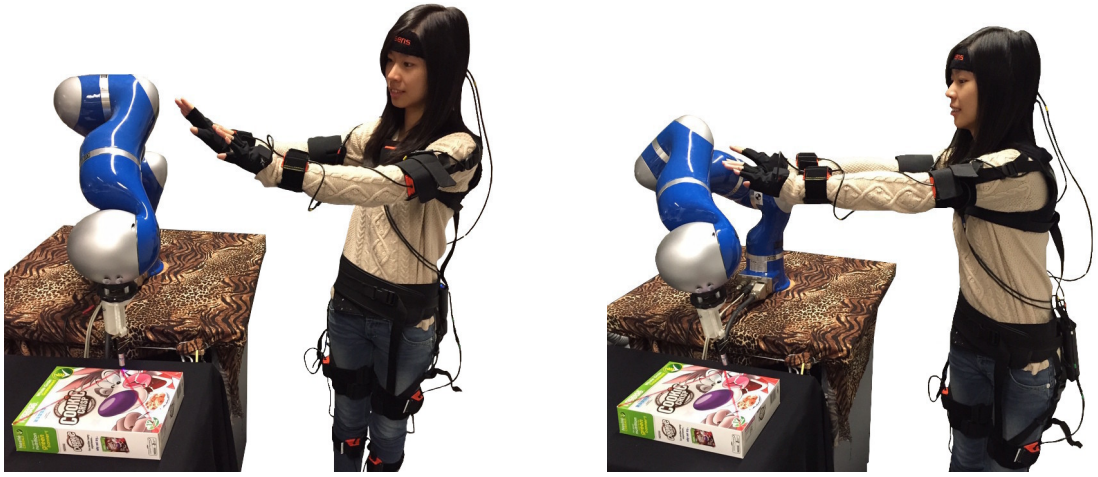


Figure 9.1: Robot-human close interaction. Left figure shows the robot's original motion, the right figure shows the adapted motion when human subject pushes her hands to block the original trajectory.

To allow the robots to handle unexpected changes, for problems that involve reaching targets around dynamic obstacles and people in particular, we present a relative distance based space representation, in which we model the relative distances between robot links, targets and obstacles. In addition, we construct the relative distance space plan in an incremental way, which gives the robot the ability to avoid not only the obstacles which are known apriori during planning phase but also the unexpected obstacles which are detected during execution. We assume that there only exists one global minima, such that we can employ a fast local method to remap from relative distance space to joint space. In contrast to some other end-effector trajectory adaptation methods, e.g. Dynamical System approach Khansari-Zadeh and Billard [2012], our method is able to adapt the trajectories of all robot links simultaneously. We apply our approach on a 7-DoF robot arm with a mock-up welding problem, as illustrated in Figure 9.1. We also demonstrate the scalability of our method on a 14-DoF dual-arm Baxter robot with a water pouring task. In both experiments, the

robots operate in relatively unstructured environments, where the robot needs to accomplish the tasks while avoiding colliding with human.

## 9.1 RELATIVE DISTANCE SPACE

This section presents the method for capturing interactions of the robot with its environment. Our objective is to create a method that will: 1) capture the pose of the robot on its own (for mimicking or pose re-targeting), 2) capture the reaching behaviour (as the task objective), and 3) capture the avoiding behaviour (obstacle avoidance). A representation that simultaneously captures these three kinds of interactions would provide a powerful tool for transferring, adapting, and planning robot motion for a wide range of reaching and manipulation tasks in environments with dynamic obstacles.

The interaction mesh representation proposed in Ho et al. [2010b] satisfies the first two requirements (capturing the pose of the robot and its interaction with reaching targets) but it is not suitable for obstacle avoidance. Maintaining relative pose with the obstacles generates artefacts in which the obstacles significantly affect the equilibrium position. To deal with this issue, we propose to use a representation we call the relative distance space.

Assume a robot has $N$ joints $q_i, i \in N$, the adaptation can be formalized as

$$\min_{q_1,\dots,q_N} \sum_{i=1}^{N} c(q_i), \tag{9.1}$$

where $c(\cdot)$ is the state-dependent cost function. Typically,

$$c(q_i) = c_{\text{pose}}(q_i) + c_{\text{goal}}(q_i) + c_{\text{obs}}(q_i) + c_{\text{rest}}(q_i), \tag{9.2}$$

where $c_{\text{pose}}(\cdot)$ is the cost for maintain particular poses, $c_{\text{goal}}(\cdot)$ is the cost for reaching goal, $c_{\text{obs}}(\cdot)$ is the cost for collision avoidance and $c_{\text{rest}}(\cdot)$ is the cost for other constraints such as joint limits. However, we argue that since the first three costs represent such closely-tied behaviours, i.e. reaching target in a particular way while

avoiding obstacles, one can unify them into one cost term in relative distance space. That is

$$c(q_i) = c_{\mathcal{D}}(q_i) + c_{\text{rest}}(q_i), \tag{9.3}$$

where $c_{\mathcal{D}}(\cdot)$ is the cost in relative distance space states that should solve pose retargeting, reaching and avoiding in a coherent, consistent way. $c^t_{\mathcal{D}}(q_i) = \|\Phi^* - \Phi^t\|$ denotes the actual cost at time $t$, where $\Phi^*$ is the desired state in relative distance space, and $\Phi^t$ is the state at time $t$. The rest of this section explicitly explains how to compute $\Phi^*$ and $\Phi^t$.

We attach $M$ number of virtual points $p_j$ ($j \in M$) to the robot's links. These virtual points are usually joint and end-effector positions. We will also attach $E$ number of virtual points $p_e$ ($e \in E$) to the centre of the obstacles in the environment. The relative distance space models the edges ($\phi_{jl}$) between each $p_j$ and $p_l$, $l \in M \cup E$ and $j \neq l$. Note that $p_j$ always represents a robot link and $p_l$ represents another object (other robot links, targets, obstacles), meaning that we model three different types of distances (Figure 9.2):

1. $\phi_{jl} = \phi_{\text{link}}$, if $p_l$ is a point on different robot link.
2. $\phi_{jl} = \phi_{\text{goal}}$, if $p_l$ is a point on target.
3. $\phi_{jl} = \phi_{\text{obs}}$, if $p_l$ is a point on obstacle.

Note that we ignore the fourth type where $j, l \in E$, since the edges between two environmental objects are not controllable.

We define the relative distance between robot links as

$$\phi_{\text{link}} = w_{jl}\|p_j - p_l\|, \tag{9.4}$$

where $w_{jl}$ is the weighting factor of the edge between $p_j$ and $p_l$. For $\phi_{\text{goal}}$ and $\phi_{\text{obs}}$, the relative distance can be Euclidean, e.g. $\phi_{jl} = w_{jl}d_{jl}, d_{jl} = \|p_j - p_l\|$. However, this will cause a series of problems. For example, distant targets will have a dominant influence on the motion. We can apply a non-linear growth model $\psi(j, l)$ on the

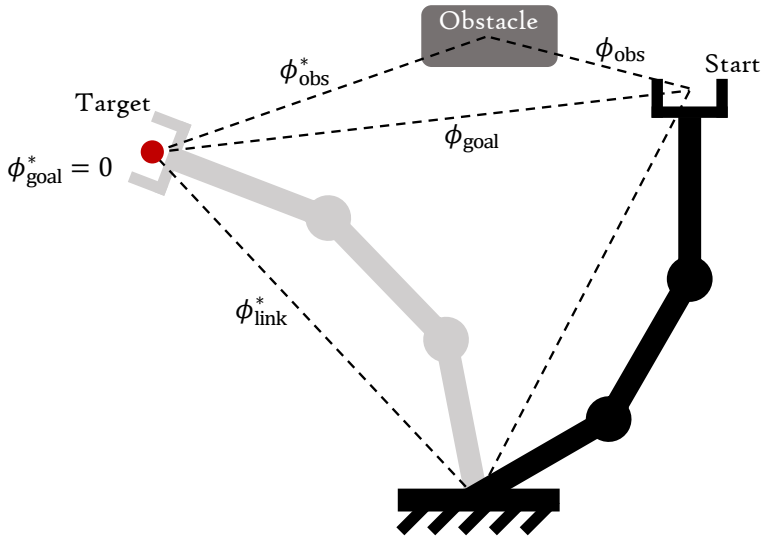



Figure 9.2: Desired and current relative distances. The target state $\Phi^* = [\phi^*_{link}, \phi^*_{goal}, \phi^*_{obs}]$ is constructed before execution, and start/current state $\Phi = [\phi_{link}, \phi_{goal}, \phi_{obs}]$ is computed during each control iteration. Not all of the relative distances are shown here, we only show one example for each type of distances. Note that $\phi_{obs}$ is not required if there is no obstacle.

distance metric to generate a smoother and more robust behaviour for targets and obstacles:

$$\phi_{jl} = w_{jl}\psi(j,l). \tag{9.5}$$

Different non-linear models can be applied here. In general, from a reaching and avoiding point of view, distant obstacles should not affect the robot, and distant target should not introduce unacceptable large effort. An inverse exponential model has the property that starts from the origin and quickly converges to a maximum value, based on which here we show one possible model for handling the interactions with targets and obstacles

$$\psi_{jl} = 1 - e^{-kd_{jl}}, \tag{9.6}$$

where $k > 0$ is a constant. The relative distance increases exponentially with $d_{jl}$, and converges to a maximum value 1 ($\phi_{jl}$ converges to $w_{jl}$). A distant obstacle ($d_{jl} \gg 0$) will not affect the robot if we set $\phi^*_{obs} = w_{jl} = w_{safe}$, i.e. $\phi^*_{obs} \approx \phi_{obs} = w_{safe}$, where $w_{safe}$ is a non-negative constant. For reaching task, we set $\phi^*_{goal} = 0$, meaning that

a distant target can only introduce a prescribed maximum effort to the system, i.e. $\|\phi^*_{\text{goal}} - \phi_{\text{goal}}\| \leq w_{jl}, \forall d \geq 0$.

In order to solve a motion transfer, adaptation and planning problem, we require the current state $\Phi^t$, which we compute using Equations 9.4-9.6, and a desired state

$$\Phi^* = [\phi^*_{\text{link}}, \phi^*_{\text{goal}}, \phi^*_{\text{obs}}] \tag{9.7}$$

$\phi^*_{\text{link}}$ constrains robot poses that can be used for imitation problems, where the reference value $\phi^*_{\text{link}}$ can be computed from demonstration data. Minimizing the difference between the demonstrated and current relative distances will then result in transferring the motion based on the relative distances between the links. However, from target reaching point of view, the robot pose is often used as a secondary task, along side a primary reaching task, or it is not used at all. In this case, the relative link distance term can be ignored entirely. $\phi^*_{\text{goal}}$ is usually set to zero for reaching tasks. One can also set $\phi^*_{\text{goal}}$ to other values, e.g. keeping the end-effector and target with particular distance. $\phi^*_{\text{obs}} = w_{\text{safe}}$, as discussed earlier.

We construct the desired relative distance space target $\Phi^*$ by combining all three distance terms: $\phi^*_{link}$, $\phi^*_{\text{goal}}$ and $\phi^*_{\text{obs}}$. The state is, however, only valid if we keep updating the positions of the links, obstacles and target. We use an operational space controller to track the changes in the environment. For this we require the Jacobian of the relative distance space.

First we compute end-effector Jacobian of the points $p_l$ using standard kinematics tools as

$$\mathbf{J}^{\text{eff}} = \frac{\partial \Phi^{\text{eff}}(\mathbf{q})}{\partial \mathbf{q}} \in \mathbf{R}^{3M \times N}, \tag{9.8}$$

where $\Phi^{\text{eff}}(\mathbf{q})$ is the joint space to end-effector space forward map. Our goal is to find the Jacobian between relative distance space and joint space

$$\mathbf{J} = \frac{\partial \Phi(\mathbf{q})}{\partial \mathbf{q}} = W \frac{\partial \Psi}{\partial \mathbf{q}} \in \mathbf{R}^{X \times N}, \tag{9.9}$$

where $X = \frac{(M+E)(M+E-1)}{2}$, $W$ is the weighting matrix and $\Psi = [\psi_{jl}], j \in M, l \in M \cup E$. The distances between two obstacles or obstacle and target are not considered, so the number of controllable distances is $X$.

$$J_{x,i} = \frac{\partial \phi_{jl}}{\partial q_i} = w_{jl} \frac{\partial \psi_{jl}}{\partial q_i}, \tag{9.10}$$

where $x \in X$, and $\partial \psi_{jl} \in [\frac{\partial \psi_{\text{link}}}{\partial q_i}, \frac{\partial \psi_{\text{goal}}}{\partial q_i}, \frac{\partial \psi_{\text{obs}}}{\partial q_i}]$ depend on edge types. If pose retargeting is required,

$$\frac{\partial \psi_{\text{link}}}{\partial q_i} = \bar{d}_{j_1 j_2} = \frac{(p_{j_1} - p_{j_2}) \cdot (J^{\text{eff}}_{j_1,i} - J^{\text{eff}}_{j_2,i})}{d_{jl}}, \tag{9.11}$$

otherwise $\frac{\partial \psi_{\text{link}}}{\partial q_i} = 0$. Here, $\cdot$ is the dot product, and $J^{jnt}_{j,i} \in \mathbf{R}^{3\times 1}$ is the position Jacobian of point $p_j$ w.r.t. the joint $i$.

For target reaching and collision avoidance, $\phi_{jl} \in \{\phi_{\text{goal}}, \phi_{\text{obs}}\}$, the first derivative of Equation 9.6 yields

$$\frac{\partial \psi_{jl}}{\partial q_i} = k \bar{d}_{jl} e^{-k d_{jl}}, \tag{9.12}$$

where $\bar{d}_{jl}$ is the relative distance Jacobian to end-effector space

$$\bar{d}_{jl} = \frac{(p_j - p_l) \cdot J^{\text{eff}}_{j,i}}{d_{jl}}. \tag{9.13}$$

Note that the Jacobian entries for goal and obstacles are the same, however, since they have different desired value, $\phi^*_{\text{obs}} \neq \phi^*_{\text{goal}}$, the effect of their Jacobian entries are different.

Now we have the desired relative distance space state $\Phi^*$, current state $\Phi^t$ and the Jacobian that are required by the cost function (Equation 9.1 and 9.3). In general, this problem can be solved by any optimization based planners. However, from a real-time implementation point of view, we choose a Jacobian-pseudo-inverse IK type controller due to its simplicity and efficiency. We discuss the performance in Section 9.2.2.

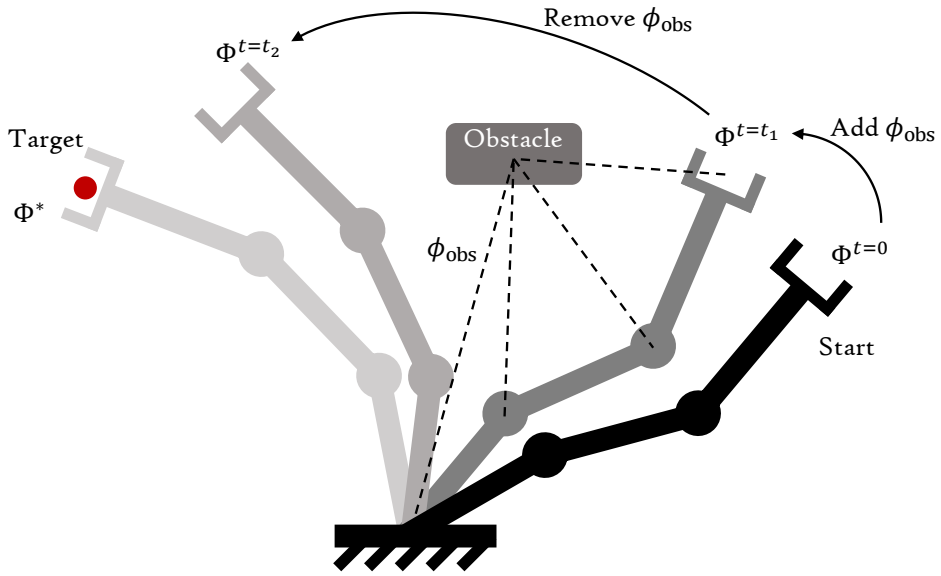



Figure 9.3: Incremental planning structure, i.e. modifying the relative distance state online. The desired state $\Phi$ is computed without obstacle. The robot starts from a state ($\Phi^{t=0}$) with no local obstacle. Obstacle is detected at time $t_1$, new entries will be added into both $\Phi^*$ ($\phi_{\text{obs}}^{t=t_1} = w_{\text{obs}}$) and $\Phi^{t=t_1}$. At time $t_2 > t_1$, the obstacle is no longer close to the robot, the entries for the obstacle are removed.

## 9.2 INCREMENTAL PLANNING STRUCTURE

For the obstacles which are known in advance, their relative distances can be encoded during planning phase. However, when we deal with unexpected obstacles, such as humans walking into the workspace of the robot, we have to modify the distance relationship space on the fly in order to avoid the costly replanning. This will involve adding and removing obstacle vertices, as illustrated in Figure 9.3.

### 9.2.1 *Incremental Planning Structure*

Assume we have a desired alternate space target

$$\Phi^* = [\phi_0^*, \phi_i^*, \ldots, \phi_{M+E}^*] \in \mathbb{R}^{(M+E)\times(M+E)} \tag{9.14}$$

where $M + E$ is the number of vertices (links, obstacles and targets combined), and

$$\phi_i^* = [\phi_{i0}^*, \phi_{i1}^*, \ldots, \phi_{i,M+E}^*]^T \tag{9.15}$$

is the vector that describes the desired distances between vertex $i$ and all other vertices. When a new obstacle $k$ is detected, the original goal $\Phi^*$ is no longer valid. We want to have a new target in alternate space in the form of

$$\Phi^*_{new} = \begin{bmatrix} \Phi^* & \phi^*_k \\ \phi^{*T}_k & 0 \end{bmatrix} \in \mathbb{R}^{(M+E+1)\times(M+E+1)}. \tag{9.16}$$

Note that $\Phi^*$ is still valid since it only depends on old vertices, meaning that we only need to compute $\phi^*_k$ and reuse the old plan as part of the new plan. The key to achieving real-time implementation is to minimise online computation. In our case, it is straight forward to get $\phi^*_k$ and modify the plan without heavy computation. From Equation 9.15 we have

$$\phi^*_k = [\phi^*_{k0}, \phi^*_{k1}, \ldots, \phi^*_{k,M+E}]^T \tag{9.17}$$

$$= [w^0_{\text{safe}}, w^1_{\text{safe}}, \ldots, w^{M+E}_{\text{safe}}]^T \tag{9.18}$$

where $w^m_{\text{safe}}, m \in M \cup E$ are the distances that the obstacle needs to keep from other objects (robot links). In practice these distances may vary based on the shape of the links and obstacle, their velocities, etc. When we add new obstacles, we only need to resize the distance space, keeping the old plan for the existing vertices, and fill in $\phi^*_k$ to get a new plan $\Phi^*_{new}$. We can continuously add or remove vertices to the distance space during execution without the need to perform replanning.

### 9.2.2 *Complexity Analysis*

In this section we analyse the computational complexity of our approach. In our experiments we assume that the state is valid when the robot's pose is collision free and the end-effector gets closer to or is at the target position. The computation can be separated into two main steps: *construction phase* and *solving phase*. In the construction phase, we compute the current relative distance space state, $\Phi$, and the relative distance space Jacobian, $\mathbf{J}$. The desired state $\Phi^*$ is calculated once before execution, and it will get modified when new obstacles are detected, also, during construction phase.

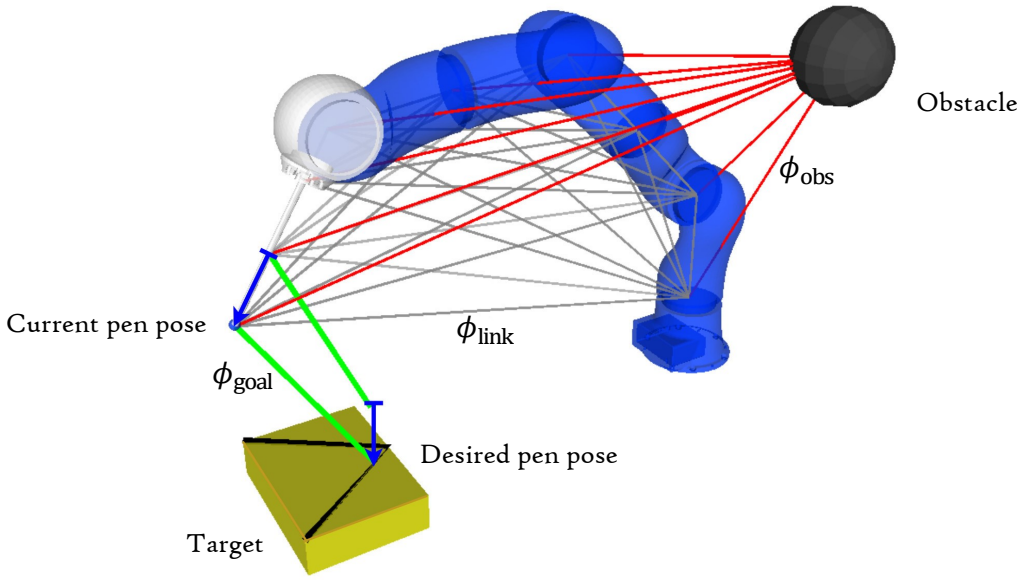



Figure 9.4: LWR mock-up welding experiment setup. The LWR robot arm is mounted with a laser pen, the task is to use the laser pen to weld along the target surface. We add an additional virtual point along the pen, and set its desired position to be above the real laser tip, such that the robot will keep the pen orthogonal to the surface. The lines represent the current relative distances, $\phi_{\text{link}}$ in grey, $\phi_{\text{obs}}$ in red, and $\phi_{\text{goal}}$ in green.

In solving phase, we solve operational space control problem using Equation 9.1 and 9.3. The order of complexity of the method is $O(\frac{1}{2}M(M+E))$ in the worst case where we consider edges between all robot links and all obstacles and targets. Furthermore, if we consider a reaching problem, without the pose re-targeting, we can omit the edges between the robot links entirely, which reduces the computational complexity to $O(ME)$.

We analyse the computational time of our method with different total number of edges $X = M + E$ on a reaching problem. The increase of the construction time is negligible compared to the increase of the solving time. An evaluation of maximum controlling speed with different $X$ is illustrated in Table 9.1. For example, $X = 10$ can be used for single arm ($M = 7$) robot in simple environment (1 target and 2 obstacles, i.e. $E = 3$). In contrast, $X = 20$ should be more than enough for single arm robot in most complex environment (e.g. KUKA LWR, Figure 9.4), $X = 30$ should be sufficient for a dual-arm upper body robot (e.g. Baxter robot, Figure 9.9), and $X = 50$ for humanoids. The result suggests that the proposed method can solve the

Table 9.1: Maximum controlling frequency with different space size $X = M + E$.

| | Evaluation scenarios | | | | | |
|---|---|---|---|---|---|---|
| Space Size ($X$) | 10 | 20 | 30 | 50 | 70 | 100 |
| Control Speed (Hz) | $750 \pm 50$ | $630 \pm 20$ | $600 \pm 30$ | $490 \pm 20$ | $350 \pm 10$ | $215 \pm 5$ |

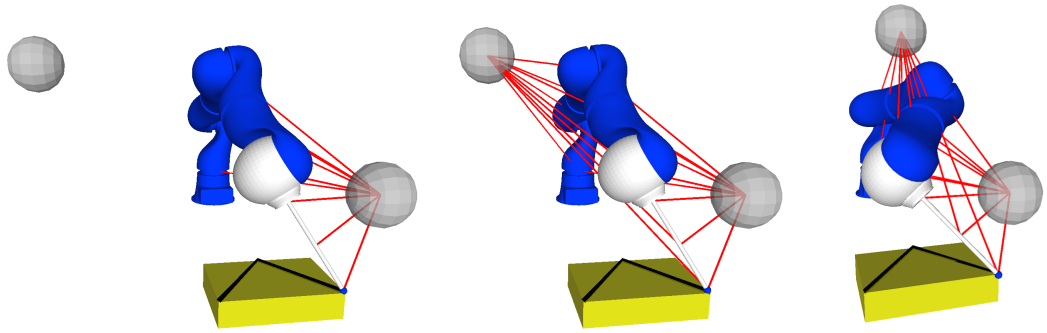


Figure 9.5: A new obstacle (unconnected one) is detected during execution, then new relative distances will be added into the original state when it gets close

adaptation problem in most common scenarios very efficiently. We have used a 3GHz Intel Core 2 Quad CPU.

## 9.3 EXPERIMENTS

We evaluate our approach with two different experiments. The first experiment uses a 7-DoF KUKA LWR robot arm to mock-up a dynamic welding task (Section 9.3.1), and the second experiment is a liquid pouring task in a close robot-human interaction scenario on a 14-DoF dual-arm Baxter robot (Section 9.3.2). The experiment setup will be detailed in each section accordingly. The tasks are implemented using the EXtensible Optimization Toolset (EXOTica Ivan et al. [2018]), which is a planning framework for solving robotics motion planning problems.

### 9.3.1 *LWR Mock-up Welding Task*

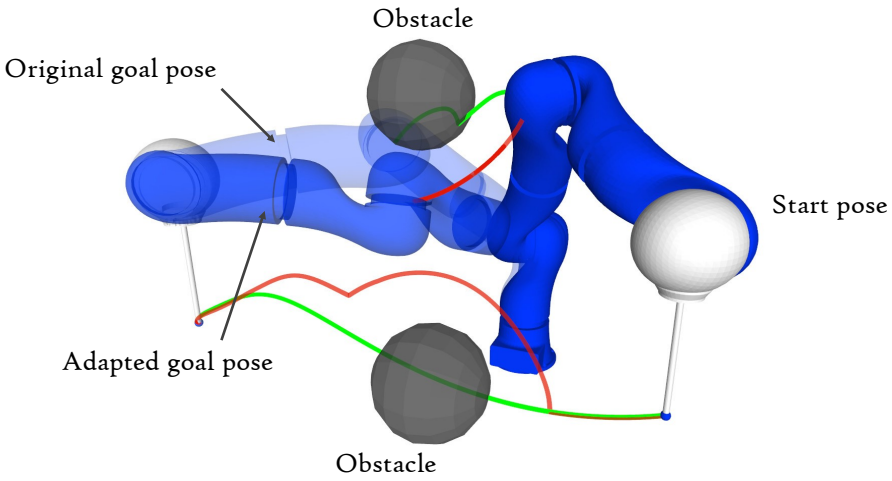



Figure 9.6: Example of trajectory adaptation, where the green lines are the original (end-effector and elbow) trajectories and the red ones are the adapted trajectories under multiple obstacles constraint.

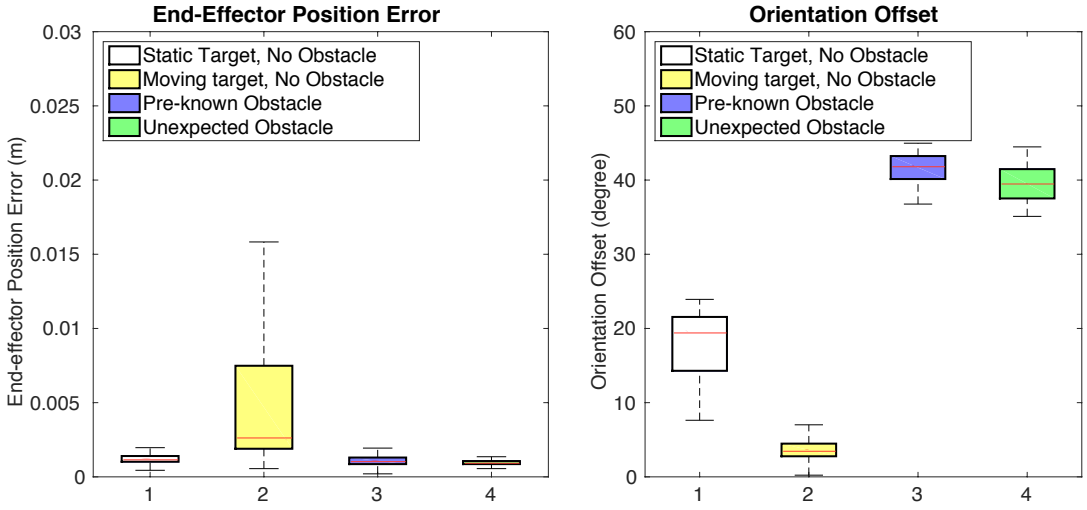



Figure 9.7: Position and orientation error analysis of LWR welding mock-up task. Left: End-effector position error; right: Laser pen orientation offset.

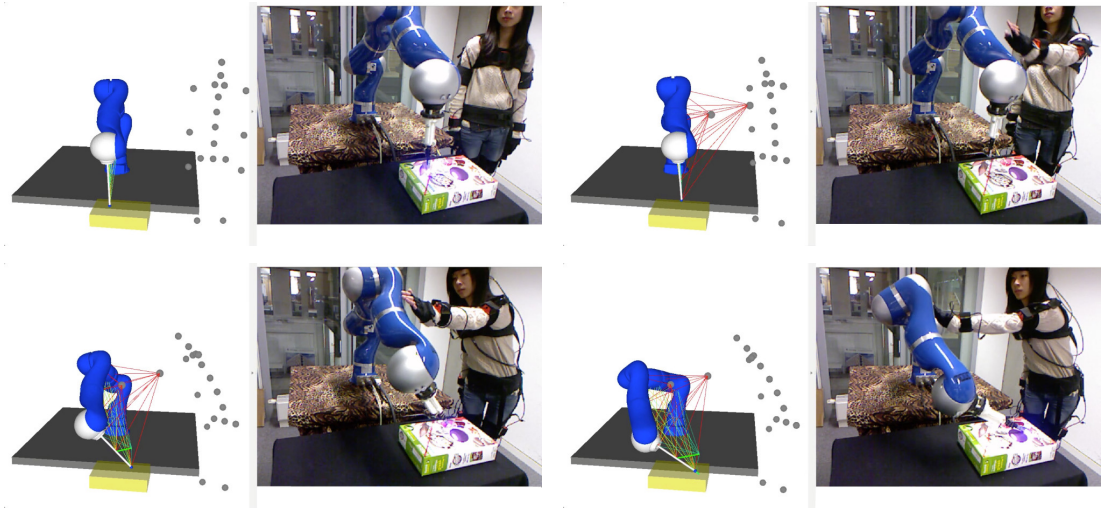


Figure 9.8: Experiment results on LWR robot hardware. Each figure contains two sub figures, where the right one is the real world environment, and left is the corresponding simulated environment.

In the first experiment we aim to show that an accurate manipulation task can be accomplished by only using relative distance based representation. The experiment setup is illustrated in Figure 9.4. In addition, we add an extra virtual end-effector along the physical end-effector's $z$ axis and a corresponding virtual target to keep the laser pen orthogonal to the target plane. We set different weighting factors for the laser tip, virtual point and obstacles ($w_{\text{tip}} > w_{\text{obstacle}} \gg w_{\text{virtual}}$), so that the pen will be kept orthogonal to the target if there is no other constraints. The orientation will be sacrificed to ensure physical end-effector position and collision free constraints in presence of obstacles. We use a real-time object pose recognition and tracking framework Pauwels et al. [2014] to detect and track the target.

The robot links' collision bodies are represented by a set of spheres with radius of 7cm and the safety threshold $w_{\text{safe}} = 5$ cm. The number of $\phi_{\text{obs}}$ can vary based on the number of obstacles. Since this is a reaching and avoiding problem, the robot pose constraint is not considered, i.e. $\phi^*_{\text{link}} = \phi_{\text{link}} = 0$.

We run the experiment with four different scenarios: 1) static target without obstacles, 2) moving target without obstacles, 3) static/moving target with dynamic obstacles present before planning started, and 4) static/moving target with unexpected obstacles present during execution. We record the laser tip position error and the pose offset over a same time duration across the four scenarios. The position error is

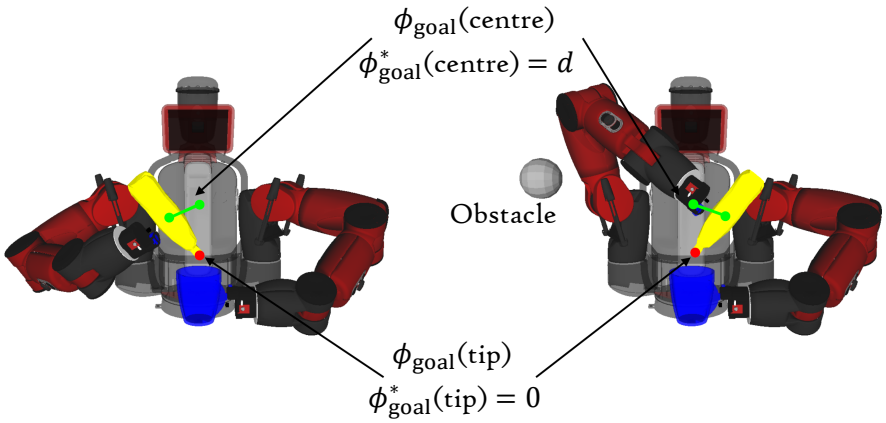



Figure 9.9: Baxter water pouring experiment. The robot holds a cup with one hand and a bottle with the other hand. The main task is to keep the bottle tip above the cup and avoid obstacles, the secondary task (lower weighted) is to keep a certain orientation between the bottle and the cup.

illustrated in Figure 9.7, where the $y$ axis is the Euclidean error between real laser tip position and desired ones. We can see that during all experiments, the errors of the laser tip are very small ($1.1 \pm 0.44$ mm). The error during the second scenario is larger due to the fast movement of the target. In the presence of obstacles, the orientation is sacrificed to ensure laser tip position and collision free constraints. Examples of adapted motion in simulation is shown in Figure 9.5. Note that the robot can adapt not only end-effector trajectory, but also the trajectories of all links. Figure 9.6 shows an example of adapting end-effector and elbow trajectories simultaneously.

In real world experiments, human subject's motion are tracked in real-time using XSENS motion tracking system. A set of obstacles are created to represent the human subject, as shown in Figure 9.8. Each figure consists two subfigures, left and right ones, where the right one is the snapshot of the real world environment and left one is the corresponding simulated environment. These results show that we can accomplish accurate manipulation tasks under dynamic obstacle constraints.

### 9.3.2 *Baxter Liquid Pouring Task*

In this experiment, we evaluate the scalability of our method on a 14-DoF Baxter robot (Figure 9.9). The robot holds a cup with one hand, keeps it horizontal and uses the other hand to grasp a bottle to simulate a water pouring task. We use one big

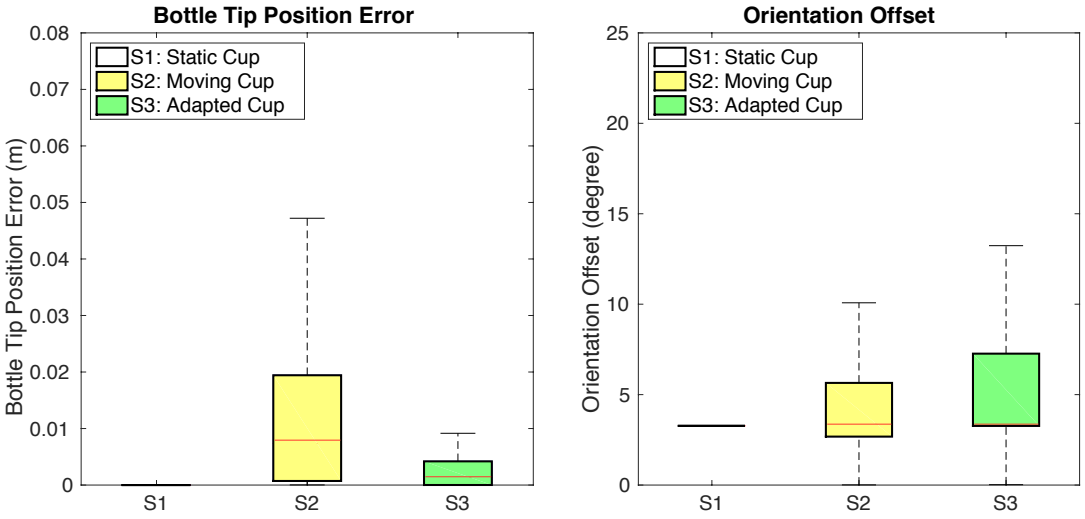



Figure 9.10: Position and orientation error analysis of Baxter water pulling task. Left: Bottle tip position error; right: Bottle orientation offset.

unified relative distance space state to encode the tasks for both hands as well as the possible collision avoidance constraints. Similar to the laser pen orientation in last experiment, here we add extra virtual point (centre of the bottle) to maintain the cup and bottle's orientation. Two goal distances are specified, i.e. $\phi^*_{\text{goal}}(tip) = 0$ and $\phi^*_{\text{goal}}(centre) = d_c$, where the first one is used to make sure the bottle's tip is at correct position and the later one is used to control the pouring angle. In practice, we set $d_c$ to zero, meaning that the bottle should be kept orthogonal to the cup.

The experiment consists of three different scenarios: 1) perturb the robot from the side on which the hand is holding the bottle; 2) perturb the robot from the side on which the hand is holding the cup, where the cup's desired position is not fixed, meaning the robot can shift both arms to avoid the human; and 3) move the cup randomly by moving the robot's hand. The task here is to keep the bottle tip directly above the cup and keeps the bottle as orthogonal as possible. The result in Figure 9.10 shows that in most cases the bottle can be placed in the correct place with an acceptable mismatch (0.1cm-1cm). The error in the third scenario is larger due to the fact that the robot can not follow the cup when it is moved by human with high velocity. Orientation offsets are similar across three scenarios, which suggests that the robot is able to "pour" the liquid into the cup with an acceptable pouring angle under dynamic obstacle constraint. Examples of adapted motions under each scenario are

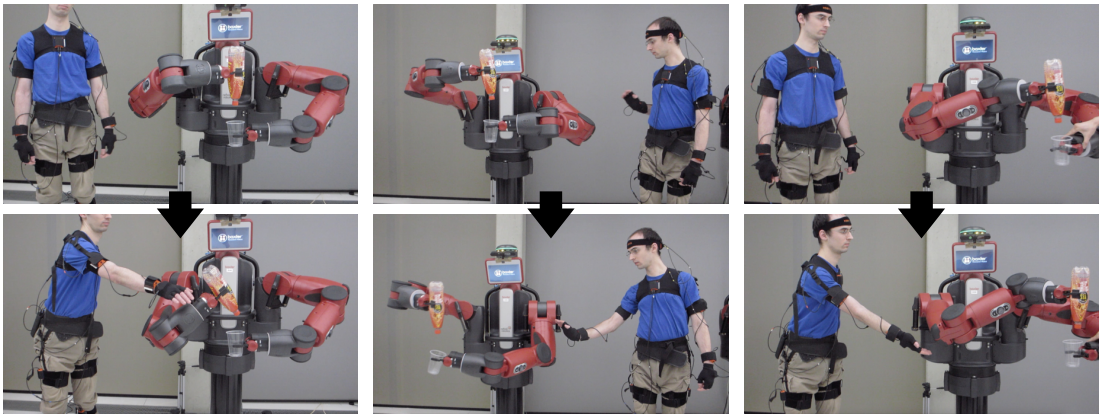


Figure 9.11: Experiment 2: Baxter robot water pouring task. In the first scenario (column 1), the human subject only disturb the robot from the left, the cup's position is fixed, the robot needs to adapt its right arm (with bottle) to fill the water while avoiding human; in the second scenario (column 2), the robot gets perturbed from the right, which means it needs to move its left arm (with the cup) to another collision free pose, and meanwhile the relationship between the bottle and cup needs to be maintained; in the third scenario (column 3), the cup's position is controlled by another human subject, the robot needs drive its right hand with the bottle to follow the cup while avoiding the human.

illustrated in Figure 9.11. A supplementary video of the experiments is available at https://youtu.be/A1dhiLyLo5U.

## 9.4 DISCUSSION

The experiments show that the proposed method can be used for solving accurate manipulation tasks in the presence of moving target and dynamic obstacles. The robot can avoid not only the existing obstacles, but also obstacles that are arbitrarily added into the scene during the execution.

The current approach has a few limitations, one of which is the local minima problem. Although relative distance space plan can adapt to environmental changes, the robot still fails to converge to the target in some situations, e.g. trapped in large non-convex obstacles, where a global replanning is required to find another valid plan. Fortunately, such bottle neck can be eliminated with an integration of the real-time motion planning approach (HDRM) described in Chapter 6.

# 10

## SUMMARY

Robots have evolved from the primary form such as the Edinburgh Freddy Robot[1] to advanced humanoid robot such as NASA Valkyrie[2] over the past several decades. However, the applications are still limited to industry and laboratory where the interactions with environment and people is highly constrained. The main challenge is for motion synthesis algorithms to generate motion trajectories that can be executed robustly and safely in cluttered and changing environments with close interactions with other robots and people.

In this thesis, we have tried to address the challenge in three phases as highlight in Figure 1.1, which are then detailed in three main parts: Part I end-pose planning, Part II motion planning and Part III motion adaptation. The three phases are interrelated, for instance, the output of end-pose planning, i.e. the goal state, is the input to motion planning; the output of motion planning, i.e. a trajectory, is the input to motion adaptation. In each part, we start with the preliminaries in the related area, followed by a series of contributions with an increasing level of complexity.

In Part I, we have summarised the preliminaries of collision-free inverse kinematics and reachability map in Chapter 2. In Chapter 3, we have proposed the novel forward and inverse dynamic reachability maps (DRM/IDRM) which enable real-time end-pose planning capability for humanoid robots in cluttered environments. As the original IDRM can only solve single-arm reaching problems on flat ground, we have further extended the DRM/IDRM method, in Chapter 4, allowing the humanoid robot to plan end-poses for bi-manual manipulation tasks on uneven terrain.

In Part II, we have generalised a brief but comprehensive overview of search-based motion planning algorithms in Chapter 5. In Chapter 6, a novel resolution complete

1 The Freddy and Freddy II Robots developed during the 1960s and 1970s at the University of Edinburgh. http://www.aiai.ed.ac.uk/project/freddy/

2 The NASA Valkyrie humanoid robot constructed by NASA-JSC in 2015 and delivered to the University of Edinburgh in Spring 2016. http://valkyrie.inf.ed.ac.uk/

motion planning algorithm is proposed, namely the Hierarchical Dynamic Roadmap (HDRM), that is capable of solving extremely complicated motion planning problems in real-time for fixed-base robotic arms. The complexity of motion planning problems is exacerbated with the type of robot base considered, whereas, in Chapter 7, we have proposed a generalised sampling-based planner for solving motion planning problems for humanoid robots with floating-bases. The method is able to efficiently plan smooth, balanced and collision-free motion for humanoid robots in complex environments.

In Part III, we have summarised the classical motion transfer techniques as well as state-of-the-art topological representations that have been applied in robotics in Chapter 8. While existing methods are inefficient dealing with unexpected perturbations, in Chapter 9, we have proposed a novel Distance Mesh representation that enables real-time motion adaptation and collision avoidance capability in unstructured and changing environments.

By utilizing the proposed end-pose planning, motion planning and motion adaptation techniques, we obtain a robotic framework that significantly improves the level of autonomy. The proposed methods have been validated on various state-of-the-art robot platforms, such as Universal Robot UR5, KUKA Light Weight Robot, Rethink Robotics Baxter, Clear Path Husky, NASA Valkyrie and many others, showing that our methods are truly applicable for solving practical problems.

# 11

# FUTURE DIRECTIONS

Despite the fact that significant improvement has been made in this thesis pushing forward the frontier of robot motion synthesis in clutter, we are still unable to fully deploy the robot unsupervised into real world applications. In this chapter, we state a list of open questions that have been made apparent due to this thesis.

## 11.1 MOTION SYNTHESIS WITH OPTIMIZATION AND SYSTEM DYNAMICS

So far, most of the work developed in this thesis considers only the robot's kinematics without taking into account the dynamics, e.g. velocity, acceleration and force. In most of the methods, the system dynamics are considered during a post-planning optimization step. Attempts had been made exploring techniques to include dynamics into some of the algorithms such as DMesh or HDRM. However, adding dynamics will double or even triple the state space's dimensionality making offline storage or online planning intractable with current commodity hardware.

It would be interesting trying to integrate dynamic properties into motion synthesis by developing more advanced configuration-workspace encoding techniques or new alternate space representations, while still managing to keep acceptable storage size and planning speed. This will allow the planners to generate not only valid, but more smooth and optimal trajectories. Similarly, the dataset or valid plan generated by IDRM or HDRM can also be used as warm start in optimization-based methods for faster convergence.

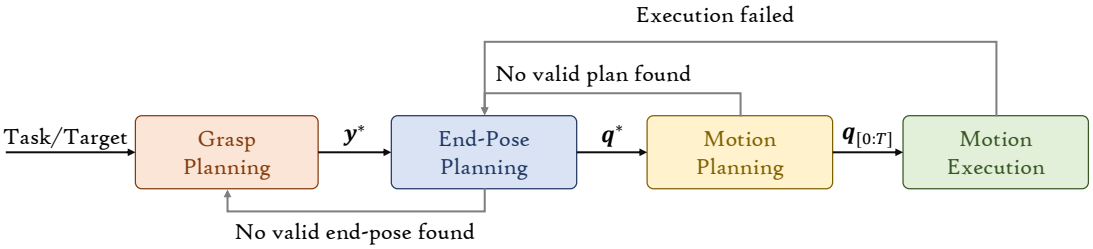



Figure 11.1: Illustration of closed-loop motion synthesis framework.

## 11.2 CLOSED-LOOP MOTION SYNTHESIS FRAMEWORK

In this thesis, we have addressed the motion synthesis problem separately (Equation 1.1 – 1.3) with a naïve combination as shown in Figure 3.7, whereas a high-level decision-making agent is missing, i.e., a system that can decide when and where to invoke end-pose planning, motion planning or motion adaptation, as shown in Figure 11.1. When a failure is caught at a certain phase, the system should be able to recall previous functions to recover from the failure. For example, after an end-pose has been found, if the footstep planner reports that the end-pose is not reachable, the system needs to automatically recall the end-pose planner to find another end-pose. Similarly, during execution time, the system needs to decide when to use motion adaptation methods for adapting local obstacles, and when to replan entirely if the original trajectory runs into local minima. With such an intelligent system, we can then deploy the robot into real world applications with decision making and failure recovery capabilities.

## 11.3 MOTION PLANNING IN CLUTTER USING DEEP LEARNING

Deep learning has drawn significant attention in the artificial intelligence community in the past few years (LeCun et al. [2015]), especially in Computer Vision and Natural Language Processing. Deep reinforcement learning has also shown promising result on controlling virtual agents and robots (Mnih et al. [2015], Lillicrap et al. [2015]). However, those approaches have rarely been validated on real robots, or only on very simple system in obstacle-free environments. To the best of the author's knowledge,

it has not been shown for deep learning approaches to plan and control robots to operate in complex environments as those shown in this thesis.

Lack of sufficient training data is normally the bottleneck of applying deep learning methods to high dimensional problems. Fortunately, with the proposed methods such as DRM and HDRM, we are able to generate a huge number of end-pose and motion planning problems with correct solutions, which can be used as training data. It should be possible, with enough training data, to train a deep reinforcement learning network to plan complex motion. We had already tried such approach, where a deep network can be trained for solving motion planning problems in simple environments, however, we were unable to generate collision-free trajectories in complex environments. We had only adopted some existing deep reinforcement learning methods, where customized and well designed robot specific learning methods might produce better results. Thus, last but by no means the least, the future work should include applying deep learning techniques for solving robot planning problems in cluttered and changing environments.

# BIBLIOGRAPHY


P. Abolghasemi, R. Rahmatizadeh, A. Behal, and L. Bölöni. Real-time placement of a wheelchair-mounted robotic arm. In *IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN)*, pages 1032–1037, 2016a. (Cited on page 16.)

P. Abolghasemi, R. Rahmatizadeh, A. Behal, and L. Bölöni. A real-time technique for positioning a wheelchair-mounted robotic arm for household manipulation tasks. In *AAAI workshop on artificial intelligence applied to assistive technologies and smart environments (ATSE)*, 2016b. (Cited on page 16.)

R. A. Al-Asqhar, T. Komura, and M. G. Choi. Relationship descriptors for interactive motion adaptation. In *ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 45–53, 2013. (Cited on pages 4 and 113.)

J. Bialkowski, S. Karaman, and E. Frazzoli. Massively parallelizing the RRT and the RRT. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3513–3518, 2011. (Cited on page 66.)

R. Bohlin and L. E. Kavraki. Path planning using lazy PRM. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 521–528, 2000. (Cited on page 66.)

F. Burget and M. Bennewitz. Stance selection for humanoid grasping tasks by inverse reachability maps. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5669–5674, 2015. (Cited on pages 19, 29, 33, and 35.)

S. R. Buss. Introduction to inverse kinematics with jacobian transpose, pseudoinverse and damped least squares methods. *IEEE Journal on Robotics and Automation (JRA)*, pages 1–19, 2004. (Cited on page 15.)

B. Coppin. *Artificial intelligence illuminated*. Jones & Bartlett Learning, 2004. (Cited on page 69.)

H. Dai, A. Valenzuela, and R. Tedrake. Whole-body motion planning with centroidal dynamics and full kinematics. In *IEEE International Conference on Humanoid Robots (Humanoids)*, pages 295–302, 2014. (Cited on pages 3 and 6.)

S. Dalibard, A. Nakhaei, F. Lamiraux, and J. P. Laumond. Whole-body task planning for a humanoid robot: a way to integrate collision avoidance. In *IEEE International Conference on Humanoid Robots (Humanoids)*, pages 355–360, 2009. (Cited on page 93.)

R. Deits and R. Tedrake. Footstep planning on uneven terrain with mixed-integer convex optimization. In *IEEE International Conference on Humanoid Robots (Humanoids)*, pages 279–286, 2014. (Cited on page 45.)

E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, pages 269–271, 1959. (Cited on page 60.)

A. D'Souza, S. Vijayakumar, and S. Schaal. Learning inverse kinematics. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 298–303, 2001. (Cited on page 15.)

M. Elbanhawi and M. Simic. Sampling-Based Robot Motion Planning: A Review. *IEEE Access*, pages 56–77, 2014. (Cited on pages 2, 7, 57, 65, and 92.)

P. Englert and M. Toussaint. Reactive phase and task space adaptation for robust motion execution. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 109–116, 2014. (Cited on page 112.)

P. E. Gill, W. Murray, and M. A. Saunders. SNOPT: An SQP Algorithm for Large-Scale Constrained Optimization. *Society for Industrial and Applied Mathematics (SIAM)*, pages 99–131, 2005. (Cited on page 95.)

Z. Y. Guo and T. C. Hsia. Joint trajectory generation for redundant robots in an environment with obstacle. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 157–162, 1990. (Cited on page 15.)

H. Halfar. General purpose inverse kinematics using lookup-tables. In *IEEE International Conference on Industrial Technology*, pages 69–75, 2013. (Cited on page 18.)

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*, pages 100–107, 1968. (Cited on page 61.)

K. Hauser, T. Bretl, K. Harada, and J. C. Latombe. Using motion primitives in probabilistic sample-based planning for humanoid robots. In *Algorithmic foundation of robotics*, pages 507–522. Springer, 2008. (Cited on pages 92 and 93.)

E. J. Van Henten, E. J. Schenk, L. G. van Willigenburg, J. Meuleman, and P. Barreiro. Collision-free inverse kinematics of the redundant seven-link manipulator used in a cucumber picking robot. *Biosystems Engineering*, 106(2):112–124, 2010. (Cited on page 3.)

E. S. L. Ho and T. Komura. Character motion synthesis by topology coordinates. In *Computer Graphics Forum*, pages 299–308, 2009. (Cited on pages 4 and 113.)

E. S. L. Ho and H. P. H. Shum. Motion adaptation for humanoid robots in constrained environments. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 3813–3818, 2013. (Cited on page 113.)

E. S. L. Ho, T. Komura, S. Ramamoorthy, and S. Vijayakumar. Controlling humanoid robots in topology coordinates. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 178–182, 2010a. (Cited on pages 5 and 113.)

E. S. L. Ho, T. Komura, and C. L. Tai. Spatial relationship preserving character motion adaptation. *ACM Transactions on Graphics (TOG)*, page 33, 2010b. (Cited on pages 113 and 118.)

E. S. L. Ho, H. Wang, and T. Komura. A Multi-resolution Approach for Adapting Close Character Interaction. In *ACM Proceedings of the 20th ACM Symposium on Virtual Reality Software and Technology*, pages 97–106, 2014. (Cited on pages 4 and 113.)

A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard. OctoMap: An Efficient Probabilistic 3D Mapping Framework Based on Octrees. *Autonomous Robots*, pages 189–206, 2013. (Cited on pages 88 and 105.)

M. Howard, S. Klanke, M. Gienger, C. Goerick, and S. Vijayakumar. A novel method for learning policies from variable constraint data. *Autonomous Robots*, pages 105–121, 2009. (Cited on page 91.)

D. Hsu and Z. Sun. Adaptively combining multiple sampling strategies for probabilistic roadmap planning. In *IEEE Conference on Robotics, Automation and Mechatronics*, pages 774–779, 2004. (Cited on page 66.)

D. Hsu, J. C. Latombe, and R. Motwani. Path planning in expansive configuration spaces. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 2719–2726, 1997. (Cited on pages 92 and 101.)

D. Hsu, R. Kindel, J. C. Latombe, and S. Rock. Randomized Kinodynamic Motion Planning with Moving Obstacles. *International Journal of Robotics Research (IJRR)*, pages 233 – 255, 2002. (Cited on page 65.)

V. Ivan, D. Zarubin, M. Toussaint, T. Komura, and S. Vijayakumar. Topology-based representations for motion planning and generalization in dynamic environments with interactions. *International Journal of Robotics Research (IJRR)*, pages 1151–1163, 2013. (Cited on pages 2, 5, 8, 57, 91, and 113.)

V. Ivan, Y. Yang, W. Merkt, M. Camilleri, and S. Vijayakumar. EXOTica: a library for easy creation of tools for optimisation and planning. In *Robot Operating System (ROS) Volume 3*. Springer, 2018. (Cited on pages 10, 101, and 126.)

K. B. Kaldestad, S. Haddadin, R. Belder, G. Hovland, and D. A. Anisi. Collision avoidance with potential fields based on parallel processing of 3D-point cloud data on the GPU. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 3250–3257, 2014. (Cited on page 112.)

M. Kallman and M. Mataric. Motion planning using dynamic roadmaps. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 4399–4404, 2004. (Cited on pages 67 and 68.)

F. Kanehiro, E. Yoshida, and K. Yokoi. Efficient reaching motion planning method for low-level autonomy of teleoperated humanoid robots. *Advanced Robotics*, pages 433–439, 2014. (Cited on page 93.)

S. Karaman and E. Frazzoli. Sampling-based algorithms for optimal motion planning. *International Journal of Robotics Research (IJRR)*, pages 846–894, 2011. (Cited on pages 65, 66, and 92.)

S. Karaman and E. Frazzoli. Sampling-based optimal motion planning for nonholonomic dynamical systems. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 5041–5047, 2013. (Cited on page 65.)

S. Karaman, M. R. Walter, A. Perez, E. Frazzoli, and S. Teller. Anytime Motion Planning using the RRT*. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 1478–1483, 2011. (Cited on pages 2, 4, and 111.)

L. E. Kavraki, P. Svestka, J. C. Latombe, and M. H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transaction Robotics and Automation (TRA)*, pages 566–580, 1996. (Cited on pages 2, 7, 57, 64, 92, and 101.)

S. M. Khansari-Zadeh and A. Billard. Realtime avoidance of fast moving objects: A dynamical system-based approach. In *Workshop on Robot Motion Planning: Online, Reactive, and in Real-Time at IROS]*, 2012. (Cited on pages 112 and 117.)

O. Khatib. Real-time obstacle avoidance for manipulators and mobile robots. In *IEEE Transaction Robotics and Automation (TRA)*, pages 500–505, 1985. (Cited on page 112.)

S. Kohlbrecher, A. Romay, A. Stumpf, A. Gupta, O. von Strykand F. Bacim, D. A. Bowman, A. Goinsand R. Balasubramanian, and D. C. Conner. Human-robot Teaming for Rescue Missions: Team ViGIR's Approach to the 2013 DARPA Robotics Challenge Trials. *Journal of Field Robotics (JFR)*, page 352–377, 2015. (Cited on page 16.)

T. Koolen, J. Smith, G. Thomas, S. Bertrand, J. Carff, N. Mertins, D. Stephen, P. Abeles, J. Englsberger, S. McCrory, J. van Egmond, M. Griffioen, M. Floyd, S. Kobus, N. Manor, S. Alsheikh, D. Duran, L. Bunch, E. Morphis, L. Colasanto, K. L. H.

Hoang, B. Layton, P. Neuhaus, M. Johnson, and J. Pratt. Summary of Team IHMC's virtual robotics challenge entry. In *IEEE International Conference on Humanoid Robots (Humanoids)*, pages 307–314, 2013. (Cited on pages 4 and 16.)

Y. Koren and J. Borenstein. Potential field methods and their inherent limitations for mobile robot navigation. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 1398–1404, 1991. (Cited on page 2.)

J. J. Kuffner. Effective sampling and distance metrics for 3D rigid body path planning. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 3993–3998, 2004. (Cited on page 24.)

J. J. Kuffner and S. M. LaValle. RRT-connect: An efficient approach to single-query path planning. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 995–1001, 2000. (Cited on pages 2, 57, 65, 92, and 101.)

J. J. Kuffner, S. Kagami, K. Nishiwaki, M. Inaba, and H. Inoue. Dynamically-Stable Motion Planning for Humanoid Robots. *Autonomous Robots*, page 105–118, 2002. (Cited on page 93.)

J. J. Kuffner, K. Nishiwaki, S. Kagami, M. Inaba, and H. Inoue. Motion planning for humanoid robots. In *Proceedings of the International Symposium of Robotics Research (ISRR)*, pages 365–374, 2005. (Cited on pages 3, 6, 92, and 93.)

J.C. Latombe. *Robot Motion Planning*. Springer, 1991. (Cited on page 72.)

S. M. Lavalle. Rapidly-Exploring Random Trees: A New Tool for Path Planning. Technical report, 1998. (Cited on pages 2, 7, 57, 64, 92, and 101.)

S. M. LaValle. *Planning algorithms*. Cambridge university press, 2006. (Cited on page 2.)

Steven M LaValle, Michael S Branicky, and Stephen R Lindemann. On the relationship between classical grid search and probabilistic roadmaps. *International Journal of Robotics Research (IJRR)*, 23(7-8):673–692, 2004. (Cited on page 72.)

Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, pages 436–444, 2015. (Cited on page 136.)

C. Y. Lee. An Algorithm for Path Connections and Its Applications. *IRE Transactions on Electronic Computers*, pages 346–365, 1961. (Cited on page 60.)

K. Leibrandt, C. Bergeles, and G. Z. Yang. On-line collision-free inverse kinematics with frictional active constraints for effective control of unstable concentric tube robots. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3797–3804, 2015. (Cited on page 15.)

D. Leidner, A. Dietrich, F. Schmidt, C. Borst, and A. Albu-Schäffer. Object-centered hybrid reasoning for whole-body mobile manipulation. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 1828–1835, 2014. (Cited on page 18.)

P. Leven and S. Hutchinson. A Framework for Real-time Path Planning in Changing Environments. *International Journal of Robotics Research (IJRR)*, pages 999–1030, 2002. (Cited on pages 67 and 69.)

T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. *CoRR*, abs/1509.02971, 2015. (Cited on page 136.)

H. C. Lin, M. Howard, and S. Vijayakumar. Learning null space projections. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 2613–2619, 2015. (Cited on page 91.)

H. Liu, X. Deng, H. Zha, and D. Ding. A Path Planner in Changing Environments by Using W-C Nodes Mapping Coupled with Lazy Edges Evaluation. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4078–4083, 2006. (Cited on page 88.)

Z. Mao and T. C. Hsia. Obstacle avoidance inverse kinematics solution of redundant robots by neural networks. *Robotica*, page 3–10, 1997. (Cited on page 15.)

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015. (Cited on page 136.)

A. S. Morris and A. Mansor. Finding the inverse kinematics of manipulator arm using artificial neural network with lookup table. *Robotica*, pages 617–625, 1997. (Cited on page 18.)

S. Murray, W. Floyd-Jones, Y. Qi, D. Sorin, and G. Konidaris. Robot motion planning on a chip. In *Robotics: Science and Systems (RSS)*, 2016. (Cited on pages 2, 23, and 69.)

J. Nakanishi, A. Radulescu, and S. Vijayakumar. Spatio-temporal optimization of multi-phase movements: Dealing with contacts and switching dynamics. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5100–5107, 2013. (Cited on page 91.)

J. Nasir, F. Islam, U. Malik, Y. Ayaz, O. Hasan, M. Khan, and M. S. Muhammad. RRT*-SMART: A Rapid Convergence Implementation of RRT*. *International Journal of Advanced Robotic Systems*, 10(7):299, 2013. (Cited on page 2.)

T. Nierhoff, S. Hirche, W. Takano, and Y. Nakamura. Full body motion adaption based on task-space distance meshes. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 1865–1870, 2014. (Cited on pages 113 and 114.)

P. Oh, K. Sohn, G. Jang, Y. Jun, and B. K. Cho. Technical Overview of Team DRC-Hubo@UNLV's Approach to the 2015 DARPA Robotics Challenge Finals. *Journal of Field Robotics (JFR)*, 2017. (Cited on pages 4 and 16.)

J. Pan and D. Manocha. GPU-based parallel collision detection for real-time motion planning. In *Algorithmic Foundations of Robotics IX*, pages 211–228. Springer, 2011. (Cited on pages 8 and 111.)

J. Pan and D. Manocha. GPU-based Parallel Collision Detection for Fast Motion Planning. *International Journal of Robotics Research (IJRR)*, pages 187–200, 2012. (Cited on page 66.)

J. Pan, S. Chitta, and D. Manocha. FCL: A general purpose library for collision and proximity queries. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 3859–3866, 2012. (Cited on page 84.)

C. Park, J. Pan, and D. Manocha. High-DOF Robots in Dynamic Environments Using Incremental Trajectory Optimization. *International Journal of Humanoid Robotics (IJHR)*, page 1441001, 2014. (Cited on pages 6 and 111.)

D. H. Park, H. Hoffmann, P. Pastor, and S. Schaal. Movement reproduction and obstacle avoidance with dynamic movement primitives and potential fields. In *IEEE International Conference on Humanoid Robots (Humanoids)*, pages 91–98, 2008. (Cited on page 112.)

K. Pauwels, V. Ivan, E. Ros, and S. Vijayakumar. Real-time object pose recognition and tracking with an imprecisely calibrated moving RGB-D camera. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2733–2740, 2014. (Cited on page 128.)

G. Pratt and J. Manzo. The DARPA Robotics Challenge [Competitions]. *IEEE Robotics and Automation Magazine (RA-M)*, 20(2):10–12, 2013. (Cited on page 31.)

N. Ratliff, M. Zucker, J. A. Bagnell, and S. Srinivasa. CHOMP: Gradient optimization techniques for efficient motion planning. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 489–494, 2009. (Cited on pages 2, 57, and 91.)

K. Rawlik, M. Toussaint, and S. Vijayakumar. On stochastic optimal control and reinforcement learning by approximate inference. *Robotics: Science and Systems (RSS)*, 13(2):3052–3056, 2012. (Cited on page 91.)

L. H. O. Rios and L. Chaimowicz. PNBA*: A Parallel Bidirectional Heuristic Search Algorithm. *XXXI Congressa da Sociedade Brasileira de Computaçao*, 2011. (Cited on page 82.)

A. Romay, S. Kohlbrecher, D. C. Conner, A. Stumpf, and O. von Stryk. Template-based manipulation in unstructured environments for supervised semi-autonomous humanoid robots. In *IEEE International Conference on Humanoid Robots (Humanoids)*, pages 979–986, 2014. (Cited on page 16.)

G. Sánchez and J. C. Latombe. A single-query bi-directional probabilistic roadmap planner with lazy collision checking. In *Robotics Research*, pages 403–417. Springer, 2003. (Cited on pages 66, 83, and 101.)

S. Schaal. Dynamic movement primitives-a framework for motor control in humans and humanoid robotics. In *Adaptive Motion of Animals and Machines*, pages 261–280. Springer, 2006. (Cited on page 112.)

J. Schulman, Y. Duan, J. Ho, A. X. Lee, I. Awwal, H. Bradlow, J. Pan, S. Patil, K. Y. Goldberg, and P. Abbeel. Motion planning with sequential convex optimization and convex collision checking. *International Journal of Robotics Research (IJRR)*, pages 1251–1270, 2014. (Cited on pages 2 and 57.)

H. Schumann-Olsen, M. Bakken, O. H. Holhjem, and P. Risholm. Parallel Dynamic Roadmaps for Real-Time Motion Planning in Complex Dynamic Scenes. In *Proceedings of the IEEE 3rd Workshop on Robots in Clutter*, 2014. (Cited on pages 69 and 82.)

L. Sciavicco and B. Siciliano. A solution algorithm to the inverse kinematic problem for redundant manipulators. *IEEE Journal on Robotics and Automation (JRA)*, pages 403–410, 1988. (Cited on page 15.)

B. Siciliano. Kinematic control of redundant robot manipulators: A tutorial. *Journal of Intelligent and Robotic Systems*, 3(3):201–212, 1990. (Cited on page 3.)

D. Spensieri, J. S. Carlson, R. Bohlin, J. Kressin, and J. Shi. Optimal Robot Placement for Tasks Execution. *Procedia CIRP*, pages 395–400, 2016. (Cited on page 16.)

I. A. Şucan and S. Chitta. Moveit! *Online at http://moveit.ros.org*, 2013. (Cited on page 22.)

I. A. Şucan and L. E. Kavraki. Kinodynamic motion planning by interior-exterior cell exploration. In *Algorithmic Foundation of Robotics*, pages 449–464. Springer, 2009. (Cited on pages 92 and 101.)

I. A. Şucan, M. Moll, and L. E. Kavraki. The Open Motion Planning Library. *IEEE Robotics and Automation Magazine (RA-M)*, pages 72 – 82, 2012. (Cited on pages 22, 84, and 101.)

R. Tedrake, M. Fallon, S. Karumanchi, S. Kuindersma, M. Antone, T. Schneider, T. Howard, M. Walter, H. Dai, R. Deits, M. Fleder, D. Fourie, R. Hammoud, S. Hemachandra, P. Ilardi, C. Pérez-D'Arpino, S. Pillai, A. Valenzuela, C. Cantu, C. Dolan, I. Evans, S. Jorgensen, J. Kristeller, J. A. Shah, K. Iagnemma, and S. Teller. A summary of team MIT's approach to the virtual robotics challenge. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 2087–2087, 2014. (Cited on pages 4 and 16.)

Russ Tedrake. Drake: A planning, control, and analysis toolbox for nonlinear dynamical systems, 2014. URL `http://drake.mit.edu`. (Cited on pages 22, 32, 45, and 93.)

L. G. Torres, C. Baykal, and R. Alterovitz. Interactive-rate motion planning for concentric tube robots. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 1915–1921, 2014. (Cited on page 15.)

N. Vahrenkamp, T. Asfour, and R. Dillmann. Robot placement based on reachability inversion. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 1970–1975, 2013. (Cited on pages 7, 19, and 35.)

A. Voelz and K. Graichen. Distance Metrics for Path Planning with Dynamic Roadmaps. In *Proceedings of the International Symposium on Robotics (ISR)*, pages 1–7, 2016. (Cited on pages 67 and 68.)

Y. Yang, V. Ivan, and S. Vijayakumar. Real-time motion adaptation using relative distance space representation. In *IEEE International Conference on Advanced Robotics (ICAR)*, pages 21–27, 2015. (Cited on pages 10 and 91.)

Y. Yang, V. Ivan, Z. Li, M. Fallon, and S. Vijayakumar. iDRM: Humanoid motion planning with realtime end-pose selection in complex environments. In *IEEE International Conference on Humanoid Robots (Humanoids)*, pages 271–278, 2016a. (Cited on page 10.)

Y. Yang, V. Ivan, W. Merkt, and S. Vijayakumar. Scaling Sampling-based Motion Planning to Humanoid Robots. In *IEEE International Conference on Robotics and Biomimetics (ROBIO)*, pages 1448–1454, 2016b. (Cited on pages 10, 45, and 57.)

Y. Yang, W. Merkt, H. Ferrolho, V. Ivan, and S.Vijayakumar. Efficient Humanoid Motion Planning on Uneven Terrain Using Paired Forward-Inverse Dynamic Reachability Maps. *IEEE Robotics and Automation Letters (RA-L)*, 2(4):2279–2286, 2017. (Cited on page 10.)

Y. Yang, W. Merkt, V. Ivan, Z. Li, and S.Vijayakumar. HDRM: A Resolution Complete Dynamic Roadmap for Realtime Motion Planning in Complex Environments. *IEEE Robotics and Automation Letters (RA-L)*, 3(1):551–558, 2018. (Cited on page 10.)

E. Yoshida. Humanoid motion planning using multi-level DOF exploitation based on randomized method. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3378–3383, 2005. (Cited on page 6.)

F. Zacharias, C. Borst, and G. Hirzinger. Capturing robot workspace structure: representing robot capabilities. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3229–3236, 2007. (Cited on page 17.)

F. Zacharias, C. Borst, and G. Hirzinger. The Capability Map: A Tool to Analyze Robot Arm Workspaces. *IEEE International Conference on Humanoid Robots (Humanoids)*, page 1350031, 2013. (Cited on pages 7, 15, 17, and 24.)