Automatic motion of manipulator using sampling based motion planning algorithms - application in service robotics

From the Faculty of Physics and Electrical Engineering
University of Bremen

For obtaining the academical degree of
Doktor-Ingenieur, Dr-Ing
by

M.Sc. Christos Fragkopoulos

Primary Supervisor:	Professor Dr-Ing Axel Gräser
Secondary Supervisor:	Professor Dr-Ing Kai Michels
Third Supervisor:	Professor Dr-Ing Udo Frese
Date of Submission:	31.05.2013
Date of Examination:	11.03.2014

Abstract

The thesis presents new approaches for autonomous motion execution of a robotic arm. The calculation of the motion is called motion planning and requires the computation of robot arm's path. The text covers the calculation of the path and several algorithms have been therefore implemented and tested in several real scenarios.

The work focuses on sampling based planners, which means that the path is created by connecting explicitly random generated points in the free space. The algorithms can be divided into three categories: those that are working in configuration space(C-Space)(C-Space is the set of all possible joint angles of a robotic arm) , the mixed approaches using both Cartesian and C-Space and those that are using only the Cartesian space. Although Cartesian space seems more appropriate, due to dimensionality, this work illustrates that the C-Space planners can achieve comparable or better results.

Initially an enhanced approach for efficient collision detection in C-Space, used by the planners, is presented. Afterwards the N dimensional cuboid region, notated as R_q, is defined. The R_q configures the C-Space so that the sampling is done close to a selected, called *center*, cell. The approach is enhanced by the decomposition of the Cartesian space into cells. A cell is selected appropriately if: (a) is closer to the target position and (b) lies inside the constraints. Inverse kinematics(IK) are applied to calculate a centre configuration used later by the R_q. The *CellBiRRT* is proposed and combines all the features. Continuously mixed approaches that do not require goal configuration or an analytic solution of IK are presented. R_q regions as well as Cells are also integrated in these approaches. A Cartesian sampling based planner using quaternions for linear interpolation is also proposed and tested.

The common feature of the so far algorithms is the feasibility which is normally against the optimality. Therefore an additional part of this work deals with the optimality of the path. An enhanced approach of CellBiRRT, called *CellBiRRT**, is developed and promises to compute shorter paths in a reasonable time. An on-line method using both CellBiRRT and CellBiRRT* is proposed where the path of the robot arm is improved and recalculated even if sudden changes in the environment are detected.

Benchmarking with the state of the art algorithms show the good performance of the proposed approaches. The good performance makes the algorithms suitable for real time applications. In this work several applications are described: Manipulative skills, an approach for an semi-autonomous control of the robot arm and a motion planning library. The motion planning library provides the necessary interface for easy use and further development of the motion planning algorithms. It can be used as the part connecting the manipulative skill designing and the motion of a robotic arm.

Kurzfassung

Diese Dissertation stellt neue Methoden zür autonomen Bewegung eines Roboterarmes vor. Die Bewegungsplanung erfordert die Berechnung des Weges des Roboterarmes. Diese Arbeit beffast sich mit der Berechnung des Pfades. Dafür wurden verschiedene Algorithmen implementiert und in realen Szenarien getestet.

Die Arbeit konzentriert sich auf die Probenahme basierte Planungen, das bedeutet, dass der Weg durch die Verbindung von zufällig generierten Punkte im freien Raum geschaffen wird. Die Algorithmen lassen sich in drei Kategorien unterteilen. Diejenigen, die nur im Konfigurationraum (C-Space , Konfigurationraum ist die Menge aller möglichen Gelenkwinkeln eines Roboterarms) arbeiten, die Gemischten, die sowohl im Kartesischen als auch im C-Space arbeiten und die Verfahren, die nur im kartesischen Raum arbeiten. Obwohl kartesischen Raum wegen der Dimensionalität besser geeignet zu sein scheint, zeigt diese Arbeit, dass die C-Space Plannungsalgorithmen vergleichbare und bessere Ergebnisse erzielen können.

Als erstes wird ein Verfahren zür effiziente Kollisionserkennung in C-Space vorgestellt, weil es in Algorithmen benutzt wird. Danach wird die N-dimensionale Quader Region, notiert mit R_q , definiert. Der R_q konfiguriert den C-Raum, so dass die Probenahme in der Nähe einer ausgewählten(genannt *center*) Zelle erfolgt. Der Algorithmus wird durch die Dekomposition des kartesischen Raumes in Zellen verbessert. Eine Zelle wird gewählt, wenn sie näher an der Zielposition ist und wenn sie innerhalb der physikalischen oder definierten Grenzen liegt. Inverse Kinematik (IK) wird angewendet, damit eine *center* Konfiguration, die später von der R_q verwendet wird, berechnet werden kann. Eine Vereinigung von allen bisherigen Funktionen ist mit dem *CellBiRRT* Algorithmus geschafft. Danach werden gemischte Methoden vorgestellt, die keine Ziel Konfiguration oder eine vorhandene analytische Lösung von IK haben. R_q Regionen sowie Zellen werden auch in gemischten Ansätze integriert. Eine kartesische Probenahme basiertes Methode mit Quaternionen für lineare Interpolierung wird vorgestellt und getestet.

Die Eigenschaft der bisherigen Algorithmen ist die Ausführbarkeit, die normalerweise gegen die Optimalität steht. Deshalb befasst sich ein zusätzlicher Teil dieser Arbeit mit der Optimalität des Pfades. Eine Erweiterung von CellBiRRT, genannt als *CellBiRRT**,ist entwickelt und es verspricht kürzere Wege innerhalb einer angemessenenen Zeit zu berechnen. Ein Online Verfahren, das sowie CellBiRRT als auch CellBiRRT* verwendet, wird präsentiert. Der Weg des Roboterarmes wird online verbessert und neu berechnet, auch wenn plötzliche Änderungen in der Umgebung erkannt werden.

Der Vergleich mit dem Stand der Technik beweisst die gute Leistung der vorgeschlagenen Verfahren. Die gute Leistung zeigt auch, dass die Algorithmen für Echtzeit Anwendungen geeignet sind. Verschiedene Anwendungen werden beschrieben: Manipulative Tätigkeiten, ein Algorithmus für eine halb-autonome Steuerung des Roboterarms und eine Software-Bibliothek für die Bewegungsplanungberechnung. Die Software-Bibliothek sorgt für die nötige Schnittstelle mit der einfachen Nutzung und der Weiterentwicklung der Bewegungplanungsalgorithmen. Es kann als Verbindungsteil zwischen der manipulativen Tätigkeiten und der Bewegung eines Roboterarmes verwendet werden.

Acknowledgements

I would like to thank Prof. Dr. Axel Gräser for his support, encouragement and comments throughout my work at the Institute of Automation in the University of Bremen. I would like to thank also Professor Kai Michels and Professor Udo Frese for being the second and third reviewer of my work. I would like also to thank Prof. Dr. Walter Anheier, Prof. Dr. Karl-Ludwig Krieger and Prof. Dr. Walter Lang.

I would like to mention Ahmeed Eldeep, Khizar Abbas for their contribution to this thesis. I also thank my colleagues Andrian Leu, Danijela Ristic and Saravanan Natarajan for their helpful comments during the reviewing of the work. Additionally I thank also the IAT colleagues with whom I have worked with : Sorin Grigorescu, Darko Ojdanic, Uwe Lange, Torsten and Stefan Heyer, Bashar Enjarini and Oliver Prenzel.

My deepest gratitude goes to my parents, my brother and sister, for their support especially at the beginning while I was student and later when I was researcher in the University of Bremen. I thank also my girlfriend Valia for her patience, understanding and help reviewing part of this thesis.

Contents

List of Figures

List of Tables

List of Algorithms

Glossary

ADL Activities of daily living

CBiRRT Constraint Bidirectional RRT - A bidirectional RRT approach meant to solve tasks with constraints

CellBiRRT Cell BiDirectional RRT - a bidirectional sampling based motion planning algorithm that uses N-Cuboid domains and cells in order to create efficiently areas for sampling. The algorithm seems to over performs the planners belonging to the same category like IKBiRRT and CBiRRT

DoF The degrees of freedom of the system

FRIEND Functional robot with dexterous arm and user-frIENdly interface for Disabled people - the robotic platform

GJK Gilbert-Johnson-Keerthi - Collision detection package for convex objects (ONLY)

IK Inverse Kinematics - It is the inverse kinematics algorithm using either analytical/geometrical or numerical approach

IKBiRRT Inverse Kinematics Bidirectional RRT - A bidirectional RRT approach where the goal is calculated using the Inverse kinematics. Can be used with Workspace Goal Regions

Jacobian An NxM matrix where M is the DoF and N is the location parameter of the robot arm

MASSiVE Multi-layer Architecture for Semi-autonomous Service robots with Verified task Execution. The software structure of the system FRIEND

OBB Oriented bounding Box - An approximation of a polyhedral using bounding box. The orientation of the bounding box is relative to the orientation of the object

OGR Object's grasping region - A region where the object can have. Used by the planner to calculate the necessary end effector frame

PRM Probabilistic Roadmap Planner - a multi query sampling based motion planning algorithm based. It constructs a graph (learning phase) and later attempts to find a solution from the graph(execution phase)

RRT Rapidly exploring Random Trees - a single query sampling based motion planning algorithm based on the exploration of trees

RRT* Rapidly exploring Random Trees star - a single query sampling based motion planning algorithm that asymptotically attempts to reach an optimal solution

SWIFT Speedy Walking via Improved Feature Testing - Collision detection package

WGR Workspace Goal Region - A six dimensional space where the end effector is able to move

WLN Weighted Least Norm

Chapter 1

Introduction

1.1 Robotics - What is it ?

This is a usual question that arises to someone that hears or reads the word "Robotics". A specific definition of this word is difficult to be given, however the field "Robotics" concerns mostly the study of the machines that can replace humans in the execution of a task. The replacement involves physical activities such as object manipulation as well as mental activities, like taking decision for humans.

The history of robotics begins hundreds of years ago, and not with the same machines like nowadays. The idea remained the same: the humans tried to use machines in order to ease their life, mostly trying to mimic the nature. History and especially Greek mythology includes examples of the first attempts of humans to build such a device (like Titan Prometheus,bronze slave Hephasteus). Mostly robots in the past ages were machines meant to be used in wars, mostly motivated by economic reasons. In these ages words like "automaton" was used since the term "robotics" is introduced much later by the Asimov (beginning of 1940s). Nowadays, the focus in the robotic research is more close to science-fiction: robots should not only serve the people, but they should be able to interact, learn and modify their environment. Although till now, it may be a very optimistic thought, however it may be not such a futuristic in twenty years.

From the early on 1940s till nowadays the research in "Robotic" is done having one specific law and target at the same time: *the robot should serve and obey to the humans.*

1.2 Please robot bring my meal, fill my glass with water, take a book I want to read....

As already said, the main law that the robot should obey is to serve the humans. Tasks like "*please prepare my meal*" or "*please bring my shoes*","*serve water*" should be done normally *automatically* by the robots. The word *automatic* is very trivial since it hides many other subtasks which can be divided into two main categories:

- environment recognition and *sensing*
- automatic planning

Regarding the first part,*sensing*, many devices have been developed that are able to provide necessary information for the robot. Sensors like tactile, force / torques , cameras, laser scanners and many other provide analog or digital signal that can be read and analyzed by the robot. The automatic planning task computes the robot motion around obstacles. A motion is done in combination with sensors, providing flexibility and autonomous behavior for the robot. Using all these before, tasks like "*serve my meal*" or "*grasp and open a book*" may be possible to be executed by the robot. But are the robots now capable of doing that?

1.3 Service Robots

Service robots are mostly developed in order to answer the above questions. The last decades, however, the research was focused mainly on industrial and military robots, leaving the sector of "service" robots in a second place. Nevertheless, nowadays the future of service robots seems to be very promising, since their technologies are interested by not only military sectors. For instance service robots should overcome difficulties like autonomous maneuvering, recognition and manipulation of unknown objects. That is one reason why during the last decade the market of service robots is increased. Moreover, the number of aged people arises also, which subsequently increases the interest of developing "clever" robotic solution capable of helping people to accomplish for instance household activities. The average living age of humans is increased, and that grows the number of elderly as well as the needs for serving the corresponding people. Due to all the above reasons, there is an increased intention to use service robots in the field of rehabilitation robotics, where the target is to support not only elderly people, but also humans with disabilities.

1.4 The goal of the thesis and the proposed approach

The main goal remains the ability of a system to execute a task and to compile it automatically into a set of low level motions. Regarding the fact that normally the objects are not static, meaning that the surrounding environment changes, the existence of algorithm that calculates the low-level motions is obligatory. That is done by a motion planning algorithm. The robot must recognize its environment, locate the objects of interests, and manipulate the objects in respect to the application. Such as applications are handling, grasping and placing of objects. All of the above show the necessity of existence of an efficient motion planning algorithm.

Figure 1.1 illustrates briefly the concept of autonomous motion planning. The command/task is given to the system by selecting an appropriate scenario. The scenario, which is the higher level, consists of many sub-tasks. The high level command executes the sub-tasks which consequently contains the necessary goals for the motion planner. The motion planning task is given via information like start pose,goal pose/location or multiple goal locations. Normally the goal for

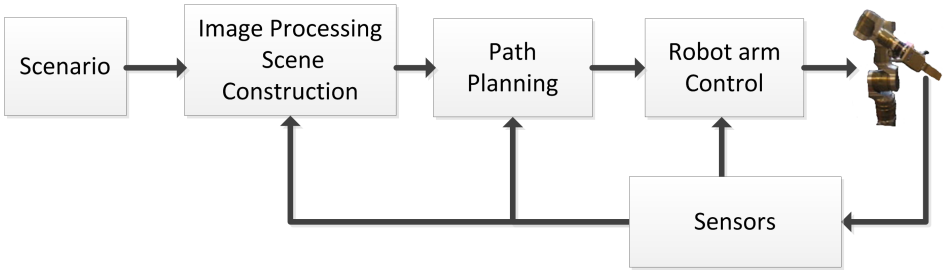


Figure 1.1: Robot Control

a manipulator (robot arm) is a location or frame indicating the position and orientation of the end-effector.

The environment for each motion planning algorithm is constructed previously by the sensors, like stereo camera. That means that when the robot starts planning its motion, the environment is assumed that does not change. The algorithms used to detect and locate the objects by the vision system are not considered in this work. It is assumed that the environment is constructed well with very high accuracy.

In this work several motion planning algorithms are going to be described and analyzed. The developed algorithms are sampling based approaches. The algorithms are designed to work for robotic arms. The common attribute of all of them is that they create random connected graphs, called trees, in configuration or cartesian space. In this work a comparison of using Cartesian and Configuration space is presented. In addition benefits and drawbacks of each one are described.

Some important issues, discussed in this thesis, are the flexibility and the practical implementation of each approach. By flexibility is considered the ability of finding solutions fast without changing the algorithm's parameter. By practical implementation is considered the usage of the algorithms in real scenarios. Feasibility concerns the ability to compute a solution in a deliverable time. All of these topics are going to be discussed in later chapters.

1.5 Organization of the thesis

This thesis is organized into several chapters. Each one deals with a specific subject. Each chapter has its own experimental results, discussions and implementation issues so that the reader would be able to understand the contents of the text. The chapters are organized so that each one needs the contents of the previous chapters. The aim of the text is to keep the reader to a continuous interest.

The first three chapters contain the introductory material that is necessary for further understanding of the text. The rest of the text describes the developed approaches. The chapter four describes the collision detection approaches and the chapter five describes a configuration space planner called CellBiRRT. The chapter six includes some mixed sampling based approaches. Benchmarking with the state of the art planners is done in chapter seven. The chapter eight presents an enhancement of the CellBiRRT, called CellBiRRT*, that is used to compute shorter paths. The

chapter nine describes some practical applications, manipulative skills for real scenarios as well as a developed motion planning library that contains all the developed algorithms. Conclusions are at the end of the thesis.

1.6 Contribution of the thesis

The contributions presented in this thesis are the following:

- Development of a dynamic efficient collision detection approach. The line between two configurations is sampled and checked efficiently for collisions. The approach estimates the maximum traveled curve of the end effector path of the robot, and reduces the samples that have to be checked. The improvement over the state of the art is (a) the use of the OBBs (Oriented Bounding Boxes) for the calculating near and far obstacles and (b) the reduction of the samples that have to be checked for collision by calculating the moment when an object shall be taken into account for collision or not. Chapter 4 explains in detail the method.

- Development of a new sampling based motion planning algorithm named *CellBiRRT* (**C**ell **Bi**directional-Rapidly exploring Random Tree). The CellBiRRT has the following main contributions: (a) the N-Dimensional cuboid regions applied for generating random configurations (see figure 1.2) using the last expanded configuration and (b) the combination of the Cartesian and configuration space using Cells in cartesian space in order to place the N-dimensional cuboid region in a probably better position. The CellBiRRT uses also the characteristics of the RRT described in the chapter 3.

  Shortly the steps of the CellBiRRT, which is a bidirectional approach, are: (a) Trial to connect to the goal configuration with a (normally) small probability (b) Creation of random configuration (c) Attempt to connect from the nearest neighbor to the random configuration (d) Attempt to connect to the opposite tree.

  The step (b) is very important factor for the total performance of a probabilistic planner. That is the main contribution of the CellBiRRT over the state of the art motion planning. The CellBiRRT subdivides initially the workspace(only the position of the end effector and not the orientation) of the robot arm into cells. Each cell has a center position(x,y,z). In CellBiRRT algorithm the step (b) is done as follows:

  (a) CellBiRRT searches the closest configuration from the last expanded configuration (q_a) to the opposite tree. The closes configuration is called q_b

  (b) Identifies the cell, using forward kinematics, where the q_a belongs to.

  (c) Explores the neighbor cells, removes those that are in collision, and selects the cell $cell_{sel}$ that is closer to the position(x,y,z) of the q_b. In order to do that the forward kinematics for the q_b shall be applied.

  (d) The selected cell is used in order to compute a configuration (q_c) which is close to the q_b. The computation is done using inverse kinematics (IK). The figure 1.3 shows that procedure.

(e) Around the q_c is generated an N-dimensional cuboid, where N is the degrees of freedom of the system. The q_c is the center of the N-dimensional cuboid. Inside this area is generated a random configuration. The figure 1.2 shows the N-dimensional cuboid.

(f) If the (c) or (d) are not successful, the q_a is used as q_c for the step (e).

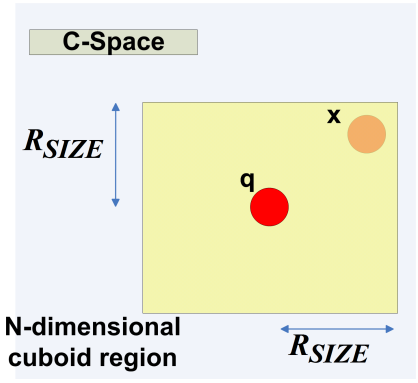
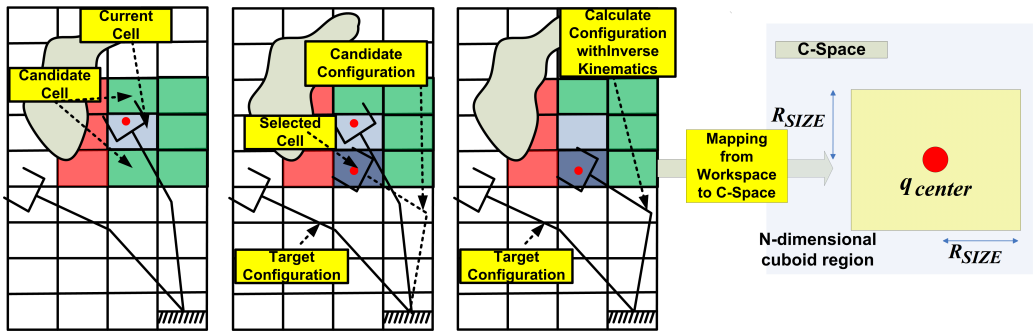


Figure 1.2: N-dimensional cuboid region around a configuration q. The random configuration x is generated inside the N-dimensional cuboid region. The figure shows an example for a 2D cuboid region.



(a) Selecting a cell from the neighbor cells

(b) Selecting a configuration

(c) Generating random configuration around this configuration

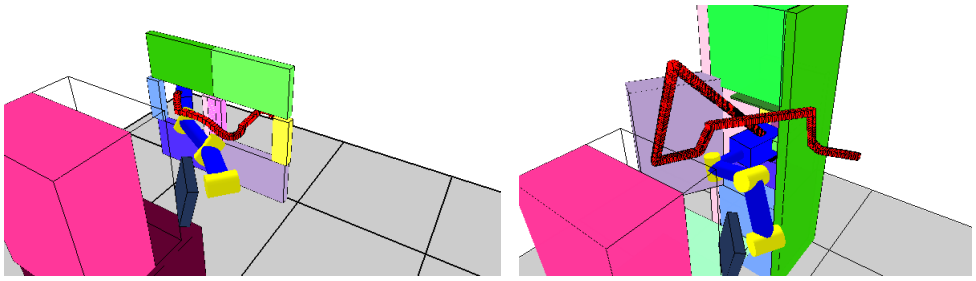
Figure 1.3: Steps (c) till (e) of the CellBiRRT

Experimental results are presented for a seven degrees of freedom robot arm.

- Developing of sampling based algorithms without the requirement of goal configuration which sometimes may not be available or possible to be computed. In contrast to the CellBiRRT these algorithms do not require a goal configuration. They are forward directional RRT and they use jacobian or analytical inverse kinematics algorithms as well as the N-dimensional cuboid regions that are presented on the *CellBiRRT*. That group of planners attempts incrementally to reach the goal location. The planners combine both cartesian and configuration

space. The contribution of this thesis over the state of the art are: (a) the development of motion planning algorithms using the weighted least norm, which takes into account the joint limits of the manipulator($RRT - J_{wln}$).(b) The development of a forward directional RRT using analytical inverse kinematics instead of jacobian iterative approach in order to connect a configuration with the target location. (c) Another contribution of this thesis is the development of a cartesian RRT planner, a planner that connects configurations completely in the workspace (cartesian space). The calculated path from this planner which is called CartesianRRT is straight line segments in the Cartesian space.

The figure 1.4 shows some examples of paths created by this group of planner.



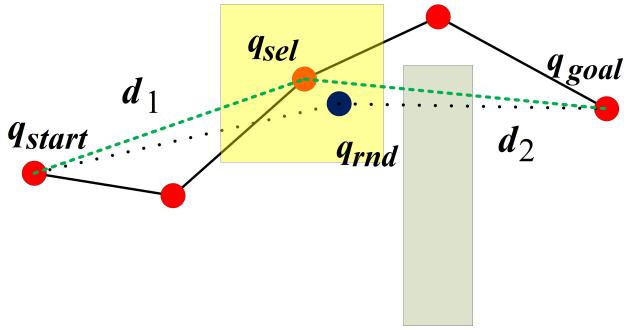
(a) $RRT - J_{wln}$ with smoothing (b) Cartesian RRT without smoothing

Figure 1.4: Example of resulted paths

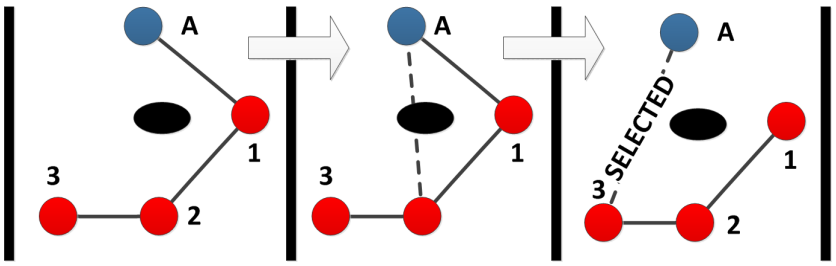
- Benchmarking between some of the state of the art and the proposed motion planning algorithms. The thesis provides a comparison between the proposed approaches and some state of the art motion planners which are dedicated for manipulators. The latter is also a contribution of this thesis, since a comparison between motion planning algorithms designed for manipulators was not documented. The details of the benchmarking are explained in the chapter 7.

- Developing of anytime CellBiRRT algorithm, called *CellBiRRT**, a planner that can provide lower cost paths. The cost is related to the length of the resulted path. The CellBiRRT* uses all the characteristics of the RRT, CellBiRRT and the RRT*. The contribution of this thesis is: (a) the usage of an additional pruning procedure, in order to improve the performance and to lower the final cost of the path (b) the usage of the N-Cuboid regions in order to provide good configuration candidates. If a path exists already, the random configuration may be generated with the N-Cuboid regions around a point of the path. The random configuration should have an estimated cost surely less or equal to the present cost (c) the algorithm ensures that each calculated path has a lower cost than the previous one and (d) the integration of the CellBiRRT* into the FRIEND system and the execution of the CellBiRRT* while the robot arm is moving in order to improve online the path. The details are explained in the chapter 8.

  The figure 1.5 presents some of the CellBiRRT* contributions.



(a) N-Cuboid regions used for generating efficiently random configurations



(b) Pruning procedure improves the quality of the path

Figure 1.5: Procedures in CellBiRRT*

- Extension of the planners in order to use multiple goal regions. Multiple goal regions have been used already in the literature. The implementation of multiple goal regions in the presented motion planners as well as the definition of the object goal region are also contributions of the thesis.

- Integration of all algorithms to the MASSiVE platform and realization of different manipulative skills and share control in the rehabilitation robotic system FRIEND. Several manipulative skills have been developed in order to realize the following scenarios: ADL (activities of daily life) and the ReIntegraRob scenario. The scenarios are described in chapter 9. The contribution of this thesis is the implementation of the manipulative skills as well as an approach of share control which is going to help the user to accomplish faster and manually scenarios. The scenarios are new and therefore the manipulative skills are also new.

- Development of a new motion planning library designed especially for manipulators. The motion planning library uses interfaces. The interfaces help the developer to implement its own planners or to use the implemented planners. In this thesis several examples on different robot arms illustrate the advantage of using the presented library. Chapter 10 describes the design and the examples of this motion library.

Chapter 2

Technical and theoretical background

This chapter presents technical and theoretical background which is important for later understanding. In this chapter the rehabilitation system FRIEND is going to be described, transformation matrices, DenavitHartenberg parameter needed to calculate forward kinematics for a manipulator, quaternions, calculation of minimum distances, and inverse kinematics are going to be described[Pau81, Cra05, SSVG09].

2.1 Rehabilitation system FRIEND

Life expectancy around the world has been increased over the last decades. The number of people that need support in daily life is increased compared to previous decades. As the number of the population increases the field of rehabilitation has become a challenge. Statistical results in United States of America , Europe, Canada and Japan show that the number of elderly together with the number of people having disability is estimated to be around 200 million[GLFG11]. All the above are the reasons why robotics society has increased its interest in the field of rehabilitation/care robotics[dLR08]. Robotic technology has been developed rapidly in the last decades which enables the opportunity to people with disabilities to take part in daily life.

In literature the field of rehabilitation robotics is divided into two main categories: *therapy* and *assistive* robots[dLR08]. Although therapy robots are very important and the field has really developed in the last years, the thesis is dedicated to assistive robots since the rehabilitation robotic system FRIEND(**F**unctional **R**obot with dexterous arm and user-fr**IEN**dly interface for **D**isabled people),figure 2.1, belongs to this category. The focus of FRIEND system is the autonomous manipulation. The aim is the user to give a simple command through a *human machine interface* and the robot to fulfill autonomously human's demand. In this field of research other institutes and companies have developed their own assistive robots. For instance Willow Garage in United States are building the PR2 robot[CGCG10], Fraunhofer institute in Germany are developed the Care-O-Bot robotic system[RCF+09]. These two projects combine manipulation and mobile platform in order to help mostly elderly but not severe disable people. Few system are developed in order to support disable people for instance in ADL (*Activities of daily living*) scenarios. Such systems

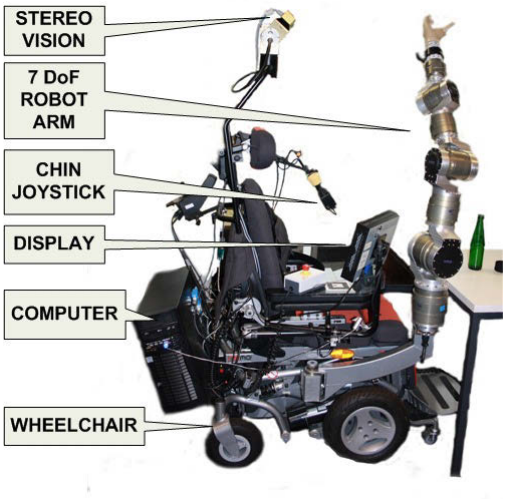


Figure 2.1: Rehabilitation robotic system FRIEND and its equipment

are *MANUS*[WGW+00], *Handy 1*[Top01] and the rehabilitation robotic platform KARES(KAIST Rehabilitation Engineering Service System)[BCC+04]. Most comparable to FRIEND system is MANUS. A 6 degrees of freedom (DoF) robotic arm (MANUS) mounted to wheelchair together with the complexity of its usage (with joystick) remain the main disadvantages of the system. FRIEND is equipped with a dexterous manipulator, idea that is not used in other projects combining wheelchair mounted manipulator. FRIEND's as well as PR2's and Care–O–Bot's manipulator has seven DoF robotic arm and that enables the possibility to execute more complicated tasks compared to a manipulator with less DoF (like in MANUS).

The history of FRIEND system begins from 1997 when the first version was developed by the Institute of Automation (IAT) of the University of Bremen. This version named as FRIEND I[MRL+01] and had two cameras,a wheelchair, computer system and a six DoF robotic arm mounted. This system had few autonomous possibilities and for that reason the FRIEND II system was build by the IAT in 2003[PMC+07]. The main difference was the replacement of the six DoF robot arm with a new seven DoF arm. That helped the FRIEND system to accomplish autonomously complicated tasks like *pour in* and *grasp bottle*. That was the first serious attempt to put a complicated system close to the market. The eliminations of FRIEND I and FRIEND II tries to vanish the new FRIEND III (or simply FRIEND) system (figure 2.1). FRIEND system is equipped with a seven DoF robot arm with an analogy of weight:payload almost 2:1,a wheelchair, an intelligent tray with infrarot sensors, a stereo vision camera and a Time of Light (ToF) camera. The aim of the system is to sustain for 1.5 hours independently giving the possibility to the user to fulfill ADL activities or even to go back to work. The autonomous behavior remain the main challenge of the system. The user should do as less work as possible and the responsibility goes mainly to the system.

The software architecture of the system provides the possibility for different software and hardware modules to cooperate together[MPFG06]. It is a multilayer structure, starting like a pyramid

from the top level called abstract and going to the lower level called *SkillLayer* and *HardwareLayer*. The manipulative skills belong to *SkillLayer* and are responsible for the motion of the robot arm. The motion planning module, which has the planning algorithms, is used by the manipulative skills and calculates the necessary trajectories. That part of software is going to be explained in this thesis. Several algorithm are developed and joined in the module of manipulation planning. Each method has advantages and disadvantages that are explained in later chapters. For better understanding of the rest of the text, there are two chapter explaining the theoretical background. This chapter has a briefly mathematical introduction to robotics and the third chapter an introduction to planning algorithms.

2.2 Rigid body transformations

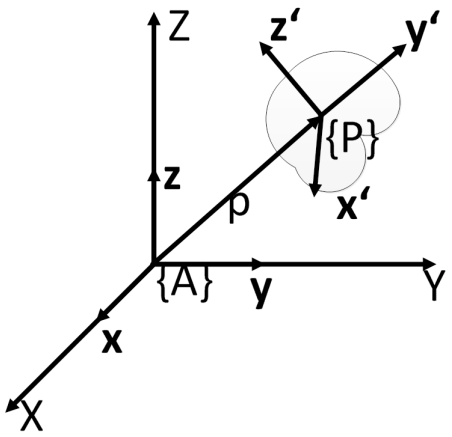


Figure 2.2: Origin and object's frame

A rigid body pose can be described completely by having its position and orientation with respect to an origin/reference frame. In figure 2.2, let {A} be the origin orthogonal frame and **x**,**y**and **z** the unit vectors of the origin frame axis.

Let **p** denotes the vector that goes from the origin to the point P of the center of the object. The vector **p** based on the reference frame {A} is given by the equation:

$$\mathbf{p} = p_x \cdot \mathbf{x} + p_y \cdot \mathbf{y} + p_z \cdot \mathbf{z}; \tag{2.1}$$

where p_i corresponds to the corresponding coordinate of the vector. The equation 2.1 can be written also in a matrix form:

$$\mathbf{p}^A = \begin{bmatrix} p_x & p_y & p_z \end{bmatrix} \cdot \begin{bmatrix} \mathbf{x} \\ \mathbf{y} \\ \mathbf{z} \end{bmatrix} \tag{2.2}$$

This equation describes the position of the coordinate system {P} based on the reference coordinate system {A}. The second important parameter is to define the orientation of the system {P} in

respect to the origin one. In simple words we have to define a formula in order to calculate the rotation of the frame $\{P\}$ in respect to the frame $\{A\}$.

Using the same approach like previously, the rotation of the rigid body can be described in respect to the $\{A\}$ as a 3x3 matrix:

$$R_P^A = \begin{bmatrix} \mathbf{x}_P^A & \mathbf{y}_P^A & \mathbf{z}_P^A \end{bmatrix} \tag{2.3}$$

where the symbol R_P^A is interpreted as follows: The rotation matrix of the $\{P\}$ based on the $\{A\}$.

In the bibliography homogeneous transformations are used. That allow us to represent affine transformations by a matrix (Affine transformation is a combination of single transformations such as translation or rotation). Homogeneous coordinates embed three-dimensional space R^3 into the P^3, the three-dimensional projective space, which is R^4. As a result, inversions or combinations of linear transformations are simplified to inversion or multiplication of the corresponding matrices.

Using homogeneous transformation, a three dimensional point(x^*,y^*,z^*) can be written using now four coordinates and represented using a matrix:

$$\begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} \tag{2.4}$$

where,

$$x^* = \frac{x}{w}, y^* = \frac{y}{w}, z^* = \frac{z}{w} \tag{2.5}$$

where w is the weighted factor (different from zero). Using the previous example, a vector $\mathbf{r}$ can be written based on the frame $\{A\}$ as follows:

$$\mathbf{r}^A = \mathbf{r}_P^A + R_P^A \cdot \mathbf{r}^P \tag{2.6}$$

This equation using homogeneous coordinates can be written as follows:

$$\begin{bmatrix} \mathbf{r}^A \\ 1 \end{bmatrix} = \begin{bmatrix} R_P^A & \mathbf{r}_P^A \\ 0_{1x3} & 1 \end{bmatrix} \cdot \begin{bmatrix} \mathbf{r}^P \\ 1 \end{bmatrix} \tag{2.7}$$

A transformation matrix is then defined as:

$$T_2^1 = \begin{bmatrix} & & x_2^1 \\ & (R_2^1)_{3x3} & y_2^1 \\ & & z_2^1 \\ 0_{1x3} & & 1 \end{bmatrix} \tag{2.8}$$

The figure 2.3 illustrates an example of a translation pp'' between frames $\{P\}$ and $\{P''\}$. The new frame is located on the position:

$$\mathbf{p'} = \mathbf{p} + \mathbf{pp''} \tag{2.9}$$

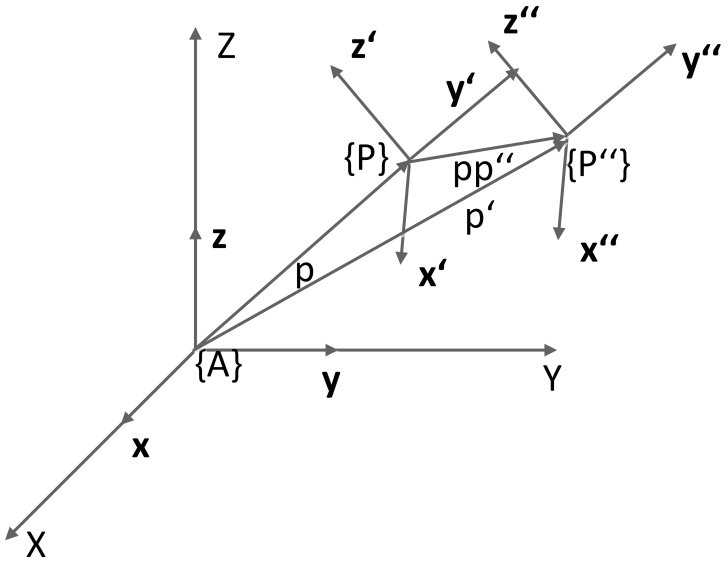


Figure 2.3: Example with translation frame

The coordinates of the **p**'(x'^A, y'^A, z'^A) can be calculated using the formula:

$$\begin{bmatrix} x'^A \\ y'^A \\ z'^A \\ 1 \end{bmatrix} = T_P^A \cdot \begin{bmatrix} pp_x''^P \\ pp_y''^P \\ pp_z''^P \\ 1 \end{bmatrix} \tag{2.10}$$

If transformation frames are used instead of points, the transformation $T_{P''}^P$ can include in general some rotations, indicating that the frame $\{P''\}$ has an additional rotation based on the frame $\{P\}$. The equation: $T_{P''}^A = T_P^A \cdot T_{P''}^P$ holds. That characteristic of the frames is going to be used in the following chapter where the forward kinematics procedure for a manipulator is going to be discussed.

2.3 Manipulator's end-effector pose and orientation - Forward Kinematics

As already mentioned the multiplication of frames implies a transformation in coordinate systems. That is very important if there is a kinematic chain like a manipulator. The image 2.4 illustrates an example in 2D for a 3 link planar manipulator. Obeying to the multiplication rule for frames, the end-effector frame(coordinate system $\{3\}$) is equal to:

$$T_3^0 = T_1^0 \cdot T_2^1 \cdot T_3^2 \tag{2.11}$$

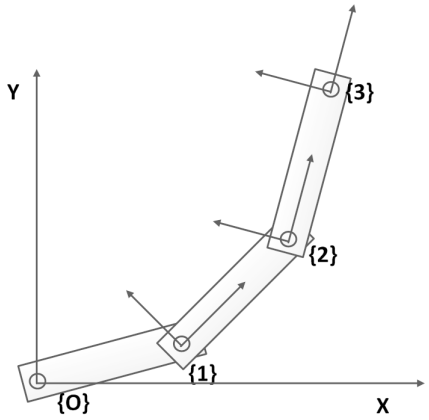


Figure 2.4: Kinematic chain for a 3 link planar manipulator

In general having N -joint robotic arm and {0} the reference coordinate system, the last / end-effector frame can be calculated as follows:

$$T_N^0 = T_1^0 \cdot T_2^1 \cdot \ldots \cdot T_N^{N-1} \tag{2.12}$$

The procedure of finding the frame for all links of a manipulator in a given position of its joints is called *Forward Kinematics*. If f is a function of configuration (e.g. joint angles) the *forward kinematics* are equal to :

$$\mathbf{y} = f(q) \tag{2.13}$$

where q is a configuration, which is a point with coordinates the joint values of the robot.

In this thesis the term *TCP* or *Tool Center Point* refers to the end effector's transformation frame notated as T_N^0. The vector $\mathbf{y}$ refers to the X,Y,Z and Roll , Pitch and Yaw angles. The angles correspond to the rotation part of the T_N^0

For manipulators the calculation of the T_i^{i-1} is not trivial and for that reason it is decided to be used the Denavit and Hartenberg parameters, called DH parameters. Assigning the $\mathbf{z}$ axis as the rotation axis for a link, and given θ_i the joint rotation on $\mathbf{z}_i$ axis, translation d_i between the joint i and i-1 in $\mathbf{z}_i$ axis, the translation a_i along the $\mathbf{x}_{i-1}$ axis and the twist/rotation by α_i about the $\mathbf{x}_{i-1}$, the transformation matrix T_i^{i-1} is equal to(see reference [Pau81]):

$$T_i^{i-1} = \begin{bmatrix} c\theta_i & -s\theta_i \cdot c\alpha_i & s\theta_i \cdot s\alpha_i & a_i \cdot c\theta_i \\ s\theta_i & c\theta_i \cdot c\alpha_i & -c\theta_i \cdot s\alpha_i & a_i \cdot s\theta_i \\ 0 & s\alpha_i & c\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.14}$$

The end-effector frame of manipulator can be calculated using the formulas 2.13 and 2.14. The symbols cx and sx refer to $cos(x)$ and $sin(x)$ respectively.

2.4 Euler angles and Quaternions

The end effector orientation of a manipulator can be expressed wit the help of three variables called *Euler* angles. *Euler* angles are used especially to describe 3D rotations. They have different application like in aircrafts and are used also in robotics. The parameters called *yaw*, *pitch* and *roll* are counterclockwise rotations and are defined as follows:

$$yaw : R_z(\alpha) = \begin{bmatrix} cos\alpha & -sin\alpha & 0 & 0 \\ sin\alpha & cos\alpha & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.15}$$

$$pitch : R_y(\beta) = \begin{bmatrix} cos\beta & 0 & sin\beta & 0 \\ 0 & 0 & 0 & 0 \\ -sin\beta & 0 & cos\beta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.16}$$

$$roll : R_x(\gamma) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & cos\gamma & -sin\gamma & 0 \\ 0 & sin\gamma & cos\gamma & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.17}$$

The yaw, pitch and roll rotations are placed together in order to create the final rotation matrix which can be calcualated as follows:

$$R(\alpha, \beta, \gamma) = R_z(\alpha) \cdot R_y(\beta) \cdot R_x(\gamma) \tag{2.18}$$

All the rotations are applied to a specified fixed-reference frame. It is also very important to note that the order of the rotations plays a significant role. The order is: (a) roll, (b) pitch and (c) yaw. If the order is changed the final result is different.

Using Euler angles to describe rotation differences is not the best solutions. For instance an orientation, has not unique values for α , β and γ. Another well known problem is the singularities when the amount of rotation around an axis goes to 0^o or 180^o. Quaternions parameters eliminate the issue of singularities and they are more efficient for interpolating rotations.

A quaternion is considered as a four dimensional complex number given by: $h = a+b\cdot i+c\cdot j+d\cdot k$, where $a, b, c, d \in R$ are the four independent parameters of the quaternion. The "i,j,k" are the imaginary numbers and that leads to $i^2 = j^2 = k^2 = i \cdot j \cdot k = -1$. One important characteristic of quaternions is that the multiplication is not commutative which is a common characteristic of rotations.

For the rest of the thesis the quaternion are normalized meaning that the $a^2 + b^2 + c^2 + d^2 = 1$. The quaternion are represented by a 3D rotation by an angle θ around an axis given by the unit vector $\mathbf{v}$= $[v_1, v_2, v_3]$:

$$h = cos(\frac{\theta}{2}) + (v_1 \cdot sin(\frac{\theta}{2}))i + (v_2 \cdot sin(\frac{\theta}{2}))j + (v_3 \cdot sin(\frac{\theta}{2}))k \tag{2.19}$$

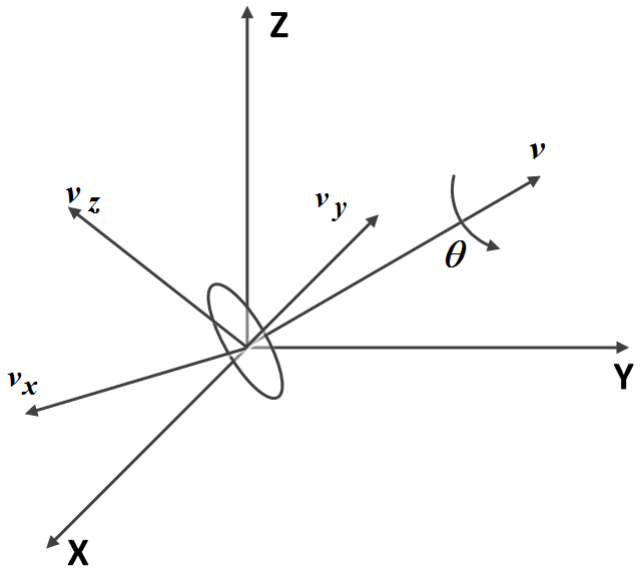


Figure 2.5: Axis and Angle rotation using unit quaternion

The reason why quaternions are used in this thesis is their efficiency of interpolating between two rotational frames. The so called *Gimbal lock* is not present if quaternions are used. The main "heavy" computations using quaternions are the conversion from a matrix to quaternion and the other way around[Sho85, DKL98]. Quaternions are going to be used also in order to create random uniform rotations[Arv92a]. In a later chapter is going to be discussed the drawbacks using Euler angles which are:

- insufficient creation of uniform random rotations
- insufficient interpolation due to singularity problems

The following section describes the notion of a configuration of a system, the configuration space and the workspace of a robotic arm. The position and rotation of the end effector are used to describe the *location* of the TCP.

2.5 The configuration space and robot's workspace

2.5.1 Definitions

The definition of configuration space of a robot is very important since it is going to be used often in this thesis. The formulation of configuration space started from a work of Lorenz–Perez[LP81], who gave the concept of the planning in general form. As the interest for motion planning increased the notion of configuration space become clear. The configuration space is based on a *configuration* of a robot which specifies the position of states(joints) of the robot. That specification is unique, it

means that two configurations correspond to different states of the robot. For instance the image 2.6 present an example of two different configurations for a planar 2D robotic arm.

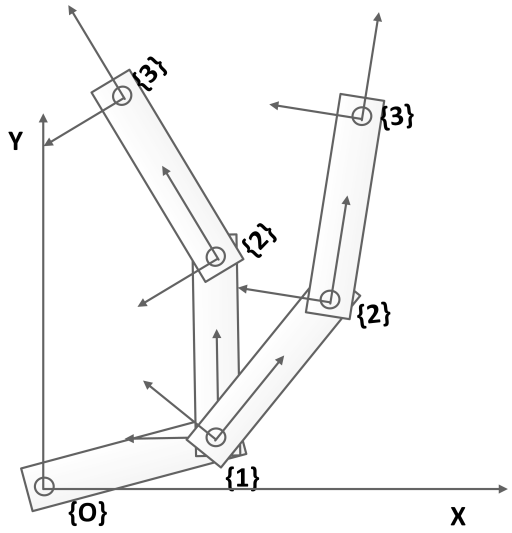


Figure 2.6: Example of two configurations for a planar robot arm

The configuration space of a robot or simply *C-Space* is the space that includes all possible configurations of the robot. The dimensionality of the configuration space is equal to the number of *degrees of freedom*. The *degree of freedom of a system or DoF* is equal to the number of independent parameters needed to describe a configuration. For example the DoF of a robot arm is equal to the number of joints. In conclusion a configuration can defined in abstract level as a point with coordinates: q_i=$\{q_{i1}, q_{i2}, q_{i3}, ..., q_{iN}\}$ where N is the DoF of the system, and the C-space is defined as $C = \bigcup_{i=0}^{\infty} q_i$, where q_i is a configuration. Is is obvious that if N is the DoF of the robot the C-Space belongs to R^N ,$C \in R^N$. For easier further understanding the configuration is considered as a point.

The *workspace* or the operation space of a robot arm is the space which represents the points of the real environment (R^3 for 3D or R^2 respectively for 2D) that the robot end effector can reach. The workspace is simply the volume where the robot can work. The workspace of the robot arm is our Euclidean / 3D world and consists of the position and orientations of the end effector of the robot arm. In this thesis, the symbol W represents the workspace of the robot i.e the placement of the robot in 3D space. Mathematically the workspace consists of the three positions and three orientation variables of end effector.

2.5.2 Configuration space obstacles

Since the C-Space is defined, any other sub-space $X \subseteq C$ can be defined also. The configuration obstacle space is a sub-space. As definition the configurations space obstacles C_{obs} is the sub-space from the C-Space where the robot collide. Let denote as $O \subset W$ the obstacle space and a link

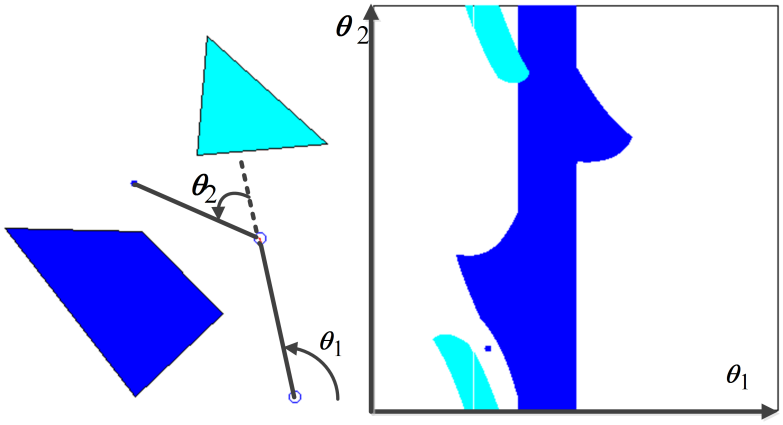


Figure 2.7: 2 DoF planar robot arm with its C_{free} and C_{obs} regions. The left image depicts the robot arm with two obstacles and the right image corresponds to the configuration space obstacles.[2]

body of the robot as $L(q) \subset W$. The C-Space obstacles is defined as follows[LaV06][CLH+05]:

$$C_{obs} = \{q \in C | L(q) \cap O \neq \emptyset\} \tag{2.20}$$

The C_{obs} space is closed set, and that is obvious since the robot should avoid the surrounding objects. It is obvious that the configuration free space is equal to : $C_{free} = C \backslash C_{obs}$ (equal to $C_{free} = C - C_{obst}$). The C_{free} is an open set meaning that it cannot even "touch" the C_{obs}. The configuration free space is the set of configurations at which robot does not collide.

A challenge using configurations space is the calculation of C_{free} and consequently the C_{obs}. For 2 DoF robot arm may be possible to be calculated, however an exact representation of C_{obs} in high dimensions is extreme difficult and till now it has not been done. The reason why there is such a difficulty is that configuration space for an N-Dof robotic arm contains all the possible joint angles. The C_{obs} is calculated only if all possible joint angles are taken into account. For each configuration, a collision detection should be done. Surely the whole procedure is not going to finish in a desirable time. It is very time consuming to examine all joints angles. A small resolution between two consecutive configurations can be considered. It is a challenge to calculate the most appropriate resolution. However, even in that case, the calculation of the C_{free} is time consuming. The image 2.7 represents an example of C_{free} and C_{obs} for a planar 2D robot arm. From this example can be noticed that there is not a direct mapping from 3D space to C-Space. The shape of the obstacles differ completely in C-Space. For all the above reasons, the designing of a motion planner with a pre-calculation of C_{free} is till nowadays impractical for high dimensional C-Spaces. However, as it is going to be explained in later chapter, sampling based approaches overcome those difficulties and although they can work in C-Space they are very fast and efficient.

Since configuration space uses points to define the state of the system, it would be really useful to calculate distances between the states of the system. That is described in the following subsection.

2.5.3 Distances-Metric

All sampling based algorithms require a function that can calculate the distance between two configurations (points). The distance between two configurations q_1 and q_2 can be calculated as the maximum displacement of every point of the robot between the two configurations. In other words if $\mathbf{A}(q_1)$ and $\mathbf{A}(q_2)$ are the set of all points of the robot for the two configurations in workspace $\mathbf{W}$ and $\lambda_{q_1} \in \mathbf{A}(q_1)$ and $\lambda_{q_2} \in \mathbf{A}(q_2)$ two points on the robot on the corresponding configurations, the distance is equal to:

$$d(q_1, q_2) = \max_{\lambda \in A(q)} ||\lambda_{q_1} - \lambda_{q_2}|| \tag{2.21}$$

The symbol $||.||$ denotes the Euclidean distance. The function 2.21 is not so intuitive to be calculated since it requires the calculation of the displacement of each point of the robot arm.

Since equation 2.21 is not appropriate, the distance between two configurations is going to be calculated in C-Space. For that reason *metric spaces* are going to be introduced. A metric space (X, ρ) is a topological space which has a function f: X×X->R with the following characteristic: (a) non-negativity (b) Reflexivity (c) Symmetry and (d) Triangle Inequality. The function defines the distance between two points in metric space, if the four conditions are fulfilled.

The most common family of metrics, symbolized as L_p in the literature[LaV06], is equal to:

$$L_p : d_p(q, q') = (\sum_{i=1}^{N} |q_i - q_i'|^p)^{(1/p)}, \tag{2.22}$$

where N is DoF and p denotes the metric. For instance when p=2, the normal Euclidean distance in R^N is defined. The L_1 corresponds to *Manhattan* metric. In the case where the p goes to ∞ the metric is equal to:

$$L_\infty = d_\infty(q_i, q_i') = \max_{1 \leq i \leq N} \{|q_i - q_i'|\} \tag{2.23}$$

The L_p metric can be used with the same way also in the vector space. The L_p norm in vector space R^N for a vector $\mathbf{q}$ is equal to:

$$L_p : ||\mathbf{q}|| = (\sum_{i=1}^{N} |q_i|^p)^{(1/p)}, \tag{2.24}$$

2.6 Calculating minimum distance for convex polyhedral

As already mentioned, every motion planning algorithm should deliver a path where the robot should follow within an allowable accuracy. A path surely should be collision–free meaning that the robot should not collide with the environment and with himself. For that reason collision detection is an important tool whose performance influences robotics and in general all those tasks that involve motion and calculations of penetration between convex objects. Non-convex objects are not considered in this thesis, since the FRIEND recognition system decomposes a non-convex

[2]Thanks to the Java program in the web site: http://ford.ieor.berkeley.edu/cspace/

object to convex objects. Another reason is that the implementation of the GJK (Gilbert–Johnson–Keerthi) algorithm in the FRIEND system can handle only convex objects. This chapter discusses the basic collision detection algorithm that is applied to the system FRIEND.

One methodology for collision detection is to calculate the C_{obs}, like in figure 2.7, and then to compute the distances between the robot and the objects. That could be the best solution, especially for C-Space motion planners. However as already discussed, the computation endeavor is extreme high and it increases as the dimensionality of the space increases also. That is the reason why the workspace for a robot arm is used for the purpose of calculating distances between objects (consequently collision detection). We have to recall that the position of each link of the robot arm can be calculated since (a) the values of joint angles are known and (b) forward kinematics can be calculated fast using the formula 2.12.

A robot arm is a rigid body like the obstacles. For that reason, an idea of representing the arm as a 3D model in Cartesian space is meaningful. In literature exist many algorithms that can calculate collision detection, penetrations and distance between polyhedral using 3D modeling. Example of algorithms are: GJK[GJK88], the SWIFT (Speedy Walking via Improved Feature Testing) algorithm[EL01] and many others[3].

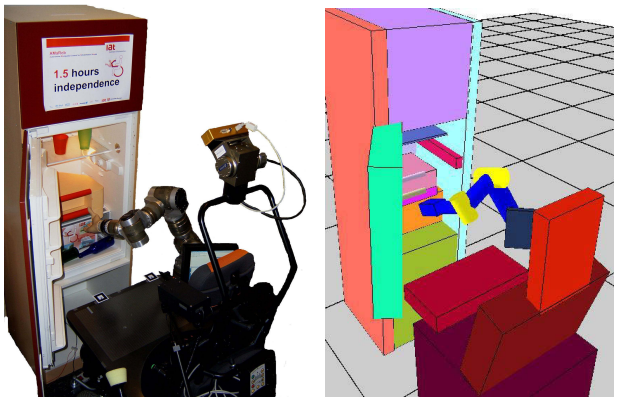
In this thesis the enhanced version of GJK algorithm [Cam97][4] and the SWIFT algorithm have been tested. Both give similar results with the SWIFT algorithm to be slightly faster. However, for calculating minimum distances the enhanced GJK package is used, because it provides addintinal information like the pair of closer points between two polyhedral.

All the algorithms are integrated into the software module which is called $MVRServer$ (MVR means Mapped Virtual Reality)[FIG05]. It models the robot and the objects of its environment using only primitive objects e.g. cuboid, sphere and cylinder. The reason is that primitives have simple geometry and the calculation of minimum distances can be done very fast. The figure 2.8 presents an example how the real world is modeled into primitives in MVR. At the beginning of this work[FIG05] the MVRServer had only the GJK as basic collision detection algorithm. During the thesis the MVRServer is enhanced so that other packages are able to be used like the SWIFT. A good advantage of this improvement is that a comparison between several packages can be done.

An important feature that is added in the thesis is the support of bounding boxes for the objects. Each object in MVR has its own bounded box called oriented bounded box (OBB)[GLM96, ZF95]. The bounding box covers each object and is used in order to improve the performance of the collision detection. In a later chapter is explained the algorithm that is developed that improves the total performance of a motion planner compared to the normal approach. The figure 2.9 depicts an example of bounding boxes with extra size for visualization purposes as well as the minimum distances between the robot arm and the environment.

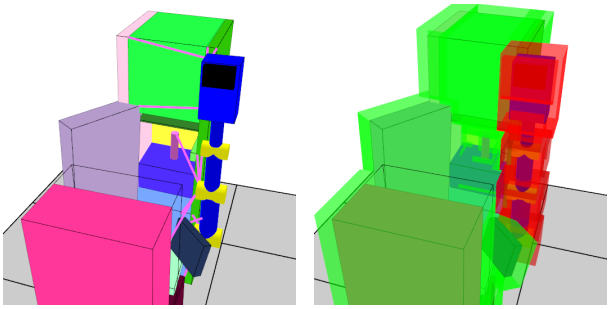
[3]Many collision detection packages are available in the web site : http://gamma.cs.unc.edu/

[4]Source code is available on the Internet in web site: http://www.cs.ox.ac.uk/stephen.cameron/distances/



(a) Real scene (b) MVR Scene

Figure 2.8: Real scene and the corresponding 3D modeled scene



(a) MVR Scene (b) MVR Scene and the OBBs

Figure 2.9: MVR Scene, minimum distances between the robot arm and an illustration of OBBs

2.7 Inverse Kinematics

The inverse kinematics, as the word *inverse* implies, is the process of determining the joint angles by a given end effector (TCP) position and orientation. The equation 2.13 defines the forward kinematics as a function of the joint angles. If the function f^{-1} exists, the inverse kinematics for a given TCP P_{TCP} is solvable and the following equation gives the configuration:

$$q = f^{-1}(P_{TCP}). \tag{2.25}$$

The inverse kinematics are very important for every motion planner since motion specifications, assigned to the end effector in the operational space, are transformed into the corresponding joint space motions. Furthermore the goal of task is normally interpreted as a frame, that is the final position and orientation of the end effector.

In literature, the calculation of f^{-1} is not trivial. Two types of methodologies exist: *closed* and *numerical* form. The *closed* form solution (or analytic) are in general faster than numerical solution. The disadvantage of closed form solutions is that they require either algebraic or geometric intuition in order to find the necessary equations. Another disadvantage of closed form solutions is the inability to be applied to many systems, since they are depended by the structure of the robot. On the other hand the numerical solutions are applicable to all kind of manipulators, but they are slow and they cannot provide all the possible solutions.

The total number of inverse kinematics solutions depends on the *redundancy* of the system. A system (e.g. robot arm) is called redundant if it has more DoF than it is needed in order to describe its position and orientation. A robot arm is called redundant if by having three variables for position (x,y,z) and three for orientation (roll , pitch and yaw) the number of joints are greater(not equal) than six. A non-redundant robot arm has a restrict number of inverse kinematic solutions for a given pose while a redundant robot arm has an (theoretical) infinite number of solutions. For instance a six DoF robot arm has up to 16 possible solutions. The selection from an infinite set of solutions is not trivial and depends on the given task. One solution could be to select the closest one to a given configuration. Another propose is to define a set of possible solutions and to pass this set to the planner. Another idea could be to calculate randomly solutions around a workspace region and to pass the random solutions to the planner.

In this thesis the developed planner works in C-Space and it is applied to a 7 DoF robot arm. As follows, the robot arm is redundant. The kinematics of the manipulator are designed in a manner that each joint has the rotation axes shifted for 90^o degrees relative to the previous one. For this seven degrees of freedom robot arm an analytical inverse kinematic solution has been developed, called *KCC*[IG97, IG98, IG00]. In the image 2.10 is presented the seven DoF robot arm together with the 90^o shifting between a joint and its previous.

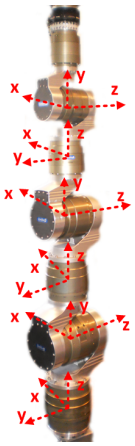


Figure 2.10: 7 DoF Robot arm and the coordinate systems of each joint

Redundancy is one parameter of the *KCC* algorithm. Since this manipulator is redundant, its elbow can have theoretical unlimited number of positions for a given end effector position and orientation. However, all of them lie on a circle (see figure 2.11). The *redundancy angle* α, shown

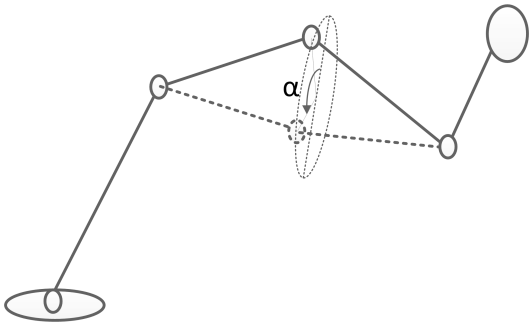


Figure 2.11: Example of redundancy angle for a manipulator

in figure 2.11, is the parameter that controls the amount of inverse kinematics solutions that the algorithms returns, since it rotates the elbow of the manipulator. The amount of possible solutions is computed as follows:

$$N_{IKsolution} = \frac{360^o}{\alpha}, \alpha \in [0, 360] \tag{2.26}$$

2.7.1 Differential Kinematics - Jacobian - Singularities

In a later chapter a Jacobian based approach is going to presented. For that reason, a short introduction to *Jacobian* matrix, its characteristic and calculations is going to be done. The term *differential kinematics* describes the relationship between the joint velocities and the corresponding end-effector linear and angular velocity. The geometric representation is done by the *Jacobian* matrix. If **p** represent a matrix with the position (x,y,z) and orientation (Roll, Pitch and Yaw –Euler angles) of the end effector, the relationship between the two type of velocities are calculated by the equation:

$$\dot{\mathbf{p}} = J(q) \cdot \dot{\mathbf{q}} \tag{2.27}$$

The dimension of jacobian matrix J(q) is m× n, where m is the number of variables representing the position and orientation of TCP and n is the dimensionality of C-Space. It is clear that for a redundant manipulator where n> m the Jacobian matrix is not a square matrix. The Jacobian matrix is given by the following equation (recall the equation 2.13):

$$J = \begin{bmatrix} \frac{\partial y_1}{\partial q_1} & \cdots & \frac{\partial y_1}{\partial q_n} \\ \vdots & \cdots & \vdots \\ \frac{\partial y_m}{\partial q_1} & \cdots & \frac{\partial y_m}{\partial q_n} \end{bmatrix} \tag{2.28}$$

The inverse transformation of equation 2.27 gives the relation between the end effector differentials and the joint velocities vectors:

$$\dot{\mathbf{q}} = J(q)^{-1} \cdot \dot{\mathbf{p}} \tag{2.29}$$

For redundant manipulators the calculation of inverse Jacobian matrix is not possible, and for that reason the $J(q)^{-1}$ is substituted by the *Jacobian transpose* (J^T) or *Jacobian pseudo inverse* $J^+ = (J^T \cdot (J \cdot J^T)^{-1})$. Both can replace the inverse in equation 2.29[SSVG09].

Using differential kinematics, the manipulator's joint angles of the robot arm can be calculated by the help of a given target TCP and thus following the steps:

1. Compute Jacobian
2. Compute J^{-1} or equivalent
3. Compute $\Delta\theta = J(q)^{-1} \cdot \Delta p$
4. $q = q + \Delta\theta$

That means that by doing small steps Δp in workspace, the differential kinematics forces the manipulator to reach a particular end effector location by transforming the joint angles. The procedure finishes when the Δp is smaller than a tolerance value ϵ.

Let now have the case where m=n. The matrix loses its rank, when the manipulator is in *singular* configuration. The inverse of the jacobian is not possible to be calculated since the determinant becomes zero. That leads to unexpected velocities and consequently behavior. The singularities for a manipulator can be devided to boundary and internal. Boundary singularities occur when the arm is stretched whereas the internal one can occur everywhere inside its workspace. Internal singularities exist normally when two axis of motion are aligned.

The calculation of Jacobian can be done either by taking the derivatives of forward kinematics or geometrically. The first one is not an efficient method if the dimensionality of the system is high. Therefore geometrical computation of the jacobian matrix is more appropriate. Using these approaches the Jacobian matrix can be calculated by the equation [SSVG09, OS84]:

$$J_e^O = [J_1(q) \dots J_N(q)], where\ J_i(q) = \begin{bmatrix} J_{P_i} \\ J_{O_i} \end{bmatrix} = \left\{ \begin{bmatrix} \mathbf{z}_{i-1} \\ \mathbf{0} \end{bmatrix} \right\} \left\{ \begin{bmatrix} \mathbf{z}_{i-1} \times (\mathbf{p}_e - \mathbf{p}_{i-1}) \\ \mathbf{z}_{i-1} \end{bmatrix} \right\} \tag{2.30}$$

The vectors $\mathbf{z}_{i-1}$, $\mathbf{p}_e$ and $\mathbf{p}_{i-1}$ (e is abbreviation of the end-effector) are computed as follows

- $\mathbf{z}_{i-1}$ is taken from the third column of the T_{i-1}^O
- $\mathbf{p}_e$ is computed by the first three elements of the fourth column of the T_e^O
- $\mathbf{p}_{i-1}$ is computed by the first three elements of the fourth column of the T_{i-1}^O

The equation 2.30 is used through this thesis and computes the Jacobian based on the reference frame $\{O\}$.

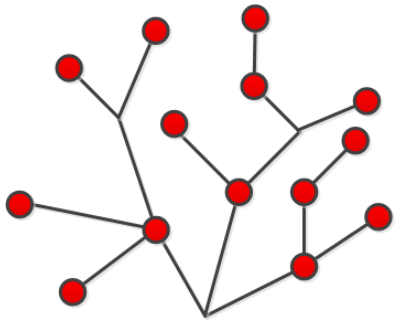


Figure 2.12: An example of Tree

2.8 Graphs

Graphs are going to be used later and it would be meaningful to be introduced at that point. The following chapter uses the notion of graph and tree (a category of graphs). A graph G=(V,E)[Die10] is a pair of sets connecting *vertices*(*nodes*)V with *edges* E. An edge connects two nodes. The edges set E is a subset of $[V]^2$. The order of a graph is equal to the cardinality of the set V that is $|G| = card(V) = |V|$ while the graph's size is equal to the cardinality of the set E. The degree of a vertex is the number of edges connected to it.

In literature graphs are separated into sub categories:

- undirected: Undirected graph is the graph where an edge $e = (a, b)$ is identical to the $e' = (b, a)$
- directed: In directed graphs the edges can be represented by arrows and there is only one edge that can connect two vertices.
- mixed : It is a combination of two referred types
- connected: A graph is k-connected if no two vertices in G are separated by fewer than k other vertices[Die10]. The connectivity of a graph G is symbolized by $k(G)$ and if k is bigger than one the graph is connected.

In this thesis we are going to use a category of graphs called *trees*. Trees are simply acyclic connected graph where each vertex is connected with only one vertex. Simply if a graph has N vertices, a tree has N-1 edges. The figure 2.12 shows a graph that is called tree.

Chapter 3

State of the art - Motion planning

3.1 Introduction

This chapter is a small introduction to motion planning. Basic algorithms, definitions and features are going to be described in this chapter since they are going to be used through this thesis. First some important definitions regarding the planning is presented and later some basic algorithms are introduced.

Motion planning involves the automatic motion of a robot from a starting placement to a goal one, avoiding collisions with objects and obeying extra constraints if that is necessary. Historically its first formulation called the *piano problem*. The problem had one critical question which should be answered: how to move a piano, that is a complicated furniture, through a cluttered environment that involves objects like furnitures or humans. It is clear that making a robot to move autonomously in our world , taking its own "decisions" and "thinking" about the path that needs to be followed is a challenge. The quality of the path and the calculation time are influenced by several criteria. For instance some heuristics may reduce the calculation time but the quality of the path may be not improved. It is usually a compromise between quality and calculation time.

The motion planners can be classified depending on the requirements[CLH$^+$05]. An example of classification can be made by the *task*. In this case, the sub-categories are: *navigation*,*coverage*, *localization* and *mapping*. Navigation, which this thesis is dedicated to, is the task of finding a collision free path from one position to another one. By Coverage problems sensors are passed over all points in space like painting. Localization is the problem where a map is used to interpret sensor data and is used to determine the configuration of the robot. The robot works in unknown environment, collects data and constructs a representation in order to use it later one the other three sub-categories.

One important property of a motion planner is the *completeness*. The *completeness* of a motion planning can be categorized by *exact*, *resolution* and *probabilistic*. A planner is (exact) complete if the planner can find a solution if one exist. A planner is *resolution* complete if a solution exists at a given resolution of discretization (normally a grid discretization). A planner is called

probabilistically complete if the planner finds a solutions while the execution time of the algorithm approaches to infinity.

A planner can be *online* or *offline*. The environment by an offline planner is known and the planner gives its result to trajectory execution. An online planner constructs its path while the robot is moving. The sensors detect changes in the environment and the planner updates its result based on the sensors data.

3.2 Motion planning algorithms

In this chapter the basic algorithms are going to be presented. The algorithms are potential fields, roadmaps, cell decomposition and sampling based approaches like Rapidly exploring Random Trees(RRTs).

3.2.1 Potential fields

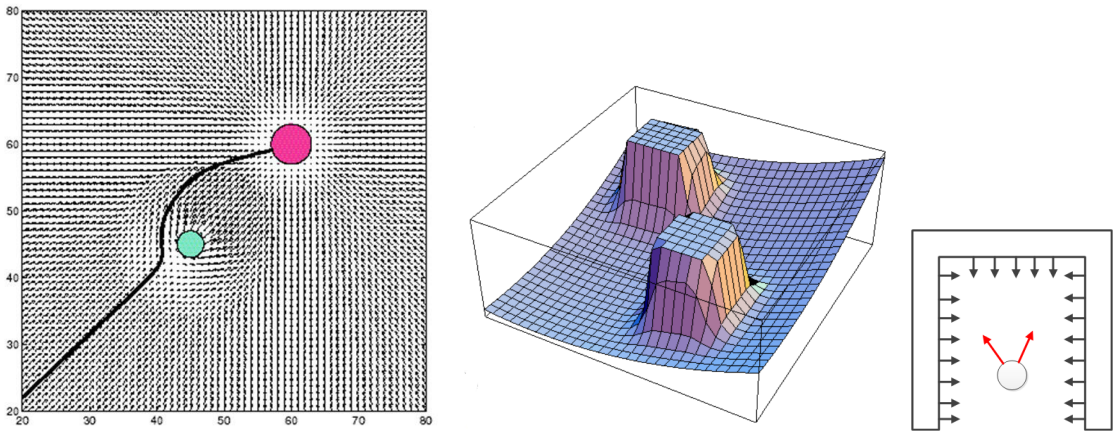
Historically potential fields was one of the first approaches in the field of the robot motion planning. This approach was first introduced around 25 years ago[Kha85]. The main idea behind potential fields is the existence of virtual potential field/forces in the C-Space. The obstacles are repulsive and the target configurations are attractive for the robot. With this approach the robot can be visualized as a magnetic ball in C-Space which rolls around the obstacles (same potential as magnetic ball) and it is attracted by the target which has the reverse potential. The figure 3.1(a) presents a simple example of motion of a planar robot in a potential field. The robot avoids the obstacle since the "charge" is the same. The image 3.1(b) shows an example of potential fields in C-Space. The obstacles are represented as mountains and the bigger they are the bigger is the repulsive force to the robot in order to avoid them.

The main drawbacks of the simple implementation of potential fields are that (a) the representation of potential fields in C-Space is not easy in high dimensions and (b) they cannot guarantee a solution. Being more precise, the potential fields may trap into local minimal and the robot may be not able to escape from them. For these situations it is necessary the developing of "heuristic" or very intelligent potential field functions that can help the robot to escape from local minimal within some time. The term *local minima* defines the space where the robot has difficulty to move, because it is surrounded by obstacles. In such a case the robot should be able return back in order to avoid them and escape from them. An example of local minima is illustrated on figure 3.1(c). In literature exist several modifications in order to avoid local minima[Lat91, Cha96, BLL91].

3.2.2 Probabilistic Roadmaps (PRM)

The word *roadmap*[Lat91] is the composition of the two words: "*road*" and "*map*". That means that a map is constructed that contains connected roads (or paths) between a start and a goal state. In other words there is an approximated mapping of the C_{free} space where the robot can

[2]Thanks to the web site document `www.cs.cmu.edu/~motionplanning/lecture/Chap4-Potential-Field_howie.pdf` for the images

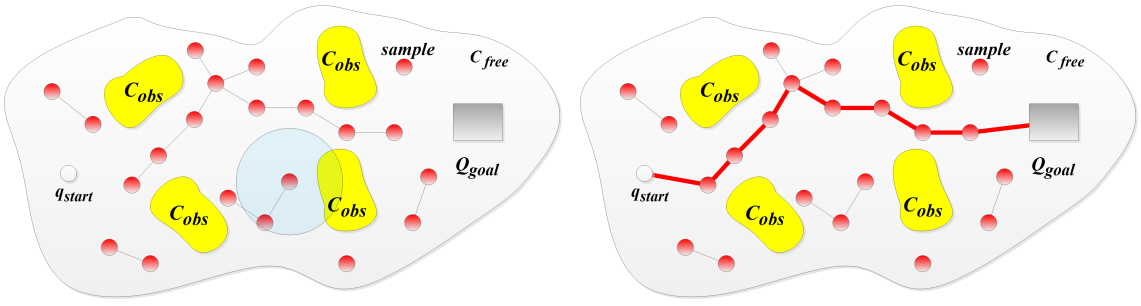


(a) Example of motion with potential field (b) Representation of potential fields (c) Local minima

Figure 3.1: Example of robot motion with potential field ,(b) representation of them and (c) local minima[2]

draw possible connections and paths and afterwards it tries to find the "best" path. The mapping is done using several methods like visibility graphs[Lat91, LPW79], Voronoi diagrams[Aur91] or some Silhouette methods like Canny roadmaps[HMP00]. The result from a map construction is a graph $G = (V, E)$ with vertices V and edges E.

In motion planning *probabilistic roadmaps (PRM)* is the common approach that is used[KSLO96, HLK06]. The PRMs are constructed as already mentioned in two phases: learning phase, where the roadmap is constructed, and query phase, where a search in the graph is done and the final path is extracted. The graph search is normally done by A^*[HNR68] or a D^* [SM93] algorithm. The construction of the roadmap is done by sampling configurations in C_{free} and consequently connecting vertices that do not belong on the same (connected) component. The PRMs are probabilistically complete. If the edges has nodes in the same connected component, the PRMs are conditionally complete. The figure 3.2 explains the construction and query phase of roadmaps.



(a) Learning phase - Graph construction (b) Query phase - Path extraction

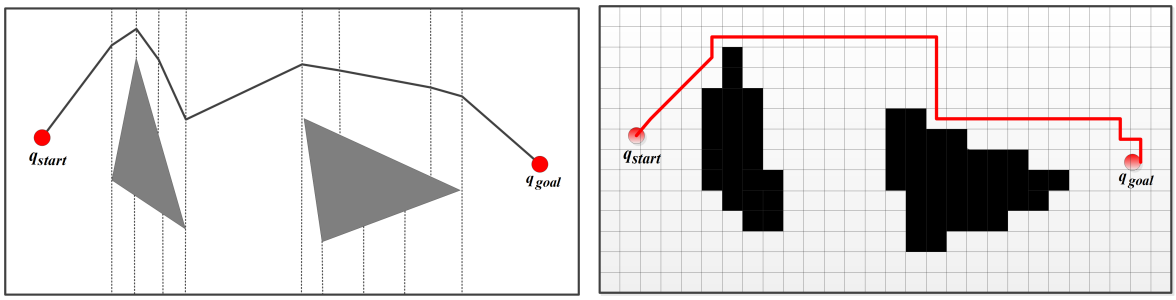
Figure 3.2: Example of probabilistic roadmaps learning(left) and query phase(right)

The PRMs approach is used widely in the case where the environment does not change[NSL99, WAS99, SHJ+05] or when few objects may move in the environment[YYG10]. The density of the graph plays also an important role and that is related with the number of samples and their distribution[LBL04]. Combination of PRMs and RRTs is also available[BCL+03].

3.2.3 Cell decomposition

Cell decomposition is an another representation of the free space of the robot. In this case the space is discretized into some specific regions called *cells*. The path later is computed using the center or some other point of the cells. Like the PRM, this approach has two phases: first is the decomposition of the space into cells and later the planner searches for a path towards all adjacent cells.

Typical decompositions of the space are the *trapezoid*[PS85] or *Morse Decomposition*[ACR+02]. The first one is based on polygonal representations while the second one allow representations of nonpolygonal and nonplanar spaces. Another interesting approach of cell decomposition is the *hierarchical approximate cell decomposition*[Lat91] where the cells are generated uniformly. That means that the cells have specific size and the space is divided into many cells. Each cell is checked whether it is in collision or not, and if it is, it is marked as non-free (figure 3.3(b)).



(a) Trapezoid cell decomposition

(b) Uniform cell decomposition.

Figure 3.3: Cell decomposition examples for a 2D robot. The left image illustrate an example of trapezoid cell decomposition and the calculated path. On the right image the uniform cell decomposition is presented. The black region is the C_{obs}

The image 3.3 illustrates two examples of space decomposition. It can be seen that the space is divided into regions and the path is constructed through these regions. The decomposition seems to be very attractive since it can be done fast. However in high dimensions, the decomposition requires effort and it is not efficient any more. Moreover by increasing the resolution e.g. reducing the size of the cells, the number of cells is increasing and consequently the searching time is also increasing. Another problem in high dimensions is the lack of knowledge of the C_{obs} space, and that leads to the fact that it is a heavy endeavor to construct a map like in figure 3.3(b). All these reasons concludes to the fact that cell decomposition is impractical in high dimensional spaces.

Cell decompositions (especially the uniform approach) has been used in the literature for manipulation motion planning with high success rate indicating the efficiency of the algorithm if the working space is below or equal to three[Ojd09a][OLJ98]. An recent approach [SZ11] introduces workspace 6D decomposition using cylindrical approach.

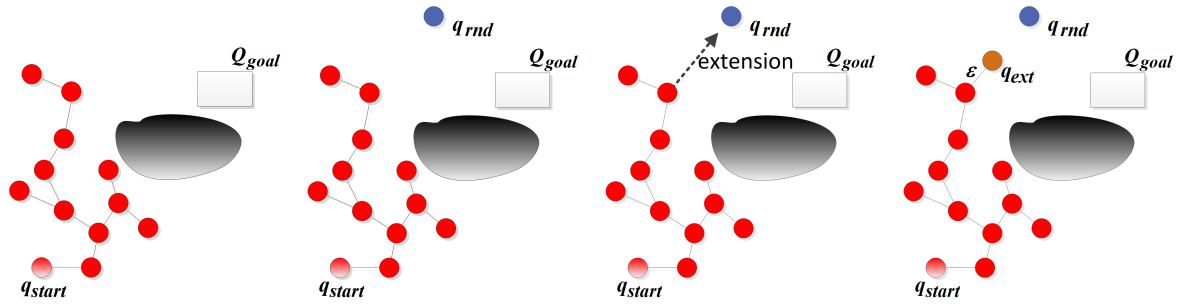
3.2.4 Rapidly exploring Random Trees - *RRT*

The RRTs started developing since almost 15 years ago, and from the beginning seemed to be promising. As it is going to be explained later the modifications from the main approach[LJ99, Lav98] can give high boost in the performance.

The main characteristic of the RRTs is that it is a *single query* approach. A tree is built incrementaly and at the same it explores the C_{free} in order to reach the goal. As the number of the samples in the configuration free space is increased the probability to reach the goal tends to one. For that reason the RRT approach is probabilisticaly complete[LJ99]. Denoting as V_n^{RRT} the set of vertices produced and n the number of iterations (consequently the time) the probability the algorithm to reach the goal is equal to:

$$P(V_n^{RRT} \cap X_{goal} \neq \emptyset) > 1 - e^{-\alpha \cdot n} \tag{3.1}$$

where α is a positive constant value.



(a) An RRT tree after N iterations (b) A random configuration q_{rnd} is created (c) Nearest neighbor is found (d) A stepwise ϵ extension is done

Figure 3.4: An example of how the RRT is constructed and explores the C_{free}

Compared to PRMs the RRTs is *single query*. A graph, which is called tree, is constructed and at the same time an exploration of the space is done. Having $T = (V, E)$ a tree, where V is the set of vertices and E the set of edges, the tree is constructed by following the steps:

1. a random configuration q_{rand} is created
2. the nearest neighbor $q_{near} \in T$ to the q_{rand} is found
3. extension from q_{near} towards q_{rand} is done

The extension is done *stepwise*. The extension in typical RRT approach takes as small step ϵ towards the q_{rand} and a new q_{ext} is created after that operation. The figure 3.4 represent an example showing the exploration of the RRT trees in the configuration free space. While this procedure continues and the number of nodes /vertices in the tree is increased the probability to reach the Q_{goal} is increased and after some time a solution is going to be extracted.

3.2.5 Other approaches

Despite the most common approaches, in literature there are several other methodologies. Theses solutions may not be general or easy to be implemented as the four described approaches, however it is worthing to be mentioned since they can solve fast several tasks. Moreover several approaches focused on the quality of the paths which is very important. The four planners focused mainly on the feasibility e.g. to extract solutions as fast as possible without considering path quality. An interesting approach to the direction of path quality planners is done in [KCT+11]. In this work an approach using a stochastic trajectory optimization framework is implemented. They update noisy trajectories in order to produce trajectories being more optimal. They claim that due to stochastic behavior it can overcome problems that other approaches have [RZBS09]. The literature in manipulation planning is extremely huge providing the community with many variations, adaptations and improvements of the planners. Many of them are discussed in the upcoming chapters.

Chapter 4

Connecting two configurations

This chapter presents an efficient collision detection algorithm that is needed by the forecoming planners e.g. CellBiRRT. In general, every planner should deliver a collision free path. A path consists of several segments and the planner guarantees that each segment is collision free. That is achieved if each segment is sampled into many points. One approach could be to check each sample for collision and also for additional constraints if it is necessary. That is the reason why collision detection calls are many during the planning (hundred till thousands calls) and it is the most expensive procedure. Therefore this chapter presents the collision detection strategy used later in every other planner such as CellBiRRT. The algorithm reduces dynamically the number of samples needed to be checked boosting the performance of the planner.

The rest of this chapter presents the collision detection strategy . Briefly the algorithm does the following:

- Computes Oriented Bounding Boxes(OBBs) that are used for calculating minimum distances between robot arm and obstacles
- Uses a formula to calculate the length of the maximum curve ($L_{max_{curve}}$) that is done by the manipulator
- Uses both OBBs and $L_{max_{curve}}$ in order to reduce the number of samples needed to be checked for collision

4.1 Calculating minimum distances - Identifying near and far obstacles

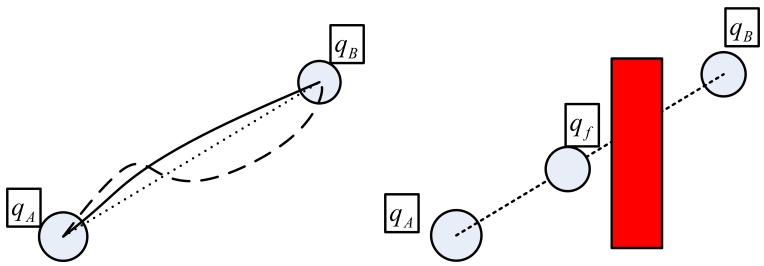
Collision detection is a basic tool whose performance influences robotics and computer graphics applications, such as motion planning, obstacle avoidance, virtual prototyping, computer animation, physics-based modeling, and , in general all those tasks that involve motion and calculations of penetration between convex objects. As already mentioned , the objects in MVR are convex.

Having two configurations q_A and q_B, a segment $\sigma_{seg} \in [q_A, q_B]$ is a continuous function that connects two points and it is collision free if each point in the segment is collision free too e.g.

$$D_{min} = (\arg\min_{q \in \sigma_{seg}}(D(q))) > \delta \tag{4.1}$$

where $\delta > 0$ is the minimum allowable distance and $D(q)$ is the minimum distance between all rigid bodies of the robot at configuration q with the environment. Consequently a path $\sigma = \bigcup_i \sigma_{seg_i}$ is collision free if each segment is also collision free.

In computer science continuous signals are sampled. The more dense the sampling is, the more accurate and closer to continuous signals are the results. The sampling is also used in a segment σ_{seg_i} since the path is defined as a continuous function. Each segment is sampled into many points and each point is checked for collision using the equation 4.1. The figure 4.1 illustrates an example of sampling between two configurations.



(a) possible segments and sampling the lines

(b) sampling direct line and collision detection

Figure 4.1: Normal sampling approach and collision detection going from q_A to q_B

In that point it is very important to define the number of samples needed to be taken. In this thesis and for simplicity the segments between two configurations are always a straight line. We define $step_i$ the distance between two samples as follows:

$$step_i = \frac{\Delta q_i}{L} \cdot Res \tag{4.2}$$

where L is equal to the length /distance between the two points, and Res is the reference resolution. The distance can be calculated from the equation 2.22. The higher the parameter Res is, the bigger is the distance between two samples which lowers the accuracy to detect a collision. The figure 4.1(b) presents an example having high accuracy. If the $step_i$ was much higher it may be not possible to detect the collision. That is the clue that the $step_i$ should be as small as possible, but this situation increases the computation time. For that purpose the dynamic reduction of intermediate steps is necessary[FG11a].

In literature, the dynamic collision detection approaches can be grouped in four methods[SSL05]: Feature tracking or static methods[Cam90], Boundary volume[SSL05], Swept volume or space time volume intersection[FH93] and Trajectory parameterization[Can86]. The first one tries for every

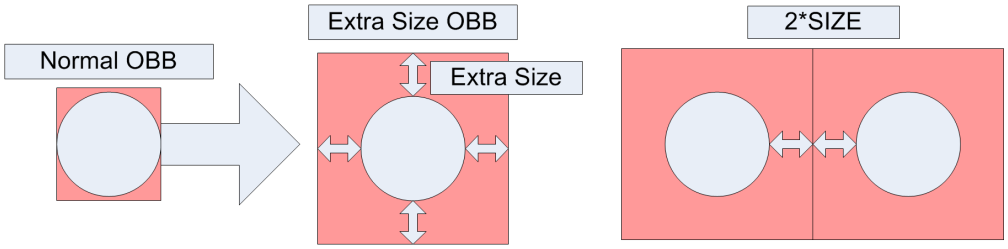


Figure 4.2: OBB creation and the additional SIZE

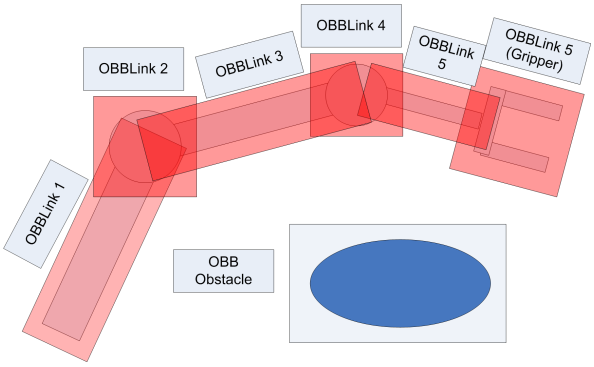


Figure 4.3: Example of robot arm and OBB construction

step in a segmented path to detect pairs of closest features of objects. The second method creates bounding volumes and in each segment the objects are tested for intersection. Actually it is assumed that if two configurations are closer than a value δ, there is no need to call any mechanism that calculates distances. The third method computes the volume that each movable object contains during the time, and attempts to find possible intersections. The last method calculates the geometry of the objects along the examined path by polynomials.

In order to calculate efficiently the collisions and distances between robot arm and objects, a specific structure is needed. In this thesis the decomposition of complex objects into primitive objects is done. Primitive objects are boxes, cylinders and spheres. If the high accuracy is not important that type of modeling is adequate[JTT98]. Space partitioning[CO90] like hierarchical volume representations (HVR) or binary space partitioning (BSP)[TN87] could be used. Another important issue is to have a boundary representation of each object. In this work Oriented Bounding Boxes (OBB) haven been used[GLM96, ZF95]. Other representations like Axes Align Bound Boxes or ellipsoids fits[EB97] or k-DOPS[KHM$^+$98] are referred in the literature. However the OBBs are easy to be constructed and they follow the orientation of the primitive object, which requires less computations.

In this thesis the system is able to detect far and near obstacles and OBBs are used for that puprosed. Instead of taking the exact OBBs, an additional $SIZE$ is given to each one as the picture 4.2 depicts. If OBB_{Size} denotes the regular size of the OBB the following equation denotes the

Algorithm 1 GREEDY-DIST-TO-OBSTACLE (Obstacle,RobotArm)

```
1: MinDistance=1000000; //big value;
2: for (i==1; i ≤ RobotArm.NumberLinks; i++) do
3:    if (IntersectOBB(Obstacle,Robotarm[i])== TRUE) then
4:       TempDistance=ComputeExactMinDistance(Obstacle,Robotarm[i])
5:       if (MinDistance≥TempDistance) then
6:          MinDistance=TempDistance;
7:       end if
8:    else
9:       if (MinDistance≥2* SIZE) then
10:         MinDistance=2 * SIZE;
11:      end if
12:   end if
13: end for
14: RETURN MinDistance //is the min distance
```

resized OBB:

$$OBB_{Size'} = OBB_{Size} + SIZE \tag{4.3}$$

An example of OBBs applied to robot arm links is given in figure 4.3. It is obvious that if two OBBs are adjacent the distance between two objects is defined as $2 \cdot SIZE$. The figure 4.2 presents an example in 2D while the figure 2.9 presents the not-exact OBBs case in the robotic system FRIEND. The following definition distinguishes the case of near and far obstacles:

Definition 1 *Given two objects A and B, and their OBBs OBB_A and OBB_B with sizes $SIZE_A$ and $SIZE_B$, they considered to be far if:*

$$B = OBB_A \cap OBB_B = \emptyset \tag{4.4}$$

otherwise are considered to be near.

At this point, it is set that two near obstacles have a distance less than $2 \cdot SIZE$. The algorithm 1 presents the approach for calculating approximate distances between robot arm and obstacles. The method GREEDY-DIST-TO-OBSTACLE computes the minimum distance between robot arm and an obstacle. First it checks if two OBBs intersect and if they do the algorithm computes the exact distance between polyhedrals. The well known separation axes theorem is used in order to calculate possible penetrations between OBBs. This procedure is much faster than calculating always the exact minimum distances between polyhedral.

Let remark here that the $SIZE$ for OBBs should be bigger enough than the minimum allowable distance δ(refer equation 4.1). If the $SIZE$ is equal to zero, the exact OBBs are used and the GREEDY-DIST-TO-OBSTACLE returns mostly zero(if the OBBs do not intersect) or a value less than the δ. If the OBBs intersect, the probability the rigid bodies to intersect is high. If the $SIZE$ is equal to δ, the GREEDY-DIST-TO-OBSTACLE returns either $2 \cdot \delta$(if OBBs do not intersect)

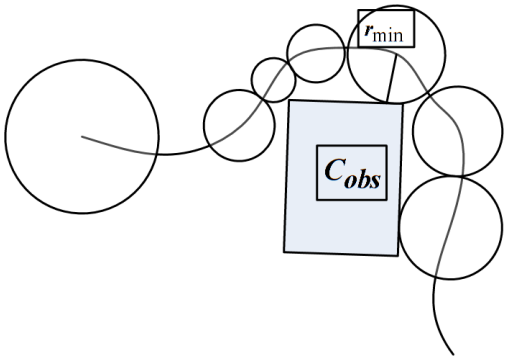


Figure 4.4: Bubbles while a robot is moving in 2D C-Space

or a value close to δ. The 3D modeling, e.g. recognition of obstacles is done by the sensors. That gives some uncertainties in the model, since no sensor can guarantee 100% accuracy. For all these reasons it is more reasonable to have a $SIZE$ much bigger than the minimum allowable distance δ.

4.2 Calculating the length of a maximum curve $L_{max_{curve}}$ - Using "Bubbles"

In this section the length of the maximum curve in cartesian space that a manipulator can travel by a displacement Δq is going to be presented. Knowing the maximum curve and the minimum distances the following challenge is faced: is it possible to neglect intermediate samples predicting that are collision free?

The idea is based on [QK93, Qui94] where the notion of *bubbles* is presented. Bubbles are defined as a local subset of the free space around a given configuration of the robot. Having r_{min} the minimum calculated distance in C-Space, the bubbles are defined as:

$$B(b) = \{q : ||b - q|| < r_{min}\} \tag{4.5}$$

where b and q are configurations. It is clear that if the distance between two configurations is smaller than the radius r_{min}, the segment σ_{b-q} is considered to be collision free. Recall that the r_{min} in equation 4.5 and the b and q are in the C-Space. Image 4.4 illustrates an example of bubbles in C-Space. However for a manipulator with high number of degrees of freedom the r_{min} in C-Space is difficult to be computed. For that reason the $L_{max_{curve}}$ is going to be computed.

In[SScL05] they defined the upper bound of the curve that a link of a robot arm could travel in the cartesian space for a step Δq_k as follows:

$$\lambda_{i,max} = \sum_{k=1}^{i} Rmax_k^i \, |\Delta q_k| \tag{4.6}$$

where the i refers to the joint and Δq_k is in radians. It is obvious that if $i = N$ the λ refers to the end effector. The appendix E describes shortly the derivation of the equation 4.6. If a link is prismatic the $Rmax_k^i$ is equal to one, otherwise $Rmax_k^i$ is the upper bound of the distances between the points of the link i and the center of the rotation of the joint k e.g. when the arm is stretched.

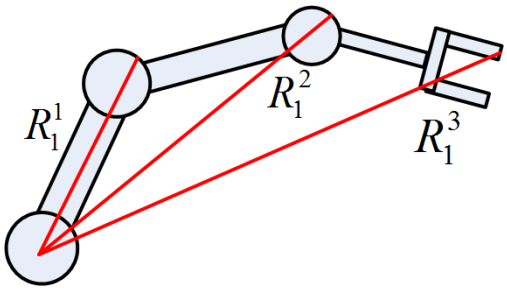
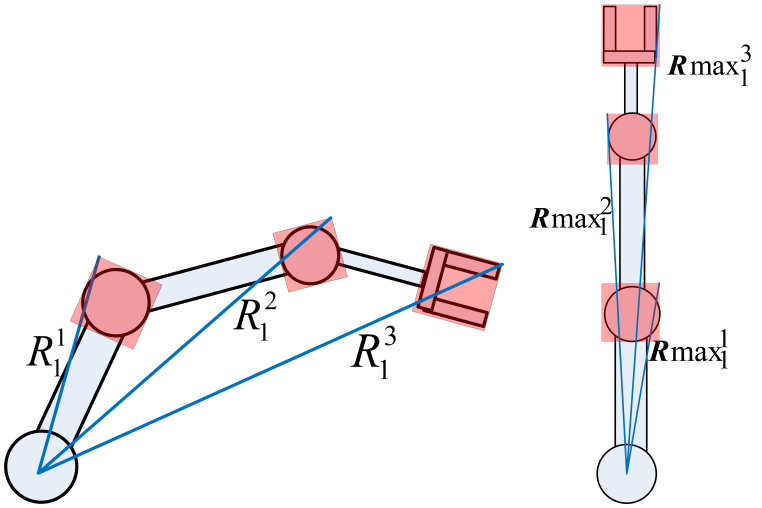


Figure 4.5: Example of the radius of a planar manipulator for a configuration



(a) The radius using only the exact OBBs

(b) The maximum radius when the arm is stretched

Figure 4.6: Example of estimation of R_k^i for a planar manipulator using the OBBs

The calculation of the radius R_k^i is critical. An example of the radius is given in the figure 4.5. Analyzing each point of the model it is going to be inefficient since a model contains triangles and consequently many points. Thus, calculating the exact R_k^i for each point may not be the best solution. The approach used in this work is an approximation and it is done using the exact OBBs (means the SIZE is equal to zero). In such a case the only computation effort is to define the maximum Euclidean distance between the center of the rotational joint and the outer vertices of the bounding box(8 vertices). Each box has eight vertices and the computation is done using only

them. The calculation of the maximum distance between each of the eight points with the center of the joint has to be done. This idea is presented in figure 4.6. The maximum curve that a manipulator is possible to follow is done when it is stretched. At that case the values of the R_k^i becomes maximum ($Rmax_k^i$). It is reasonable to calculate the R_k^i when the arm is stretched and with the help of the bounding boxes. Surely it is an over estimation and not an exact computation of the λ from equation 4.6. On the other side it is a fast computation which is done once even before the beginning of motion planning algorithm since the values of the $Rmax_k^i$ depend on the characteristics of the robot arm e.g. length and size. Summarizing the "λ_{max}" in this work is calculated as follows:

$$\lambda_{max} = \sum_{k=1}^{N} Rmax_k^N |\Delta q_k| \tag{4.7}$$

where N is the DoF of the robotic arm. Considering a path σ which is sampled uniformly into M steps, the maximum possible curve's length that the robot arm can do $L_{max_{curve}}$ is given by:

$$L_{max_{curve}} = \sum_{i=1}^{M} \lambda_{max} \tag{4.8}$$

4.3 Reducing the number of samples between two configurations q_A and q_B

In this chapter the results from the previous sections are going to be joined. As already mentioned, the number of collision checks for a segment should be high enough in order to guarantee that the segment between two configurations is collision free. Many calls of collision detection algorithm increases the total computation time. The idea proposed in this work is to use the radius R_k^i calculated before together with the algorithm 1 and the equations 4.5 and 4.6. The bubbles are constructed based on the maximum curve $L_{max_{curve}}$. Their values depend on the displacements Δq_k of each link k of the robot. The algorithm 2 presents the proposed approach.

At the beginning of each segment, the λ_{max} is calculated (line 1-2) and it is used as a constant through the algorithm. For each obstacle two values, called D_i and $ActualArc_i$, are assigned, which (a) represent the distance between the arm and the respective obstacle (D_i) and (b) the maximum possible length $ActualArc_i(L_{max_{curve}})$ of the curve that the arm may have done if the environment consisted only with the obstacle i. For each sample (line 9) there is a pair of distances for an obstacle and the traveled curve of the robot arm. The $ActualArc$ for each obstacle is checked over the samples if the value overcomes the corresponding distance between the robot arm and the obstacle(line 19 and 20). If it does, the $ActualArc$ for the corresponding obstacle is reseted (line 21) and if the distance D_j (line 23) exceeds the limit δ the algorithm returns the last valid (notated as q_k in line 28) or l configurations before the last valid q_k (line 24). The value of l depends on the environment and also if the robot has to move far from obstacles. Big l forces the arm to move far from possible colliding obstacles.

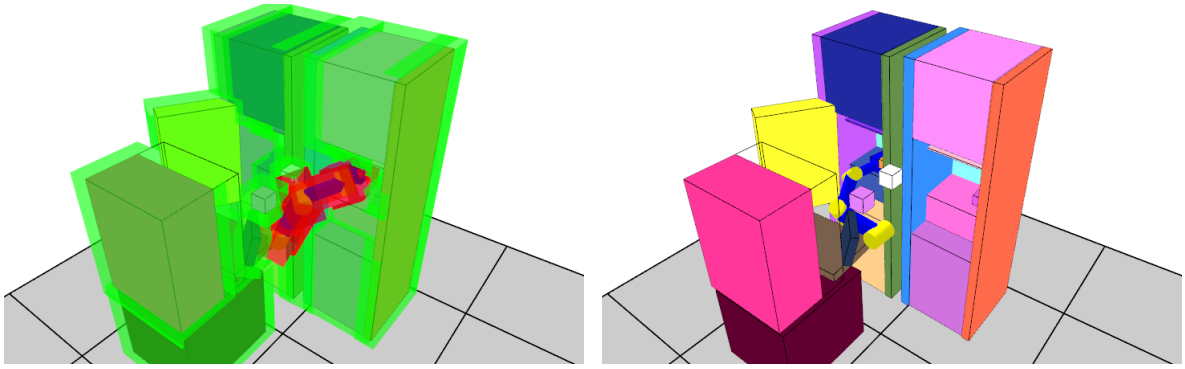
Another important issue that the algorithm 2 covers is the probability of a self collision to occur. Self-collision is done when two parts of arm collide. In the robot arm used in this work the

Algorithm 2 q = ConnectEfficient(q_A, q_B)

```
1: Steps=CalculateSteps() // see equation 4.2
2: λ_max=CalculateLMax(Steps) //see equation 4.7
3: for (i=1;i≤Obstacles;i++) do
4:    ActualArc_i=0; //all global traveled path is zero
5:    D_i = GREEDY − DIST − TO − OBSTACLE(i, RobotArm) //see algorithm 1
6:    SelfCollision=0; // for self collision
7:    StorePair(ActualArc_i,D_i) // pairs are createed
8: end for
9: for (i=1;i≤ NumberSamples;i++) do
10:   q_i=q_A + i · step
11:   SelfCollision+=λ_max;
12:   if (SelfCollision≥ λ_max) then
13:      dDistance=SelfCollisionOBB(); //computes the MinDistance between end effector and the necessary links
         based on GREEDY-DISTANCE
14:      if (dDistance≤ δ) then
15:         RETURN q=q_k − l · step
16:      end if
17:   end if
18:   for (j=1; j≤Obstacles; j++) do
19:      ActualArc_j+=λ_max; //the global calculated path for the arc_i
20:      if (ActualArc_j≥D_j) then
21:         ActualArc_j=0 //check if for the obstacle is close to robot
22:         D_j=DISTANCE-TO-OBSTACLE(j,RobotArm) //only respective obstacle is checked
23:         if (D_j < δ) then
24:            RETURN q=q_k − l · step //k≥1, is the -k- valid configuration and l is greater or equal to zero
25:         end if
26:      end if
27:   end for
28:   Store k=i; //this index stores collision free configuration
29: end for
30: RETURN q_B;
```

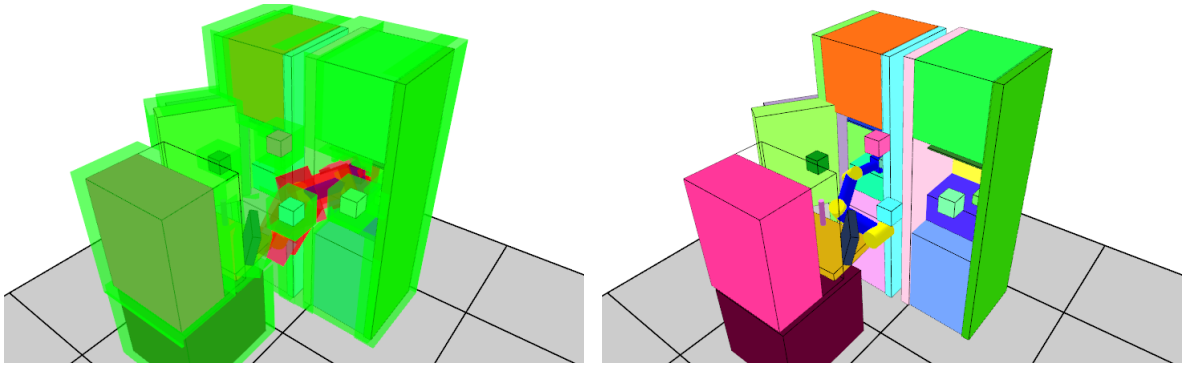
end effector can reach its first and second link and for that reason self collision checking is done. The self-collision checking follows the same strategy and is covered in lines 11-15.

Summarizing the algorithm 2 can efficiently ignore samples that are not necessary to be checked by controlling the $ActualArc_i$ of each obstacle with the corresponding distance between it and the robot arm. For that reason the algorithm guarantees that there is no missing samples. The D_i of each obstacle is equivalent with the "bubbles" in figure 4.4 and the $ActualArc_i$ is the corresponding maximum length of the path done by the manipulator. Experimental results presented in later chapter show the significant improvement in the performance of a motion planning algorithm that works in C-Space.



(a) Start Configuration (b) Goal Configuration

Figure 4.7: Simulation environment 1 with start and goal configurations in MVR. The robot should move from the start to goal configuration. The aim is to measure the collision detections and the computation time of the planning algorithm



(a) Start Configuration (b) Goal Configuration

Figure 4.8: Simulation environment 2 with start and goal configurations in MVR

4.4 Experimental results

In this chapter it is going to be shown the profit of the algorithm 2 compared to the regular algorithm 3. The algorithm 3 uses also the λ_{max} however not for each obstacle individually but globally. Moreover it uses the method *CalculateAllMinDistancesWithOBB()* which returns the global minimum distance between the robot arm and the obstacles using either OBBs (algorithm 1) or not. For the first case (where OBBs is used) the algorithm is notated as **optimized** and for the second case the algorithm is referred as **no-optimized**. The approach proposed here is referred as **efficient**.

For calculating minimum distances between polyhedral the GJK algorithm is used. The workstation consists of an Intel Core i5M@2.46Ghz. In order to study the efficiency of the algorithm,

Algorithm 3 q = ConnectOptimized(q_A, q_B)

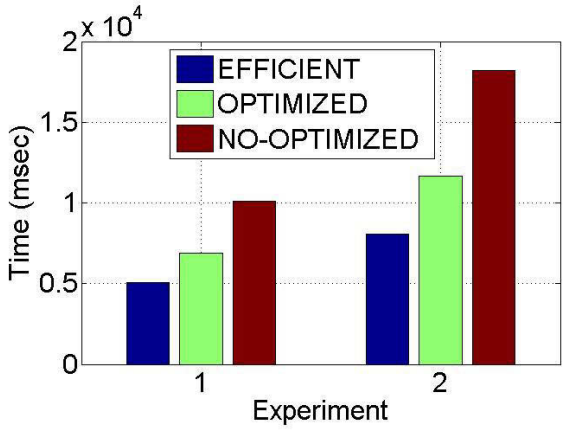
```
1: Steps = CalculateSteps() //see equation 4.2
2: λ_max=CalculateLMax(Steps)();
3: MinDistance = q_A.CalculateAllMinDistancesWithOBB() //global collision detection for all obstacles using algorithm 1 (OBBs)
4: ActualArc=0
5: for (i=1;i≤ NumberSamples;i++) do
6:    q=q_A + i · step_i
7:    ActualArc+=λ_max; //the global calculated path
8:    if (ActualArc≥MinDistance) then
9:       ActualArc=0;
10:      MinDistance=q_i.CalculateAllMinDistancesWithOBB() //global collision detection for all obstacles using algorithm 1
11:      if (MinDistance≤δ) then
12:         RETURN q=q_k − l · step // is the -k- configuration before the collision, l is a positive number
13:      end if
14:      Store k=i; //this index shows the collision free configuration.
15:   end if
16: end for
17: RETURN q_B;
```

different benchmarks are performed. The simulation environments are illustrated in the figures 4.7 and 4.8. The Collision Profit is given by the formula:

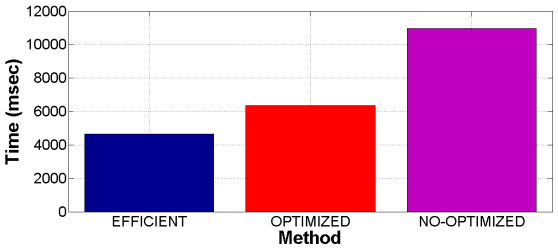
$$Profit = \frac{ExpectedCD - MeasuredCD}{ExpectedCD} \cdot 100\% \tag{4.9}$$

where the notation *ExpectedCD* refer to the expected number of collision detection checking and $MeasuredCD$ are the actual - measure number of checking. For all experiments, the sampling resolution "*Res*" (equation 4.2) for the line between q_A and q_B is equal to 1 deg. The efficiency of the proposed approach is tested in an RRT based motion planner. All experiments run with the same pseudo random generator e.g. the sampling remains the same for all tests. The performance depends mainly from the performance of *ConnectEfficient* and *ConnectOptimized*.

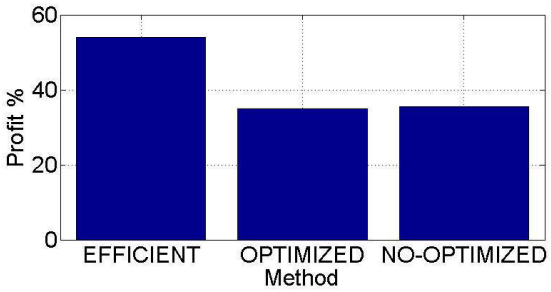
The results are really promising. The proposed approach (efficient) managed to deliver almost 25% faster result than the optimized approach. That is expected since the collision profit (figure 4.9(c)) is also 25% bigger, which means that the method considers less samples which improves the total performance. The value of the SIZE is an important parameter. From the figure 4.9(d) can be seen that 5cm is the value that resulted to shorter computation time. The dimension of the gripper (in 3D model) is equal to 10cmx10cmx20cm (width, depth, height). Experiments done with small SIZE (e.g.1, 2 cm) do not perform well since obstacles are assumed to have maximum distance equal to 2*SIZE (e.g. 2, 4 cm). That is a small value compared to the size of gripper. If the SIZE is much bigger the algorithm behaves similar to the one with small SIZE. Big SIZE cancels mostly the presence of OBBs since OBBs are going to intersect always. The latter results unfortunately to low performance.



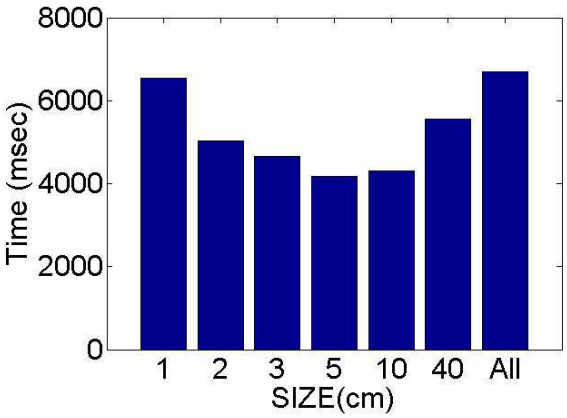
(a) Planning computation time-Method for task 1



(b) Planning computation time-Method for task 2



(c) Collision detection Profit in % for Task 2



(d) SIZE-Planner Time for Task 2

Figure 4.9: Experimental results (a)-(b) Performance influence in computation. (c) The collision detection profit % of each method. (d) The SIZE influences the computation time

4.4.1 Benchmarks with other collision detection packages

Experiments are done with other packages PQP[LGLM00] and $SWIFT++$[EL01]. The PQP package is old and it gave slower results compared to the other packages. The optimization done in algorithm 1 improved the performance of PQP much more than in GJK and SWIFT++. Compared to PQP and GJK algorithm, which have as input a pair of convex polyhedral, the SWIFT++ algorithm works faster if all necessary pairs of objects are given for calculation.

The boost in the performance depends also on the collision detection algorithm. For PQP the improvement was higher compared to the GJKs and SWIFT++s. For SWIFT++ the improvement was slightly less than GJKs (around 22% between *optimized* and $efficient$)

4.5 Discussions

The method *ConnectEfficient*, proposed in this chapter, can reduce significantly the number of the calls of collision detection as well as the computation time for a planner. That is accomplished with a combined strategy of OBBs, the length of the traveled curve of a manipulator and an algorithm for exact distance computation. In the following chapter, the bidirectional sampling based approach *CellBiRRT* is explained. The planner requires an efficient collision detection algorithm and therefore the *ConnectEfficient* is explained here.

The limitation of the algorithm is the size of the robot arm. If the length and the size of the arm are big then the value or $Rmax_k^j$ is also big. That results to the issue that the maximum traveled curve is going to exceed often the calculated minimum distances. The experimental results 4.9 show that if the SIZE is small, the algorithm does not improve the performance. That behavior is similar to when the $Rmax_k^j$ is big. Another limitation is the density of the environment. If the environment is extreme dense with obstacles, the algorithm may not improve the total performance. That is explained since OBBs intersections may occur often. Consequently, the latter calls the GJK algorithm to calculate the minimum distances.

Chapter 5

CellBiRRT- a sampling based motion planner

This chapter describes a sampling based motion planning algorithm, called *CellBiRRT*. The planner works completely in configuration space growing two bi-directional trees. The algorithm is based on the RRT that is described briefly in the 3.2.4 chapter of this thesis. The CellBiRRT configures the C_{free} space into areas called N-cuboid areas that are created based on specific rules. It uses information from the workspace in order to achieve additional features like obeying to additional constraints or to improve the performance of the planner. For that purpose cells are going to be used having some additional characteristics.

The chapter is organized into more sections. The sections are:

- *Configuring the C_{free} space.* This section describes the N-cuboid areas. The first part of the CellBiRRT motion planning algorithm is going to be presented.
- *Efficient sampling areas using cells.* Here the cells are going to be introduced illustrating the benefits of using them.
- *CellBiRRT algorithm.* In this section the *CellBiRRT* algorithm is presented and explained in detail.
- Experimental result. Experimental results analyzing each parameter of the CellBiRRT planner are presented.

The algorithm, that is going to be presented, is an RRT-based algorithm[LaV06]. In literature there is a lot of modifications and improvements of the RRT. Several studies regarding the RRTs and their efficiency have been done [JCS08, MWS07, Lav98, Bra06a, MWS07, OOV02]. Some improvements are focused on the sampling part of the algorithm[JYLVS05, LL04, YJSL05]. More recent motion planner is focused on grasping introducing work space goal regions[BSF$^+$09]. Other work is focused on manipulating objects while the motions of the robot arm should obey to additional constraints e.g. orientation or force/torque constraints[BSFK09, YG05, BS10]. For the

Algorithm 4 Bi-Directional RRT algorithm(BiRRT)

1: $T1, T2$ are the trees, q is configuration
2: $T1.Init(q_{start})$, $T2.Init(q_{goal})$
3: For each direction
4: **loop**
5: $q_{rand} \leftarrow CREATE\_RANDOM\_CONFIG()$
6: $q_{near} \leftarrow FIND\_NEAREST\_NEIGHBOUR(q_{rand})$
7: $q_{\epsilon} \leftarrow EXPAND\_WITH\_STEP(q_{near} \rightarrow q_{rand})$
8: $T1.Add(q_{\epsilon})$
9: **if** $(CONNECT\_TREES(T1, T2) == true)$ **then**
10: return SUCCESS
11: **end if**
12: SWAP(T1,T2)
13: SWAP(q_{start},q_{goal})
14: **end loop**

last two approaches more discussion is going to be done in a later chapter, where benchmarking is done. Recent work[JCS10] replaces the points in the tree with volumes and with this approach the algorithm avoids exploration close to the points belonging already to the tree. In this paper the authors claim that their algorithm improves the performance over the simple bidirectional RRT especially in cluttered environments.

For easier understanding of the rest of the chapter, there is firstly a short description how the Bidirectional RRT works (algorithm 4). In this algorithm there are three steps that influence the total performance. The first one is the the creation of random configuration (method $CREATE\_RANDOM\_CONFIG()$). The basic RRT approach considers the whole C_{free} space and it is surely not optimal since the algorithm may lose time due to that.

The expansion of the tree (method EXPAND_WITH_STEP($q_{near} \rightarrow q_{rand}$)) is the second part that influences the performance. In the simple approach the expansion is stepwise and the total performance depends on the length of the step. In this work it is considered that the expansion is done in a straight line, meaning that the expanded configuration q_{ϵ} lies on the straight line between q_{near} and q_{rand}. For that purpose a method called *Steer* is defined as follows:

$$Steer(q_a, q_b, x) = x \cdot q_a + (1 - x) \cdot q_b \quad x \in [0, 1] \tag{5.1}$$

The *Steer* method returns a configuration q between the two configurations q_a and q_b and can be used inside the method $EXPAND\_WITH\_STEP$.

The CellBiRRT requires to search for the nearest neighbor between a query point x and a set of vertices V. The nearest neighbor method is defined as follows:

Definition 2 *Given a graph G=(V,E) and a point $x \in C_{free}$, the nearest neighbor returns the closest point from the x to the graph G. Summarizing the nearest neighbor equals to:*

$$Nearest(G, x) = q_{near} = \arg\min_{v \in V}(||x - v||) \tag{5.2}$$

The third part is the connection of the two trees. A case is to consider the last expanded node q_ϵ of the T1 tree and the $Nearest(T2, q_\epsilon)$ called q_{nearT2}. The *CONNECT TREES* method calls iteratively the Extend method going from the q_{nearT2} to q_ϵ. The intermediate points generated by the *Connect* method are added to the tree T2. After each iteration the trees are swapped. The resulted algorithm is probabilistic complete but the computation time is high for practical applications. Moreover the computed paths may not be nice from the end-user perspective making the basic algorithm not so applicable for an application in rehabilitation robotics. In the following chapter the N-dimensional cuboids are presented, an approach which improves the performance of the standard RRT planner.

5.1 Configuring the C_{free} space

This section explains the first characteristic of the $CellBiRRT$ that is the N-Cuboid domains and the generation of them. One proposal for managing the C_{free} space is to create a complete structure of configuration space. For a manipulator e.g a system with high number of DoF, this attempt is impractical. A solution such as increasing the Voronoi[1] bias is a good approach[LL04]. An improvement proposed in [JYLVS05] is based on dynamic domain distribution and the visibility Voronoi region is introduced. N-dimensional spheres with variable radius are used (actually the radius is going to be increased if necessary) and the distribution is done over the boundary domains of boundary points (dynamic domain). The drawback of such a method is that it produces many nodes in the free space, since it is biased, and secondly many samples may get rejected before one belonging to the dynamic domain is found. The algorithm in [YJSL05] is based on visible Voronoi region which is the intersection of a nodes Voronoi region with the associated visibility domain. Although it seems to be ideal its computation is a hard problem.

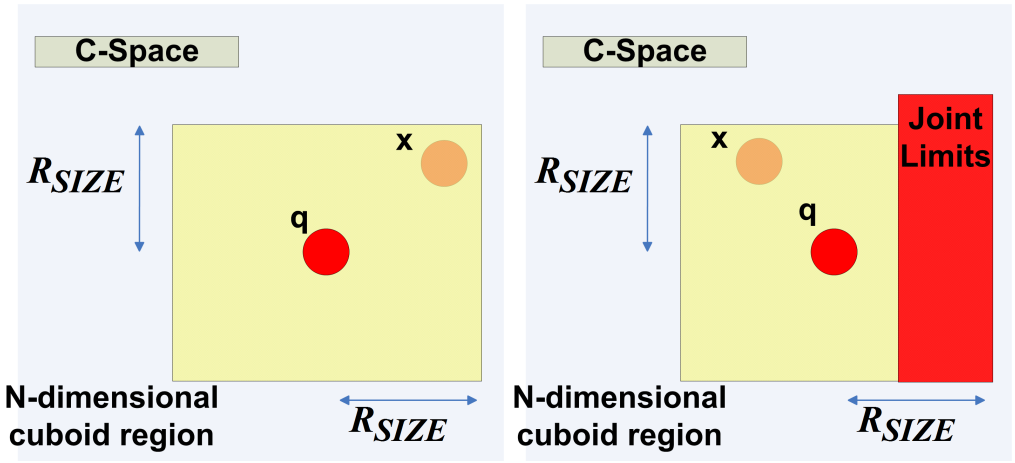
The CellBiRRT approach is based on some rules:

- Each attempt to extend from a configuration q_a towards the q_b the intermediate steps are omitted. That means that the *step* ϵ is much higher than the distance between q_a and q_b e.g $\epsilon >> ||q_a - q_b||$. For that reason the function $ConnectEfficient(q_a, q_b)$ is defined in algorithm 2 and it returns the last valid configuration. This situation as well as the case where $\epsilon < |q_a - q_b|$ are tested. In that case the $ConnectEfficient$ is substituted by the $ConnectEfficientWithStep$. That is explained in detail later.
- the last expanded node of the tree is used as a center of the N-dimensional cuboid regions.

At this point the N-dimensional cuboid regions are going to be introduced. Having a configuration q as the center of the region, the N-dimensional cuboid region R_q is the subset of the C_{free} space whose maximum absolute value (not the distance) between the center q and any point x is less or equal to a value. This value is the size of the region and it is referred as R_{size}[FG10a]:

$$R_q = \bigcup_x \{q \in C_{free}, x \in C_{free} / |q_i - x_i| \leq R_{size}\} \subset C_{free} \tag{5.3}$$

[1]Voronoi diagram, introduced by the George Voronoi, is a method of decomposing a given space into specific domains. The domains are constructed by calculating distances to a given group of subsets [Aur91].



(a) Normal N-cuboid domain case without joint limits

(b) N-cuboid domain in case of joint limit. The shape is parallelogram.

Figure 5.1: N-cbuoid domain with R_{size} for a 2D planar robot arm in C-Space

where i corresponds to the specific coordinate e.g. the joint value. Surely the x_i should not exceed the joint limits (see figure 5.1(b)). An example of the N- Cuboid domain is illustrated in figure 5.1. The R_{size}, as it is going to be shown later, plays a significant role in the final performance of the algorithm. Also the positioning of the center q is an important parameter. The CellBiRRT positions the q to the last expanded node (called *static* placement). That is the default placement and it is used mostly in this thesis. However other possibilities are examined and related benchmarks are presented in detail later.

One possibility of different placing is the shifting. Two cases of shifting are going to be examined. One situation is linear shifting and the second one is exponential. Both of them require a target configuration q_{target}. Let denote as offset D the following quantity:

$$D_{offset} = \delta_{offset} \cdot \Delta q, \Delta q = q_{target} - q_{center} \tag{5.4}$$

where δ_{offset} is a constant. The limits of the δ_{offset} depend on the type of shifting.

If q_{center} denotes the center of the N-cuboid region, the coordinates of the new center q'_{center} after the linear shifting are equal to:

$$q'_{center,i} = q_{center,i} + D_{offset,i} \ , where \quad \delta_{offset} \in [0,1], i \in [1, DoF] \in N \tag{5.5}$$

where N is a natural positive number. The exponential shifting results to the following configuration:

$$q'_{center,i} = \begin{cases} q_{center,i} + (1 - exp(\frac{D_{offset,i}}{||\Delta q||})) \cdot \Delta q_i, \text{ if } D_{offset,i} < 0 \\ q_{center,i} + (1 - exp(\frac{-D_{offset,i}}{||\Delta q||})) \cdot \Delta q_i, \text{ if } D_{offset,i} > 0 \end{cases} . \delta_{offset} \in [0,1], i \in [1, DoF] \in N \tag{5.6}$$

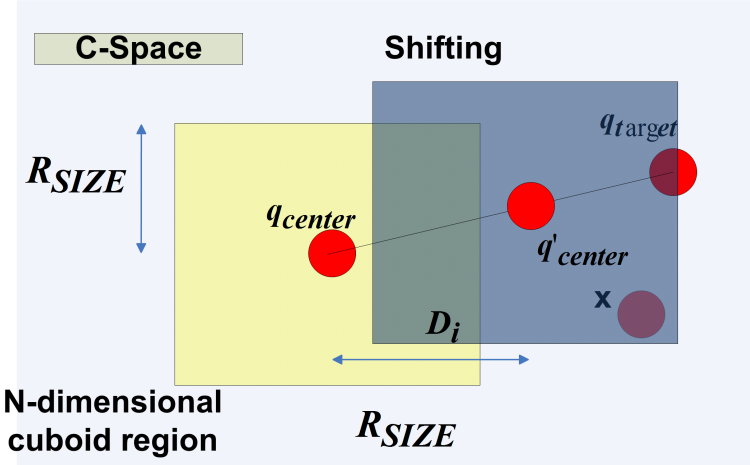


Figure 5.2: Example of linear shifting

It is obvious that the quantity $(1 - exp(\frac{D_{offset}}{|\Delta q|}))$ is between zero and one.

The common characteristic between the two shifts is that the calculated q'_{center} lies on the line between q_{center} and a target q_{target} configuration. The q_{target} configuration may belong to another tree or it may be a random configuration. One difference between the equations 5.5 and 5.6 is that for constant δ_{offset}, the q'_{center} using the 5.6 is in a different position on the line that connects $q_{target} <-> q_{center}$. The factor $(1-exp(\frac{D_{offset,i}}{|\Delta q|}))$ depends on the $D_{offset,i}$. For instance if δ_{offset} is equal to 0.5, the linear shifting moves the new center to lie always in the middle of the line between the two configurations. The image 5.2 presents an example of linear shifting towards the q_{target}.

The next step in a bidirectional approach is a trial to connect the trees. The algorithm 5 requires to connect the trees and some strategies are:

- connect the last expanded nodes between two trees
- connect the closest pair of nodes between the trees
- connect the last expanded node with the closest one to the opposite tree

All strategies are illustrated on the figure 5.3. From this example can be seen that the third approach may deliver better results since there are two trials from two different nodes in order to connect the trees.

The algorithmic part of the approach using N -Cuboid regions is presented in algorithm 5. The method **GenerateNCuboidRegion** in line 5 generates the region where the random configurations in line 6 is going to be generated. The q_{center} is calculated respectively in line 4. In case of linear and exponential shifting the q_{target} is equal to the nearest node between the last expanded node of tree T_1 and the opposite tree T_2. In case of **static** generation, the q_{center} is equal to the last inserted node of T_1. The line 8 has a difference compared to the normal approach regarding the expansion step. Instead of doing stepwise expansion, a direct connection to the q_{rand} is attempted

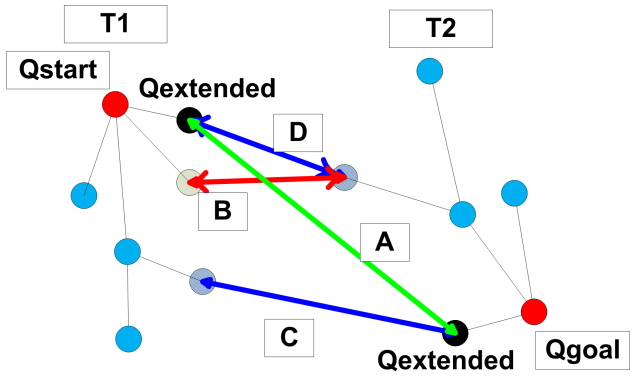
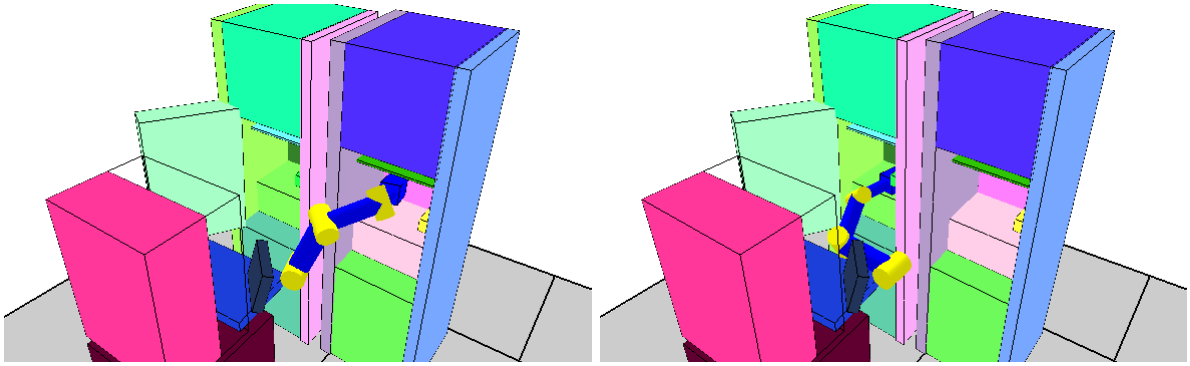


Figure 5.3: Possible connections for two bidirectional trees. (A) The last expanded nodes are attempting to be connected. (B) The closest pair of nodes is trying to be expanded from both sides. (C)-(D) The last extended node from both sides finds the closest node and attempts to expand toward that node.



(a) Start configuration (b) Goal configuration

Figure 5.4: Start(left) and goal(right) configuration. At the beginning the robot arm is inside the first fridge and in the goal location is inside the second fridge

and if it is not succeed, the **ConnectEfficient** returns back the last valid configuration. Recall algorithm 2 for information.

5.1.1 Preliminary experimental results

In this subsection experimental results comparing the simple Bi-Directional RRT and the approach described before are going to be presented. The system consists of an Intel Core Duo 1.86Ghz with 2 GB ram. The start and goal configurations are illustrated on the figure 5.4. For all tables the

Algorithm 5 Bi Directional RRT with N-Cuboid regions algorithm

1: $T1, T2$ are the trees, q is configuration
2: $T1.Init(q_{start})$, $T2.Init(q_{goal})$
3: **loop**
4: q'_{center}= **CalculateCenter(T1.LastInsertedNode(),T2)**
5: **GenerateNCuboidRegion(**q'_{center},R_{SIZE}**)**
6: $q_{rand} \leftarrow CREATE\_RANDOM\_CONFIG()$
7: $q_{near} \leftarrow FIND\_NEAREST\_NEIGHBOUR(q_{rand})$
8: $q_{\epsilon} \leftarrow$ **ConnectEfficient(**$q_{near} \rightarrow q_{rand}$**)**
9: $T1.Add(q_{\epsilon})$
10: **if** $(CONNECT\_TREES(T1, T2) == true)$ **then**
11: return SUCCESS
12: **end if**
13: SWAP(T1,T2)
14: SWAP(q_{start},q_{goal})
15: **end loop**

C_{Length} is equal to:

$$C_{Length} = \sum_{k=1}^{M-1} \sqrt{\sum_{i=1}^{N} (q_i^k - q_i^{(k+1)})^2} \tag{5.7}$$

and the $3D_{Length}$ equals to the total displacement of the end effector position during the execution of the path.

Experiments with R_{size} = 10deg is not done for linear and exponential case since the space between the start and goal configuration is not collision free. The shifting places the N-Cuboid region in the vicinity of the collision space and obstructs the sampling of collision free configuration. The *CreateRandomConfiguration* method runs continuesly till a collision free configuration is found. In this scenario the arm should move first away in order to elicit efficiently. The step-size for the Normal RRT is selected to be 11 deg.

For all cases the usage of N-Cuboid domains over performs the normal RRT. It delivers fast and with 100% success rate results. The simple RRT due to high calculation time fails some times to deliver a solution within 180 seconds which is the upper timit limit for the experimeriments. The static placement of the N-Cuboid domain seems to be faster compared to the others. The shifting returns back paths with more configurations, but after smoothing the final path is shorter (C_{length}) compared to the static case of N-Cuboid domain. Moreover the smaller the R_{size} is, the shorter the path (after smoothing) is computed by the planner. The compromise is the larger computation time.

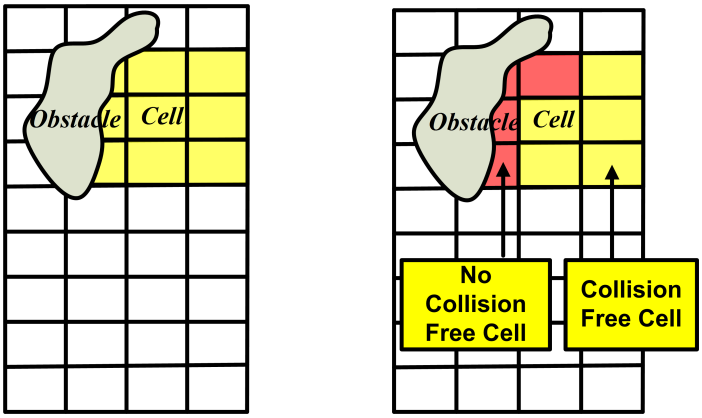
An initial consequence from this first experimental result could be that the N-Cuboid domain really improves the total performance and the final path. That is the reason why the N-Cuboid domains are used also later. The following section presents the *Cells*. Both *Cells* and *N-Cuboid* domains are included and combined in CellBiRRT.

PLANNER	Time (msec)	C_{Length}	$3D_{Length}$ (meter)	Failures	Trajectory Configurations
BiRRT	83927	1541	9,19	32	143,61
NCuboid-BiRRT(R_{size}=10deg) static	22929	2553	16,369	0	212
NCuboid-BiRRT(R_{size}=25deg) static	4370	1552	9,41	0	55,38
NCuboid-BiRRT(R_{size}=50deg) static	3110	1381	8,39	0	30,7
NCuboid-BiRRT(R_{size}=25deg, δ_{offset}=0.5, Linear)	13733	925	5,024	0	43,61
NCuboid-BiRRT(R_{size}=50deg, δ_{offset}=0.5, Linear)	3640	940	5,63	0	25,6
NCuboid-BiRRT(R_{size}=25deg, δ_{offset}=1, Exp)	9794	1027	5,68	0	47,18
NCuboid-BiRRT(R_{size}=50deg, δ_{offset}=1, Exp)	3674	1034	6,2	0	26,78

Table 5.1: Comparison between Normal Bi-RRT and N-Cuboid BiRRT. Average results of 50 trials with maximum computation time 180sec. The results are without smoothing.

PLANNER	C_{Length}	$3D_{Length}$(meter)	Trajectory Configurations
BiRRT	898	3,58	11,72
NCuboid-BiRRT(R_{size}=10deg) static	357,52	1,85	6,8
NCuboid-BiRRT(R_{size}=25deg) static	478,13	2,52	6,72
NCuboid-BiRRT(R_{size}=50deg) static	681,127	3,47	8,12
NCuboid-BiRRT(R_{size}=25deg, δ_{offset}=0.5, Linear)	379	1,96	6,64
NCuboid-BiRRT(R_{size}=50deg, δ_{offset}=0.5, Linear)	527	2,81	6,92
NCuboid-BiRRT(R_{size}=25deg, δ_{offset}=1, Exp)	393	1,98	6,48
NCuboid-BiRRT(R_{size}=50deg, δ_{offset}=1, Exp)	570	3,16	7,66

Table 5.2: Comparison between Normal Bi-RRT and NCuboid BiRRT after smoothing. Smoothing procedures removes the intermediate and redundant configurations from the path. It is called pruning (refer to [GO07])



(a) Cartesian cell decomposition of the space

(b) Cells are distinguished as collision(red) and not-collision free(yellow)

Figure 5.5: Cartesian cell decomposition with a resolution $Cell_{size}$

5.2 Efficient sampling areas using cells

This section describes an enhancement of the previous approach. The BiRRT algorithm with N-Cuboids can deliver fast results however it cannot manage complicated tasks where additional constraints (except the hardware constraints like the joint limits) exist. Such a constraint is for instance orientation or position of the end effector.

The main disadvantages of the previous algorithm are:

- They cannot manage efficiently situations where the end effector of the robot arm should stay inside limits. For instance, if a bottle should be kept up-side down during the motion,the planner should be able to deliver a path, where the end effector must lie inside limits. Therefore, another strategy in order to fulfill these requirements is needed.

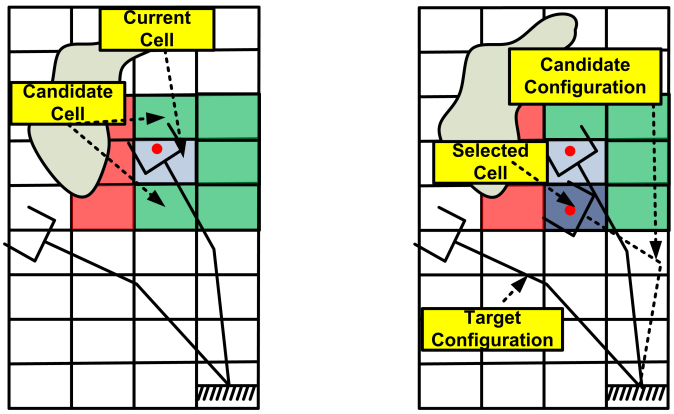
- The resulted path is a not the optimal one. Since the classical RRT cannot deliver an optimal solution, a fast computation of a better one could be also welcome.

The proposed approach does a mapping from the Cartesian space to the configuration space and generates a q_{cell} that lies inside additional constraints but it can be used also without them. The first important part of the approach is the *cell decomposition* of the space. The workspace of the robot is subdivided uniformly with a specific resolution. Only the position is subdivided. The orientation of the end effector is calculated later. An example of subdivision is given in the figure 5.5. Each dimension (x,y and z) is divided by a parameter which is the size of the cell $Cell_{size}$. This resolution influences the performance as well as the final result of the algorithm.

The cells are distinguished to collision and collision-free. A cell is considered to be in collision if its minimum distance with the rest of the objects is less than a value. The value is normally the

same with the acceptable minimum distance of the robot arm. The figure 5.5 presents an example of cell decomposition illustrating also the collision free cells.

For each cell a position with coordinates x_{cell}, y_{cell} and z_{cell} is attached. The coordinates are equal to the position of the center of the cell. The orientation of the cell , expressed by the roll-pitch and yaw, is always zero since the orientation of end effector is the main focus. Given a specific configuration of the robot arm, the challenge at this point is to calculate the position and orientation of the end effector from neighbor cell that is going to be selected. Given the orientation of the current end effector location and the center position(x,y,z) of the cell, the inverse kinematics (IK) are going to calculate a configuration for the robot arm.



(a) Current cells and the neighbor cells are presented

(b) Cell is selected. The location of the end-effector of robot arm is calculated and a configuration is selected.

Figure 5.6: A cell is selected from a group of candidates that are the neighbors of the current cell. The end- effector belongs to the current cell. Inverse kinematics calculate a set of configurations and one is selected

The position of the end effector is used in order to calculate the cell whose position belongs to (called current configuration). The image 5.6(b) illustrates an example showing the current cell, the robot with the end effector and the neighbor collision-free cells (green color). For a planar robot the number of the neighbor cells are eight. However in the 3D world the number of neighbor collision-free cells can be 26 at maximum. The criterion of selecting a cell $Cell_{Sel}$ from a set $C_{cell} = \bigcup_i Cell_i$ of cells is the following:

$$Cell_{sel} = \arg\min_{i \in C}(A \cdot D_{P_{current}-P_{cell,i}} + (1-A) \cdot D_{P_{target}-P_{cell,i}}), A \in [0,1] \tag{5.8}$$

where the symbol $D_{P_{current}-P_{cell,i}}$ denotes the euclidean distance between the center of the cell and the actual position of the end effector, while the $D_{P_{target}-P_{cell,i}}$ is the distance between the

center of the cell and the position of the target's end effector. Normally the A parameter is a small value (around 0.2) so that the cells that are closer to the target's position are in favor.

A cell has a status which is: *active* or *inactive*. Cells that are in collision are considered automatically as inactive. Later, it is going to be explained that the center's placement of a cell is used to calculate a configuration. If such a configuration cannot be computed, the cell is temporally set to inactive. It may be used later again, since it is collision free. For each cell a counter $Cell_{counter}$ is used to indicate the amount of the failures and the visits to a cell. If the counter exceeds a limit, the corresponding cell is deactivated. That allows other cells to be selected.

The second important part after selecting a cell is to calculate the new orientation for the end effector based on the selected cell. The orientation is not unique since a cell can be reselected again that may result to different orientation. The challenge is to calculate an orientation which lies between the current configuration and the target's configuration of the robot arm. For that purpose the orientation is computed as follows:

$$Or = r \cdot Or_{target} + (1 - r) \cdot Or_{current} \tag{5.9}$$

where $r \in [0, 1]$ and the symbol Or corresponds to a form of the orientation e.g. Euler angles or quaternions. A nice interpolation between two orientations can be done with the usage of quaternions. In [Kuf04] is examined two approaches regarding this interpolations. The "best" approach seems to be the *Slerp* (spherical linear interpolation). The chapter A.1 in appendix describes the mathematical background for this type of interpolation. The parameter r is computed as follows:

$$r = \frac{D_{P_{current}-P_{cell}}}{(D_{P_{current}-P_{cell}} + D_{P_{target}-P_{cell}})} \in [0, 1] \tag{5.10}$$

where the $D_{P_{current}-P_{cell}}$ and $D_{P_{target}-P_{cell}}$are described before. Obviously if the end effector is closer to the target position the parameter r reaches the one which means that the generated orientation asymptotically reaches the target's end effector orientation. The value of r is influenced also by the $Cell_{size}$. If the $Cell_{size}$ grows, the $D_{P_{current}-P_{cell}}$ may grow also and the quantity r has greater value for the same distance $D_{P_{target}-P_{cell}}$.Another important characteristic of this transformation is that the generated orientation lies between two limits: the target and the current orientation. These two orientations lie surely inside the constraints, like orientation constraints.

Since position and orientation of the end effector are calculated, a configuration of the robot arm is possible to be extracted by the inverse kinematics. That configuration is used as a new q_{center} for the N-Cuboid region. For the rest of this thesis, the symbol $q_{center_{cell}}$ refers to the case where the center configuration is generated by selecting a cell from the neighbor cells. The image 5.7 presents the final step where the mapping from the C-space to workspace and finally back to the C-Space is done.

5.3 Constraints

This section describes the background regarding the additional constraints that a task may include. The $CellBiRRT$ is capable of solving fast tasks with additional constraints and therefore this ability is explained in this section.

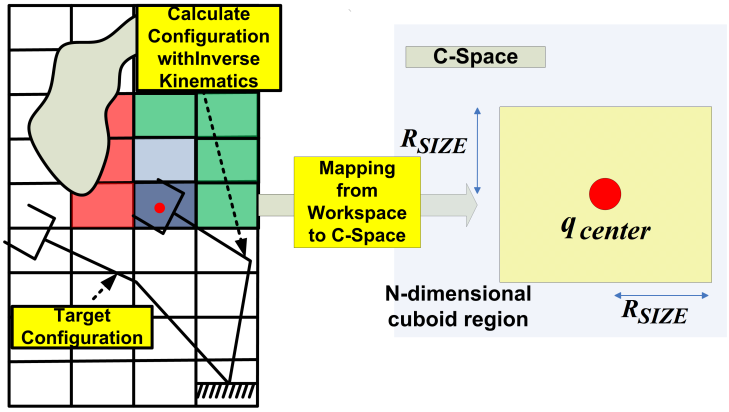


Figure 5.7: Mapping from workspace to C-Space

Constraints except the hardware constraints like joint limits or velocities are also:

- position
- orientation
- force

In this thesis position and orientation constraints are going to be examined. Thus in the workspace the constraints define the available free workspace where the end effector can move:

$$CT_C = \begin{pmatrix} Xmin & X_{max} \\ Y_{min} & Y_{max} \\ Z_{min} & Z_{max} \\ RotX_{min} & RotX_{max} \\ RotY_{min} & RotY_{max} \\ RotZ_{min} & RotZ_{max} \end{pmatrix} \tag{5.11}$$

The 6x2 matrix defines the limits of six dimensional manifold, called CT_C, and it is a subset of the workspace W ($CT_C \subseteq W$). If $\{W\}$ is the world coordinate system and $\{C\}$ the origin of the constraint manifold, the transformation matrix T_C^W defines the location of the manifold in the workspace. An example of a T_C^W is given in image 5.8. For simplicity, it is considered the T_C^W to be equal to the identity matrix (e.g the C lies in the origin). Since equation 5.11 corresponds to the limits of the end effector's location, the constraint manifold CT_q contains all configurations which end effector's location, calculated by the equation 2.13, lies inside the CT_C:

$$CT_q = \bigcup_{y \in CT_C} \{q \in C_{free} | f(q) = \mathbf{y}\} \subset C_{free} \tag{5.12}$$

The f(q) corresponds to the forward kinematics and the **y** corresponds to the location of the end effector. If the T_C^W is not equal to identity, the CT_C is compared with the rotational angles and

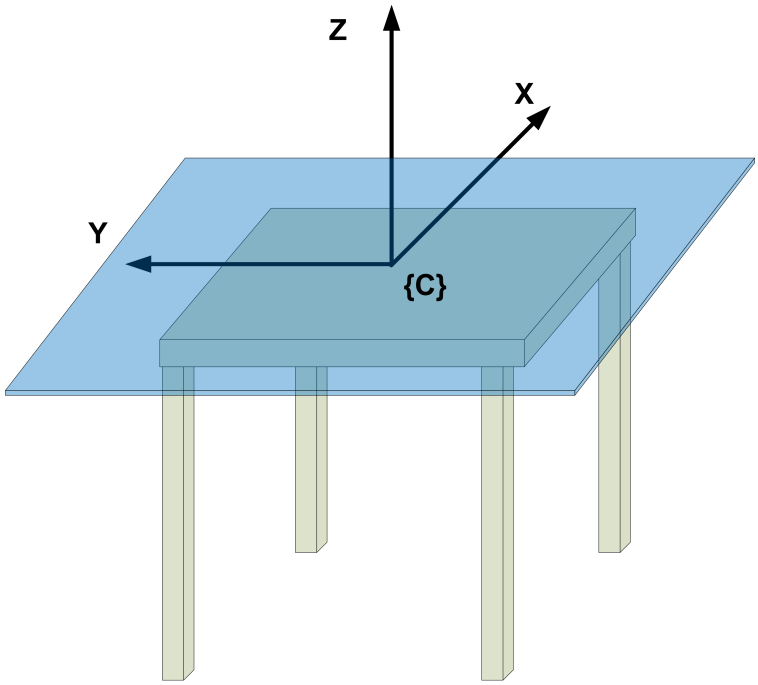


Figure 5.8: Constraint manifold T_C^W example being on the surface of the table. In this example the end effector TCP should lie inside this manifold

the position computed from the frame:

$$T_G^C = (T_C^W)^{-1} \cdot T_G^W \tag{5.13}$$

where $\{G\}$ is the gripper frame and $\{W\}$ is the world coordinate system.

The difficulty for a planner is to find the configurations for which the end effector lies inside the workspace constraints. The *Cells*, as already described before, provide this possibility. Every configuration's $q_{center_{Cell}}$ lies inside the constraints limits. Recall that the orientation of the selected cells is always between the orientation of the start and the goal location. Moreover cells where their position lies outside the constraints are rejected automatically. These are the reasons why cells are used. The location of the selected cell lies always inside the constraints and consequently every $q_{center_{Cell}}$. Moreover the N-Cuboid area with center $q_{center_{Cell}}$ and size R_q contains a lot of configurations that lie inside the CT_q. That is one advantage of using the *Cell* approach: The generated N-Cuboid areas having center $q_{center_{cell}}$ and size R_{size} are partially a part of the CT_q. Mathematical is equal to the expression:

$$R_{cm} = R_q \cap CT_q \neq \emptyset. \tag{5.14}$$

Since the CellBiRRT planner creates random configurations with high probability to be inside the constraint manifold, the combination of cells and N-cuboid regions could be an attractive solution for tasks having constraints. The experimental results prove exactly this statement.

Constraints can be represented with quaternions. The necessary step is to transfer the rotation part to quaternion. This approach is discussed briefly in the appendix B.

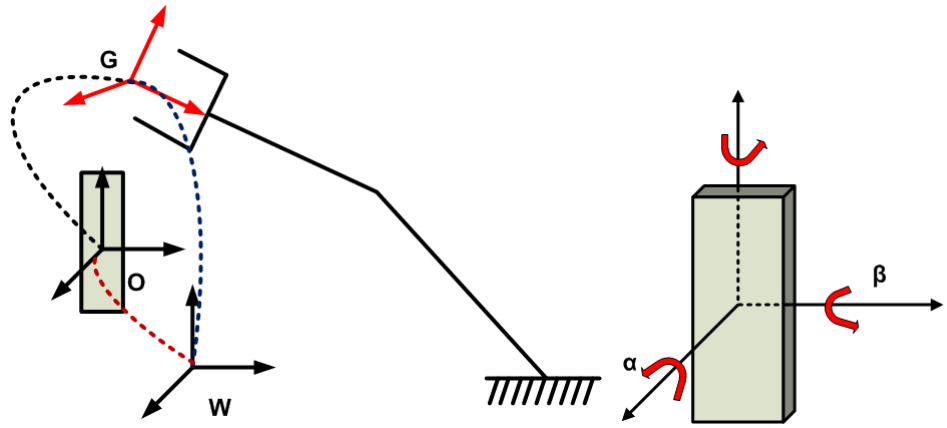
5.3.1 Constraints for gripped objects

Constraints can be extended to the case of having gripped objects. For each object, a grasping frame is assigned. Given $\{G\}$ the gripper coordinate system, $\{W\}$ the world frame and $\{O\}$ the object's frame, a grasping frame T_O^G is defined as:

$$T_O^G = T_W^G \cdot T_O^W \tag{5.15}$$

The frames are illustrated on the figure 5.9.

The frame for the constraint manifold is represented in this sub section as T_O^C. The constraint



(a) Illustration of gripper's , object's and world coordinate's system

(b) Example of α and β tolerance for a bottle

Figure 5.9: Illustration of gripper frame $\{G\}$, object's frame $\{O\}$ and grasping frame T_O^G

in equation 5.11 is extended to object's constraints. The object frame T_O^W at every state of the robot arm can be calculated by the equation 5.15. Instead of using the gripper's frame the object's frame seems to be to more appropriate for such a situation. Given the T_O^W (can be extracted by the equation 5.15) the following mapping to the object's constraint manifold is done:

$$T_O^C = (T_C^W)^{-1} \cdot T_O^W \tag{5.16}$$

From the frame T_O^C the position as well as the orientation are calculated and these values are compared with the matrix CT_C in order to check if the current location violates the constraints.

For instance if a bottle has to be kept up-right down, the T_C^W is equal to identity matrix and the constraints are:

$$CT_{bottle} = \begin{pmatrix} -\infty & \infty \\ -\infty & \infty \\ -\infty & \infty \\ -\alpha & \alpha \\ -\beta & \beta \\ -\pi & \pi \end{pmatrix} \tag{5.17}$$

where α and β are the angles representing the tolerance. For the rest of this thesis if an object is grasped, the object's constraints are considered instead of the gripper's constraints.

5.4 Cell Bi-Directional RRT algorithm

The *CellBiRRT* algorithm is presented analytically in the algorithm 7[FG10b]. The properties of this algorithm are:

- Probabilistic biasing towards a corresponding goal (for the case of backward tree the goal is the start configuration)
- Decomposes 3D Space(only the position) into cells with fixed size (can be adaptive also but it is going to increase the complexity)
- The target configuration used in cell selection is the nearest neighbor to the opposite tree. A configuration($q_{center_{cell}}$) is selected by a set of possible configurations created by inverse kinematics (IK).
- Collision free random configurations are generated inside the N-Cuboid domain with center $q_{center_{cell}}$.
- Probabilistic connecting the two trees

Firstly the algorithm attempts to connect to the starting configuration of the backward tree (within a given probability P_g). Continuously the nearest neighbor to the backward tree is selected as target configuration. The cells are created and one is selected as described earlier. Position and orientation of the end effector for this cell is computed and the inverse kinematics select a configuration as a $q_{center_{cell}}$. If Q_{IK} is the set of all possible collision free solutions from the IK, the CellBiRRT selects the one that satisfies the following criterion:

$$q_{center_{cell}} = \arg\min_{q \in Q_{IK}}(||q - q_{actual}||) \tag{5.18}$$

If the $Q_{IK} = \emptyset$ the cell is set temporally as inactive. In *CellBiRRT* a cell has one of the following statuses:

- **active**

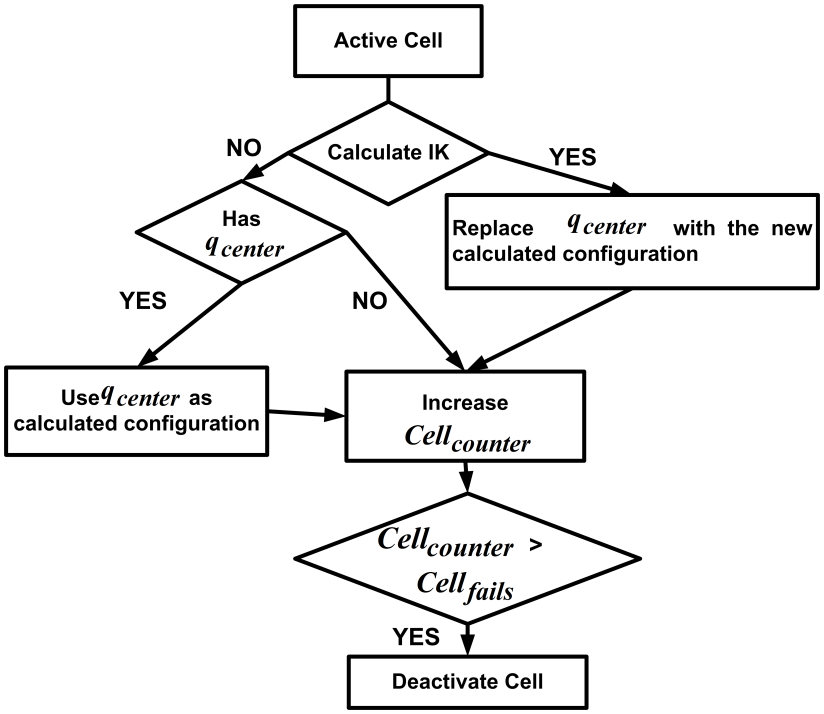


Figure 5.10: Cell managing

- **inactive**

Each cell has a *center configuration* denoted with $q_{center_{cell}}$ and a $Cell_{counter}$ that counts the number of the visits for a cell. The center configuration is the result of IK. The cells store the last computed configuration in order to use it later if that is necessary. The status of a cell is not static but adapted. If a cell is in collision, it is set as *inactive* and it cannot be selected later. A cell is *inactive* also if the $Cell_{counter}$ is over a limit $Cell_{fails}$. If all neighbor cells are examined, it is assumed, that the area is searched extensively and no better solution is able to contribute to the algorithm. The diagram in image 5.10 illustrates the described approach regarding the status management.

If a configuration from IK is not possible to be found, the previous one is used as $q_{center_{cell}}$. If neither a previous one exists nor IK cannot calculate a solution, the algorithm uses the last expanded node as $q_{center_{cell}}$. Actually, the algorithm in this situation is the same as the algorithm 5 in **static** case. When IK algorithm calculates a new configuration, the $q_{center_{cell}}$ is replaced by the new calculated solution.

5.4.1 Extension in case of local minima

An environment may have a set of possible local minima. To recognize a local minima is not easy to be done in C-Space since the mapping from Workspace to C-Space is not one by one (due to redundancy). Although RRT is probabilistic complete, its efficiency is reduced due to local minima

Algorithm 6 $ConnectEfficientWithStep(q_a, q_b, Tree, step)$

```
1: q_s = q_a
2: while (q_s ≠ q_b) do
3:    d = distance(q_s, q_b);
4:    if (d≥ step) then
5:       q_s^new = Steer(q_s, q_b, step/d); {recall the equation 5.1}
6:    else
7:       q_s^new = q_s;
8:    end if
9:    q_new = ConnectEfficient(q_s^new, q_b);
10:   if (q_new == q_b) then
11:      Tree.Add(q_new)
12:      q_s = q_new
13:   else
14:      return LastValidConfiguration
15:   end if
16: end while
17: return q_b
```

and the robot may need time in order to escape from them. As a result the algorithm may become impractical in that case.

In[BKDA06] they introduced a failure counter when an attempt to add a new configuration is failed or the new node does not contribute to a lower cost compared to the parent. The near configuration q_{near} is selected to be the first in the list of configurations sorted by a measure. This measure indicates their chance to reach the goal. In the same paper a nodes failure count is set to the maximum when one of its child nodes is removed from the ranking.

In $CellBiRRT$ the nodes can be disabled in the tree, and that may lead the algorithm to expand in a space far from local minima. Recall that an expansion is done from the nearest neighbor q_{near} towards a random configuration. In $CellBiRRT$ the q_{near} is computed by the nearest neighbor while in [BKDA06] is the result of ranking/ heuristic procedures. If a node is not selected by the nearest neighbor, because the node is deactivated, an expansion cannot be attempted from this node. Another node is selected then. Each node in $CellBiRRT$ has an attribute, called $Node_{counter}$, representing the expansion failures. An expansion fails if the ConnectEfficient(q_A,q_B) returns a new vertex which is close to the beginning one(q_A). If the $Node_{counter}$ exceeds a limit, the node is deactivated and cannot be selected by the nearest neighbor method. Thus the nodes being closer to local minima are going to be deactivated.

5.4.2 $ConnectEfficient$ vs $ConnectEfficient$ with step

If incremental expansion is necessary, the method *Connecteefficient* in CellBiRRT algorithm can be replaced by the *ConnectEfficientWithStep*. The latter calls iteratively the *ConnnectEfficient* method and therefore attempts incrementally to connect two configurations doing small steps.

Each step is added to the corresponding tree. Briefly the approach is described in algorithm 6. The algorithm returns again the last valid configuration. This approach may not improve the performance because more points are going to be inserted into the tree, but it is going to increase the resolution of the tree and it may increase the quality of the path. The notation **CellBiRRTStep** corresponds to the case of incremental expansion.

5.5 Experimental results

This section presents some experimental results done in the LWA3 robotic arm (FRIEND system). The benchmarks are done in simulation mode and the task is to examine the efficiency of the Cell-BiRRT as well as the influence of each parameter. Surely, like in every algorithm, the environment influences from the value of the parameters i.e. in this case the size of Cells ($Cell_{SIZE}$) and the size of the N-Cuboid region (R_q). The second parameter is examined in previous section. Different type of environments, from opened (C_{free} is big) to very close ($C_{obs} >> C_{free}$), are going to examined. The PC for all tests is an Intel Core i5 450M@2.4GHz and the operating system is Linux (Ubuntu 10.04). The results are divided into two parts: with and without constraints. The *CellBiRRT* algorithm is going to be compared with the simpler approach of not existing cells (referred as *CellBiRRT NoCell*). The figure 5.11 shows the task's environment and some examples of solutions. For all experimental results presented in this thesis the C_{Length} corresponds to the path length in C-Space e.g. the root of the square sum of the distances between two configurations.

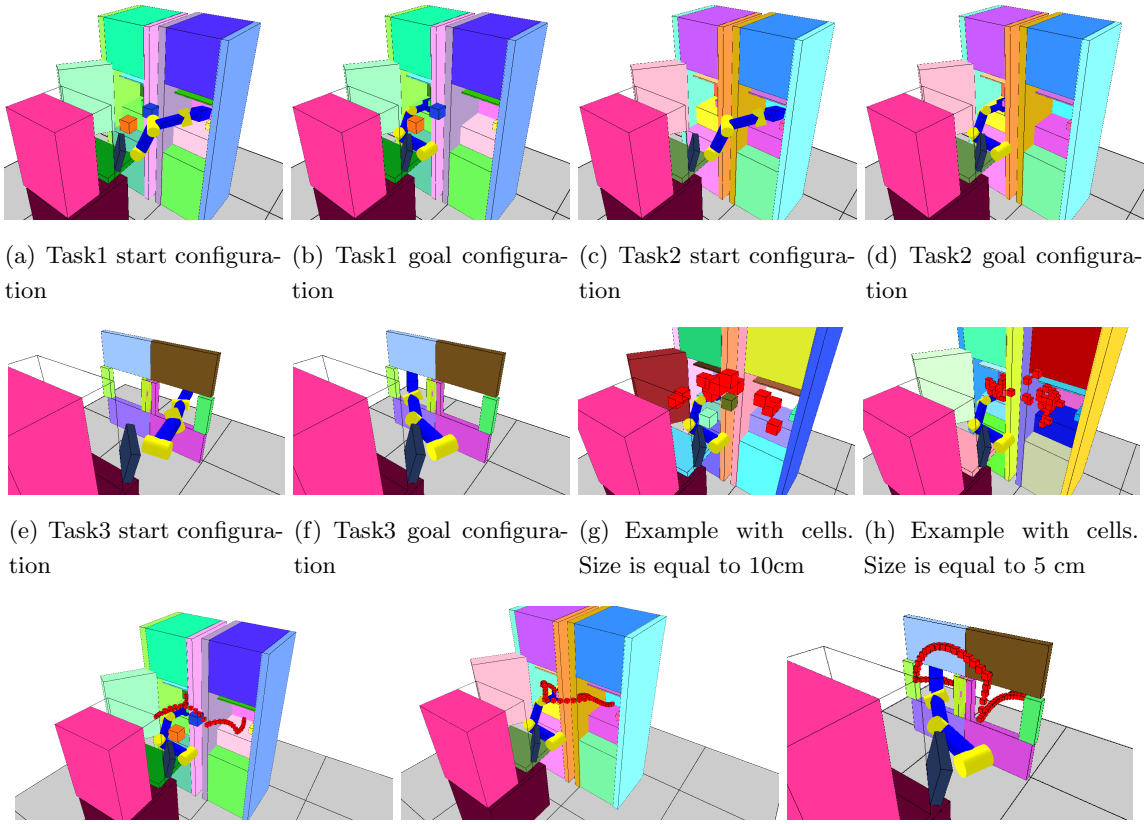
$$C_{Length} = \sum_{k=1}^{M-1} \sqrt{\sum_{i=1}^{N} (q_i^k - q_i^{(k+1)})^2} \tag{5.19}$$

5.5.1 Without Constraints

The three tasks for this setup are presented in figure 5.11. The aim of all tasks are to examine the behavior of the CellBiRRT algorithm with different parameters and in different environments. The Task 1 and Task 3 are cluttered while the Task 2 is less complicated. The robotic system should calculate a collision free path from the start to the goal configuration. The results are presented on tables 5.3- 5.5. The *CellBiRRT* as well as the *CellBiRRTWithStep* are compared. The benchmarks compare the cases between the presence of cells and the free planning without cells.

The table 5.3 shows that the presence of cells do not provide better results for that task. An explanation is that the two virtual cuboid of the Task 1 are between the start and goal configurations, a fact that complicates the calculation of cells that can contribute. The arm should move between the two cuboid and the cells cannot help the planner to provide faster solution. That makes the *CellBiRRT* to be slow for this environment. Free planning, without cells, for this environment provides faster results.

The table 5.4 illustrates clearly that the *CellBiRRT* over performs the simple *CellBiRRT NoCells*. The environment has more free space, compared to the Task 1. The presence of cells



(a) Task1 start configuration (b) Task1 goal configuration (c) Task2 start configuration (d) Task2 goal configuration

(e) Task3 start configuration (f) Task3 goal configuration (g) Example with cells. Size is equal to 10cm (h) Example with cells. Size is equal to 5 cm

(i) Example of resulted path for Task1 with smoothing [GO07] (j) Example of resulted path for Task2 with smoothing [GO07] (k) Example of resulted path Task3 with smoothing [GO07]

Figure 5.11: Three environments as well as experimental results. Task1 and Task2 seems to be similar, but Task1 has more obstacles. All the environments are artificial and cluttered for path planning

accelerates the performance by the factor of two. Moreover the *CellBiRRT* has less C-Length, which implies that the cells may provide solutions with lower length.

The table 5.5 shows the results for the Task 3. This task is the most cluttered one since the robot arm should move from one hole to the second one. The *CellBiRRT* with $Cell_{SIZE}$ equals to 15cm provides the best performance giving 98% success and also faster calculation time. It is noticable that the *CellBiRRT* over performs the *CellBiRRT NoCells*.

The *CellBiRRTStep* may not provide better results for these tests. The slower performance of the *CellBiRRTStep* can be explained by the presence of many configurations in the path, which are produced by the repeated call of *ConnectEfficient*. That may decrease the computational speed. The *CellBiRRTStep* creates more points which may lead to higher computational time.

Planner	$Cell_{size}$ (cm)	Time (msec)	C_{Length}	$3D_{Length}$ (meter)	Success %	Path Configurations
CellBIRRT	5	7871	1688	9.11	100	55.84
CellBiRRT	10	11609	1742	9.35	100	54.93
CellBiRRT	15	5893	1649	8.65	100	50.78
CellBiRRT	NO CELLS	3285	1384	8.57	100	53.32
CellBiRRTStep	5	8319	1765	9.15	100	177.72
CellBiRRTStep	10	11107	1726	8.71	100	172
CellBiRRTStep	15	6781	1447	7.58	100	144.57
CellBiRRTStep	NO	4800	1564	8.92	100	160.8

Table 5.3: Comparison between different $Cell_{size}$ value for Task1 (see figure 5.11). The tolerance $R_{size}$was 25deg (without shifting), the $P_{ConnectTrees} = 1.0$, the $P_g = 0.0$ (no goal bias) and the step equals to 11deg (*CellBiRRTStep*). Average results are from 100 trials with maximum allowable time 60sec. The results are without smoothing.

Planner	$Cell_{size}$ (cm)	Time (msec)	C_{Length}	$3D_{Length}$ (meter)	Success %	Path Configurations
CellBIRRT	5	1776	1072	5.51	100	33.81
CellBiRRT	10	1642	1133	5.36	100	32.33
CellBiRRT	15	1728	1105	5.52	100	32.79
CellBiRRT	NO CELLS	2885	1231	7.94	100	51.83
CellBiRRTStep	5	2130	1046	5.03	100	103.76
CellBiRRTStep	10	2059	1039	4.84	100	102.4
CellBiRRTStep	15	2095	1029	4.85	100	100.56
CellBiRRTStep	NO CELLS	3548	1357	8	100	138.57

Table 5.4: Comparison between different $Cell_{size}$ value for Task2 (see figure 5.11). The tolerance $R_{size}$was 25deg (without shifting), the $P_{ConnectTrees} = 1.0$, the $P_g = 0.0$ (no goal bias) and the step equals to 11deg (*CellBiRRTStep*). Average results are from 100 trials with maximum duration 60sec. The results are without smoothing.

5.5.2 With Constraints

This subsection presents experimental results when additional constraints (position and orientation) are present. Surely the tasks are difficult since the arm cannot move in the whole C_{free}.

5.5.2.1 Task 1

The first environment is illustrated on figure 5.12. The task is to move the robot arm from the start to the goal configuration avoiding the bottle in between. Several tests are done for this environment. One group of test is done with orientation constraint of ± 4 degrees in each direction (X,Y, and Z).

Planner	$Cell_{size}$ (cm)	Time (msec)	C_{Length}	$3D_{Length}$ (meter)	Success %	Path Configurations
CellBIRRT	5	51479	2655	16.62	72	92.68
CellBiRRT	10	40832	2324	14.11	94	80.94
CellBiRRT	15	28716	2611	16.10	98	88.16
CellBiRRT	NO CELLS	59297	4608	34.41	51	169.68
CellBiRRTStep	5	50046	2653	16.44	68	270
CellBiRRTStep	10	45114	2558	15.16	82	261
CellBiRRTStep	15	30717	2492	14.72	93	253
CellBiRRTStep	NO CELLS	59498	4825	34.89	54	499.26

Table 5.5: Comparison between different $Cell_{size}$ value for Task 3(see figure 5.11). The tolerance $R_{size}$was 25deg (without shifting), the $P_{ConnectTrees} = 1.0$, the $P_g = 0.0$ (no goal bias) and the step equals to 11deg ($CellBiRRTStep$). Average results are from 100 trials with maximum allowable time 120sec. The results are without smoothing.

The second group of experiments tests orientation together with position constraint. The robot is allowed to move up-down within a tolerance of ± 5mm. In summary, the tasks are the following:

- Constant orientation within a tolerance of 4 degrees in all directions (Task 1a). The Constraint matrix for this environment is the following:

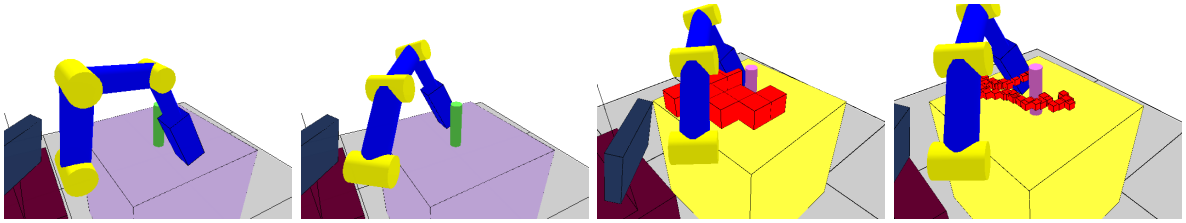
$$CT_1 = \begin{pmatrix} -\infty & \infty \\ -\infty & \infty \\ -\infty & \infty \\ -\alpha & \alpha \\ -\alpha & \alpha \\ -\alpha & \alpha \end{pmatrix} \tag{5.20}$$

  where α is the orientation tolerance.

- Constant orientation (4 deg)and tolerance on Z axes (up-down) ±0.005 m (Task 1b)

$$CT_2 = \begin{pmatrix} -\infty & \infty \\ -\infty & \infty \\ -z_a & z_a \\ -\alpha & \alpha \\ -\alpha & \alpha \\ -\alpha & \alpha \end{pmatrix} \tag{5.21}$$

For all tasks, tests with three different $Cell_{SIZE}$ are examined. The values are 5cm , 10cm and 15 cm. The case where cells do not exist is also examined. A performance comparison is done. The results for Task 1a are in the table 5.6. The results for the task 1b are presented on the table 5.7.



(a) start configuration (b) goal configuration (c) Cell generation for task 1b (d) Path result for task 1b with smoothing [GO07]

Figure 5.12: Constraint Task 1. The robot should avoid the bottle. The robot arm moves around it.

Planner	$Cell_{size}$ (cm)	Time (msec)	C_{Length}	$3D_{Length}$ (meter)	Success %	Path Configurations
CellBIRRT	5	1458	360.3	1.67	100	45.91
CellBIRRT	10	688	298	1.36	100	34.02
CellBiRRT	15	484	292	1.28	100	28.97
CellBiRRT	NO CELLS	2773	328	1.73	100	45.84
CellBiRRTStep	5	1383	387	1.68	100	54.79
CellBiRRTStep	10	710	323	1.43	100	43.22
CellBiRRTStep	15	401	306	1.3	100	38.31
CellBiRRTStep	NO CELLS	2872	380	1.88	100	55.11

Table 5.6: Comparison between different $Cell_{size}$ value for Constraint Task 1a. The tolerance $R_{size}$was 25deg (without shifting), the $P_{ConnectTrees} = 1.0$, the $P_g = 0.0$ (no goal bias) and the step equals to 11deg ($CellBiRRTStep$). Average results of 100 trials with maximum computation time 60sec. The results are without smoothing.

5.5.2.2 Task 2

The robot arm in this task should calculate its path while an object is grasped in the end effector. For this task the object being grasped is a cylindrical object like a bottle or a glass. The object should be shifted from the fridge and should be placed on the platform in front of the user. The main challenge here is the object to be manipulated in a such a way so that no water drops will come outside. That requires that the object is going to be kept up-right down within a tolerance . The figure 5.13 presents the task in the virtual environment.

In this scenario the influence of the orientation tolerance is going to be examined. The con-

Planner	$Cell_{size}$ (cm)	Time (msec)	C_{Length}	$3D_{Length}$ (meter)	Success %	Path Configurations
CellBIRRT	5	8108	444	1.77	100	82.51
CellBIRRT	10	3278	337	1.40	100	59.07
CellBiRRT	15	5103	370	1.57	100	62.58
CellBiRRT	NO CELLS	25761	342	1.73	96	68.86
CellBiRRTStep	5	15716	522	2.066	87	99.81
CellBiRRTStep	10	10520	437	1.75	88	81.71
CellBiRRTStep	15	12225	432	1.77	96	79.42
CellBiRRTStep	NO CELLS	34112	513	2.36	52	103.84

Table 5.7: Comparison between different $Cell_{size}$ value for Constraint Task 1b. The tolerance $R_{size}$was 25deg (without shifting), the $P_{ConnectTrees} = 1.0$, the $P_g = 0.0$ (no goal bias) and the step equals to 11deg ($CellBiRRTStep$). Average results of 100 trials with maximum computation time 60sec. The results are without smoothing.

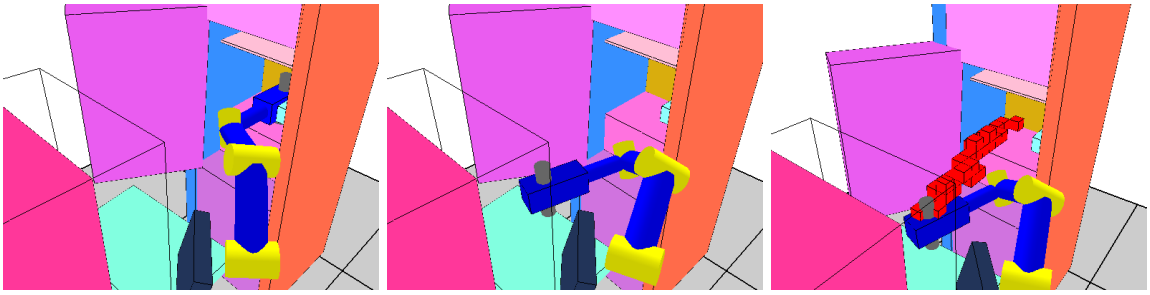
Planner	$Cell_{size}$ (cm)	Time (msec)	C_{Length}	$3D_{Length}$ (meter)	Success %	Path Configurations
CellBIRRT (4 deg)	5	1727	720	3.13	100	57.9
CellBIRRT (2 deg)	5	3245	734	3.07	100	92.7
CellBiRRT (4 deg)	NO CELLS	2517	615	3.6	100	53.14
CellBiRRT (2 deg)	NO CELLS	5684	576	3.32	100	77.75

Table 5.8: Comparison between different $Cell_{size}$ value for Constraint Task 2. The tolerance $R_{size}$was 25deg (without shifting), the $P_{ConnectTrees} = 1.0$, the $P_g = 0.0$ (no goal bias). Average results of 100 trials with maximum computation time 60sec. The results are without smoothing.

straints are:

$$CT_{bottle} = \begin{pmatrix} -\infty & \infty \\ -\infty & \infty \\ -\infty & \infty \\ -\alpha & \alpha \\ -\alpha & \alpha \\ -\pi & \pi \end{pmatrix} \tag{5.22}$$

Two cases for angle α are tested: (a) 4 deg and (b) 2 deg. Experimental results with and without cells are presented on table 5.8.



(a) start configuration (b) goal configuration (c) Cell generation

Figure 5.13: Constraint Task 2. The robot should should place the bottle on the platform in front of the user. The bottle should be kept up-right own within a tolerance

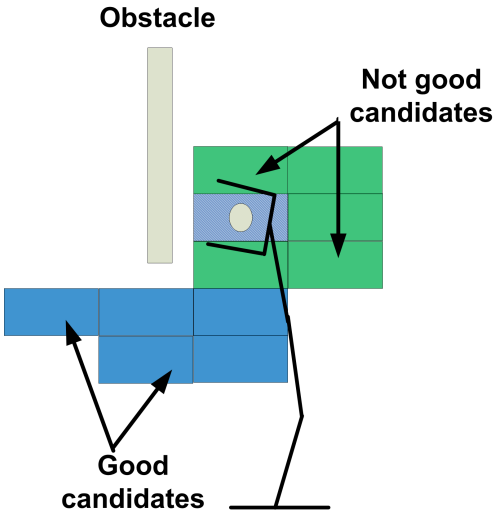


Figure 5.14: Example with bad and good cell candidates

5.6 Discussion

This chapter described the designing a sampling based algorithm which has the following properties:

- configures the sampling space with the usage of N-Cuboid areas
- replacing the N-Cuboid areas to regions being closer to the target

These two properties may improve significantly the speed of the algorithm. The experiments show clearly that N-Cuboid areas over perform the simple approach of RRT, while the cells give additional boost in the final performance. That is explained since the algorithm does not create samples to unnecessary C_{free} space and consequently it does not lose time due to that. The size of the N-Cuboid areas and the $Cell_{SIZE}$ influence the performance. Very small values may provide

results with lower path length but the penalty is the computation time. A suggestion could be to have an adaptable SIZE of the cuboids. The SIZE should be increased in case that the R_q has less free space than the size of the R_q. That may guide the samples to be generated in the C_{free} space and far from obstacles.

The $Cell_{SIZE}$ as well as the position of the cell play an important role. For instance, consider the example in figure 5.14. In this example the green cells are not considered as good candidates since there is close the obstacle . The algorithm needs to examine these cells and the manipulator may need time for that. That is the reason why the cells may delay the performance in such a situation. Moreover if the division is big (e.g. the cells are small) then more time is needed since more cells have to be examined. That is the reason why for big cells(e.g. 15cm) the algorithm behaves faster compared to the case where the $Cell_{SIZE}$ is only 5 cm. Nevertheless, the CellBiRRT performs much better if cells are present since they guide the manipulator to follow the right way.

Another important characteristic is that the CellBiRRT with cells improve the quality of the path mostly in Cartesian space. That may be explained since the cells try to bring the trees close to each other in Cartesian space. That is a goo advantage of using cells.

Regarding the performance in constraint manifold tasks, the $CellBiRRT$ with cells can provide acceptable results. The computation time for all tasks is deliverable and compared to standard cases (standard Bidirectional RRT), the algorithm with this cell decomposition performs much better.

Summarizing, the big advantage of the proposed approach is the following: without making modifications or changes in the algorithm, the method $CellBiRRT$ can be used for environments with and without constraints in the pose of the end effector (position and orientation). The CellBiRRT delivers path in a reasonable amount of time.

Algorithm 7 Cell-Bi directional RRT

Require: $T1$,$T2$ trees,q_{start}, q_{goal} start/goal configurations, R_{size} the size of N-Cuboid region, $Cell_{size}$ the size of cells , $Pg \in [0,1]$ and $P_{ConnectTrees} \in [0,1]$ probability values, $MaxTrials$ the maximum trials to expand towards a random configuration.

1: $T1.Init(q_{start})$, $T2.Init(q_{goal})$
2: Decompose3DSpaceIntoCells($Cell_{size}$)
3: **loop**
4: a=Ran(0,1)
5: **if** (a ≤ Pg) **then**
6: q_{last}=ConnectEfficient(T1.LastNode(),T2.InitialNode());
7: $T1.Add(q_{last})$
8: Path = ExtractSolution()
9: **end if**
10: $q^2_{near} = T2.find\_nearest(q_{last})$
11: $Cell = CalculateCell(q_1, q^2_{near})$
12: $q_{center_{cell}} = CalculateCenterConfigWithCell(Cell)$
13: **if** ($q_{center_{cell}} == NULL$) **then**
14: $q_{center_{cell}} = q_{last}$
15: **end if**
16: GenerateNCuboidRegion($q_{center_{cell}}$,R_{SIZE})
17: iTrial=0;
18: **while** ($iTrial \leq MaxTrials$) **do**
19: q_{rnd}=CreateRandomConfigInsideNCuboidRegion()
20: $q^1_{near} = T1.find\_nearest(q_{rnd})$
21: $q_2 = ConnectEfficient(q^1_{near}, q_{rnd})$
22: **if** ($q_2 != q^1_{near}$) **then**
23: $T1.Add(q_2)$
24: $q_{last} = q_2$
25: break;
26: **end if**
27: iTrial++;
28: **end while**
29: **if** (Ran(0,1)≤ $P_{ConnectTrees}$) **then**
30: $q^2_{near} = T2.find\_nearest(q_{last})$
31: $q_3 = ConnectEfficient(q_{last}, q^2_{near})$
32: **if** ($q_3 == q^2_{near}$) **then**
33: $T1.Add(q_3)$
34: Path = ExtractSolution()
35: **end if**
36: $T1.Add(q_3)$
37: $q_{last} = q_3$
38: **end if**
39: $SWAP(T1, T2)$
40: **end loop**

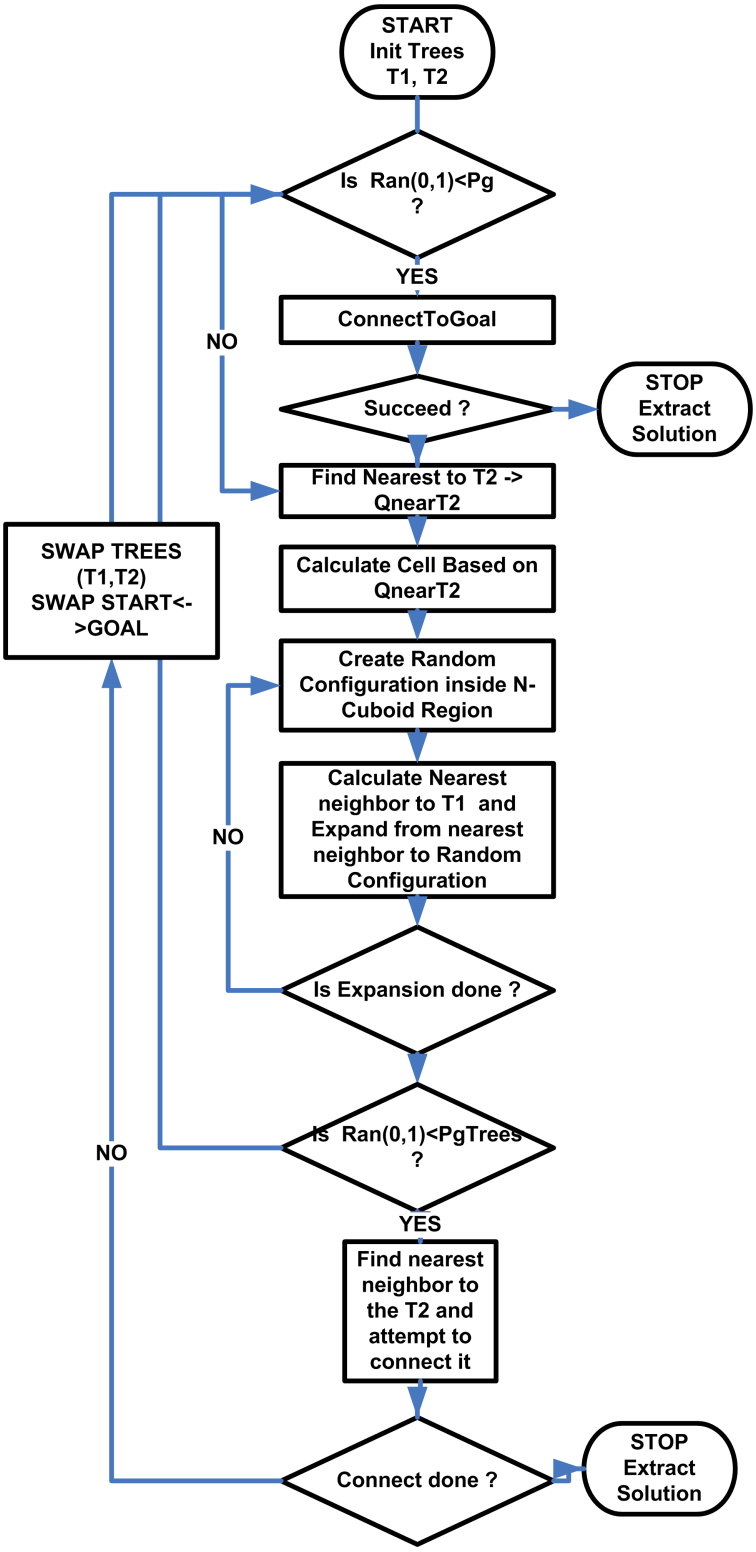


Figure 5.15: CellBiRRT flow chart

Chapter 6

Sampling based motion planning algorithms without goal configuration

This chapter deals with planners where the goal configuration is not a necessary input. The CellBiRRT described in previous chapter and in algorithm 7 requires the calculation of a goal configuration in order to be able to work. If inverse kinematics fail to accomplish this task the algorithm fails also. However, if inverse kinematics (IK) do not solve the problem it is not guaranteed that there is no solution. That is explained due to the redundancy. Each IK algorithm delivers a set of possible solutions and the discretization depends on a given resolution. If the resolution is not high enough, the IK algorithm may fail although with other resolution it may be able to deliver a solution. For these reasons the challenge in this chapter is to develop several algorithms that do not require goal configuration and are complete.

In this chapter the following algorithms are going to be described:

- $RRT - J_{wln}$ without cells
- $RRT - J_{wln}$ with cells
- $RRT - IK$ without cells
- $RRT - IK$ with cells
- *CartesianRRT*

The common characteristic of the algorithms are :

- they keep the properties of the RRT
- they use analytical (the already described KCC library [IG97, IG98, IG00]) or recursive Euler-Newton method for solving inverse kinematics (using Jacobian matrix)
- they are sampling approaches

- they do not require goal configuration but a goal frame of the end effector.

The main algorithm for all approaches described in this chapter is presented in algorithm 8.

Algorithm 8 Basic algorithm for single tree RRT with/without inverse kinematics

Require: $Pg \in [0, 1]$
1: T is the trees, q is configuration,bool ConnectToGoal
2: $T.Init(q_{start})$
3: **loop**
4: **if** $(rand() < Pg)$ **then**
5: ConnectToGoal = **ExpandTowardsTheGoal()**;
6: **if** $(ConnectToGoal ==$ true$)$ **then**
7: ExtractPath();
8: **end if**
9: **end if**
10: $ExpandRandomly(T, T.GetLastNode())$;
11: **end loop**

In this pseudo-algorithm there are two functions going to be discussed later:

- ExpandTowardsTheGoal()
- ExpandRandomly()

Each presented approach modifies these methods. The common to all approaches are the random expansion and the probabilistic expansion towards the goal (*ExpandTowardsTheGoal*). The second guarantees that the planner is biased towards the goal location in order to return a result. The expansion is done either using a jacobian based approach or an inverse kinematics(analytical) solver. Both situations are going to be discussed and compared.

The method *ExpandRandomly* is very similar to the one described in section 5.1. The main difference is that now there is no bi-directional trees and the cells are generated based on the target frame (recall that in CellBiRRT the cells are selected based on the nearest node of the opposite tree). Briefly the method is revised in algorithms 9 and 10. The method *ExpandRandomly* seperates the two cases: without and with cells. That influences the total performance. For both cases each new configuration is added to the tree. The $ExpandRandomlyWithNCuboid$ is the same like in previous chapter. The algorithm uses the R_q (N-Cuboid region) area in order to create a random configuration. The *ConnectEfficient* procedure, like the $CellBiRRT$, can be done incrementally if intermediate points are needed (use $ConnectEfficientWithStep$ in this case).

6.1 RRT-J_{wln} with or without cells

The $RRT - J_{wln}$ belongs to the group of planner where a single tree is going to be extended and inverse kinematics are not present or are too complicated to be computed. For that reason Jacobian is used and numerical solution is applied to solve the inverse kinematics.

Algorithm 9 *ExpandRandomly*(T, q_{cur})

Require: q_{cur} is the current node,T is the tree,P_{target} is the target frame

1: **if** (no cells) **then**
2: q_{ext} = ExpandRandomlyWithNCuboid(q_{cur})
3: **else**
4: $P_{cur} = ForwardKinematics(q_{cur})$
5: $BestCell = SelectCell(P_{cur})$ {Select the cell having target frame the P_{target}}
6: $q_{generated} = CalculateInvKinematic(BestCell)$
7: q_{ext}=ExpandRandomlyWithNCuboid($q_{generated}$)
8: **end if**
9: T.add(q_{ext})

Algorithm 10 *ExpandRandomlyWithNCuboid*(q_{cur})

1: CREATE-N-CUBOID(q_{cur})
2: q_{rand} ←CREATE-RANDOM-CONFIGURATION()
3: q_{near} ←FIND-NEAREST-NEIGHBOUR(Tree,q_{rand})
4: q_{expand} ←ConnectEfficient($q_{near} -> q_{rand}$)
5: return q_{expand}

In literature several groups have been worked with this kind of planners. All of them have a common specification: The tree either explores the C-Space or moves the robot towards the goal location (recall that here goal configuration is not available). For instance in [VWFS07] Jacobian transpose (J^T) has been applied in order to "pull" the end effector of the robot to the goal. Using J^T has a significant drawback which is the convergence speed. More recent work [VBA+09] improves the previous algorithm and uses the jacobian pseudo inverse (J^+). The Jacobian pseudo inverse converges faster to a solution compared to the simple transpose method.

The $RRT - J_{wln}$ approach is first illustrated in [FG11b]. The main difference compared to the state of the art approaches are:

- the weighted least norm is used
- N-Cuboid domains is used
- Cells are used

The weighted least norm (WLN), presented in [CD95], is briefly a method for calculating the inverse kinematics having additional constraints like joint limits and it is based on the jacobian pseudo inverse approach. Given $d\theta$ a small joint displacement, J the jacobian matrix and dx the end effector displacement (position and orientation), the next state θ_{new} can be calculated as follows:

$$\theta_{new} = d\theta + \theta_{old} = W^{-1}J^T[J \cdot W^{-1}J^T]^{-1} \cdot dx + \theta_{old} \tag{6.1}$$

Let notice here that if W is identity matrix the equation 6.1 is the same as having the pseudo inverse approach. The W matrix is an NxN matrix, where N is the number of joints of the robot.

In [CD95] they proposed the joint limits criteria

$$H(\theta) = \sum_{i=1}^{N} \frac{(\theta_{i,max} - \theta_{i,min})^2}{4 \cdot (\theta_{i,max} - \theta_i)(\theta_i - \theta_{i,min})} \tag{6.2}$$

and the matrix W is equal to:

$$W = \begin{pmatrix} w_1 & 0 & ... & 0 \\ 0 & w_2 & ... & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & w_n \end{pmatrix} \tag{6.3}$$

where the w_i element of the matrix equals to:

$$w_i = \begin{cases} 1 + |\frac{\partial H(\theta)}{\partial \theta_i}| & \text{if } \Delta|\frac{\partial H(\theta)}{\partial \theta_i}| \geq 0 \\ 1 & \text{if } \Delta|\frac{\partial H(\theta)}{\partial \theta_i}| < 0 \end{cases} \tag{6.4}$$

The above equations imply that when the robot reaches the joint limits, it is forced to avoid them because the $\Delta|\frac{\partial H(\theta)}{\partial \theta_i}|$ has a big value.

Using the above approach, and denoting as P to be a frame, Q a set of configurations and q a single configuration, the method $ExpandTowardsTheGoal$ has the content described in algorithm 11.

Algorithm 11 $ExpandTowardsTheGoal_{JWLN}$

Require: Pg$\in [0,1]$,q_{last} the last expanded node,f is computation of forward kinematics

```
1: P_target ←SelectTargetFrame
2: a=rand(0,1)
3: if (a ≤ Pg) then
4:     q_last= ExpansionWithJacobian(q_last, P_target,Tree);
5:     if (f(q_last) == P_target) then
6:         return PATH;
7:     end if
8: end if
```

The method $ExpansionWithJacobian$ makes iterative progress towards a target frame. The figure 6.1 shows how the function $ExpandWithJacobian$ works. The controller influence the behavior of control loop e.g. the speed of convergence. In our case the controller is proportional. Each new configuration is checked for validity and it is inserted into the tree. The progress is done with small steps and continues till the goal location is reached within a tolerance ϵ. The algorithm 12 presents the described procedure.

The method $Collision$ calls the method $ConnectEfficient$ and returns true if the connection is done successfully.

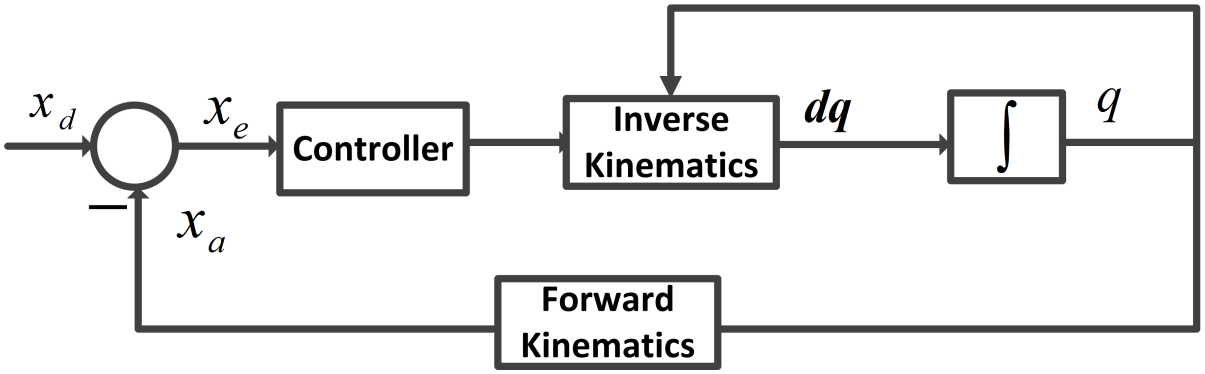


Figure 6.1: Iterative expansion till the desire location is reached

Algorithm 12 Expansion with Jacobian(q_{cur},P_{target},Tree)

Require: J normal Jacobian Matrix, q is configuration,*Step* is the minimum allowable accuracy. Under this value the expansion terminates with success

1: $Tree$ actual tree ,P is a frame
2: q_{temp}=q_{cur}
3: **repeat**
4: $P_{cur} \leftarrow ForwardKinematics(q_{temp})$
5: $\Delta P = P_{target} - P_{cur}$
6: $J = Computejacobian(q_{temp})$
7: $J_{WLN} = ComputeWLN(J)$ {as the equation 6.1 is proposing}
8: $\Delta q = J_{WLN} \cdot LimitStep(\Delta P)$ {a small differential step. The new configurations are incrementally created based on the last computed configuration }
9: $q'_{temp} = q_{temp} + \Delta q$
10: **if** $(Collision(q_{temp}, q'_{temp}) = false\ AND\ IsInsideLimits(q_{temp}) = true)$ **then**
11: Tree.Add(q_{temp})
12: **else**
13: break;
14: **end if**
15: **until** $(\Delta T \leq Step)$
16: return Tree.LastNode()

6.2 RRT-IK with/without cells

The RRT-IK(with / without cells) follows the same structure, but it requires an analytical/geometrical solution for the IK algorithm. The difference compared to $RRT - J_{WLN}$ is in the method *Expand-TowardsTheGoal* where for this case is replaced by the algorithm 13.

The important part of the algorithm 13 is the *Interpolation*. The interpolation should fulfill some requirements which are:

- ensure that the sampling is done with good resolution (too big resolution requires more steps)

Algorithm 13 *ExpandTowardsTheGoal*$_{IK}$(q_{cur},T_{target},Tree)

Require: Pg$\in$ [0, 1],q_{last} the last expanded node,f is computation of forward kinematics

```
1: P_target ←SelectTargetFrame
2: a=rand(0,1)
3: if (a ≤ Pg) then
4:    q_last= InterpolateIK(q_last,P_target,Step_pos,Step_Or,Tree);
5:    if (f(q_last) == P_target) then
6:       return PATH;
7:    end if
8: end if
```

- ensure that the interpolation is linear

The first requirement is difficult to be calculated since even small steps in cartesian space do not ensure small steps in C-Space. The intermediate steps, if one exists, are neglected by the $ConnectEfficient$. A small step value like 1 cm is good enough giving good performance also.

The second requirement is very important. The straight line between two points is sampled and each sample has a specific distance from the previous one. There is one difficulty which is that the 6D Workspace (three for position and three for orientation) has not a unique unit. The position's measure is in meter and the orientation is measured in degrees. For that reason the interpolation is splitted into two parts: one for the position and another one for the orientation. Quaternions have been applied for the linear rotational interpolation(*Slerp*).

The method calculates the number of samples (N_{pos}) for the position and the number of samples for the orientation (N_{Or}) and the biggest one is taken as a common sampling parameter (called *MaxSteps*). With this approach the interpolation remains synchronized (both are starting and finishing at the same time) and linear since both sub-interpolations are linear too.

The *Step* in algorithm 14 has two parts, the orientation ($Step_{Or}$) and the position part($Step_{pos}$), and is equal to:

$$Step = \begin{cases} ||Pos_q - Pos_{target}||/MaxSteps \\ ||Or_q - Or_{target}||/MaxSteps \end{cases} \tag{6.5}$$

The interpolation continues till a point is not collision free. Every new node is added to the tree. The inverse kinematics function F^{-1} computes a set of configurations Q and one is selected by the formula:

$$q = SelectConfig(Q, q_{base}, A) = \arg\min_{q \in Q}(A \cdot ||q - q_{base}|| + (1 - A) \cdot \frac{1}{q.GetMinDistance()}) \tag{6.6}$$

The $A \in [0, 1]$ is normally one or zero. The balanced function selects configuration that is closer to a given configuration(q_{base}) or the one that has bigger clearance (distance from the obstacles). For faster computations the A is equal to one.

Algorithm 14 InterpolateIK(q,T_{target},$Step_{pos}$,$Step_{Or}$,Tree)

Require: $F(q)$ forward kinematics,$F^{-1}(P)$ inverse kinematics, Orientation is expressed on quaternion, P is a frame

PPos, Or= F(q)
$PosSteps$=$||Pos_q - Pos_{target}||/Step_{pos}$
$OrSteps$=$||Or_q - Or_{target}||/Step_{Or}$ {refer the appendix A.2 for the quaternion distance}
$MaxSteps$= max($PosSteps$,$OrSteps$);
$q_{cur} = q$
while ($F(q_{cur})$!= P_{target}) **do**
    $P_{cur} = F(q_{cur})$
    $Pos'_{cur} = Pos_{cur} + Step_{pos}$
    $Or'_{cur} = Or_{cur} + Step_{or}$
    $P'_{cur} = (Pos'_{cur}, Or'_{cur})$ {simply join the position and orientation to a frame}
    $Q = F^{-1}(P'_{cur})$
    $q'_{cur} = SelectConfig(Q)$
    if ($CollisionFree(q_{cur}, q'_{cur})$== FALSE) **then**
        break
    end if
    Tree.add(q'_{cur})
    $q_{cur} = q'_{cur}$
end while

6.3 Cartesian RRT Planner

The Cartesian RRT planner, presented in algorithm 15, is, like the rest of the algorithms, a forward approach. It combines the normal RRT idea, but the applied space is the Workspace - W. The planner attempts each step to expand lying on a straight line between two points in 6D workspace (three dimension for position and three for orientation). The interpolation is done like previously, so it is not going to be described again. Analytical inverse kinematic algorithm and not numerical solutions (Jacobian based approaches) has been used in order to compute a set of configurations for a given position and orientation of the end effector.

The challenge of this approach is to be able to create uniform random points in workspace. The easiest approach is to sample the position and the orientation part separately (each Euler angle individually). The Euler angles however are not optimal for creating uniform rotations and may cause some difficulties that should be overcomed[Kuf04]. For instance a simple sampling approach may concatenate the samples to the poles of the SO(3) space (SO(3) is the 3 dimensional space of the rotations). A nice approach for calculating fast random rotations is done in [Arv92b]. In [Kuf04] they presented a method to generate uniform rotations based on Euler angles. In this thesis the idea of Ken Schoemake [Sho92] is applied and therefore quaternions are used. The idea is to compute the uniform random axes v and the angle θ to generate equivalent uniform quaternions (recall the basic equations of quaternions). The rotational matrix is computed by converting the quaternion back to 3x3 matrix. This method utilizes three intermediate random variables to compute four quaternion parameters that map uniformly to the unit sphere in four dimensions. The algorithm

16 presents this approach. The figure 6.2 shows the sampled points generated by the proposed approach. Concluding, the random location is generated as follows:

$$Location_{rnd} = \begin{cases} \text{UniformRandomPosition(x,y,z)} \\ \text{UniformRandomRotation(Roll,Pitch,Yaw)} \end{cases} \tag{6.7}$$

where the rotation part is calculated by the algorithm 16.

Algorithm 15 Cartesian RRT Planner

Require: q_{start},Tree, GoalTCP, Pg∈[0,1] constant

```
Tree.add(q_start)
loop
    a ∈ [0,1] Random Number
    if a ≤ Pg then
        q_last ← Tree.LastConfig()
        if Expand(q_last, GoalTCP) = true then
            trajectory ← CreateTrajectory(Tree)
            print Planning was successful !
        end if
    else
        RandTCP ← GenerateRandomTCP()
        q_rand ← InvKinematics(RandTCP)
        Q_near ← Tree.FindKNearst(q_rand)
        q_near ← FindMinCost(Q_near) {the configuration is selected using the formula argmin_{k∈Q}(A * k.CostToCome() + (1 − A) * k.CostToGo())}
        q_exp= InterpolateIK(q_near, RandTCP)
        if q_exp! = q_near then
            Tree.AddNode(q_exp)
        end if
    end if
end loop
```

Algorithm 16 Generating uniform random rotation

Result: uniform random quaternion Q = (w, x, y, z)

s = rand();

$\sigma_1 = \sqrt{1-s}$

$\sigma_2 = \sqrt{s}$

$\theta_1 = 2 \cdot \pi \cdot rand()$

$w = cos(\theta_2) \cdot \sigma_2$

$x = sin(\theta_1) \cdot \sigma_1$

$y = cos(\theta_1) \cdot \sigma_1$

$z = sin(\theta_2) \cdot \sigma_2$

return Q(w,x,y,z)

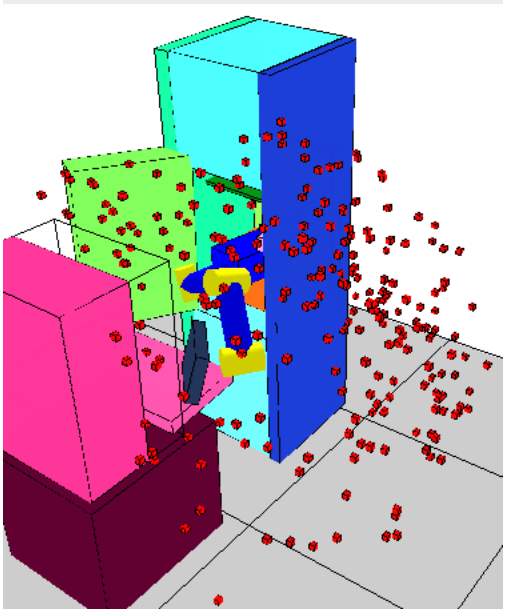
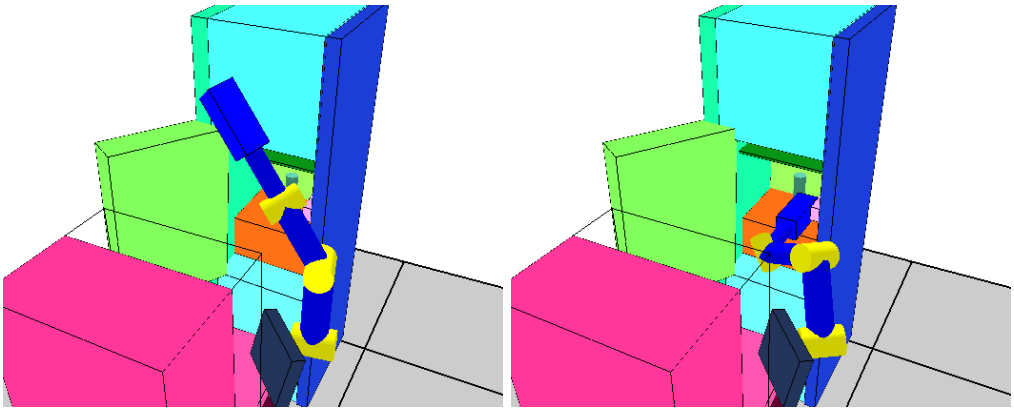


Figure 6.2: Random Sampling inside the Workspace of the robot arm based on algorithm 16

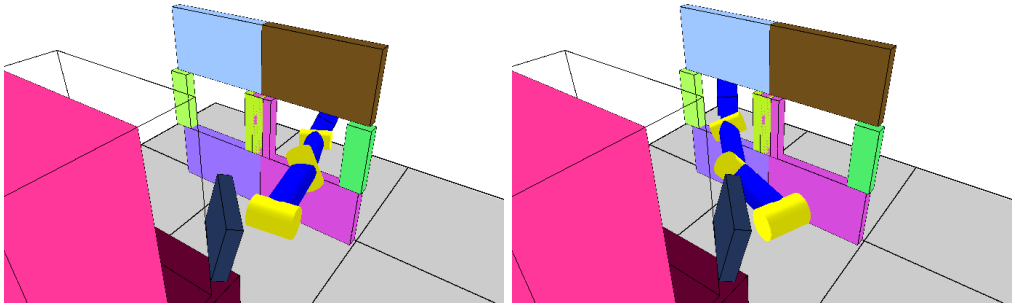


(a) start configuration (b) possible goal configuration

Figure 6.3: Task 1 with start and a (possible) goal configuration

6.4 Experimental results

Like previously , this section includes the experimental results regarding the described planners of this chapter. The experimental setup (e.g. computer system) is the same as in previous chapter. For all methods the expansion is done incrementally with small steps and for that reason it is expected that the approaches generate many intermediate points. That may cause a difficulty when the path is going to be executed by the robot arm. The difficulty exist when the arm needs



(a) start configuration (b) possible goal configuration

Figure 6.4: Task 2 with start and a (possible) goal configuration. It is the same as in figure 5.11

Planner	$Cell_{size}$ (cm)	Time (msec)	C_{Length}	$3D_{Length}$ (meter)	Success %	Path Configurations
CellBiRRT	10	169	535	2.39	100	9.54

Table 6.1: Experimental result of *CellBiRRT* algorithm for Task 1

to move fast. At that case the controller should switch very fast from point to point and that may lead to "unwanted" motions. As a consequence the execution of such a path may be reasonable to be done with low joint velocities.

6.4.1 Grasp bottle

The first task is a scenario where the robot should grasp a bottle. The scenario is a part of the **ADL** scenario (more information in chapter 9) and includes a fridge and the robot system FRIEND. The results are summarized on tables 6.2, 6.3 and 6.4. The notation $RRT - J_{WLN}$ refer to the case where the Jacobian expansion is used and not the one with inverse kinematics($RRT - IK$). As a comparison, the $CellBiRRT$ for this task is very fast and for 100 trials the average results are presented in the table 6.1.

6.4.2 Move out/in from/to a hole

The environment of this task is the same as the environment in figure 5.11. The start configuration and the goal location(position and orientation of end effector) are the same. The task for this experiment is to illustrate how this algorithm work in such a clutter environment. The experimental results are presented on tables 6.5, 6.6 and 6.7 .

The experimental results seem to be very promising especially for the $RRT - J_{WLN}$ algorithm. It manages to solve the task into a deliverable time (around 8 sec). Let remind here that the $CellBiRRT$ managed to solve the task but the time needed to calculate a path is much higher. Also the $RRT - IK$ solves the task but it needs almost double the time compared to $RRT - J_{WLN}$.

Planner	$Cell_{size}$ (cm)	Time (msec)	C_{Length}	$3D_{Length}$ (meter)	Success %	Path Configurations
RRTJWLN (Pg=0.2)	5	2734	1410	6.75	100	399
RRTJWLN (Pg=0.4)	5	1514	931	4	100	388
RRTJWLN (Pg=0.7)	5	1792	816	3.27	100	387
RRTJWLN (Pg=0.2)	15	5912	1196	5.54	100	379
RRTJWLN (Pg=0.4)	15	3127	1011	4.42	100	394
RRTJWLN (Pg=0.7)	15	1843	798	3.15	100	396
RRTJWLN (Pg=0.2)	NO	3745	4683	25.34	100	687
RRTJWLN (Pg=0.4)	NO	2328	2289	10.87	100	563
RRTJWLN (Pg=0.7)	NO	1961	1550	6.66	100	515

Table 6.2: Experimental result for $RRT - J_{WLN}$ for the task of figure 6.3

The *CartesianRRT* planner does not provide acceptable results for this environment. The reason may be the sampling procedure. The algorithm samples the available workspace whereas the other algorithms sample around a region in C-Space. Recall that *CartesianRRT* planner needs to calculate a configuration using the inverse kinematics algorithm. Consequently its speed depends highly on the inverse kinematics algorithm speed.

6.5 Discussion

The experimental results show that these algorithms are really promising since they can deliver results in a deliverable time even in clutter environments. The $RRT - J_{WLN}$ seems to provide more reliable results. Moreover does not depend on a specific inverse kinematics algorithm which is very important. For instance if the robot had 6 DoF another inverse kinematics algorithm should be used. That means that an extra work should be done for this case. That is the negative aspect of $RRT - IK$ or *CartesianRRT* or even the *CellBiRRT* which requires a goal configuration.

These approaches have some drawbacks. All the approaches generate paths with many configurations. That is caused because the expansion in cartesian space require many intermediate steps. That guarantees a collision free path since the sampling resolution is high. A pruning procedure as well as extra smoothing may be required afterwards. Moreover the velocities cannot be very high

Planner	$Cell_{size}$ (cm)	Time (msec)	C_{Length}	$3D_{Length}$ (meter)	Success %	Path Configurations
RRTIK (Pg=0.2)	5	2154	1620	6.59	100	146
RRTIK (Pg=0.4)	5	852	875	3.07	100	110
RRTIK (Pg=0.7)	5	628	704	1.97	100	105
RRTIK (Pg=0.2)	15	4706	1086	4.09	100	112
RRTIK (Pg=0.4)	15	1699	948	3.20	100	112
RRTIK (Pg=0.7)	15	460	672	1.81	100	96.5
RRTIK (Pg=0.2)	NO	2807	3567	15.89	100	314
RRTIK (Pg=0.4)	NO	1966	2446	9.39	100	263
RRTIK (Pg=0.7)	NO	1065	1394	4.5	100	190

Table 6.3: Experimental result for $RRT - IK$ for the task of figure 6.3

Planner	$Cell_{size}$ (cm)	Time (msec)	C_{Length}	$3D_{Length}$ (meter)	Success %	Path Configurations
CartesianRRT(Pg=0.2)	—	9305	772	1.49	100	163
CartesianRRT (Pg=0.4)	—	6028	747	1.45	100	161
CartesianRRT (Pg=0.7)	—	3998	742	1.33	100	148

Table 6.4: Experimental result for $CartesianRRT$ for the task of figure 6.3

when the robot arm moves through samples which are very close to each other because the controller may not be able to follow the path. Another issue of theses approaches is the singularities. When the end effector of the robot arm follows a direct line in workspace, there is no guarantee that its joints are able to follow the motion. Around singularities the velocities of the joints are going to be increased rapidly. One improvement that may help to avoid motions around singularities is

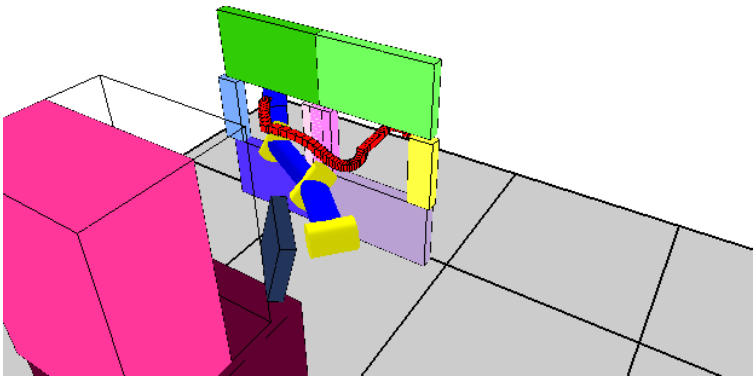


Figure 6.5: Path example resulted from the $RRT - J_{WLN}$ with smoothing [GO07]

Planner	$Cell_{size}$ (cm)	Time (msec)	C_{Length}	$3D_{Length}$ (meter)	Success %	Path Configurations
RRTJWLN (Pg=0.4)	5	16984	2335	11.27	23	528
RRTJWLN (Pg=0.7)	5	26541	1947	8.71	12	535
RRTJWLN (Pg=0.4)	15	35040	2508	12.04	100	540
RRTJWLN (Pg=0.7)	15	32764	2258	10.89	100	594
RRTJWLN (Pg=0.4)	NO	9424	6696	35.42	100	1074
RRTJWLN (Pg=0.7)	NO	7473	4181	21.32	100	916

Table 6.5: Experimental result for $RRT - J_{WLN}$ for the task of figure 6.4. Maximum computation time is 120 sec. Maximum number of nodes is 32000. Most of the failures are due to maximum limit number of nodes

to use the manipulability measure:

$$Manipulability = \sqrt{(det(JJ^T))} \tag{6.8}$$

When the arm approaches the singularity, its manipulability approaches to zero and that may be a good measure avoiding singularities or lowering the speed of the robot arm. An expansion may fail if the manipulability reaches a value smaller than a critical limit. The advantage of this approach is that the arm may avoid such as situations during the planning and not during the

Planner	$Cell_{size}$ (cm)	Time (msec)	C_{Length}	$3D_{Length}$ (meter)	Success %	Path Configurations
RRTIK (Pg=0.4)	5	19658	2428	10.62	11	265
RRTIK (Pg=0.7)	5	5900	1990	8.26	6	292
RRTIK (Pg=0.4)	15	55405	3154	13.75	100	330
RRTIK (Pg=0.7)	15	69066	3044	13.25	95	417
RRTIK (Pg=0.4)	NO	16347	7579	34.6	100	881
RRTIK (Pg=0.7)	NO	16903	4742	21.19	97	683

Table 6.6: Experimental result for $RRT-IK$ for the task of figure 6.4. Maximum computation time is 120 sec. Maximum number of nodes is 32000. Most of the failures are due to maximum limit number of nodes

Planner	$Cell_{size}$ (cm)	Time (msec)	C_{Length}	$3D_{Length}$ (meter)	Success %	Path Configurations
CartesianRRT (Pg=0.4)	–	52921	1472	3.77	76	389
CartesianRRT (Pg=0.7)	–	55595	1342	3.46	80	358

Table 6.7: Experimental result for $CartesianRRT$ for the task of figure 6.4. Maximum computation time is 120 sec

execution of the motion.

The presence of cells does not provide an improvement in the performance in case of environment like the task in figure 6.4. The main reason is that these approaches are mixed e.g. combine direct line expansion in workspace and in C-Space. Compared to $CellBiRRT$, where the Cells improve the performance, here the performance is better only in simple environments like the grasp bottle scenario.

Chapter 7

Benchmarking and comparison with the state of the art motion planners

7.1 Introduction

Till nowadays a systematic and a general report that compares different planning algorithms which are based on RRT and specialized for robotic manipulators is not available. Open Motion Planning Library is a framework where different planning algorithm like RRTConnect, SBL, PRM can be compared [OMP]. However additional modification and updates of these algorithms are not included. Another platform which includes motion planning algorithms is OpenRAVE [OPE], but there is not a report regarding the benchmarking of algorithms being applied in manipulators . In [PBK07] different benchmarking results are presented based on an open source programming system. The experiments are done in a mobile system and not in manipulators. Comparison between different planners for specific tasks has been done in several works [Bra06b, FT10].

This chapter compares the performance of all described planners as well as with some state of the art motion planning like the IKBiRRT [BSF+09], RRT-JT [BSF+09] and CBiRRT [BSFK09]. At the end of the chapter, a comparison with a graph search algorithm is presented. This benchmark shows the advantage of sampling based approaches over algorithms which use graph search and additional heuristics in order to explore the free space.

A comparison of planning algorithm focused on optimality is not done since they need more time by default due to their complexity. The IKBiRRT, RRT-JT and CBiRRT are planners similar to the described planners in this thesis e.g. are sampling based approaches.

Before proceeding to benchmarking, a short description of IKBiRRT, RRT-JT and CBiRRT is going to be done.

- RRT-JT: The algorithmic part of this planner is given in the algorithm 17. The algorithm can use the Workspace goal regions(WGR), a feature described in later chapter. The algorithm simply has two type of expansions. One is in C-Space like previously described in bidirectional RRT and the second one is a Jacobian expansion like in RRT_{JWLN} but the pseudo-inverse

Jacobian is used. The expansion is done by a configuration being selected with probability inversely proportional to the WGR. The advantage of the algorithm like in RRT_{JWLN} is the independence from an inverse kinematics algorithm. However the experimental results showed that the algorithm is not so promising, requiring a lot of time till a solution is extracted. For that purposes the planner is left from the benchmarking since it cannot contribute to a fast solution like the rest of the algorithms.

Algorithm 17 $RRT - JT$

```
1: W:Work Space Goal Region(WGR), D: Distance to WGR, T_a: Tree, T_a^b: Frame
2: loop
3:    if rand(0, 1) ≤ Eg then
4:       q_sample=WeightedSampleNode(Tree);
5:       T^0_sample'=CalculateGoalFrameBySampling(W);
6:       Q_new=ExpandJacobian(q_sample,T^0_sample');
7:    else
8:       q_rand=RandomConfigInsideNCuboid(T_a.LastConfig(), Tol);
9:       q_near=T_a.find_nearest(q_rand);
10:      Q_new= T_a.ConnectEfficient(q_near,q_rand);
11:   end if
12:   D = DistanceToNearestWGR(Q_new,W);
13:   T_a.AddNodes(Q_new);
14:   for each Di∈ D do
15:      if D_i==0 then
16:         return SUCCESS;
17:      end if
18:   end for
19: end loop
```

- IKBiRRT/ CBiRRT: The IKBiRRT and the CBiRRT are very similar and for that reason are compressed to the same algorithm 18. The main difference is the extension method. The IKBiRRT uses also the workspace goal regions(WGR) but for benchmarking purposes they are omitted. The CBiRRT differentiates from IKBiRRT in the fact that it solves problems with additional constraints. In CBiRRT the method Extend is substituted with the method ConstrainedExtend (refer to the literature for more details), which checks for constraints. If a candidate node violates them, a projection to the constraint manifold is done. The projection uses jacobian pseudo inverse expansion and attracts the end effector to return back to the constraint manifold. In the literature the IKBiRRT uses incremental connection from a q_a to q_b and for that reason the $ConnectEfficientWithStep$ method is used.

  It is good to be noticed that the CellBiRRT becomes identical to IKBiRRT if cells and N-Cuboid domains are not present. Since N-Cuboid domains provide really an improvement in performance, they are included also in IKBiRRT.

Algorithm 18 *CBiRRT/IKBiRRT*

W:Work Space Goal Region(WGR)
loop
  T_{goal}=GetBackwardTree(T_a,T_b);
  if $T_{goal}size = 0$ or rand(0, 1) $\leq$ Eg) **then**
    AddIKSolution(T_{goal},W);
  end if
  q_{rand}=RandomConfigInsideNCuboid($T_a.LastConfig()$,Tol)
  q^a_{near}=T_a.find_nearest(q_{rand});
  $q^a_{reached}$= Extend(T_a,q^a_{near},q_{rand});
  q^b_{near}=T_b.find_nearest($q^a_{reached}$);
  $q^b_{reached}$=Extend(T_b,q^b_{near},$q^a_{reached}$);
  if $q^b_{reached}$==$q^a_{reached}$ **then**
    return SUCCESS;
  end if
  SWAP(T_a,T_b);
end loop

7.2 Benchmarking sampling based approaches

The benchmarking consists of several tasks. As already mentioned the main focus is the feasibility and the time needed by the planner to achieve a result. The planners that considered for comparison are: CellBiRRT, IKBiRRT, RRT_{JWLN} and RRT_{IK}. Benchmarking with the $CartesianRRT$ and the $RRT-JT$ are not included due to their worse performance compared to the rest. There is no need to include benchmarking with the simple BiRRT since the literature confirms that the current planners are faster. The $CBiRRT$ is going to be compared with the $CellBIRRT$ in environment with constraints.

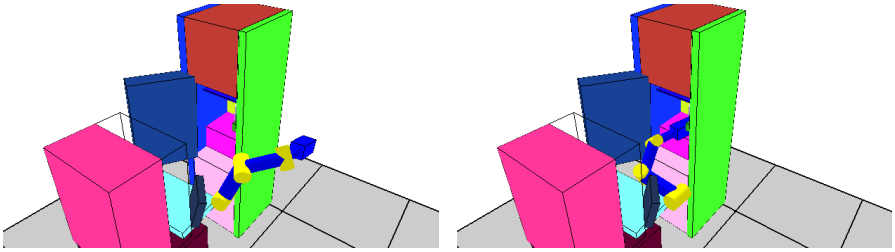
The platform consists of Intel Core i5-450M@2.4GHz CPU with 4GB Ram. For Task1 and Task2 the maximum time was 60sec while for Task3 the average was 120sec. For Task1 and Task2 all the planners had 100% success rate. For Task3 the $CellBiRRT$ had 90%, the $IKBiRRT$ had 70% , the $RRT-IK$ and the $RRT-J_{WLN}$ had 100%. The computation time here is computed as follows:

$$Time = \frac{Time_{Success} \cdot N_{success} + MaximumTime \cdot N_{Fails}}{N_{Trials}} \tag{7.1}$$

where the $Time_{Success}$ denotes the average time of the succeeded trials, $N_{success}$ is the amount of successes ,N_{fails} is the amount of failures and N_{Trial} is the amount of trials.

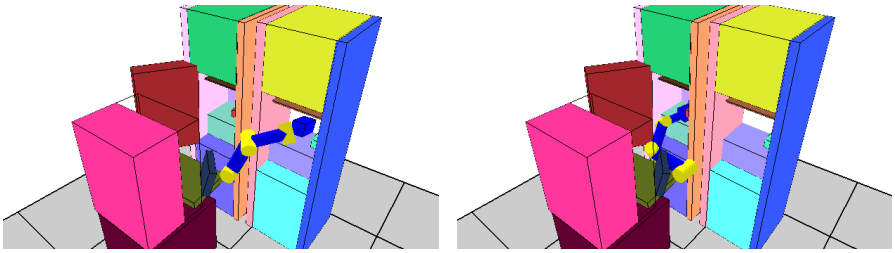
7.2.1 Task 1

The manipulator has to grasp the bottle in the fridge. The start and (possible) goal configuration are presented in the figure 7.1. The results are illustrated on figure 7.4. Clearely the CellBiRRT



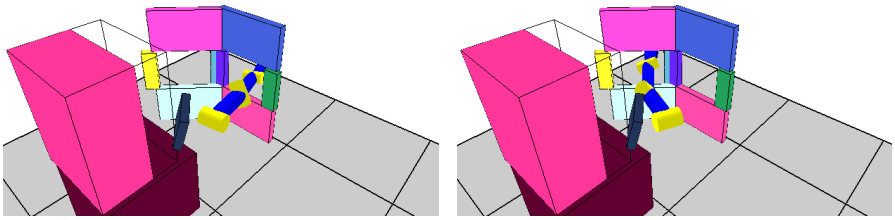
(a) Start Configuration (b) Goal Configuration

Figure 7.1: Benchmarking Start-goal Configuration Task 1



(a) Start Configuration (b) Goal Configuration

Figure 7.2: Benchmarking Start-goal Configuration Task 2



(a) Start Configuration (b) Goal Configuration

Figure 7.3: Benchmarking Start-goal Configuration Task 3

over performs the IKBiRRT. The *Cells* are improving the performance and collision free paths can be extracted faster.

7.2.2 Task 2

This task has more obstacles and therefore is more cluttered. The manipulator should move from one fridge to the other. The task is presented on figure 7.2. Again the CellBiRRT is ahead compared to the rest of the algorithms (refer tot figure 7.5).

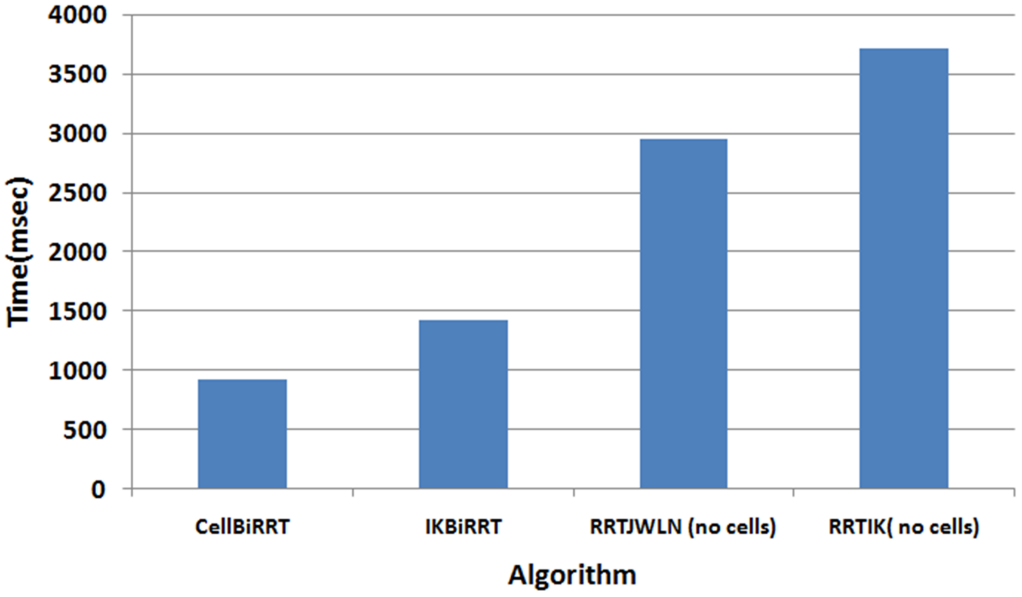


Figure 7.4: Experimental Results for the Task 1

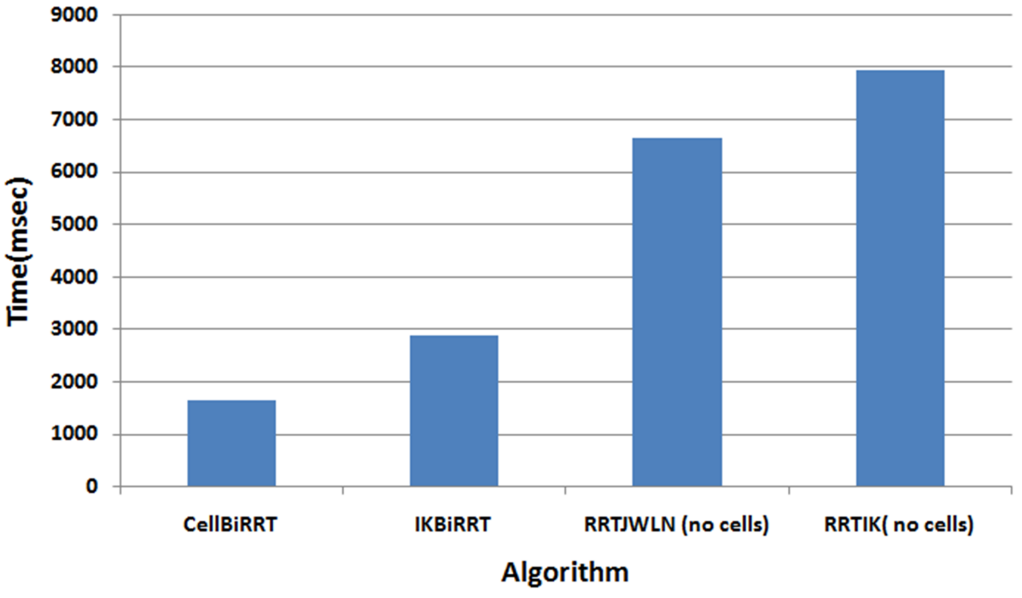


Figure 7.5: Experimental Results for the Task 2

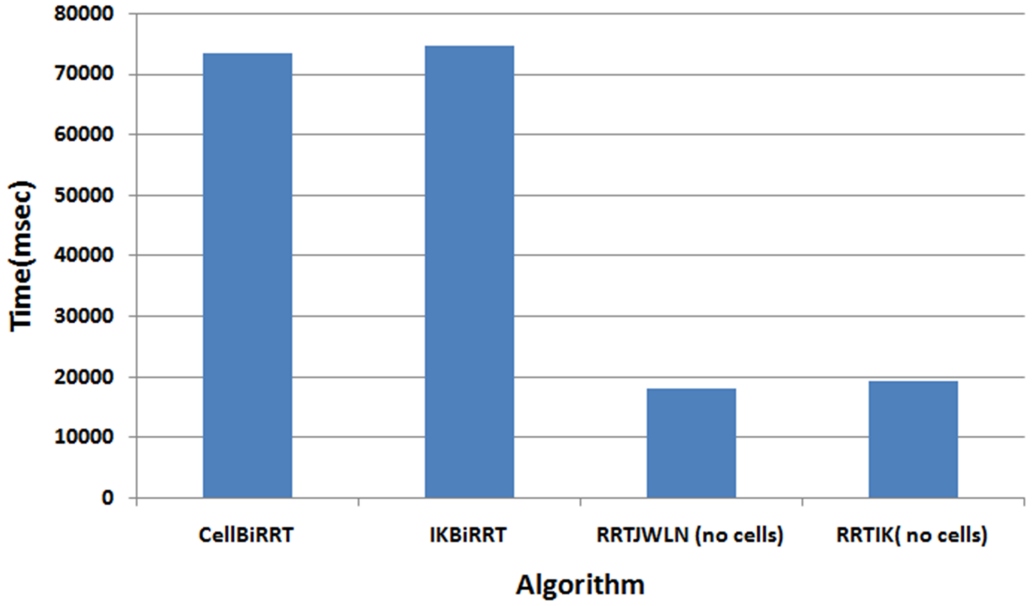


Figure 7.6: Experimental Results for the Task 3

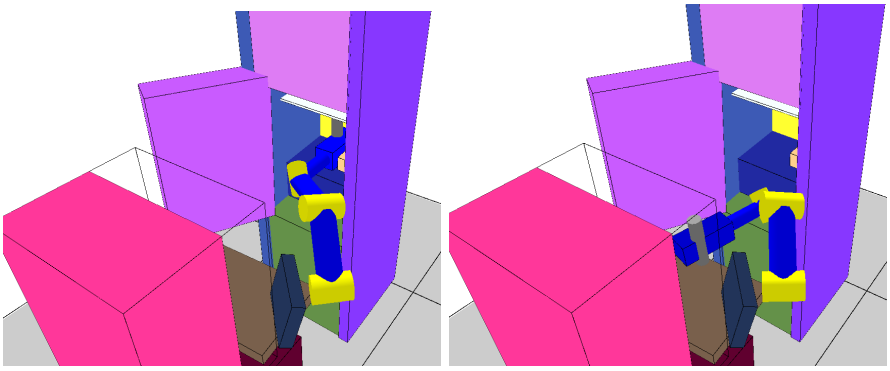
7.2.3 Task 3

This task is the most cluttered one and similar to the tasks described in chapter six and it is illustrated in figure 7.3. The environment is cluttered and it is expected the planners to need more time. Based on the experimental results from previous chapters, the mixed approaches are expected to perform better than the approaches working in C-Space.

The results are presented on figure 7.6.The CellBiRRT has almost the same computation time like the IKBiRRT, however the mixed approaches e.g. RRT_{IK} and $RRT - J_{JWLN}$ performed better.

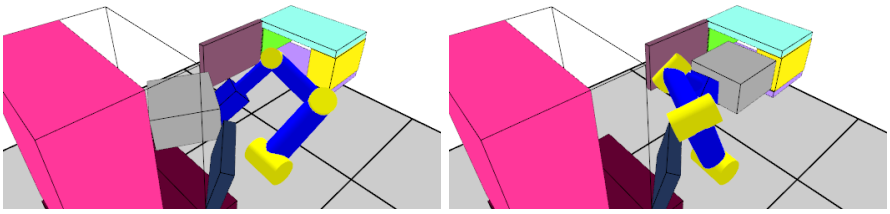
7.2.4 Constraints: Task 4

The first task with additional constraints is, like in chapter 6, the bottle. The start and goal configuration are presented in figure 7.7. The comparison is done between the CelllBiRRT and the CBiRRT and the results are on the table 7.1. The CellBiRRT is able to solve faster the task and the computation time is deliverable. The maximum computation time was 60sec.



(a) start configuration (b) goal configuration

Figure 7.7: Orientation constraints - bottle up-right down



(a) start configuration (b) goal configuration

Figure 7.8: Orientation constraints - grasped object is a mealtray and should be kept horizontal

PLANNER	Time(msec)	Success %	Path Configurations
CellBiRRT(5^o)	1211	100	46
CellBiRRT(10^o)	672	100	30
CBiRRT(5^o)	7053	100	235
CBiRRT(10^o)	3235	100	237

Table 7.1: Average results for the $Task4$ in figure 7.7. Orientation tolerance of 5^o and 10^o for object-bottle are tested. Last column is the number of configurations in the final path

7.2.5 Constraints: Task 5

The mealtray has to be transfered from the user in front of the microwave. Later the robot will place the mealtray inside the microwave. The mealtray during the robot motion should be handled so that the food is going to remain on the plate. For that reason small tolerance for its orientation

is allowed. The table 7.2 shows different results for the two tolerances. Both algorithms can solve the task with 100% success. The CellBiRRT can extract a solution faster.

PLANNER	Time(msec)	Success %	Path Configurations
CellBiRRT(5^o)	1525	100	46
CellBiRRT(10^o)	908	100	32
CBiRRT(5^o)	6582	100	295
CBiRRT(10^o)	4245	100	323

Table 7.2: Average results for the $Task5$ in figure 7.8. Orientation tolerance of 5^o and 10^o for meal-tray are tested. Last column is the number of configurations in the final path

7.3 Features Comparison

This subsection describes briefly the benefits and drawbacks of each algorithm. The table 7.3 shows the features that each method has, so that the user preferably can select the appropriate method. The notation *"NN"* denotes "not necessary". The column *IK needed* shows if the algorithm works using only analytical /geometrical solution for the inverse kinematics. In that case an IK solver is necessary to exist.

Feature / Method	WGR	Constraints	IK needed	C-Space	Workspace
CellBiRRT	YES	YES	NN	YES	NO
IKBiRRT	YES	NO	YES	YES	NO
CBiRRT	NO	YES	NO	YES	NO
$RRT - J_{WLN}$/RRT-JT	YES	NO	NO	Mixed	Mixed
$RRT - IK$	YES	NO	YES	Mixed	Mixed
CartesianRRT	YES	NO	YES	NO	YES

Table 7.3: Feature Comparison

The CellBiRRT does not need necessary an IK solver. The inverse kinematics can be computed using iterative approaches e.g. Jacobian based approaches. For that reason the "NN" is written on the table.

The CellBiRRT has most of the features and it works completely in the C-Space e.g. a segment of the path is a straight line in configuration space. From benchmarks it seems to be a reliable and fast solution for path planning. It lacks mostly on very cluttered narrow passage tasks. The mixed approaches are very promising since they can solve fast the tasks. In very cluttered environment the mixed approaches seem to solve the tasks faster. That makes the mixed approaches a good candidate but their disadvantage is the high number of configurations in the final path. The robot

arm may not be able to follow the path if the motion has to be executed fast. The CartesianRRT planner solves all the tasks and works completely in cartesian space e.g. the path segments are straight lines in cartesian space.

7.4 Benchmarking with Graph Search Planner

This section is going to compare the CellBiRRT with a cartesian cell decomposition graph search algorithm (notated in this thesis as *GSP*)[Ojd09b]. This illustrates the main advantage of sampling based approaches over "deterministic" approaches. The algorithms belonging in the second category like the GSP have normally a good advantage which is the speed and some times the quality of the path. Moreover for the same parameters they deliver the same results. However if they fail to solve a task once they are not able to replan calculating a different route. If the algorithm is trapped it may not be able to escape.

The performance of this planner to the so far described tasks is the following: (a) Task 1 : 19043msec (b) Task 2: 19sec (c) Task 3: Fail. It is noticeable that the algorithm is not able to deliver a solution for the task 3 within this time duration while the sampling based approaches manage to deliver solutions. It is also remarkable the high computation time that is needed for the first two tasks. The CellBiRRT computes paths in less than two seconds while the GSP needs almost 20 seconds i.e. ten times more. Surely this performance difference is not the same for all environments, but the presented sampling based approaches perform better in dense environments. The latter is very important for practical applications, since the environment is not dense. The algorithms can run for such a situation very fast.

7.5 Discussion

The $CellBiRRT$ as well as the $RRT - J_{WLN}$, the $CartesianRRT$ and the $RRT - IK$ provide comparable results with the state of the art motion planning algorithms. Surely the performance of each algorithm depends on many important parameters which are:

- Implementation
- Parameters of the planner

All the planners were implemented on the same platform and had identical collision detection approach. During this work all the planners in this chapter have been tested with different parameters giving the same relative results. The results show that the presented algorithms are very comparable with the state of the art planning algorithms. Additional results are available in [FAEG12].

Chapter 8

Optimality

As presented in previous chapters there are many planning algorithms available in the literature and most of them have the following similarity : can solve a large set of tasks focusing on feasibility and much less to the quality of the paths. The first part of this chapter presents the state of the art approaches that improve the quality of path while the second part describes a novel anytime sampling based approach that reduces the cost of the path attempting to reach asymptotically an optimal solution. The new planner is called *CellBiRRT** and it is based on the described planner CellBiRRT. For the rest of this chapter the cost of a path is the same with the length of the path.

8.1 Creating high quality paths

Creating high quality paths for motion planning is still a challenging task. The planners deliver results fast but smoothing and at the same time fast computation are two important parameters. If a path is short, the robot may have to move less for the same speed and therefore the execution needs shorter time. Another parameter of a path is the distance of the robot from the obstacles. The minimum distance between a robot and the obstacles is called *clearance*. Summarizing, clearance and length of the path are the most important parameters. The two parameters are in contradiction since a shorter path is mostly a path with low clearance.

In [GO07] there is a nice overview of approaches that significantly help to improve the quality of a path. The approach used in this thesis is called *pruning*. Pruning simply removes all redundant configurations e.g. removes configurations that do not contribute to the final path. For instance if a path goes from q_a to q_b through the q_c, the pruning examines the path $q_a - q_b$ and if it is valid removes the q_c. The algorithm may go against the clearance, but that can be overcomed if the minimum distance limit is high during the pruning. Algorithm 19 presents the pruning.

Algorithm 19 PRUNING(Path N) [GO07]

```
1: i←0
2: while (i< card(N) -1) do
3:   if (PathIsValid(q_i → q_{i+2})== true ) then
4:     N ← N q_{i+1}
5:     if (i>0 ) then
6:       i←i-1
7:     end if
8:   else
9:     i←i+1
10:  end if
11: end while
```

Shortcutting is another option that modifies the existent path and tries to create a new one based on some heuristics. For instance in [HNTH10] some heuristics are applied to smooth jerky trajectories for manipulators subject to collision avoidance, velocity and acceleration bounds. The approach selects randomly point in the trajectory and attempts to replace the segment with a shorter one. However this approach requires time if the shortcutting is meant to provide significant improvement. According to the [HNTH10] the approach can be applied in a parallel thread with the robot motion, which does not require additional time. In this case the shortcutting should be faster than the robot motion. Due to the randomness the approach may shorter the path in an amount and its performance depends on the number of iterations. Experimental results showed that the final length may not be improved in a significant manner if the shortcutting has to run with small number of iterations. High number of iterations requires more execution time and that is not optimal in case that the robot moves fast.

8.2 Asymptotically optimal (lowest) cost of a path

This section describes the CellBiRRT*. The main idea behind this planner is the anytime planning. The planner does not stop if a solution is found but continues searching reducing the cost of the path at the same time. That is still a challenge since the available planners promising to reduce the cost of the path require high computational time and they cannot be implemented in a real time system where the execution of sub processes like path planning should be as fast as possible.

The second challenge is the parallel execution of a planner while the robotic system is working e.g. executing the initial path. In this thesis the CellBiRRT* in combination with the CellBiRRT is applied in static as well as dynamic environment. Planning in dynamic environments requires a fast planner (like CellBiRRT) but since the robot does not collide with the obstacle, the CellBiRRT* may be applied. It may be achieved to deliver path having lower cost compared to the initial one. For that reason the notion of *replanning* is introduced e.g. the ability of the system to recalculate

its path in order to deliver a better solution. That can be done online or offline. *Online* replanning is executed in parallel to the robot motion. More details are going to be explained later.

8.2.1 Theoretical Background

One nice approach for replanning in unknown environment especially for mobile robots was presented 15 years ago by Stentz [SM93, LFG+05, Ste95]. All ideas are mainly based on the D^*(Dynamic A^*) algorithm. The D^* works like the A^*, since it has OPEN and CLOSE list, but it maintains those list by classifying states with "RAISE" and "LOWER" cost and by sorting states based on the $min_{allstates}(min(prevCost, actualCost))$. The tests are done with mobile platform and the space is divided by grids. However, in high dimensional space, like a 7D, subdivision of space in grids is not appropriate concerning memory space and speed. Other approaches like elastic bands and elastic strips [QK93] [BK02] are considered mixed approaches, where the initial global path is adapted to a changing environment. Actually they are based on the reaction of the robot caused by the distance between itself and an obstacle. The path is adapted throughout this control.

More recent works try to cope with the dynamic environment using replanning RRTs [ZKB07, FKS06]. The idea behind both algorithms is the removal of segments in the resulted path that will lead to a new rearrangement of the nodes that are still valid. [ZKB07] used extra smoothing and pruning procedures in order to search for disconnected subtrees, invalid nodes and edges. The algorithm tries to continue the searching using also information from previous solutions. Evolutionary algorithms (EAs) are used in order to bias more efficient RRTs in dynamic environments [MWS07]. Probabilistic path replanning based on the sensors data is presented in [PJS06]. Other approach uses partial motion planning to plan safe in dynamic environment [PF05]. It uses the dynamics of the system and moving obstacles and ensures that a critical situation with inevitable collision is not going to appear. Randomized kinodynamic motion planning presented in [LJ99] and [HKcLR00] is a nice solution especially for non-holomonic motion planning tasks. In a recent paper [KSV10], based on dynamic roadmaps presented in [LH00] and [LH02], a probabilistic roadmap is implemented that is able to replan very fast when the environment changes. Although this method seems very attractive, it may not be very appropriate in a complex environment since it is affected by the roadmap construction and its efficiency.

Anytime planners [LFG+05, vdBFK06, FS06] are developed promising to solve the optimality. The difference with replanning RRT's methods is that anytime planning continues to grow trees even if a solution is founded. Anytime path planning tries to reduce the cost (normally from initial result) of the path, attempting to reach an optimal solution, if that is possible. In [vdBFK06] an initial roadmap is built and tries to improve the initial path iteratively concerning any possible change in the environment. Anytime RRT developed in [FS06] generates new solutions over the time and additionally it attempts the generated paths to have less cost than previous solutions. However this method is slow and does not have additional heuristics in order to achieve acceptable results very rapidly. In [KF10, KWP+11] an idea of steering and rewiring of nodes in the tree is proposed. In their work the RRT^* is introduced promising to solve the asymptotically optimality challenge. In the same works the authors prove theoretically the optimality of RRT^*. Practical implementations, especially for manipulators, is done in [PKS+11]. They improved also the speed

of the RRT^* by reducing the calls of the collision checking procedure. A bidirectional RRT^*(Bi-RRT^*) with additional heuristics is implemented in order to improve the performance [AS11] . While RRT^* needs significant a lot of time in order to find a solution, the Bi-RRT^* managed to reduce the total cost faster than the RRT^*.

Regarding the practical aspects for dynamic environments a recent work [YYG10] tries to cope with them. The difficulty comes especially from the fact that the system should first identify if a change in the environment is occurred, and then to react accordingly. In [YYG10] a system is developed where parallel threads are generated when changes in the environment occur, and new trajectories are computed if the arm is decelerating. They include also tests with a PRM and RRT planner. However, compared with this work, they do not include the possibility of reproducing trajectories with lower cost. The new trajectories are generated from replanning while the robot is moving.

8.2.2 The CellBiRRT*

Before starting explaning the algorithm new symbols and definitions are introduced. A node in a tree has a $FailureCounter$ declaring the expansion failures and can be *active* or *inactive*. It becomes *inactive* if a solution is already present and at the same time the $FailureCounter$ or the cost of the node exceeds a limit and it is denoted as N_{FAILS}. If a node is deactivated, it is no longer used inside the nearest neighbor routines. "$Cost$" of a node is the cost-to-come, that is the accumulative cost of the path till this node. The "$Trajectory.Cost$" refers to the total trajectory length. The "||..||" and L_{A-B} are the distance between two nodes A and B. The distance measure depends on the metric (refer to section 2.5.3). For the rest of this section it is the normal euclidean distance in configuration space(C-Space).

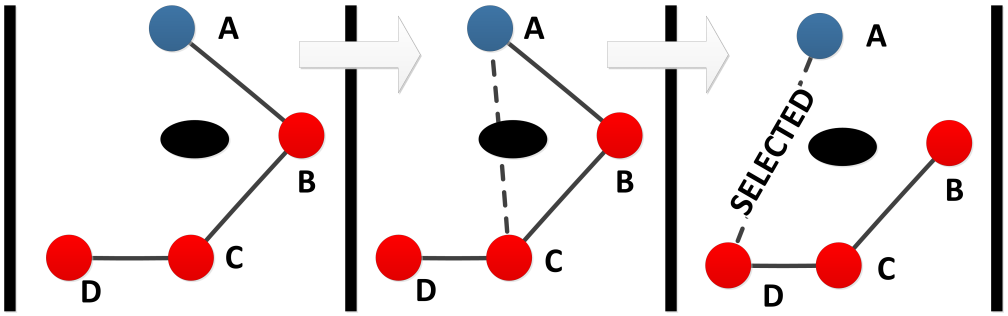


Figure 8.1: PRUNE method in the CellBiRRT*

The main algorithm depends on some important methods, which are going to be described first. The first one is an extra "$rewiring$" step, called $PRUNE$. Starting from a node, the method explores iteratively all of its parents trying to find the shortest possible connection. It is based on the triangular inequality. The metric should obey to it. The figure 8.1 illustrates an example. In this one the L_{A-C} is smaller than $L_{A-B} + L_{B-C}$, but it is not collision free. The L_{A-D} is smaller than the $L_{A-C} + L_{C-D}$ and at the same time collision free. The new parent of node "A" is now the node "D". The procedure continues recursively till the starting node of the tree is reached.

With this method the cost for a node can be reduced(see algorithm 20). A node can be deactivated if its improved cost is bigger than a limit(normally the calculated trajectory cost), and activated otherwise. This rewiring differs from the one presented in [KF11].In [KF11] the rewiring is done using the nearest neighbors while the proposed one reduces recursively the path of a node starting from its parent. The proposed one may improve the performance since it reduces the number of intermediate nodes lowering consequently the cost.

Algorithm 20 Bool=PRUNE(q,Cost)

```
1: bSucces=false
2: actual = q→parent
3: while (actual != start) do
4:    if (CollisionFree(actual,q,Cost)==TRUE) then
5:       q→parent =actual
6:       q.Cost=q→parent.Cost +||q, q → parent|| //calculate NEW cost-to-come
7:       bSucces=true
8:    end if
9:    actual=actual→parent
10: end while
11: if (q.Cost ≤ Cost) then
12:    q.Activate();
13: end if
14: RETURN bSuccess
```

The $CollisionFree(q_1, q_2)$ method works as in previous sections. The method checks additionaly each sample if the estimated cost of the sample $Cest_{q_{sample}}$ ($Cest_{q_{sample}} = q_1.Cost + q_{sample}.Cost_{est}$ where $q_{sample}.Cost_{est}$ is given by the equation 8.1) exceeds a limit. This limit is the calculated trajectory cost. It returns true if it succeeds.

$$q_{sample}.Cost_{est} = \begin{cases} \|q_1 - q_{sample}\| + \|q_{sample} - q_{goal}\| & \text{if } q_{sample} == q_{rnd} \\ \|q_1 - q_{sample}\| + \|q_2.Cost\| & \text{otherwise} \end{cases} \tag{8.1}$$

The second method, called $Extend\_rrts$ and presented in algorithm 21, is very similar to the normal extension part of the RRT^*. The method $ExtendWithStep$ makes the stepwise extension from the nearest neighbor q_{near} towards the q_{sample}. The stepwise configuration q_{step} is computed by the formula:

$$q_{step} = \begin{cases} q_{near} + (\Delta q) \cdot (\frac{step}{||\Delta q||}) & \text{if step} \leq ||\Delta q|| \\ q_{sample} & \text{if step} > ||\Delta q|| \end{cases} \tag{8.2}$$

where Δq is equal to the difference $q_{sample} - q_{near}$. An extension fails if the CollisionFree(q_{near} ,q_{step}) fails. At this case the $q_{near}.FailureCounter$ is increased.

In line 7 the "Near" routine returns the nearest neighbors. Given a Tree and a configuration $q \in C_{free}$ the Near returns the set of all configurations that are close to q. Two different cases have been examined. If A is the number of points in a Tree and d the dimensionality of the space, the Near routines returns:

Algorithm 21 Extend_rrts(q_{sample},Cost,N_{FAILS})

1: q_{near}=Tree.Nearest(q_{sample},N_{FAILS}); {$Tree$ is the actual tree where the Extend is done}
2: q_{extend}=ExtendWithStep(q_{near},q_{sample},step,Cost);
3: **if** (q_{extend} == q_{near}) **then**
4: q_{near}.FailureCounter++;
5: return q_{near};
6: **end if**
7: Q_{near}=Tree.Near(q_{extend},N_{FAILS});{find nearest neighbor from equations 8.3 or 8.4}
8: L_{near}=PopulateSortedList(Q_{near},q_{extend},Cost);
9: q_{parent}=FindBestParent(L_{near},q_{extend},Cost);
10: **if** (q_{parent} == $NULL$) **then**
11: return q_{near};
12: **end if**
13: q_{new}=Tree.Add(q_{sample}, q_{parent});
14: RewireVertices(L_{near},q_{new});
15: PRUNE(q_{new},Cost);
16: return q_{new}

- the set of vertices that lie inside a radius

$$r(A) = min(\gamma_{RRT^*} \cdot (log(A)/A)^{(1/d)}, \eta) \tag{8.3}$$

  where, γ_{RRT^*} is a constant and η is the maximum extend e.g. $step \cdot \sqrt{d}$ (d is the DoF of the system).

- the K nearest neighbors. In such a case, based on the [KF11], the k is equal to :

$$k = K_{RRT^*} \cdot (log(A)) \tag{8.4}$$

  where K_{RRT^*} is a constant and it is equal in this work with $2 \cdot e$.

The next method, called *PopulateSortedList*(algorithm 22) , returns a sorted list of the nearest neighbors. It differentiates from the one used in [PKS$^+$11, AS11] since it calls the $PRUNE$ approach for each nearest neighbor. The *Steer* in line 7 is a function which connects two configurations q_a and q_b using the formula: Steer(x,q_a,q_b) = (1-x) $\cdot q_a$ + x$\cdot q_b$, where $x \in$[0,1]. The σ_{near} contains the path going from q_{near} to q_{new}. If the $q_{near}.Cost$ exceeds the trajectory cost, it is deactivated. The list of nearest neighbors are sorted by ascending order of the cost(line 11).

The procedure $FindBestParent$(algorithm 23) takes the sorted list L_{near} and returns the first node where the path σ_{near} is collision free and does not exist the $Cost$(Trajectory cost). It should be noticed again that if $CollisionFree$ fails the $FailureCounter$ of the corresponding node increases.

The rewiring procedure(algorithm 24) differs from the normal approach [PKS$^+$11, KF11] since it can increase the $FailureCounter$ or activate a node. A node is activated (if it is inactive) if its cost is less than the path's cost.

The CellBiRRT* uses the same approach for creating random configurations like the CellBiRRT. It uses Cells and N-Cuboid domains in order to reduce the space for creating random configurations.

Algorithm 22 PopulateSortedList(Q_{near},q_{sample},Cost)

```
1: L_near.clear();
2: for q_near ∈ Q_near do
3:    PRUNE(q_near,Cost);
4:    if (q_near.Cost > Cost) then
5:       q_near.Deactivate();
6:    end if
7:    σ_near=Steer(q_near, q_new)
8:    c_near=q_near.Cost + Cost(σ_near)
9:    L_near.add(c_near, q_near,σ_near)
10: end for
11: L_near.sort();
```

Algorithm 23 q_{min}=FindBestParent(L_{near},q_{sample},Cost)

```
1: for (c_near, q_near, σ_near) ∈ L do
2:    if CollisionFree(σ_near,Cost) then
3:       return q_near;
4:    else
5:       q_near.FailureCounter++;
6:    end if
7: end for
8: return NULL;
```

The collision free random configuration(q_{rnd}) (algorithm 25) is created if the sum of the distances $D_{rnd} = \|q_{rnd} - q_{start}\| + \|q_{rnd} - q_{goal}\|$ is less than a $Score$ and the current path cost. The $Score$ can have two possible values, which are selected by a probability P_{rand}. The values are :

- if a path is found, the approach selects randomly one node(q_{sel}) from the path and the score is equal to $\|q_{sel} - q_{start}\| + \|q_{sel} - q_{goal}\|$ (figure 8.2).
- if path is not found or the $a > P_{rand}$, the Score is equal to a maximum value(normally a very big number).

The $CreateRndConfig$ generates collision free random configurations(q_{rnd}) which are good candidates for reducing the total cost of the trajectory. Moreover the P_{rand} parameter distinguishes the space where the q_{rnd} is created. The space is either around a given path as the figure 8.2 depicts or the one that is calculated like in CellBiRRT e.g. using the cells. Even if the actual path is not an optimal one, the algorithm may elicits to a better solution, since it searches for collision-free configurations with D less than the path's cost.

The CellBiRRT* is presented in algorithm 26. First a trial to connect to the goal is done (using either the method $Extend\_rrts$ or $Connect\_rrts$). The $Connect\_rrts$ repeats recursively the $Extend\_rrts$ and terminates if the $Extend\_rrts$ fails or succeeds. The approach, like the RRT^*, samples a configuration, updates and sorts the list with the nearest neighbors. Finally the opposite

Algorithm 24 RewireVertices(L_{near},q_{new},Cost)

```
1: for (c_near, q_near, σ_near) ∈ L do
2:    if CollisionFree(σ_near,Cost) then
3:       q_new.RemoveParent();
4:       q_new.AddParent(q_near)
5:       q_new.CalculateCost()
6:       if (q_new.Cost < Cost) then
7:          q_new.Activate();
8:       else
9:          q_new.Deactivate();
10:      end if
11:   else
12:      q_near.FailureCounter++;
13:   end if
14: end for
```

tree tries to connect with the forward one. If the last attempt succeeds the trees are connected and the solution is extracted, otherwise the trees are swapped.

8.2.2.1 Probabilistic completeness

The probabilistic completeness of RRTs is proved in [LaV06]. In [KF11] is proved that RRT^* shares the same properties. Random sampling is also kept in our approach and nodes are deactivated since a solution is found. These features are also kept in the CellBiRRT*.

8.2.2.2 Asymptotic Optimality

Our algorithms holds the same properties as RRT^*. Random sampling with additional heuristics as well as node deactivation, if a node does not contribute to a better solution, do not remove a property of the RRT^*. These properties are present in the CellBiRRT*.

8.3 On-line $CellBiRRT*$ replanning

This section describes the structure for on-line replanning e.g. online recalculation of the path while the system is moving. The base algorithm for searching a solution rapidly is the $CellBiRRT$ (or *initial planner*) and it is referred as *initial planner*. The algorithm described in this chapter ($CellBiRRT*$) has been used in order to calculate new trajectories. The figure 8.3 shows briefly the different states that this implementation has. Every planning procedure in our system works as follows: A planning with the simpler *initial planner* is done, in order to search fast for a solution. Once it is found the system calculates the motion of the arm, and the arm starts moving. In order for the replanning to start working, a start configuration is needed. For faster calculations the end of current segment is the next point (q_{i+1}),and it is considered as the root for the replanning.

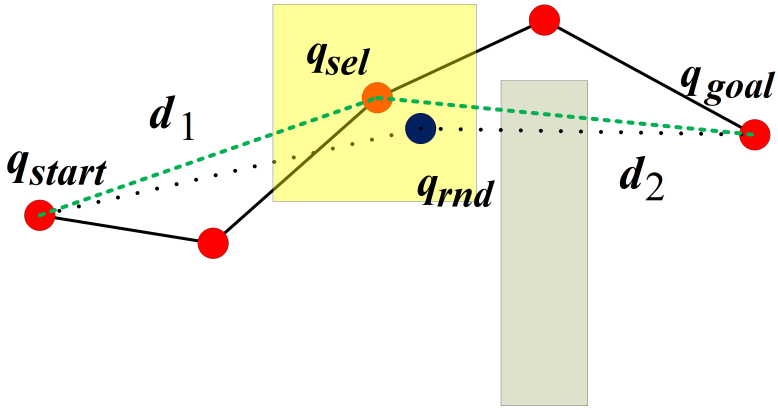


Figure 8.2: Create random configuration based on Trajectory. A random node q_{sel} is selected from the trajectory. A random configuration q_{rnd} is created inside the region close to q_{sel}. If the $\|q_{rnd} - q_{start}\| + \|q_{rnd} - q_{goal}\|$ is less than the $\|q_{sel} - q_{start}\| + \|q_{sel} - q_{goal}\|$ and $q_{rnd} \in C_{free}$ the q_{rnd} is valid

From that point the $CellBiRRT*$ starts being executed in a parallel thread. Nowadays this is not a difficult task, since most of the systems have at least two cores and support multi threading. If the environment is not updated, the motion continues. When the robot reaches the point q_{i+1} it stops the created parallel thread and if a new solution is found, the robot follows the new one. The procedure continues again for each segment. The robot arm with this on-line replanning method follows a better path, without waiting for an optimal solution during the initial planning.

The updates of the environment can be identified using sensors like stereo cameras and laser scanners. When the environment is updated and a collision is not anticipated the arm continues moving with the old trajectory. It is important to mention that if the environment is updated only the segment q_i-q_{i+1} is checked for collision, and not the complete path. At the same time, as the diagram shows, the re-planing is started and if it succeeds, the new trajectory substitutes the old one. If it fails, the old path remains, but every segment is checked for collision, since the environment was updated during the motion.

If a collision is unavoidable, the point where the arm should stop is calculated and from that point the $CellBiRRT$(initial planner) is started. The arm moves till that point and when it reaches the point, the system checks if the planner has returned a solution. If the environment is updated again and a collision is expected, the planner stops immediately and the procedure is repeated. If the planner returns a result the new path is given to the robot and the motion restarts. During multiple updates of the environment it is necessary a solution to be found very fast even though this solution is not an optimal one. That is very important, since the $CellBiRRT*$ takes longer time than the normal planner.

The on-line replanning presented here can deal with static , as well as dynamic environments. Moreover, it improves the calculated trajectory for every segment during the robot arm motion. With the proposed strategy the system runs faster, since it calculates rapidly trajectories, and improves the already existing paths in the background without affecting the system's performance (e.g. time).

Algorithm 25 q_{rnd}=CreateRndConfig(q,Trajectory,q_{start},q_{goal})

Output: random configuration q_{rnd}
while ($q_{rnd}.Collide == true$ OR $d_{start} + d_{goal} > Score$ OR $d_{start} + d_{goal} > Trajectory.Cost$) **do**
  if ($PathFound == true$ and $rand(0,1) < P_{rand}$) **then**
    q_1 = SelectRandomNode(Trajectory);
    $Score = \|q_1 - q_{start}\| + \|q_1 - q_{goal}\|$
    CalculateRegion(q_1);
  else
    Score = ∞;
    q_1 = CalculateConfigWithCells(q); // use the cells described in section 5.4
    CalculateRegion(q_1);
  end if
  q_{rnd}=RANDOM_CONFIG(); {Uniform sampling }
  $d_{start} = \|q_{rnd} - q_{start}\|$
  $d_{goal} = \|q_{rnd} - q_{goal}\|$
end while
return q_{rnd}

Algorithm 26 *CellBiRRT**

Ta,Tb Trees,q_{init},q_{goal},BestTrajectory is the resulted trajectory
$N_{FAILS} = \infty$ till first solution
BestTrajectory.clear()
Ta.Init(q_{init}),Tb.Init(q_{goal})
for $i = 1 \rightarrow N$ **do**
  a=rand(0,1);
  if (a $\leq$ Pg) **then**
    if (Ta.ConnectToGoal(q_{goal})==SUCCESS) **then**
      BestTrajectory=ExtractPath();
    end if
  end if
  q_{near}=Tb.FindNearest(Ta.LastNode(),N_{FAILS});
  q_{sample}=SampleWithCells(Ta,q_{near},BestTrajectory);
  q_{extend}=Ta.Extend_rrts(q_{sample},BestTrajectory.Cost,N_{FAILS})
  bSucces=Tb.Connect_rrts(q_{extend},BestTrajectory.Cost,N_{FAILS})
  if (bSuccess==true) **then**
    BestTrajectory=ExtractPath();
  end if
  Swap(Ta,Tb);
  Swap(q_{init},q_{goal});
end for

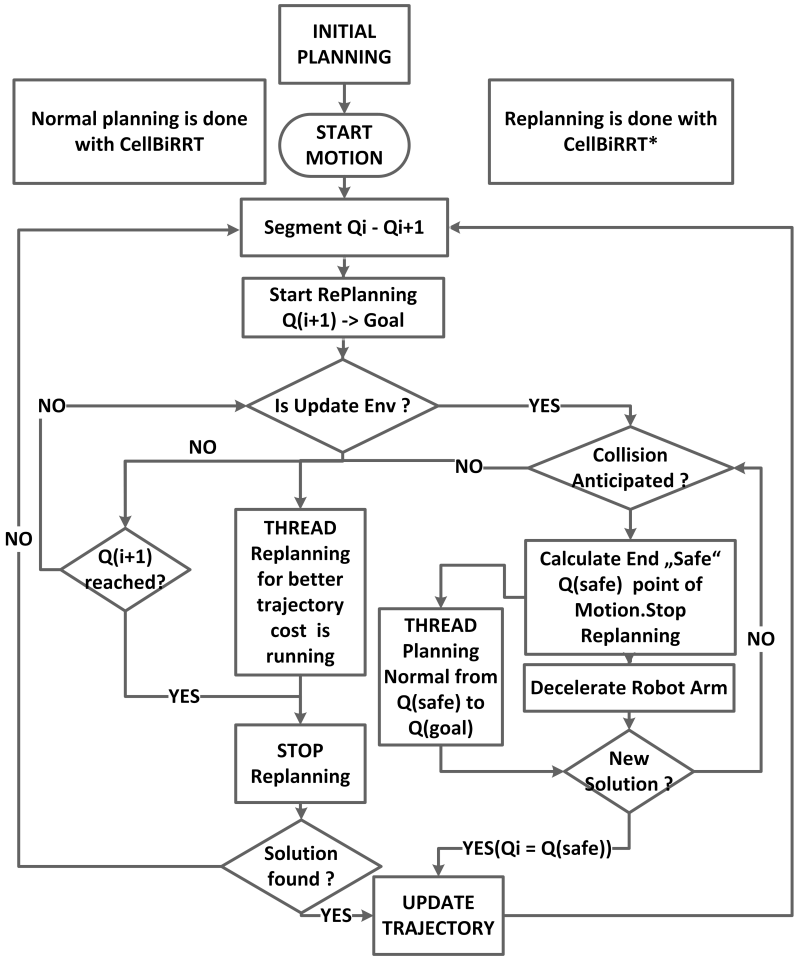
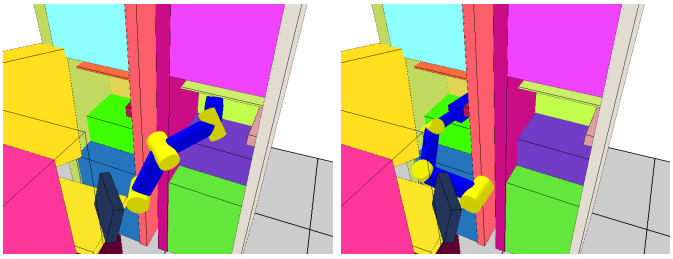


Figure 8.3: Basic Transition States of Robot arm Motion with On-line Replanning algorithm

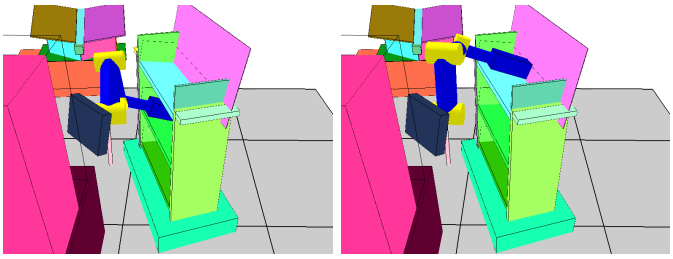
Algorithm 27 *BiRRT**

1: Ta,Tb Trees,q_{init},q_{goal}
2: Ta.Init(q_{init}),Tb.Init(q_{goal})
3: **for** $i = 1 \rightarrow N$ **do**
4: q_{sample}=Sample(i); //sampling is done based on the [AS11]
5: q_{extend}=Ta.Extend_rrts(q_{sample});
6: **if** ((q_{extend} !=q_{sample}) **then**
7: Tb.Connect_rtts(q_{extend});
8: **end if**
9: Swap(Ta,Tb);
10: Swap(q_{init},q_{goal});
11: **end for**



(a) start configuration (b) goal configuration

Figure 8.4: Task1 1- Virtual environment of the robot with start-goal configuration.



(a) start configuration (b) goal configuration

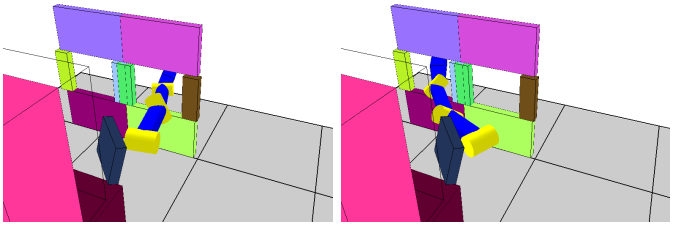
Figure 8.5: Task 2- Virtual environment of the robot with start-goal configuration.

8.4 Experimental Results

Two kind of experiments have been performed. The first one, done in a Intel i5-450M@2.4GHz system, illustrates the performance of the $CellBiRRT*$ in simulation environment. A comparison with the $Bi-RRT^*$, presented by [AS11] in algorithm 27, is done. The second type of experimental results examines the performance of the on-line replanning using $CellBiRRT*$ in the real system.

The figures present the average results between $BiRRT^*$, $CellBiRRT*$ for two scenarios. The simulation environments are illustrated on figures 8.4, 8.5 and 8.6. For the tests it is selected P_{cell} to be 0.9,P_{rand} to be 0.8 and P_g to be zero (simply goal biasing is not included). A maximum of 20000 iterations for Task 1 and Task 3 and 15000 for the Task 2 is specified. The parameters of $Bi-RRT^*$ are the same as in $CellBiRRT*$.

Figure 8.7 as well as the tables 8.1 - 8.3 show clearly that the $CellBiRRT*$ over performs the $BiRRT^*$ for the three tasks. The figure 8.7 illustrates that the $CellBiRRT*$ can deliver faster shorter paths. The tables 8.1 - 8.3 present the result until the first solution is found. The $CellBiRRT*$ for the first two tasks can deliver a path in a deliverable time compared to the $BiRRT^*$ that needs more execution time. Recall also that according to the literature, the $BiRRT^*$ over performs the RRT^*. That makes the $CellBiRRT*$ more appropriate planner. In the last task, which is more cluttered, the $CellBiRRT*$ had 100% success , but the execution time according to table 8.3 makes the planner not appropriate for such an environment.



(a) start configuration (b) goal configuration

Figure 8.6: Task 3- Virtual environment of the robot with start-goal configuration.

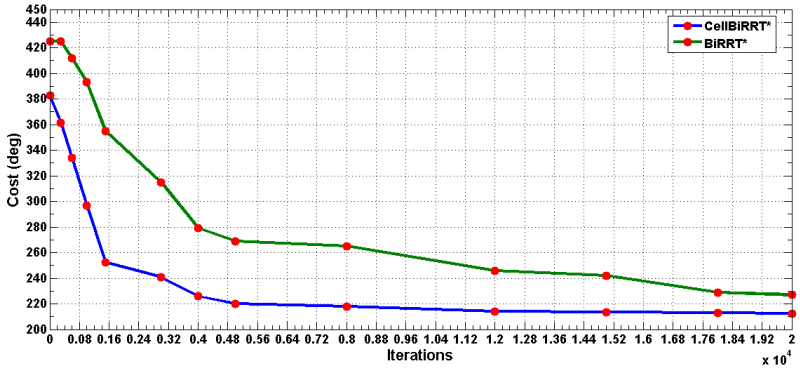
Method	Initial Cost (deg)	Initial Time(msec)
$BiRRT^*$	425	13586
$CellBiRRT*$	382	6495
CellBiRRT	1082	1805

Table 8.1: Average Results for 20 runs for $BiRRT^*$, $CellBiRRT*$ and $CellBiRRT$ - Task 1 - Maximum 20000 Iterations - Shortcutting in the initial paths is not done

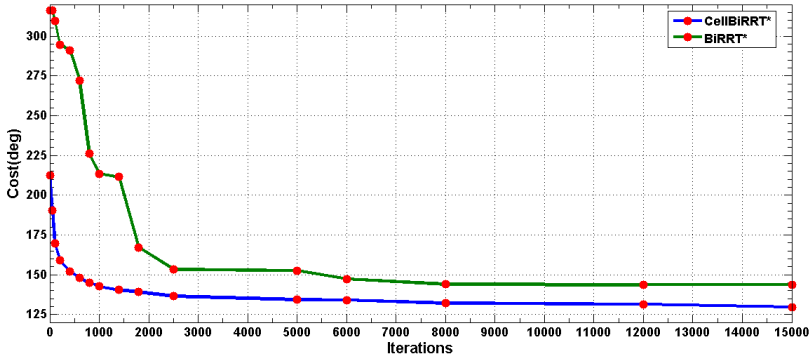
The figure 8.8 presents results of $CellBiRRT*$ over the time with different parameters. It can be seen that for values N_{FAILS}=10 , $Cell_{SIZE}$ = 5 and $step$ = 11 the algorithm can give good results for all tasks. The figure 8.9 compares the $CellBiRRT*$ with the $BiRRT^*$ over the time. It can be seen again that the $CellBiRRT*$ over performs the $BiRRT^*$ especially in more complicated tasks like the Task 1.

The table 8.4 represents the advantage of executing the $CellBiRRT*$ in a parallel thread while the robot arm is moving. A given trajectory is executed by the robot arm and the planner attempts to improve the given path while the robot is moving. The robot follows a third order polynomial with initial start and goal both velocity and acceleration equal to zero (simple point to point motion). From the table can be seen that the presence of $CellBiRRT*$ improves the performance of the system in a cluttered environment. The new paths are shorter and they are created during the motion.

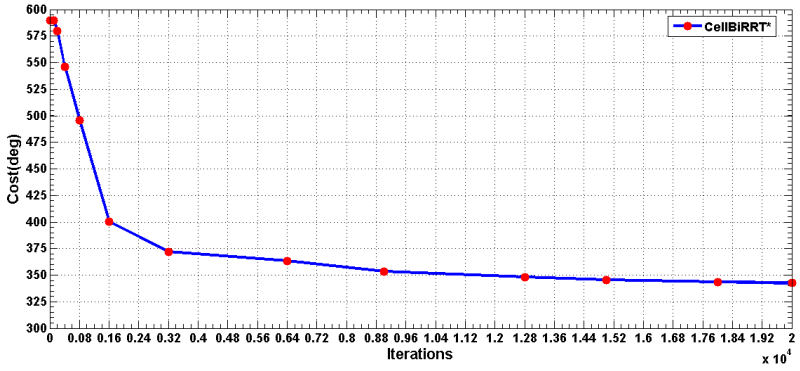
The figures 8.10a till 8.10d illustrate the case where a dynamic environment and on-line replanning exist. The on-line replanning tries to reduce the path cost while the arm is moving. A person approaches while the arm moves in the free space. The system identifies the new object, plans a new path and improves it with the $CellBiRRT*$



(a) Results for Task 1. *step* is equal to 25deg



(b) Results For Task 2.*step* is equal to 25deg



(c) Results For Task 3.*step* is equal to 25deg

Figure 8.7: A graphical comparison between $CellBiRRT*$ and $BiRRT^*$ based on the number of iterations. The $BiRRT^*$ fails in 16 out of 20 runs for the Task 3. N_{FAILS} is 100 for all tests.

Method	Initial Cost	Initial Time(msec)
$BiRRT^*$	316	8574
$CellBiRRT*$	212	2567
CellBiRRT	594	655

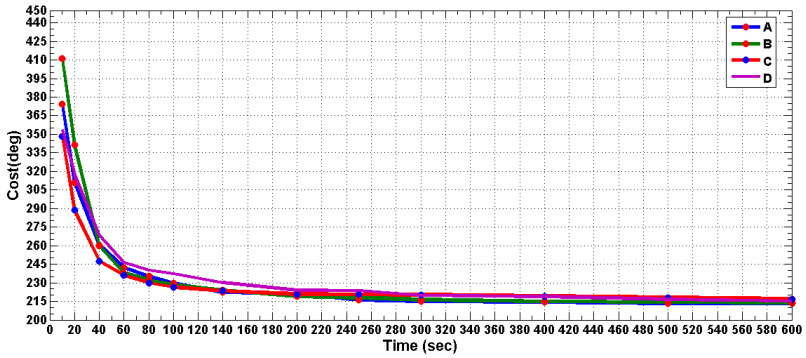
Table 8.2: Average Results for 20 runs for $BiRRT^*$, $CellBiRRT*$ and $CellBiRRT$ - Task 2 - Maximum 15000 Iterations - Shortcutting in the initial paths is not done

Method	Initial Cost	Initial Time (msec)
$BiRRT^*$ (4 success)	2689	37619
$CellBiRRT*$	590	65004
CellBiRRT	2470	24640

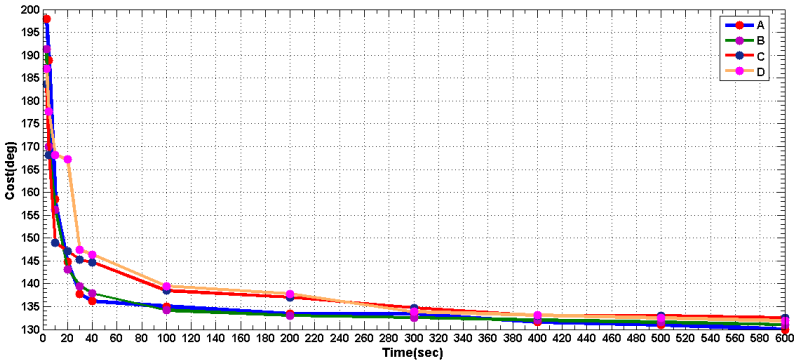
Table 8.3: Average Results for 20 runs for $BiRRT^*$, $CellBiRRT*$ and $CellBiRRT$ - Task 3 - Maximum 20000 Iterations maximum - Shortcutting in the initial paths is not done

Task	Initial Cost (deg)	Final Cost	Improvement
1	328	298	9%
2	209	169	19.2%
3	891	461.73	48.2%

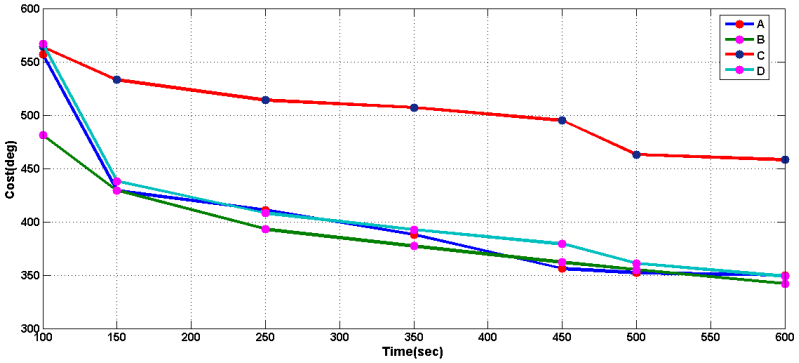
Table 8.4: On-line replanning in static environment. The environment does not change. The Cell-BiRRT* improves a pre-calculated path while the robot arm is moving. The table presents the average improvement for 20 runs for $CellBiRRT*$



(a) Results Cost - Time for Task 1 for $CellBiRRT*$

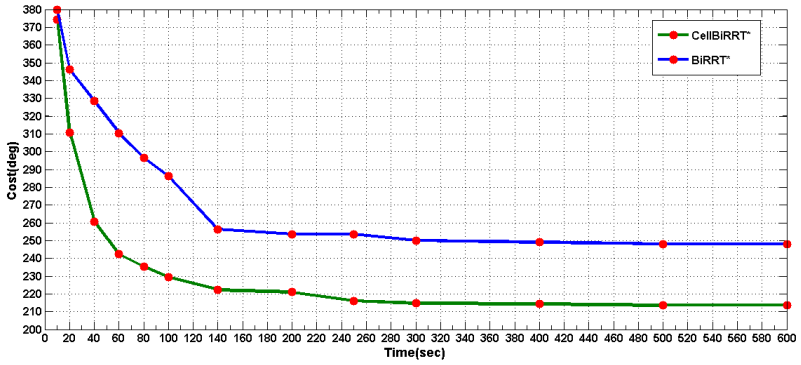


(b) Results Cost - Time for Task 2 for $CellBiRRT*$

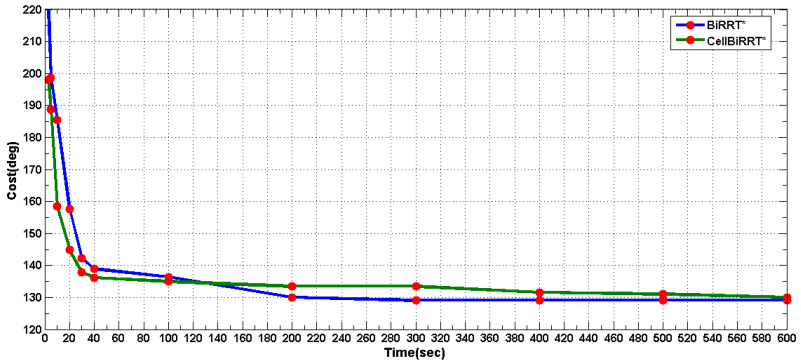


(c) Results Cost - Time for Task 3 for $CellBiRRT*$

Figure 8.8: Results Cost - Time for the three Tasks. The parameters are: A=($N_{FAIL} = 10, Cell_{SIZE} = 5, step = 11$),B=($N_{FAIL} = 10, Cell_{SIZE} = 15, step = 11$), C=($N_{FAIL} = 10, Cell_{SIZE} = 5, step = 25$), D=($N_{FAIL} = 1000, Cell_{SIZE} = 5, step = 11$)

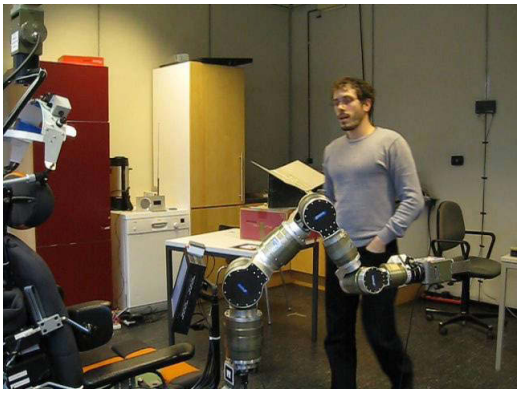


(a) Comparison $CellBiRRT*$ and $BiRRT*$ for Task 1

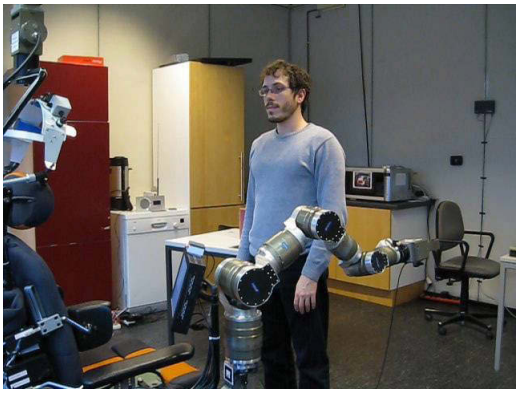


(b) Comparison $CellBiRRT*$ and $BiRRT*$ for Task 2

Figure 8.9: Comparison between $CellBiRRT*$ and $BiRRT*$ for the first two tasks. The Task 3 is not included because the $BiRRT^*$ failed to deliver many solutions. The *step* is equal to 11 deg

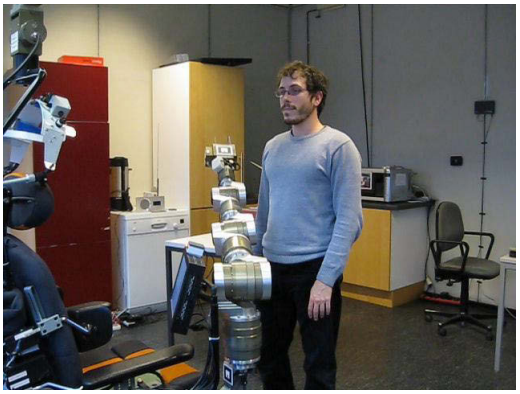


(a) Person approaches the robot while it is moving



(b) System detects collision with the person and computes new trajectory avoiding him

(c) Executes new trajectory, and improves it with $CellBiRRT*$ thread



(d) End of motion

Figure 8.10: Sequence of robot arm motion using on-line $CellBiRRT*$ replanning

Chapter 9

Extensions and Application of motion planning algorithms - Grasping - Control and design manipulative skills for ADL and library scenarios - Motion Planning Library

This chapter deals with the extensions and applications of the planning algorithms. Till that point the described planners deliver a path between a starting configuration and a goal one. However there are situations where the tasks may involve more than one goal i.e. grasping a bottle. For such a reason an extension in the sense of multiple goals is necessary.

First an enhancement using Workspace Goal Regions is presented. Later an approach of sharing control of the robot arm using the planning algorithms is described. The user is able to execute tasks with a shared autonomy between him and the system. The third part of this chapter presents briefly the implementation of manipulative skills done in parallel with the development of the planning algorithms. The last section shows in UML diagram the planning structure with the interfaces and structures. All of them are combined in a motion planning library.

9.1 Workspace(WGR) and Object Goal Regions (OGR)

This section describes an important feature that can be included in any planning algorithm. The workspace goal region first introduced by [BKDA06, VBA+09] and is formulated better in [BSF+09]. The workspace has three dimensions for position and three for orientation, therefore the workspace goal regions is six dimensional. The workspace goal regions (WGR) defines the area where the end

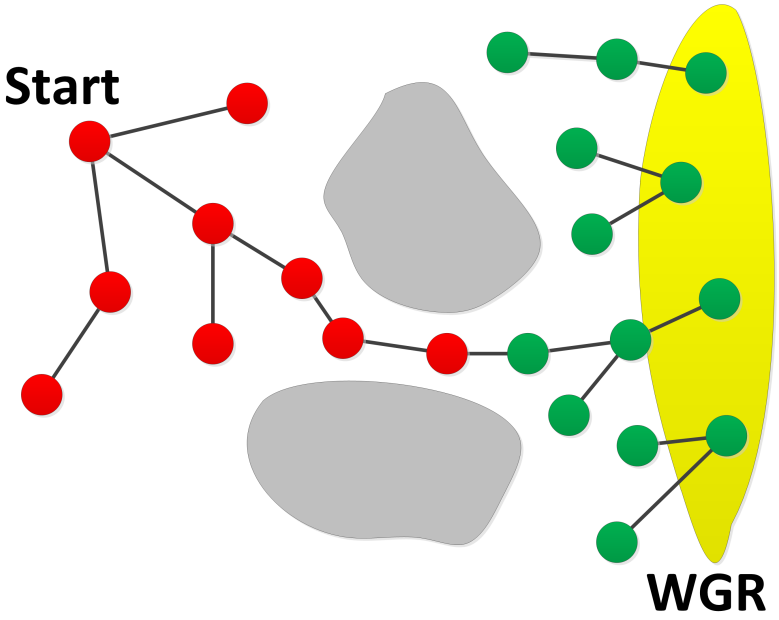


Figure 9.1: Workspace Goal Region(WGR)

effector of the robot arm can move. This area is limited by the maximum and minimum limits of the location (position and orientation) and equals to:

$$WGR = W = W^w = \begin{pmatrix} x_{min} & x_{max} \\ y_{min} & y_{max} \\ z_{min} & z_{max} \\ RotX_{min} & RotX_{max} \\ RotY_{min} & RotY_{max} \\ RotZ_{min} & RotZ_{max} \end{pmatrix} \tag{9.1}$$

This area is mapped to a region in configuration space as the image 9.1 illustrates. The symbol w presents the reference coordinate system where the WGR is defined. In this work the reference frame is equal to the robot basis e.g. the world coordinate frame. The rotational parts (e.g. RotX,RotY and RotZ) correspond to the Euler angles. It is clear that a task can have many of this regions each one allocating a target space for the robot arm's end effector.

The WGR regions can be integrated to all planners described so far. The reason is that all planners require a goal frame (a configuration can be calculated by inverse kinematics). The goal frame can be calculated randomly by the WGR. The random location is computed by the equation 9.1. Simply, if T notates a frame and $\{e\}$ the end-effector, a random frame is given by:

$$T_e^w = T_{sample}^w = random(W^w) \tag{9.2}$$

A work space goal region can be extended to refer to an object. For instance consider the situation of a cylindrical object like a bottle (see figure 9.2). Consider $\{o\}$ the coordinate system

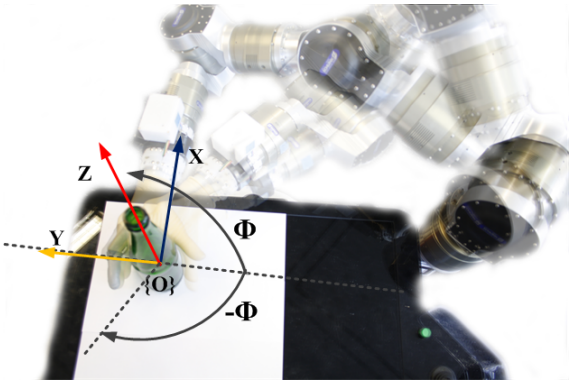


Figure 9.2: Different grasping poses of the robot arm around a cylindrical object e.g. bottle

of the object and the T_o^e the relative frame between the end effector $\{e\}$ and the object frame. The frame T_e^o corresponds to one grasping pose. For that reason the **o**bject's **g**rasping **r**egion (OGR) like the WGR can be defined. The OGR^o simply assigns all possible positions and rotations that an object can have and consequently can be grasped. Considering T_{sample} a sample of OGR^o, the random goal T_e^w is defined as follows:

$$(T_e^w)' = (T_o^w \cdot T_{sample})_o^w \cdot (T_e^o) \tag{9.3}$$

Normally during the grasping procedure some offsets are expected. Being T_{offset}^e the offset frame , the final frame is equal to:

$$T_e^w = (T_e^w)' \cdot T_{offset}^e \tag{9.4}$$

Each planner is necessary to be modified in a such a way so that the WGR (e.g. OGR) are included. That can be done by inserting a probability of generating random goal. The algorithm 28 is used in the bidirectional approach like the *CellBiRRT* and the algorithm 29 in the forwards directional approaches. It is an extension of all algorithms used mostly to accomplish manipulative tasks.

The image 9.3 illustrates a sequence of motion done by the $RRT - J_{WLN}$ attempting to grasp a cylindrical object like a bottle. The OGR^o of this object is equal to:

$$OGR^{bottle} = \begin{pmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ -\pi & \pi \end{pmatrix} \tag{9.5}$$

9.2 Share control of robot arm

The planners described in previous chapters can be used to control the robot arm. The FRIEND system is dedicated to serve autonomously disabled people. However practical experimental results

Algorithm 28 WGR for Bi-directional approaches

Require: $T1$(forward),$T2$(backward) trees,q_{start}, q_{goal} start/goal configurations, $PgRandomGoal \in [0,1]$ random value

1: $T1.Init(q_{start})$, $T2.Init(q_{goal})$
2: **loop**
3: a=Ran(0,1)
4: **if** (a $\leq$ PgRandomGoal) AND T1 == T2) **then**
5: P_{sample}=SampleWGR() {Sample the WGR like previously described }
6: $q_{goal} = CalculateConfigWithIK(P_{sample})$
7: $T1.Add(q_{goal})$
8: **end if**
9: ...continue to the algorithm ...
10: $SWAP(T1, T2)$
11: **end loop**

Algorithm 29 WGR for forward approaches

Require: T(forward) trees,q_{start} start configuration,P_{goal} target frame, $PgRandomGoal \in [0,1]$ random value

1: $T.Init(q_{start})$
2: **loop**
3: a=Ran(0,1)
4: **if** (a $\leq$ PgRandomGoal) AND T1 == T2) **then**
5: P_{sample}=SampleWGR() {Sample the WGR like previously described }
6: **end if**
7: ...continue to the algorithm ...
8: **end loop**

(a) (b) (c) (d)

Figure 9.3: Sequence of motions in order to grasp a bottle on the table done by $RRT - J_{wln}$

had shown that the system may fail to execute autonomously a task(for instance due to bad sensing). The user is asked then to control manually the robot arm.

In this thesis two types of share control of robot arm are attempted. Normally, the tasks are:

- to grasp an object
- to maneuver around the obstacles

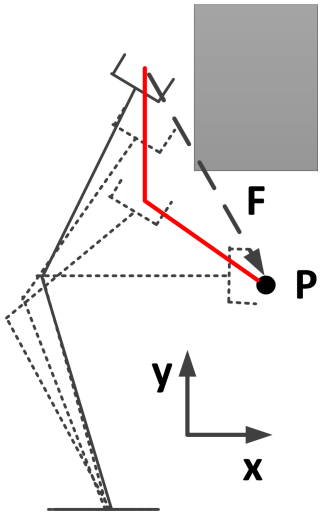


Figure 9.4: Motion example of a planar two DoF robot arm from a semi-autonomous/share control between the robotic system and the user. The robot moves incrementally from the initial configuration to the target location P. The robot end effector is attracted by the "force" coming from the target location

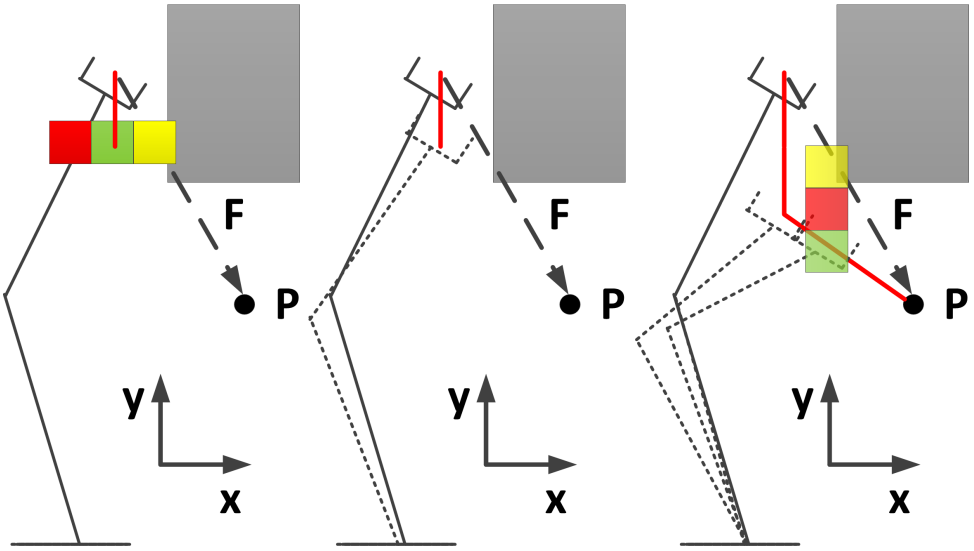
For that reason two types of share control are examined.

9.2.1 Target Oriented Share Control

The main idea of this approach is to have an attractive goal. The end effector is moved incrementally towards the target location. The robot is allowed normally to move towards one of the main directions : up / down, left/right and forward/backward, but it is not possible to move simultaneously in two directions. With this approach its movement is followed by an additional motion in order to reach the goal location. Let consider the planar robot illustrated on image 9.4. The robot failed to go autonomously to the target location and the user is asked to reach the target manually. The user simply commands the robot to move down. At that time the system adds an additional offset in $\widehat{X}$ direction. This offset depends on the distance $\vec{F}$ which is the vector that connects the current end effector position with the target's one. The system assists the user to reach faster the target location by attempting to move simultaneously to additional directions. The step of each motion equals to:

$$step_i = \begin{cases} & \text{step} & \text{, if } motion_i \text{ is selected} \\ \text{OR} & F_i & \text{if } F_i \neq 0 \\ \text{OR} & 0 & \end{cases} \tag{9.6}$$

The $step_i$ in the equation denotes the length of the step and the i the direction of motion e.g. x,y or z direction. The F_i denotes the value of the i coordinate of the vector $\vec{F}$. This equations defines that if the user does not select the i direction, the robot arm either does not move towards



(a) Green cell is selected (b) Result of the motion towards -$\hat{Y}$ (c) Result of the motion towards +$\hat{X}$

Figure 9.5: Example of share control for stepwise direct line motion in direction -$\hat{Y}$ and +$\hat{X}$. The yellow cuboid is in collision, the red one is rejected due to the $\vec{F}$ direction. Therefore the green one is selected. Orientation and position are calculated as in previous chapters and the robot's end effector moves directly to this location

it or it moves with a step of the F_i. In order to eliminate big motions due to a big value of the $|\vec{F}|$, the value of the maximum F_i is limited to the given step (equation 9.7).

$$F_i = min(F_i, step) \tag{9.7}$$

This approach uses cells to decide the next state of the end-effector. Consider the example of the image 9.4. At the beginning the user decides to move on -$\hat{Y}$ direction e.g. down (see figure 9.5). Due to the direction of motion, three cuboids are generated and tested for (a) collision and (b) decreasing the distance between end-effector and target location. In this example the green one is selected and the red one is rejected. In any case if the green and the yellow cells were in collision the remaining one is selected. If all of them were in collision the robot doubles its step and checks again otherwise the user is informed that the required motion cannot be done. Continuing the same procedure, the robotic arm gradually reaches the target location and the user does at least the half of the work since the robotic system simultaneously moves the end effector towards the +$\hat{X}$ direction. The same procedure is followed if the user selects to move towards the +$\hat{X}$ direction. In this situation the system moves simultaneously towards the -$\hat{Y}$ direction (see figure 9.5(c)).

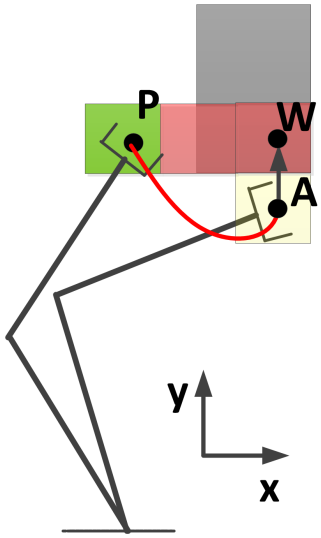


Figure 9.6: The user decides to move the robot arm in positive $\vec{Y}$ axes. The robot normally is not able to move since its next state is in collision. The user should have moved the robot's end effector backwards in order to free his place. With share control the robot is able to move at once the path AP. The red color denotes cells being in collision

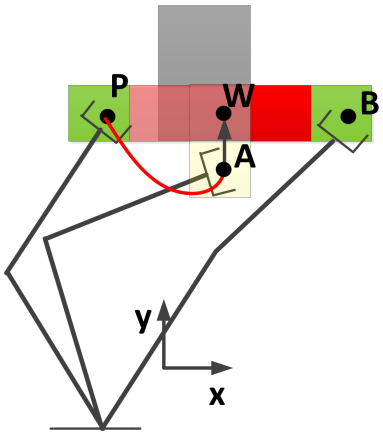


Figure 9.7: Example with two possible configuration corresponding to the location B and P. the P is selected because the manipulability is bigger

9.2.2 Semi autonomous maneuvering around obstacles

The semi autonomous maneuvering follows almost the same strategy as the target oriented share control. Now a target location is not given. Cells are used also here in order to identify possible free space around the obstacle.

Consider the example in figure 9.6. The robot is close to the obstacle and the user decides to move the robot arm far from obstacles. In order to accomplish it the user should move stepwise

backwards and later to go to the position P. For such a situation an automatic solution is necessary.

The workspace around the actual end effector location is divided again into cells. Consider the same example but now the robot has to move stepwise to $+\vec{Y}$ direction. The new location is calculated but because the robot arm is in collision, cells are examining the near area. The size of cell is equal to the step. The searching is stopped when a cell is collision free. Each cell is examined if there is (a) collision or (b) if the possible solution e.g. configuration, is more appropriate from the others. All possible solutions are included into a set Q. The selection is done based on the manipulability and the calculated distance from the obstacles. If the robot has difficulty to move like being stretch or close to singularity the manipulability is very low. These situations have to be excluded. The second criteria is the minimum distance. The configuration that has bigger distance from the obstacles is favored. Putting the two criteria together, a configuration is selected which satisfies:

$$q = \arg\min_{q \in Q}(A \cdot \frac{1}{Manipulability} + (1 - A) \cdot \frac{1}{MinimumDistance}) \tag{9.8}$$

where $A \in [0, 1]$. The A should have a big value, so that the configurations that have big manipulability are favored. Consider the example in the image 9.7. There are two configurations corresponding to locations P and B. The distances AB and AP are the same. The red cells indicate that are in collision. Although the configuration corresponding to the location B has bigger distance from the obstacle, the configuration that corresponds to location P is selected. The reason is obvious. The manipulability is bigger and that may assists the robot arm to move later to another position. The position B is not appropriate since the robotic arm is almost stretched.

9.3 Manipulative Skills

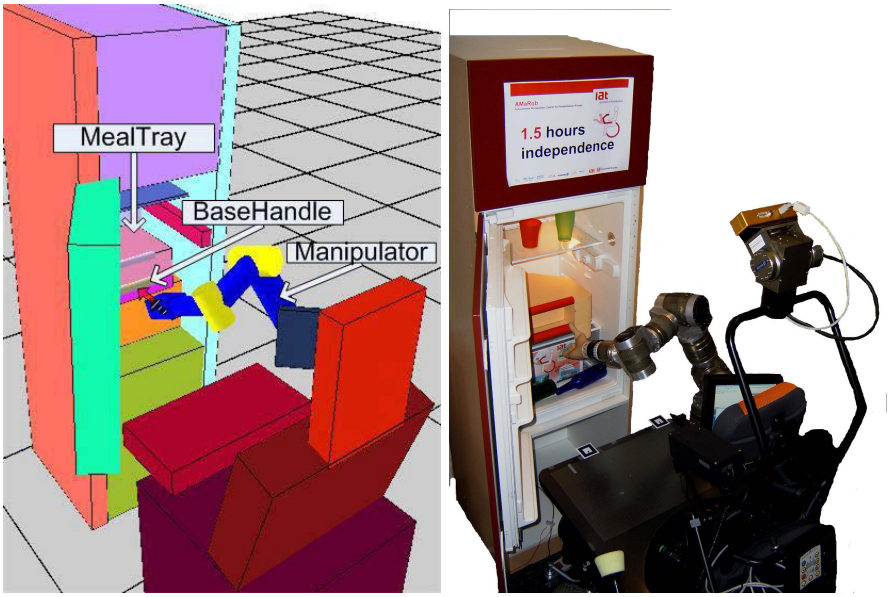
Manipulative skills are very crucial since several tasks should be accomplished in a given scenario. For instance the user asks to grasp a bottle, to fill a glass with water, to cook e.t.c. All these tasks include several sub motions which are executed in a sequence. The design of the sub motions is discussed in this section.

Manipulative skills include the ability of the system to execute some tasks like to grasp an object, to place it to another position, to close a door and many others. Automatic control e.g. motion planning is important so that the robotic system will be able execute the sub-tasks automatically. The planning algorithms described so far can be used to calculate the path for a motion. Preferably the *CellBiRRT* has been used but the others can be applied too.

The described scenarios are:

- Activities of Daily Life (ADL) (project AMaRoB)
- Working on a library (project ReIntegraRob)

Both scenarios require skills,which simply include a sequence of motions done by the robotic arm. In order to accomplish these sequences, three helper skills are developed:



(a) Grasping a mealtray modeled in MVR

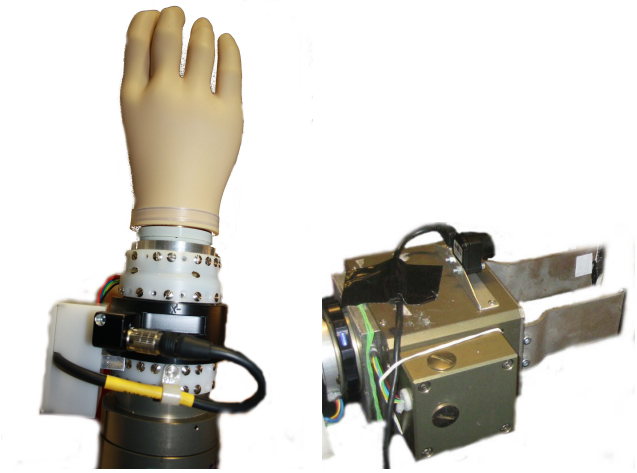
(b) Grasping a mealtray in reality

Figure 9.8: Task "Grasp a mealtray" presented in 3D modeling and in reality

- PlanAndMoveGripperToLocation: This method places the end effector to a given location/frame. The inverse kinematics calculate the necessary target configuration if needed. This method requires the planning parameters (extracted from a database) and the final location/ frame of the end effector. The difference from previous versions is the presence of some additional parameter: the WGR and the constraints.
- PlanAndMoveObjectToLocation: This skill places a gripped object to a given location/frame. The skills automatically detects if there is a grasped object and consequently calculates the target location of the end effector. This method requires the planning parameters and the final location/ frame of the object.
- GraspObject: This skill executes the scenario of automatic grasping an object. Given the name of the object the skill loads from a database the WGR region corresponding to the object, the planning parameter and finally the robot executes automatically two sub motions. The first one is the coarse approach till a pre location. The robot places the end effector close to an object. The second one is a direct line motion till the grasping location(final location). The image 9.8 shows an example where the robot arm has to grasp a mealtray.

9.3.1 ADL Scenario

This scenario has been conducted from 2008 till 2011 and the system is illustrated in figure 2.1. This scenario is done within the research project AMaRob. The aim of the project is to work on



(a) (b)

Figure 9.9: Two different grippers: the left one used in the ADL scenario and the right one in the Library scenario

the fundamentals of a rehabilitation robotic system that is suitable for everyday use. The goal is to support people with multiple handicaps for at least 1.5 hours continuesly without further support by care personal (refer to project's web site [IATb]).

The system FRIEND should execute the following sub tasks: Grasp a mealtray from a fridge, moving away from a fridge, open the microwave(it is not done by the robotic arm), placing the mealtray in the microwave, closing the door, warming the food (it is done not by the robotic arm), grasping the mealtray from microwave, placing it on the tray/tablet and serving the person. Except the opening of the door of the microwave and the warming of the food, the rest is done by the robotic arm. The gripper used for this purposed is presented in the figure 9.9(a) .

All the manipulative skills use the same strategy which has two phases:

- Learning phase: During this phase the system learns / stores the relative location between the end effector and an object or a location. That is done by moving manually the end effector to the desire location and afterwards storing the relative frame to a database. That strategy is very helpful for instance in case of grasping an object. With this procedure an initial relative location between the end effector and the target object is stored in the database.

- Execution phase: During the execution phase the relative frame is used to calculate the final end effector frame. The end effector frame is calculated as it is described in previous sections, by multiplying the relative frame with the object's frame.

The rest of the text describes shortly the manipulative skills developed to fulfill all the tasks:

- *CloseDoorByFTSControl*(see figure 9.10): The manipulator closes the door of the microwave with the thumb of the end-effector. Firstly the robot moves parallel to the microwave, and

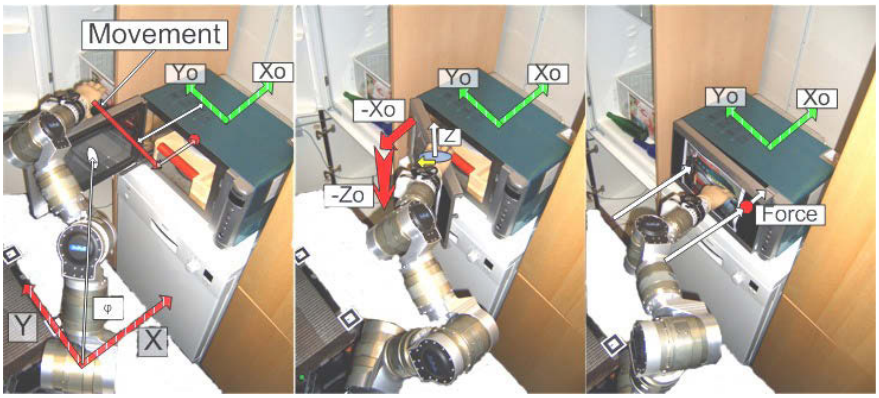
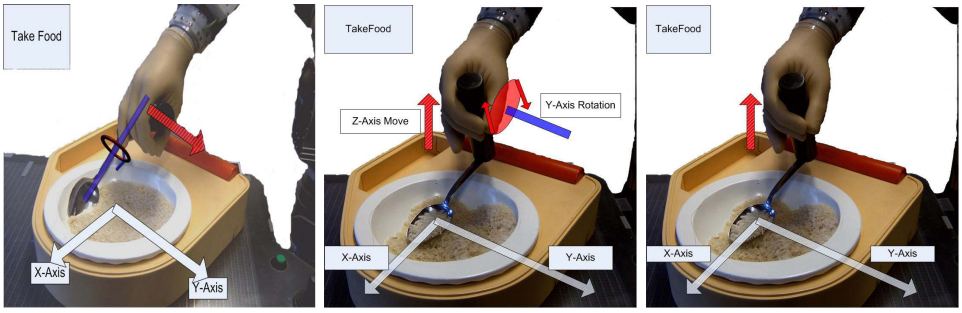


Figure 9.10: Sequence of motion to close the door of the microwave



(a) (b) (c)

Figure 9.11: Sequence of motion for the skill TakeFood

later it pushes the door to close. The force sensor, that is mounted on the end-effector, detects if the door is closed. The skill *FTSensorMonitoring*(described later) informs the manipulator that the required force is applied.

- *TakeFood* (see figure 9.11): That skill is a composition of a sequence of motions. The spoon being grasped by the robot arm should be rotated in order to take the food from the mealtray. Experiments shown that a motion parallel to the $\widehat{Y}$ of the mealtray can take safely the food. Afterwards the spoon is raised up and then the food is served slowly to the user.

- *MoveObjectInByFTSControl*(see figure 9.12): The object e.g. mealtray is going to be placed inside the microwave. Due to the fact that the sensors (cameras) do not provide high accuracy, the force sensor is used for fine placing of the mealtray into the microwave. Each time the mealtray is in contact with the inner sides (left or right part) of the microwave, the robotic arm moves backwards towards the force direction. This reaction provides higher accuracy by placing the mealtray in the microwave.

- *FTSensorMonitoring*: The abbreviations *FT* comes from the Force - Torque. The Force

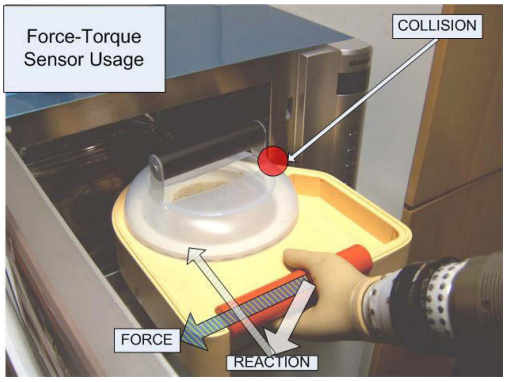


Figure 9.12: Move the mealtray into the microwave using the Force sensor

sensor mounted on the gripper of the robotic arm (see figure 9.9) is able to measure the forces that are applied on the gripper. It is an additional advantage and it has been used within this project. The skill reacts sending the necessary messages to the robotic arm e.g. contact or not contact. The monitoring of the force is executed in parallel with the robot motion and it tries to detect if a peak e.g. sudden change of forces is appeared. That is done at real time so that the robot will be able to react as fast as possible. The data taken from the sensors are filtered with an average filter and secondly the derivative of the signal detects sudden changes. The image 9.13 presents an example where the end effector collides with an obstacle. The force signal changes suddenly and it is detected by the system as a collision. The robot arm returns back to a safer position.

Basic functions like the *PlanAndMoveGripperToLocation*,*PlanAndMoveObjectToLocation* and *GraspObject* are used by every skill in order to execute the sub motions.

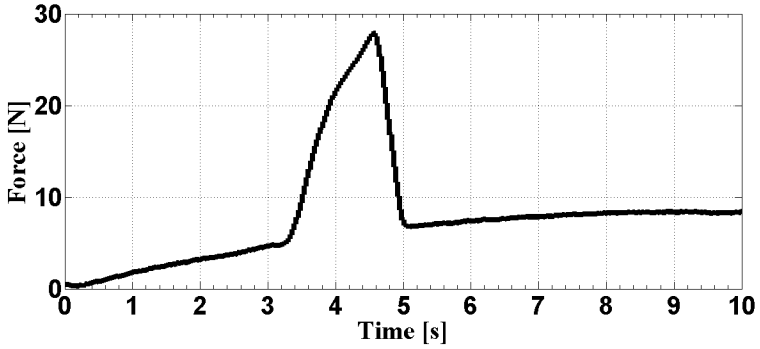
The ADL scenario has been successfully demonstrated in many exhibitions like RehaCare@2009 and Hannover Messe@2010. Many videos demonstrating the scenarios are in the web page [IATa].

9.3.2 Library Scenario

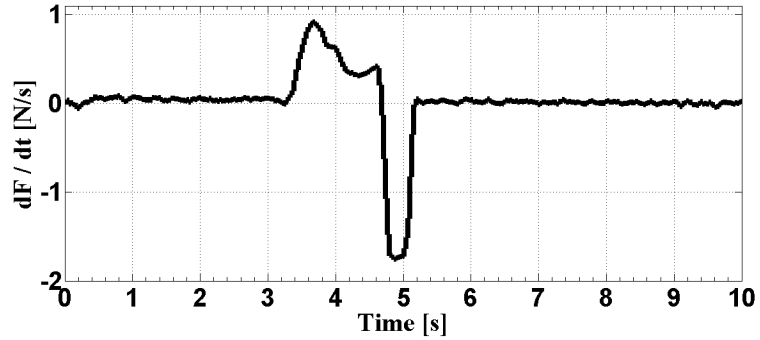
Comparing library scenario and ADL scenario there is a difference which is the replacement of a hardware part. The open/close "anthropomorphic" hand gripper is replaced by an industrial open/close parallel gripper (image 9.9(b)). The parallel gripper has some advantages and drawbacks. The advantage is its simplicity since a parallel gripper has only one degree of freedom(DoF) compared to a situation where multiple joints on the gripper are present. However the disadvantage is the lack of multiple sensing, grasping points or even flexibility that a gripper with more DoF may provide. The manipulative skills, described in this section, are therefore developed for a parallel gripper.

The aim of this project is the system FRIEND to give the ability to the user to return back to working life. His task is to catalog books. For that reason four skills are developed:

- Grasp a book from book cart



(a) Signal captured by the FTSensor



(b) Derivative of the captured signal

Figure 9.13: The signal by the FTSensor and its derivative. If the derivative exceeds a limit the robot arm stops its motion and reacts either by moving backwards or it stops

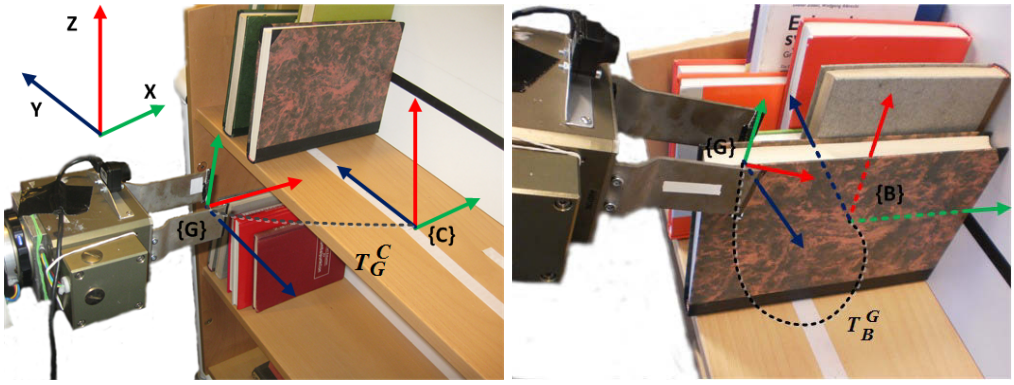
- Place a book on the book holder
- Grasp a book from the book bolder
- Place a book to the book cart

All the above skills use almost the same strategy: The calculation of a relative frame between the end effector and an object and later to apply relative motion towards the object.

- Grasp a book from book cart: The skill contains mainly three parts: the first one is the coarse approach of the end effector in front of the book cart. The second motion is a visual servoing task which description is out of the scope of this thesis. The last motion is the fine tuning to grasp reliably the book. In this thesis the first and the last part are going to be described briefly.

  The image 9.14 depicts of the end-effector's and the book cart's frame. The relative frame is calculated as follows:

$$T_G^W = T_C^W \cdot T_G^C = T_C^W \cdot (Rot_X(\pi) \cdot Rot_Y(\pi) \cdot Trans_C) \tag{9.9}$$



(a) Coarse approach of end effector in front of the book cart

(b) Fine tuning for reliable book grasping

Figure 9.14: Coarse approach of end effector in front of the book cart and the fine tuning for reliable grasping [HFHG12]

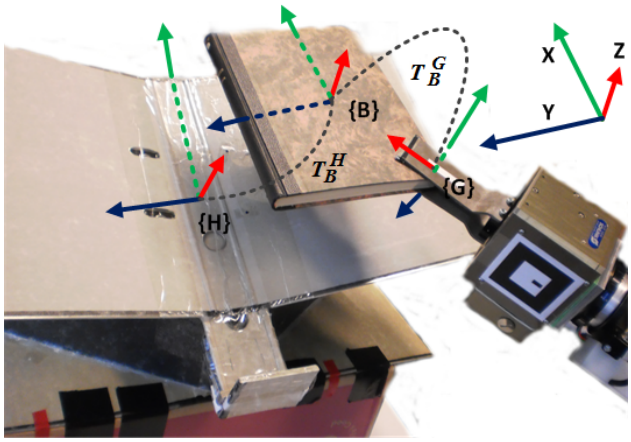
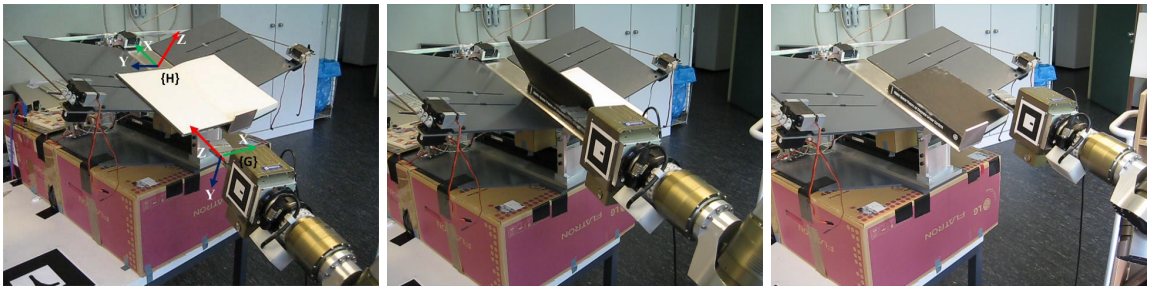


Figure 9.15: Place book on a book holder [HFHG12]

where $Trans_Y$ is a translation in $\{C\}$ coordinate system for better alignment (closer of further to the book cart). The grasping is done by rotating and moving the end effector towards the book. The book should be placed between the plates as the image 9.14(b) shows. The gripper is ready to close and the robot arm detaches the book from the shelf.

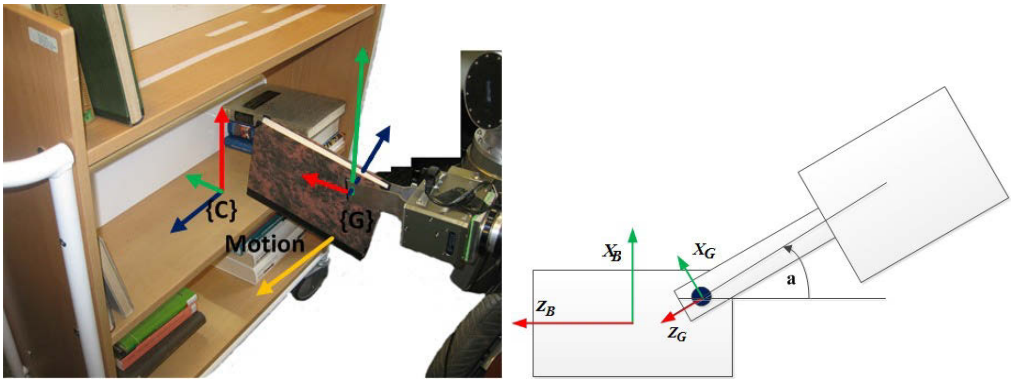
- Place a book on the book holder: The strategy of the skill is similar to the one done in ADL scenario. The image 9.15 presents the frames needed for the placing. The relative frames T_B^G and T_B^H are needed. The frame T_B^G is calculated when the book is grasped by the gripper. The book is recognized by the sensors and the relative location between the book and the gripper is calculated using the actual location of the book and the end-effector. The frame T_B^H is extracted from the database. The T_B^H can have two values. The reason is the placement



(a) (b) (c)

(d) (e) (f)

Figure 9.16: Sequence of motion for grasping a book from book holder



(a) (b)

Figure 9.17: Place a book back on the book cart's shelf

of the book on the book shelf. The books are placed levelly - straightly on the shelf resulting to two possible T_B^G and T_B^H frames.

- Grasp a book from the book bolder: The image 9.16 illustrates the sequence of motion done by the end-effector to close the book and to grasp it from the book holder. Similarly to previous skills, the frame T_H^G is required. The frame is extracted by the database. Afterwards

the end effector moves towards the book holder and closes the book (figures 9.16(b)- 9.16(f)) .

- Place a book to the book cart: Like the other tasks the calculation of the end effector's location has to be done (see figure 9.17). This scenario has many similarities with the first task since the relative frame between the gripper and the book cart is used. The calculations remain the same except the last step where an additional rotation on the gripper is done. The rotation α (see figure 9.17(b)) is done around the $\hat{Y}_G$ axis. The aim is the book to lie parallel to the shelf. The rotation α can be computed easily by calculating the rotation from the relative frame T_B^G.

One main difference , except the described skills, between the ADL scenario and the library scenario is the reliability. The basic kills like *PlanAndMoveGripperToLocation* have been developed so that multi threading control is available. For instance, the planner and the motion of the robot are able to stop simultaneously. The reliability is totally increased. The software is developed so that the platform will be close to real product.

The library scenario has been successfully demonstrated in the RehaCare@2012 exhibition [IATc].

9.4 Motion Planning Library for Manipulators

This section describes motion planning library. It includes all the described so far planning algorithms as well as some additional features. The requirements of such a library are:

- to be independent of the platform e.g. to be able to be applied to other robotic systems
- to be as simple as possible for the end-user programmer
- to be reliable and stable e.g. to support multi threating and stable code

These three requirements are necessary to be fulfilled.

The library , called *Open Motion Planning Library for Manipulators*, has the following features:

- Supports a wide range of basic planning algorithm like RRT,RRT-JT,CellBiRRT e.t.c
- Supports planning with additional constraints like in position and in orientation
- Supports several trajectory generation profiles (using polynomials or Reflexxes Library [Krö10])
- Supports offline and online path smoothing like Pruning or Online Shortcutting [HNTH10])
- Supports multi threading
- Supports efficient collision detection using bubbles and OBBs for faster computation time as well as different collision detection packages (GJK,SWIFT, PQP)

Algorithm 30 Example using the Motion Planning Library

MotionPlanning::Planner::CPlanner<Tplanner, Tparameters> Planning;
Planning.ConnectToMVR();
Planning.SetEnvironment(...) ; // set the environment of the planner
IPlannerInterface *pPlanner = Planning.GetPlanner();
pPlanner− >InitPlanner(MVR,RobotDHParam, MinDistance,RobotName) ; // RobotDHParam is an IKinematics and the RobotName is used in order to control the input of the robot. Internally initializes the object that inherits by the IKinematics and implements the kinematics of the specified robot
pPlanner− >SetParameter(.. ,PlannerType ,...) // set global parameter like start, goal, WGR, constraints , the planner type from the group *Tplanner* like CellBiRRT, $RRT - J_{WLN}$
pPlanner− >SetParameterPlanner (....) // set specific planning parameter if needed
Planning− >Solve() // try to solve the task
Planning− >Smooth(...) // smooth the path
Planning− >ExecuteTrajectory(....) // executes the trajectory based on the profile

The library is based on interfaces which define the complete structure of the library. Every programmer should follow the interfaces or classes. The core of the library is encapsulated into the following interfaces and classes:

- CPlanner [template class]: This class is a template, and as argument can be any implementation of the *IPlannerInterface*. This container is the basis and it is very important since it is in the top level and acts as an intermediate between the calling application and the planner. The class returns the interface and later the programmer can use this interface for the rest of teh work. This class initialize the environment and can add or delete elements on it.

- IPlannerInterface [interface]: It contains the necessary methods used by the planner. The implementation of this interface should have all the planning methods like CellBiRRT, CartesianRRT e.t.c. In addition methods like *SetParameters*, *Initialization* and *StartPlanning* are included.

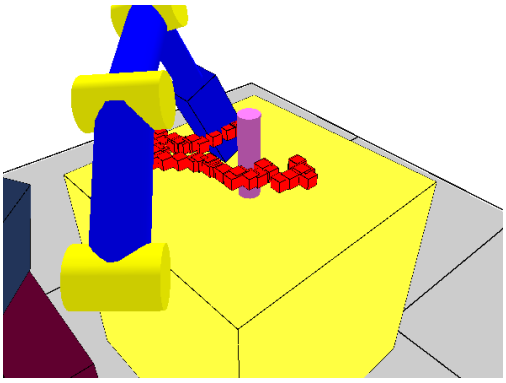
- IKinematics[interface]: This method contains the kinematics of the robot arm. Each robot arm should have its own kinematic structure. The rest of the classes and methods should use this interface. The code remains global and can be used by any robot.

- CVertex [class]: This modules uses partially the IKinematics and the collision detection class(in this thesis is the MVRserver). It contains the data of a state of the system, it can check for collision and it can solve inverse or forward kinematics.

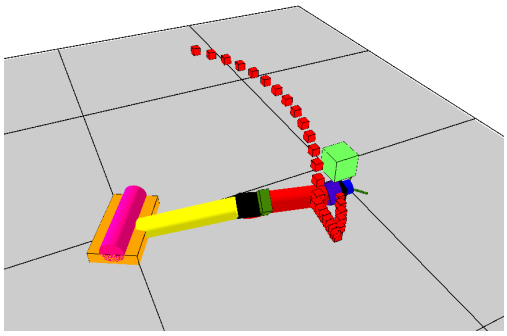
- CEdge [class]: This module is responsible for the connection between two nodes. Calculates the norm (distance) between two nodes and can check if the segment between two nodes is collision free.
- IGraph [interface]: This interface contains the methods that are used by graphs. Functions like *FindNearesT*, *AddNode*, *Find-K-Nearest* are included. The programmer should implement the functions of the interface accordingly.

The figure 9.18 presents some examples where this library is applied in several robotic arms. The parameters of the planner remain the same for all experiments and robots. This is an advantage of using the proposed planners since the planners can solve easy as well as difficult tasks.

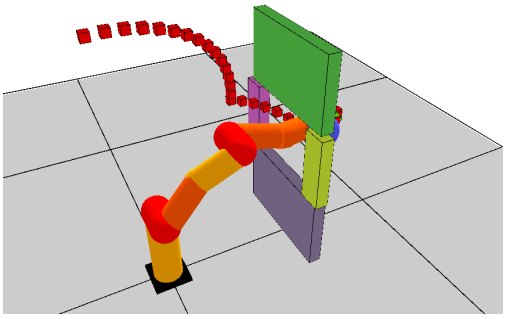
The UML diagram of the implementation is shown on figure 9.19. An example using the library is presented in the pseudo-code 30. The red color indicates the changes that are necessary to be done if the group of planners changes. The blue one are the places that need to be adapted if the robot type changes. The pink color denotes the places where the planner type from the group of planners is changed. It is clear that there are only three places that have to be modified if a new robot or a new planner is used.



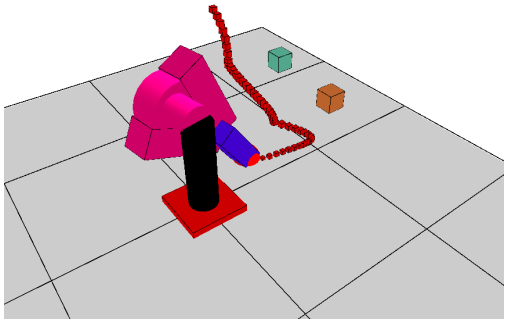
(a) SCHUNK 7DoF robotic arm



(b) WAM 7DoF robotic arm



(c) LWR 7DoF robotic arm



(d) PUMA560 6DoF robotic arm

Figure 9.18

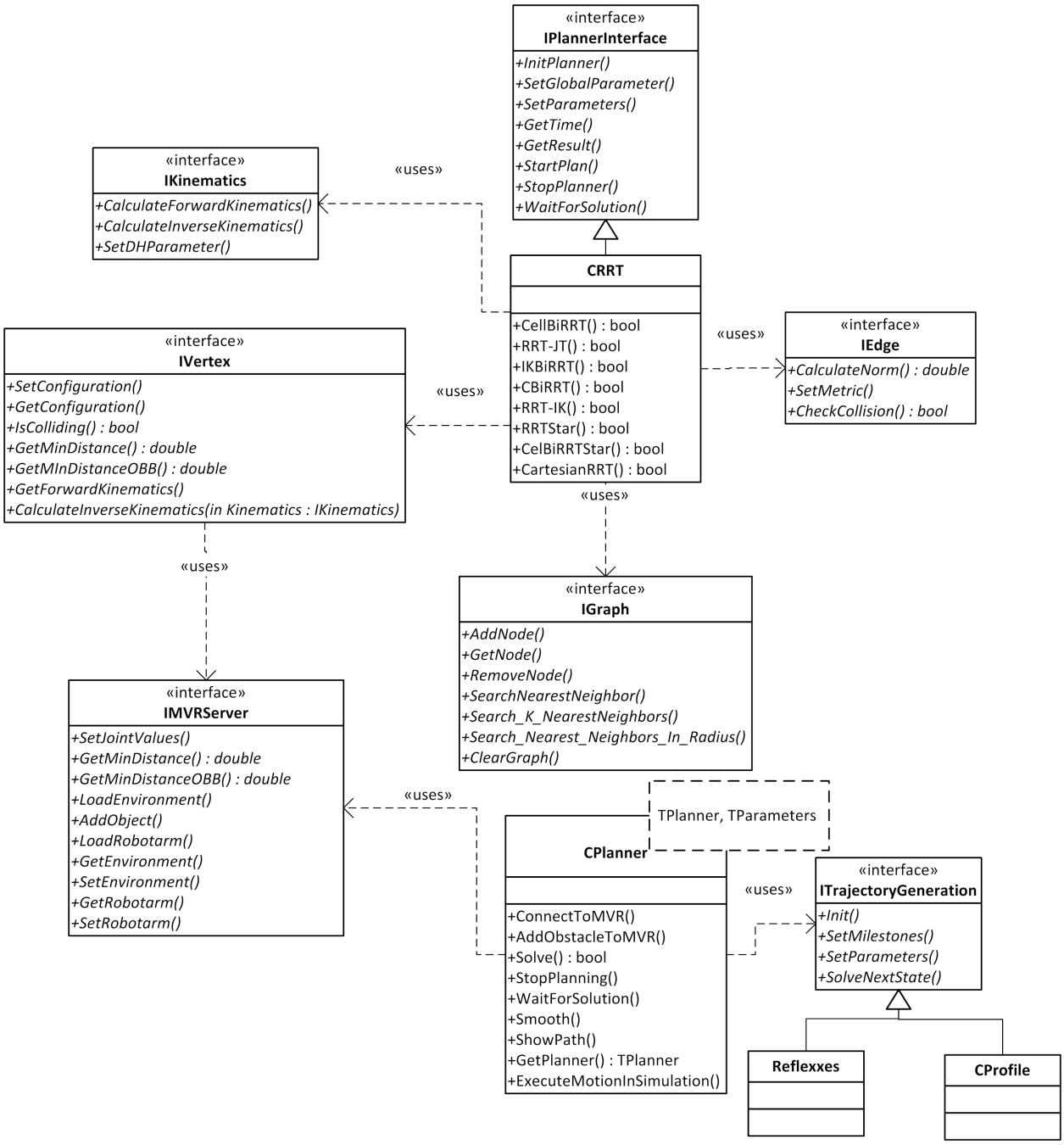


Figure 9.19: UML diagram of the Motion Planning Library

Chapter 10

Conclusions

In robotics research community the autonomous manipulation remains a challenge. That autonomous behavior requires the design of motion planning algorithms that are efficient to solve a task. This thesis describes several motion planning algorithms that accomplish these requirements: they can solve fast several kind of tasks (from very open to very cluttered tasks) without adding complicated and special heuristics that may be needed by a task. Another important topic that is discussed in the thesis is the optimality. A new algorithm is presented promising to minimize the cost of the path in a reasonable time. The performance of all planners is really comparable (even better) with the state of the art explaining the reason of using them in several applications.

The first described algorithm is called CellBiRRT and it is a sampling based approach. The planner works in configuration space, however it uses the Cartesian space in order to generate random configurations. The CellBiRRT uses the main characteristic of the bidirectional RRT i.e. expands randomly two bidirectional trees attempting to connect each other in each iteration. The CellBiRRT involves two main contributions: the first one is the generation of N-dimensional cuboid regions, notated with R_q, and the second one is the appropriate selection of cell. The region R_q is a space around a configuration q. The last expanded configuration from a tree or the one generated by the cells is used as configuration q. Random configurations are generated inside this space. The second contribution, the cells, are generated by cell decomposition of the Cartesian space. The algorithm detects the cell where the end-effector belongs to and it examines its neighbor cells. The location of the end-effector is calculated by the position of the selected cell. The center configuration of the cell is calculated by the inverse kinematics. The center configuration is used as configuration q for the N-dimensional cuboid region. Experimental results showed that the CellBiRRT without cells but with the region R_q provides better results compared to the Bidirectional RRT. The cells improve the performance and also improve the quality of a path. The planner solves efficiently tasks with or without additional constraints on the end effector. The main drawback of the planner is that the calculated path does not remain the same after each execution. That is a characteristic of the sampling based approaches. However, the feasibility of the planner is very high and the calculated path does not vary much. For all the above reasons the CellBiRRT is selected as the main planner for the described applications.

The second part of the thesis presents forward directional sampling based approaches ($RRT - J_{WLN}, RRT_{IK}, CartesianRRT$) e.g. approaches where only one tree grows. The expansion is probabilistically controlled which means that the tree expands either towards the goal or towards a random configuration. The regions R_q as well as the cells are also applied here(except the CartesianRRT). The expansion towards the goal is done using either analytical inverse kinematics (IK) or Jacobian based approaches. Except CartesianRRT, the rest of the algorithms are using cells, Jacobian based approach for inverse kinematics and regions R_q. The main challenge for the algorithms is the linear interpolation of the end-effector between two locations in Cartesian space. That is accomplished by using quaternions (Slerp) for the rotational part and linear interpolation for the position part. Experimental results show that the planners are really challenging. Especially the Jacobian based approaches provide nice and comparative results in simple as well as very cluttered environments with the CellBiRRT. In cluttered environments may calculate a solution faster compared to the CellBiRRT. Despite the fact that they are very efficient, the resulted path contains high number of configurations. A smoothing as well as fast controlling of the joints velocities during the execution may be necessary.

The presented planners are focused on the efficiency and not in the optimality. Optimality is therefore an important topic. The main disadvantage of the state of the art planning algorithms focused on optimality is the efficiency. That group of planners provides better qualitative paths compared to efficient algorithms like the CellBiRRT, but it requires additional time to accomplish it. The CellBiRRT*, presented in this work, can compute low cost paths (paths with small length) while the computation time is deliverable. The CellBiRRT* uses the characteristics of the RRT*, the CellBiRRT and additional heuristics in order to create configurations that have good probability to contribute to lower cost path. Configurations that their cost is bigger than the path's length are set as inactive and they do not contribute further. Since CellBiRRT* requires more time compared with the CellBiRRT, an On-line computation of a path is developed. The path is being re-calculated on the fly e.g. while the robot arm is moving even if changes on the environment happens. The pre-calculated path is sampled and the CellBiRRT* attempts to recalculate the path from these samples. That executed path may have now better quality.

All the presented algorithms are evaluated and compared with the state of the art motion planning algorithms (IkBiRRT, CBiRRT, RRT-JT, BiRRT*, Cartesian Cell Decomposition planner). Test are done in very cluttered as well as easier environments. The proposed approaches manage to deliver very good results compared to the state of the art planning algorithms showing that they are competitive or even faster methods.

The described planning algorithms are organized into a library, called "open motion planning library for manipulators"(OMPLFM). The main aim of the library is the generality e.g. the ability to be used by other institution. The library is built with interfaces, giving the flexibility to the user either to follow the present or to create his own implementation.

Future perspective of this work is to apply additional constraints regarding the safety. For instance the robot should not not move straight towards the user, at least when the target location is far from him. Moreover adapting dynamically the velocities and accelerations of the robot arm in respect to safety could be a potential work.

An enhancement of the proposed idea of cells is to use boxes with unequal width, height and depth. In this thesis, the dimensions are equal. If the dimensions are not equal, it may improve the performance when additional constraints in the end-effector exist.

References

[ACR+02] E. U. Acar, H. Choset, A. A. Rizzi, P. Atkar, and D. Hull, *Morse decompositions for coverage tasks*, International Journal of Robotics Research **21** (2002), 331–344. 30

[Arv92a] James Arvo, *Fast random rotation matrices*, 1992. 16

[Arv92b] James Arvo, *Graphics gems iii*, Academic Press Professional, Inc., San Diego, CA, USA, 1992, pp. 117–120. 79

[AS11] Baris Akgun and Mike Stilman, *Sampling heuristics for optimal motion planning in high dimensions*, Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on, sept. 2011, pp. 2640 –2645. 100, 102, 107, 108

[Aur91] Franz Aurenhammer, *Voronoi diagrams a survey of a fundamental geometric data structure*, ACM Comput. Surv. **23** (1991), no. 3, 345–405. 29, 47

[BCC+04] Z. Bien, M.-J. Chung, P.-H. Chang, D.-S. Kwon, D.-J. Kim, J.-S. Han, J.-H. Kim, D.-H. Kim, H.-S. Park, S.-H. Kang andK. Lee, and S.-C. Lim, *Integration of a rehabilitation robotic system (kares ii) with human-friendlyman-machine interaction units*, Autonomous Robots **16** (Nov. 2004), 165–191. 10

[BCL+03] K.E. Bekris, B.Y. Chen, A.M. Ladd, E. Plaku, and L.E. Kavraki, *Multiple query probabilistic roadmap planning using single query planning primitives*, Intelligent Robots and Systems, 2003. (IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on, vol. 1, oct. 2003, pp. 656 – 661 vol.1. 30

[BK02] O. Brock and O. Khatib, *Elastic strips: A framework for motion generation in human environments*, International Journal of Robotics research, vol. 10, 202, pp. 1031–1052. 99

[BKDA06] D. Bertram, J. Kuffner, R. Dillmann, and T. Asfour, *An integrated approach to inverse kinematics and path planning for redundant manipulators*, Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on, may 2006, pp. 1874 –1879. 61, 115

[BLL91] J. Barraquand, B. Langlois, and J.-C. Latombe, *Numerical potential field techniques for robot path planning*, Advanced Robotics, 1991. 'Robots in Unstructured Environments', 91 ICAR., Fifth International Conference on, june 1991, pp. 1012 –1017 vol.2. 28

REFERENCES

[Bra06a] David Brandt, *Comparison of a and rrt-connect motion planning techniques for self-reconfiguration planning*, Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on, oct. 2006, pp. 892 –897. 45

[Bra06b] David Brandt, *Comparison of a* and rrt-connect motion planning techniques for self-reconfiguration planning.*, IROS, IEEE, 2006, pp. 892–897. 87

[BS10] Dmitry Berenson and Siddhartha Srinivasa, *Probabilistically complete planning with end-effector pose constraints*, IEEE International Conference on Robotics and Automation (ICRA '10), May 2010. 45

[BSF+09] Dmitry Berenson, Siddhartha S. Srinivasa, Dave Ferguson, Alvaro Collet, and James J. Kuffner, *Manipulation planning with workspace goal regions*, Robotics and Automation, 2009. ICRA '09. IEEE International Conference on, may 2009, pp. 618 –624. 45, 87, 115

[BSFK09] Dmitry Berenson, Siddhartha S. Srinivasa, Dave Ferguson, and James J. Kuffner, *Manipulation planning on constraint manifolds*, Robotics and Automation, 2009. ICRA '09. IEEE International Conference on, may 2009, pp. 625 –632. 45, 87

[BT97] S.A. Bazaz and B. Tondu, *Online computing of a robotic manipulator joint trajectory with velocity and acceleration constraints*, Assembly and Task Planning, 1997. ISATP 97., 1997 IEEE International Symposium on, aug 1997, pp. 1 –6. 157

[Cam90] S. Cameron, *Collision detection by four-dimensional intersection testing*, Robotics and Automation, IEEE Transactions on **6** (1990), no. 3, 291 –302. 34

[Cam97] ———, *Enhancing gjk: computing minimum and penetration distances between convex polyhedra*, Robotics and Automation, 1997. Proceedings., 1997 IEEE International Conference on, vol. 4, apr 1997, pp. 3112 –3117 vol.4. 20

[Can86] John Canny, *Collision detection for moving polyhedra*, Pattern Analysis and Machine Intelligence, IEEE Transactions on **PAMI-8** (1986), no. 2, 200 –209. 34

[CD95] Tan Fung Chan and R.V. Dubey, *A weighted least-norm solution based scheme for avoiding joint limits for redundant joint manipulators*, Robotics and Automation, IEEE Transactions on **11** (1995), no. 2, 286 –292. 75, 76

[CGCG10] S. Cousins, B. Gerkey, K. Conley, and W. Garage, *Sharing software with ros [ros topics]*, IEEE Robotics and Automation Magazin (Jan 2010), 12–14. 9

[Cha96] H. Chang, *A new technique to handle local minimum for imperfect potential field based motion planning*, Robotics and Automation, 1996. Proceedings., 1996 IEEE International Conference on, vol. 1, apr 1996, pp. 108 –112 vol.1. 28

[CLH+05] H. Choset, K. M. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L. E. Kavraki, and S. Thrun, *Principles of robot motion: Theory, algorithms, and implementations*, MIT Press, Cambridge, MA, 2005. 18, 27

[CO90] Stephen Cameron and B Of, *Efficient intersection tests for objects defined constructively*, Int. J. Robotics Res **8** (1990), 3–25. 35

[Cra05] J.J Craig, *Introduction to robotics, mechanics and control*, Pearson Prentice Hall, Upper Saddle River,NJ, 2005. 9

[Die10] Reinhard Diestel, *Graph theory*, 4th ed., Graduate Texts in Mathematics, Volume 173, Springer-Verlag, Heidelberg, July 2010. 25

[DKL98] Erik B. Dam, Martin Koch, and Martin Lillholm, *Quaternions, interpolation and animation*, Tech. report, 1998. 16

[dLR08] H. F. Machiel Van der Loos and David J. Reinkensmeyer, *Rehabilitation and health care robotics.*, Springer Handbook of Robotics (Bruno Siciliano and Oussama Khatib, eds.), Springer, 2008, pp. 1223–1251. 9

[EB97] E.Rimon and S. Boyd, *Obstacle collision detection using best ellipsoid fit*, Journal of Intelligent and Robotic Systems (1997). 35

[EL01] Stephen A. Ehmann and Ming C. Lin, *Accurate and fast proximity queries between polyhedra using surface decomposition*, Computer Graphics Forum (Proc. of Eurographics), 2001. 20, 43

[FAEG12] Christos Fragkopoulos, Khizar Abbas, Ahmed Eldeep, and Axel Graeser, *Comparison of sampling based motion planning algorithms specialized for robot manipulators*, Robotics; Proceedings of ROBOTIK 2012; 7th German Conference on (2012), 1 –6. 95

[FG10a] C. Fragkopoulos and A. Gräser, *Extended rrt algorithm with dynamic n-dimensional cuboid domains*, Optimization of Electrical and Electronic Equipment (OPTIM), 2010 12th International Conference on, may 2010, pp. 851 –857. 47

[FG10b] Christos Fragkopoulos and Axel Graeser, *A rrt based path planning algorithm for rehabilitation robots*, Robotics (ISR), 2010 41st International Symposium on and 2010 6th German Conference on Robotics (ROBOTIK) (2010), 1 –8. 59

[FG11a] C. Fragkopoulos and A. Graser, *Dynamic efficient collision checking method of robot arm paths in configuration space*, Advanced Intelligent Mechatronics (AIM), 2011 IEEE/ASME International Conference on, july 2011, pp. 784 –789. 34

[FG11b] C. Fragkopoulos and A Gräser, *Sampling based path planning for high dof manipulators without goal configuration*, IFAC World Congress, 2011, pp. 11568–11573. 75

[FH93] A. Foisy and V. Hayward, *A safe swept volume method for collision detection*, The Sixth International Symposium of Robotics Research, 1993, pp. 61–68. 34

[FIG05] J. Feuser, O. Ivlev, and A. Gräser, *Collision prevention for rehabilitation robots with mapped virtual reality in rehabilitation robotics*, International Conference of rehabilitation robotics, 2005, pp. 461–464. 20

REFERENCES

[FKS06] D. Ferguson, N. Kalra, and A. T. Stentz, *Replanning with rrts*, IEEE Int Conf on Robotics and Automation, 2006. 99

[FS06] David Ferguson and Anthony (Tony) Stentz, *Anytime rrts*, Proceedings of the 2006 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS '06), October 2006, pp. 5369 – 5375. 99

[FT10] D. Flavigne and M. Ta'i'andx, *Improving motion planning in weakly connected configuration spaces*, Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on, oct. 2010, pp. 5900 –5905. 87

[GJK88] E.G. Gilbert, D.W. Johnson, and S.S. Keerthi, *A fast procedure for computing the distance between complex objects in three-dimensional space*, Robotics and Automation, IEEE Journal of **4** (1988), no. 2, 193 –203. 20

[GLFG11] Sorin Grigorescu, Thorsten Lüth, Christos Fragkopoulos, and Axel Gräser, *A bci-controlled robotic assistant for quadriplegic people in domestic and professional life*, Robotica **30** (2011), 419–431. 9

[GLM96] S. Gottschalk, M.C. Lin, and D. Manocha, *Obb-tree: A hierarchical structure for rapid interference detection*, ACM Siggraph, 1996. 20, 35

[GO07] Roland Geraerts and Mark H. Overmars, *Creating high-quality paths for motion planning*, Int. J. Rob. Res. **26** (2007), no. 8, 845–863. viii, xi, xiii, 52, 63, 66, 85, 97, 98

[HFHG12] S. Heyer, C. Fragkopoulos, T. Heyer, and A. Graser, *Reliable hand camera based book detection and manipulation in library scenario*, Optimization of Electrical and Electronic Equipment (OPTIM), 2012 13th International Conference on, may 2012, pp. 1486 –1492. x, 128

[HKcLR00] David Hsu, Robert Kindel, Jean claude Latombe, and Stephen Rock, *Randomized kinodynamic motion planning with moving obstacles*, 2000. 99

[HLK06] David Hsu, Jean-Claude Latombe, and Hanna Kurniawati, *On the probabilistic foundations of probabilistic roadmap planning*, The International Journal of Robotics Research **25** (2006), no. 7, 627 – 643. 29

[HMP00] H. Hirukawa, B. Mourrain, and Y. Papegay, *A symbolic-numeric silhouette algorithm*, Intelligent Robots and Systems, 2000. (IROS 2000). Proceedings. 2000 IEEE/RSJ International Conference on, vol. 3, 2000, pp. 2358 –2365 vol.3. 29

[HNR68] P.E. Hart, N.J. Nilsson, and B. Raphael, *A formal basis for the heuristic determination of minimum cost paths*, Systems Science and Cybernetics, IEEE Transactions on **4** (1968), no. 2, 100 –107. 29

[HNTH10] K. Hauser and V. Ng-Thow-Hing, *Fast smoothing of manipulator trajectories using optimal bounded-acceleration shortcuts*, Robotics and Automation (ICRA), 2010 IEEE International Conference on, may 2010, pp. 2493 –2498. 98, 130

[IATa] IAT, *Adl scenario - multimedia*, http://www.iat.uni-bremen.de/sixcms/detail.php?id=589. 126

[IATb] ———, *Friend system*, www.friend4you.eu. 124

[IATc] ———, *Library scenario - multimedia*, http://www.iat.uni-bremen.de/sixcms/detail.php?id=1291. 130

[IG97] O. Ivlev and A. Graeser, *An analytical method for the inverse kinematics of redundant robots*, Third ECPD International Conference on advanced Robots, Intelligent Automation and Active Systems, 1997, pp. 416–421. 22, 73

[IG98] O. Ivlev and A. Gräser, *Resolving redundancy of series kinematic chains through imaginary links*, In Computational Engineering in Systems Applications CESA'98 IMACS Multiconference, Tunisia, 1998. 22, 73

[IG00] ———, *The optimized kinematic configuration control algorithm for redundant robots*, 16th IMACS World Congress on Scientific Computation, Applied Mathematics and Simulation, 2000. 22, 73

[JCS08] L. Jaillet, J. Cortes, and T. Simeon, *Transition-based rrt for path planning in continuous cost spaces*, Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on, sept. 2008, pp. 2145 –2150. 45

[JCS10] L. Jaillet, J. Corteands, and T. Simeandon, *Sampling-based path planning on configuration-space costmaps*, Robotics, IEEE Transactions on **26** (2010), no. 4, 635 –646. 46

[JTT98] P. Jimnez, F. Thomas, and C. Torras, *Collision detection algorithms for motion planning*, LECTURE NOTES IN CONTROL AND INFORMATION SCIENCES, 229, Springer, 1998, pp. 305–343. 35

[JYLVS05] L. Jaillet, A. Yershova, S.M. La Valle, and T. Simeon, *Adaptive tuning of the sampling domain for dynamic-domain rrts*, Intelligent Robots and Systems, 2005. (IROS 2005). 2005 IEEE/RSJ International Conference on, aug. 2005, pp. 2851 – 2856. 45, 47

[KCT+11] Mrinal Kalakrishnan, Sachin Chitta, Evangelos Theodorou, Peter Pastor, and Stefan Schaal, *STOMP: Stochastic Trajectory Optimization for Motion Planning*, International Conference on Robotics and Automation, 2011. 32

[KF10] Sertac Karaman and Emilio Frazzoli, *Incremental sampling-based algorithms for optimal motion planning*, CoRR **abs/1005.0416** (2010). 99

[KF11] S. Karaman and E. Frazzoli, *Sampling-based algorithms for optimal motion planning*, International Journal of Robotics Research, 2011, pp. 846 – 894. 101, 102, 104

REFERENCES

[Kha85] O. Khatib, *Real-time obstacle avoidance for manipulators and mobile robots*, Robotics and Automation. Proceedings. 1985 IEEE International Conference on, vol. 2, mar 1985, pp. 500 – 505. 28

[KHM+98] J.T. Klosowski, M. Held, J.S.B. Mitchell, H. Sowizral, and K. Zikan, *Efficient collision detection using bounding volume hierarchies of k-dops*, Visualization and Computer Graphics, IEEE Transactions on **4** (1998), no. 1, 21 –36. 35

[Krö10] T. Kröger, *On-line trajectory generation in robotic systems*, Springer Tracts in Advanced Robotics, vol. 58, Springer, Berlin, Heidelberg, Germany, jan 2010. 130

[KSLO96] L.E. Kavraki, P. Svestka, J.-C. Latombe, and M.H. Overmars, *Probabilistic roadmaps for path planning in high-dimensional configuration spaces*, Robotics and Automation, IEEE Transactions on **12** (1996), no. 4, 566 –580. 29

[KSV10] U. Kunz, T.and Reiser, M. Stilman, and A. Verl, *Real-time path planning for a robot arm in changing environments*, IEEE/RSJ Conference on Intelligent Robots and Systems, 2010, pp. 5906–5911. 99

[Kuf04] James J. Kuffner, *Effective sampling and distance metrics for 3d rigid body path planning*, ICRA, 2004, pp. 3993–3998. 55, 79, 149

[KWP+11] S. Karaman, M.R. Walter, A. Perez, E. Frazzoli, and S. Teller, *Anytime motion planning using the rrt**, Robotics and Automation (ICRA), 2011 IEEE International Conference on, may 2011, pp. 1478 –1483. 99

[Lat91] Jean-Claude Latombe, *Robot motion planning*, Kluwer Academic Publishers, 1991. 28, 29, 30

[Lav98] Steven M. Lavalle, *Rapidly-exploring random trees: A new tool for path planning*, Tech. report, 1998. 31, 45

[LaV06] S. M. LaValle, *Planning algorithms*, Cambridge University Press, Cambridge, U.K., 2006, Available at http://planning.cs.uiuc.edu/. 18, 19, 45, 104

[LBL04] Steven M. LaValle, Michael S. Branicky, and Stephen R. Lindemann, *On the relationship between classical grid search and probabilistic roadmaps*, The International Journal of Robotics Research **23** (2004), no. 7-8, 673–692. 30

[LFG+05] Maxim Likhachev, David Ferguson, Geoffrey Gordon, Anthony (Tony) Stentz, and Sebastian Thrun, *Anytime dynamic a*: An anytime, replanning algorithm*, Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS), June 2005. 99

[LGLM00] Eric Larsen, Stefan Gottschalk, Ming C. Lin, and Dinesh Manocha, *Fast distance queries with rectangular swept sphere volumes*, Proc. of IEEE Int. Conference on Robotics and Automation, 2000, pp. 3719–3726. 43

[LH00] Peter Leven and Seth Hutchinson, *Toward real-time path planning in changing environments*, 2000. 99

[LH02] P. Leven and S. Hutchinson, *A framework for real-time path planning in changing environment*, The international Journal of Robotics and Research, vol. 21, 2002, pp. 999–1030. 99

[LJ99] Steven M. LaValle and James J. Kuffner Jr., *Randomized kinodynamic planning*, The International Journal of Robotics Research **20** (1999), no. 5, 378–400. 31, 99

[LL04] S.R. Lindemann and S.M. LaValle, *Incrementally reducing dispersion by increasing voronoi bias in rrts*, Robotics and Automation, 2004. Proceedings. ICRA '04. 2004 IEEE International Conference on, vol. 4, 26-may 1, 2004, pp. 3251 – 3257 Vol.4. 45, 47

[LP81] Tomas Lozano-Perez, *Automatic planning of manipulator transfer movements*, Systems, Man and Cybernetics, IEEE Transactions on **11** (1981), no. 10, 681 –698. 16

[LPW79] Tomás Lozano-Pérez and Michael A. Wesley, *An algorithm for planning collision-free paths among polyhedral obstacles*, Commun. ACM **22** (1979), no. 10, 560–570. 29

[MPFG06] C. Martens, O. Prenzel, J. Feuser, and A. Gräser, *Massive: Multi-layer architecture for semi-autonomous service-robots with verified task execution*, 10th Int. Conf. on Optimization of Electrical and Electronic Equipments OPTIM, 2006, pp. 107–112. 10

[MRL+01] C. Martens, N. Ruchel, O. Lang, O. Ivlev, and A. Gräser, *A friend for assisting handicapped people*, IEEE Robotics and Automation Magazines (Mar 2001), 57–65. 10

[MWS07] S.R. Martin, S.E. Wright, and J.W. Sheppard, *Offline and online evolutionary bi-directional rrt algorithms for efficient re-planning in dynamic environments*, Automation Science and Engineering, 2007. CASE 2007. IEEE International Conference on, sept. 2007, pp. 1131 –1136. 45, 99

[NSL99] C. Nissoux, T. Simeon, and J.-P. Laumond, *Visibility based probabilistic roadmaps*, Intelligent Robots and Systems, 1999. IROS '99. Proceedings. 1999 IEEE/RSJ International Conference on, vol. 3, 1999, pp. 1316 –1321 vol.3. 30

[Ojd09a] D. Ojdanic, *Using cartesian space for manipulation motion planning- application in service robots*, Shaker Verlag -Germany, 2009. 31

[Ojd09b] Darko Ojdanic, *Using cartesian space for manipulator motion planning - application in service robotics phd*, Shalker Verlag, 2009. 95

[OLJ98] L. Overgaard, R. Larsen, and N. Jacobsen, *Industrial applications of the amrose motion planner*, Practical Motion Planning in Robotics: Current Approaches and Future Directions (1998). 31

[OMP] *Open motion planning library*, `http://ompl.kvrakilab.org`. 87

[OOV02] G. Oriolo, M. Ottavi, and M. Vendittelli, *Probabilistic motion planning for redundant robots along given end-effector paths*, Intelligent Robots and Systems, 2002. IEEE/RSJ International Conference on, vol. 2, 2002, pp. 1657 – 1662 vol.2. 45

[OPE] *Open rave*, `http://openrave.programmingvision.com/en/`. 87

[OS84] David E. Orin and William W. Schrader, *Efficient computation of the jacobian for robot manipulators*, The International Journal of Robotics Research **3** (1984), no. 4, 66–75. 24

[Pau81] R.P Paul, *Robot manipulators: Mathematics, programming and control*, MIT Press, Cambridge, MA, 1981. 9, 14

[PBK07] E. Plaku, Kostas E. Bekris, and L. E. Kavraki, *Oops for motion planning: An online open-source programming system*, IEEE International Conference on Robotics and Automation (ICRA) (Rome, Italy), 2007, pp. 3711–3716. 87

[PF05] S. Petti and T. Fraichard, *Safe motion planning in dynamic environments*, Intelligent Robots and Systems, 2005. (IROS 2005). 2005 IEEE/RSJ International Conference on, 2005, pp. 2210 – 2215. 99

[PJS06] R. Philippsen, B. Jensen, and R. Siegwart, *Toward online probabilistic path replanning in dynamic environments*, IEEE ICRA, Beijing 2006, pp. 2876–2881. 99

[PKS+11] Alejandro Perez, Sertac Karaman, Alexander Shkolnik, Emilio Frazzoli, Seth Teller, and Matthew R. Walter, *Asymptotically-optimal path planning for manipulation using incremental sampling-based algorithms*, Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on, sept. 2011, pp. 4307 –4313. 99, 102

[PMC+07] O. Prenzel, C. Martens, M. Cyriacks, C. Wang, and A. Gräser, *System controlled user interaction within the service robotic control architecture massive*, Robotica (Special Issue) **25** (Mar. 2007). 10

[PS85] F. Preparata and M. I. Shamos, *Computational geometry: An introduction*, pp. 198–257, Springer-Verlag, 1985. 30

[QK93] S. Quinlan and O. Khatib, *Elastic bands: connecting path planning and control*, Robotics and Automation, 1993. Proceedings., 1993 IEEE International Conference on, may 1993, pp. 802 –807 vol.2. 37, 99

[Qui94] S. Quilan, *Real-time modification of collision-free paths*, Stanford University, 1994. 37

[RCF+09] U. Reiser, C. Connette, J. Fischer, J. Kubacki, A. Bubeck, F. Weisshardt, T. Jacobs, C. Parlitz, M. Haegele, and A. Verl, *Care-o-bot 3 creating a product vision for service robot applications by integrating design and technology*, International Conference on Intelligent Robots and Systems, Oct 2009. 9

[RZBS09] N. Ratliff, M. Zucker, J.A. Bagnell, and S. Srinivasa, *Chomp: Gradient optimization techniques for efficient motion planning*, Robotics and Automation, 2009. ICRA '09. IEEE International Conference on, May 2009, pp. 489–494. 32

[SHJ+05] Zheng Sun, D. Hsu, Tingting Jiang, H. Kurniawati, and J.H. Reif, *Narrow passage sampling for probabilistic roadmap planning*, Robotics, IEEE Transactions on **21** (2005), no. 6, 1105 – 1115. 30

[Sho85] Ken Shoemake, *Animating rotation with quaternion curves*, SIGGRAPH Comput. Graph. **19** (1985), no. 3, 245–254. 16, 149, 151

[Sho92] ———, *Graphics gems iii*, Academic Press Professional, Inc., San Diego, CA, USA, 1992, pp. 124–132. 79

[SM93] Anthony Stentz and I. Carnegle Mellon, *Optimal and efficient path planning for unknown and dynamic environments*, International Journal of Robotics and Automation **10** (1993), 89–100. 29, 99

[SScL05] Fabian Schwarzer, Mitul Saha, and Jean claude Latombe, *Adaptive dynamic collision checking for single and multiple articulated robots in complex environments*, IEEE Tr. on Robotics **21** (2005), 338–353. 37, 159

[SSL05] F. Schwarzer, M. Saha, and J.-C. Latombe, *Adaptive dynamic collision checking for single and multiple articulated robots in complex environments*, Robotics, IEEE Transactions on **21** (2005), no. 3, 338 – 353. 34

[SSVG09] B. Siciliano, L. Sciavicco, L. Villani, and O. Giuseppe, *Robotics; modelling, planning and control*, Springer, 2009. 9, 23, 24

[Ste95] Anthony Stentz, *The focussed d* algorithm for real-time replanning*, In Proceedings of the International Joint Conference on Artificial Intelligence, 1995, pp. 1652–1659. 99

[SZ11] C. Scheurer and U.E. Zimmermann, *Path planning method for palletizing tasks using workspace cell decomposition*, Robotics and Automation (ICRA), 2011 IEEE International Conference on, may 2011, pp. 1 –4. 31

[TN87] William C. Thibault and Bruce F. Naylor, *Set operations on polyhedra using binary space partitioning trees*, SIGGRAPH Comput. Graph. **21** (1987), no. 4, 153–162. 35

[Top01] M. Topping, *Handy 1, a robotic-arm to aid independence for severely disabled people*, Integration of Assistive Technology in the Information Age, IOS Press, Netherland, 2001, pp. 142–147. 10

[VBA+09] N. Vahrenkamp, D. Berenson, T. Asfour, J. Kuffner, and R. Dillmann, *Humanoid motion planning for dual-arm manipulation and re-grasping tasks*, Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on, oct. 2009, pp. 2464 –2470. 75, 115

REFERENCES

[vdBFK06] Jur van den Berg, David Ferguson, and James Kuffner, *Anytime path planning and replanning in dynamic environments*, Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), May 2006, pp. 2366 – 2371. 99

[VWFS07] M. Vande Weghe, D. Ferguson, and S.S. Srinivasa, *Randomized path planning for redundant manipulators without inverse kinematics*, Humanoid Robots, 2007 7th IEEE-RAS International Conference on, 29 2007-dec. 1 2007, pp. 477 –482. 75

[WAS99] S.A. Wilmarth, N.M. Amato, and P.F. Stiller, *Maprm: a probabilistic roadmap planner with sampling on the medial axis of the free space*, Robotics and Automation, 1999. Proceedings. 1999 IEEE International Conference on, vol. 2, 1999, pp. 1024 –1031 vol.2. 30

[WGW+00] L. De Witte, M. Goossens, R. Wessels, D. Van der Pijl, G. Gelderblom, W. Van Hoofd, D. Tilli, B. Dijcks, and K. Van Soest, *Cost-effectiveness of specialized assistive technology: the manus robot manipulator*, Annual International Society of Technology Assessment inHealth Care Meeting, 2000, p. 284. 10

[YG05] Zhenwang Yao and K. Gupta, *Path planning with general end-effector constraints: using task space to guide configuration space search*, Intelligent Robots and Systems, 2005. (IROS 2005). 2005 IEEE/RSJ International Conference on, aug. 2005, pp. 1875 – 1880. 45

[YJSL05] A. Yershova, L. Jaillet, T. Simeon, and S.M. LaValle, *Dynamic-domain rrts: Efficient exploration by controlling the sampling domain*, Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on, april 2005, pp. 3856 – 3861. 45, 47

[YYG10] E. Yoshida, K. Yokoi, and P. Gergondet, *Online replanning for reactive robot motion: Practical aspects*, Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on, oct. 2010, pp. 5927 –5933. 30, 100

[ZF95] G. Zachman and W. Felger, *The box tree: Enabling real time and exact collision detection of arbitrary polyedra*, SIVE, 1995, pp. 104–113. 20, 35

[ZKB07] Matthew Zucker, James Kuffner, and Michael Branicky, *Multipartite rrts for rapid replanning in dynamic environments*, Proc. IEEE Int. Conf. Robotics and Automation, April 2007. 99

Appendix A

Quaternion

A.1 Slerp(Spherical Linear) interpolation

Ken Shoemake in [Sho85] presented an approach for linear interpolation between two rotations. Given $\lambda \in [0,1]$ an arbitrary number, q_1 and q_2 two quaternions, the *Slerp* interpolation is defined by the following equation:

$$Slerp(q_1, q_2, \lambda) = q_1 \cdot (q_1^{-1} \cdot q_2)^{\lambda} \tag{A.1}$$

or by using 4D geometry:

$$Slerp(q_1, q_2, \lambda) = \frac{sin((1-\lambda)\cdot\theta)}{sin\theta} \cdot q_1 + \frac{sin(\lambda\theta)}{sin\theta} \cdot q_2 \tag{A.2}$$

where θ is computed by the following equation $q_1 \cdot q_2 = cos\theta$. In Slerp interpolation the existed path, which is normally a straight line, is transform to an equivalent spherical path. The image A.1 illustrate this feature of the interpolation. The benefit of this method is an exact linear transformation between two rotations.

In [Kuf04] a nice pseudo code regarding the usage of the equation A.2 is illustrated. The pseudocode is in algorithm 31. The parameter ϵ has a very small value and it is used in the case where two rotations are very close.

A.2 Distance between quaternions

The distance between two normalized quaternion express the angle between the two quaternions, and is expressed as the dot product between the quaternions.Given two quaternions $q_1(w_1, x_1, y_1, z_1)$ and $q_2(w_2, x_2, y_2, z_2)$ the distance is equal to:

$$cos\theta == q_1 \cdot q_2 = w_1 \cdot q_2 + x_1 \cdot x_2 + y_1 \cdot y_2 + z_1 \cdot z_2 \tag{A.3}$$

Algorithm 31 Slerp(q_1, q_2, λ)

q_1 and q_2 are the quaternions (w,x,y,z)
$d = q_1 \cdot q_2$ //compute the inner product of two quaternions
if (d<0) **then**
    $q_2 = -q_2$
end if
if $(||1 - d|| < \epsilon)$ **then**
    r=1-λ
    s=f
else
    a=arccos(λ)
    $\gamma = \frac{1}{sina}$
    r= $sin((1 - \lambda) \cdot a) \cdot \gamma$
    s= $sin(\lambda \cdot a) \cdot \gamma$
end if
// set the interpolation quaternion
w=$r \cdot w_1 + s \cdot w_2$
x=$r \cdot x_1 + s \cdot x_2$
y=$r \cdot y_1 + s \cdot y_2$
Q=$\frac{q}{||q||}$ { normalize the quaternion q(w,x,y,z)}
Return Q;

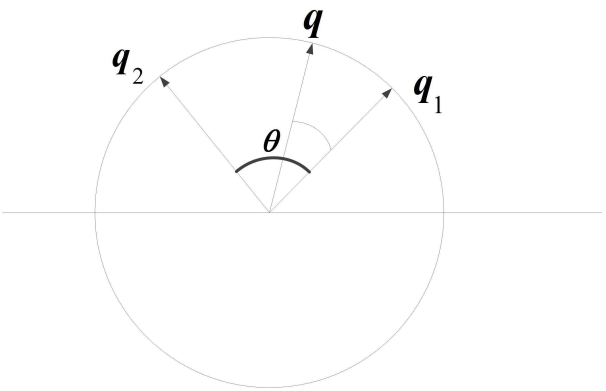


Figure A.1: Slerp interpolation between two quaternions q_1 and q_2

A.3 Quaternion to matrix

Given a quaternion q=w+x·i + y·j + z·k the equivalent 3x3 rotation matrix is given by the following equation:

$$R = \begin{bmatrix} 1\text{-}2{\cdot}y^2\text{-}2{\cdot}z^2 & 2{\cdot}x \cdot y\text{-}2{\cdot}z \cdot w & 2{\cdot}x \cdot z + 2 \cdot y \cdot w \\ 2{\cdot}x \cdot y + 2{\cdot}z \cdot w & 1\text{-}2{\cdot}x^2\text{-}2 \cdot z^2 & 2{\cdot}y \cdot z\text{-}2{\cdot}y \cdot z\text{-}2{\cdot}x \cdot w \\ 2{\cdot}x \cdot z - 2{\cdot}y \cdot w & 2{\cdot}y \cdot z + 2{\cdot}x \cdot w & 1\text{-}2{\cdot}x^2\text{-}2{\cdot}y^2 \end{bmatrix} \quad \text{(A.4)}$$

A.4 Matrix to quaternion

In [Sho85] an approach of converting a matrix to quaternion is presented. This conversion is unfortunately not trivial but it can be done nowadays easily. Given a matrix M_{3x3} the components of the quaternion q=w+x·i + y·j + z·k can be computed by the algorithm 32.

Algorithm 32 Matrix to Quaternion

$w^2 = 1/4 * (1 + M_{11} + M_{22} + M_{33})$
if $(w^2 > \epsilon)$ **then**
  $w = \sqrt{w^2}$
  $x = (M_{23} - M_{32})/4w$
  $y = (M_{31} - M_{13})/4w$
  $z = (M_{12} - M_{21})/4w$
else
  w=0
  $x^2 = -1/2 * (M_{22} + M_{33})$
  if $(x^2 > \epsilon)$ **then**
    $x = \sqrt{x}$
    $y = M_{12}/2x$
    $z = M_{13}/2x$
  else
    x = 0
    $y^2 = 1/2 * (1 - M_{33})$
    if $(y^2 > \epsilon)$ **then**
      $y = \sqrt{y}$
      $z = M_{23}/2y$
    else
      y=0
      z=0
    end if
  end if
end if

Appendix B

Constraints with quaternions

Given a normalized quaternion q=(w,x,y,z), the constraints in equation 5.11 can be written as follows:

$$CT_T = \begin{pmatrix} X_{min} & X_{max} \\ Y_{min} & Y_{max} \\ Z_{min} & Z_{max} \\ q_{min} & q_{max} \end{pmatrix} \tag{B.1}$$

where the q_{min} and q_{max} correspond to minimum and maximum rotations of the end effector represented by quaternions.

The transformation T from rotation matrix to quaternion is defined as:

$$T(R \to q) : R(\alpha, \beta, \gamma) \xrightarrow{T} q(w, x, y, z) \tag{B.2}$$

where $R(\alpha, \beta, \gamma)$ is rotation matrix given by the equation 2.18. Equivalent is defined the $T(q \to R)$ the transformation from quaternion to rotation 4x4 matrix. Every frame, and equivalently rotations, can be expressed with the help of quaternions. The advantage is less intermediate calculations(for instance Slerp interpolation is fast). By applying only the necessary transformations(equation B.2), the constraint limits and the rest of the computations can be expressed by quaternions.

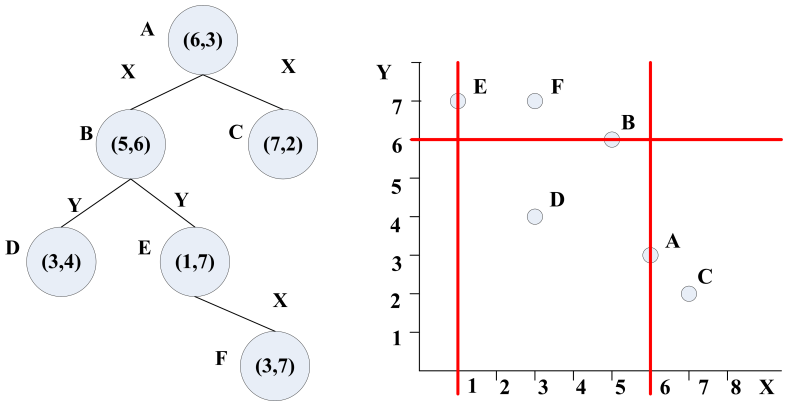
Appendix C

KD-Trees

In this appendix, **KDtrees** are presented briefly. In literature there is an huge number of references as well as modifications of the KDTrees. KDTrees is a specific structure that decomposes the space with a specific manner. This specific decomposition assists the search of the nearest neighbors. The *KDTrees* are binary trees and they have the following functionalities:

- Insert (complexity from O(log n) till O(n))
- Search (complexity from O(log n) till O(n))
- Delete (complexity from O(log n) till O(n))

The insertion and the search are going to be explained through examples. The delete function in a kd tree is more complicated since needs balancing of the tree. The normal procedure in that case is to remove the node and to reconstruct the tree again from this node.



(a) Points and the dimension splitting in KDTree

(b) Points represented in 2D with the splitting dimension

Figure C.1: KDTree example

Consider the points A(6,3), B(5,6), C(7,2), D(3,4), E(1,7) and F(3,7) as the figure C.1(a) shows. The A s the first point. The initial splitting dimension is X. Based on X the B and C points are inserted regarding if the point lies on the left or on the right part for this dimension (e.g. to be less or bigger than the A_x value). After splitting by X the next dimension is Y and the procedure is the same. The third splitting is done again by the X axes and the procedure continues iteratively. The figure C.1(b) shows the cutting dimensions and the points in 2D.

The searching works almost with the same way. Consider that there is a query point Q in figure C.2(a). Let consider first the circle with radius equals to $|QA|$. It is clear that the radius does not contain any point (for example B lies outside the circle) from the space being left to the A. This space is completely discarded and consequently all the points in the tree belonging on the left side of A are discarded. Now the point A and C are taken into account. Finally the point C is closer than A (figure C.2(b)). This example shows that the KDTree can be very efficient structure for nearest neighbour queries. From seven points the algorithm controlled only two and that improves the performance. If the tree contains many points this approach is going to improve the query of the nearest neighbor. As the dimensionality of the space increases the improvement is reducing and after almost 20 degrees of freedom the KDTrees do not provide a significant improvement in the performance compared to the brute force search.

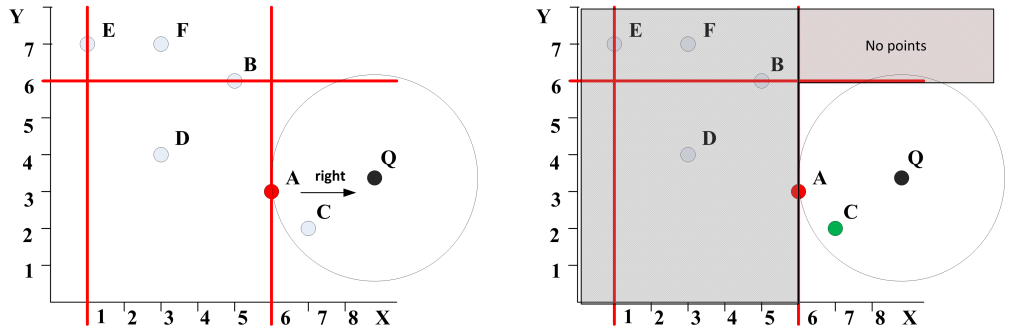


Figure C.2: KDTree Search example

Appendix D

Trajectory generation

The trajectory profile, the calculation of velocities in relation with the time, is crucial for the robot arm motion. In this work a third order polynomial is used since the constraints are: Initial and final state are given and initial and final velocities are zero. The motion is done point-to-point and therefore the robot velocities are zero at the begging and at the end of each segment. Therefore the following equation describe the position of a joint:

$$\begin{aligned} q &= a + b \cdot t + c \cdot t^2 + d \cdot t^3 \\ \dot{q} &= b + 2 \cdot c \cdot t + 3 \cdot d \cdot t^2 \\ \ddot{q} &= +2 \cdot c + 6 \cdot d \cdot t \end{aligned} \tag{D.1}$$

Following this profile there are some important issues that should be taken into account: the robot should avoid any backward motion, and therefore the velocity should not change its sign. The robot should not exceed a given maximum velocity $|\dot{q}|_{max}$ and a given maximum acceleration $|\ddot{q}|_{max}$. Solving the equation D.1 the parameters are calculated as follows (more information is in [BT97]):

$$\begin{aligned} \mathrm{a} &= q_o \\ \mathrm{b} &= \dot{q_1} \\ \mathrm{c} &= \tfrac{3(q_1 - q_o)}{\Delta t^2} - \tfrac{(2 \cdot \dot{q}_o + \dot{q}_1)}{\Delta t} \\ \mathrm{d} &= \tfrac{2(q_o - q_1)}{\Delta t^3} + \tfrac{\dot{q}_o + \dot{q}_1}{\Delta t^2} \end{aligned} \tag{D.2}$$

where the notations $< o >$ and $< 1 >$ refer to start and goal positions respectively and Δt the duration of the motion.

Solving the equations D.1and D.2, and considering the constraints, the following minimum travel time are calculated:

- With Velocity Constraints:

$$t_v = \frac{1.5 \cdot |q_1 - q_o|}{|\dot{q}_{max}|} \tag{D.3}$$

- With acceleration constraints:

$$t_a = \sqrt{\frac{6 \cdot |q_1 - q_o|}{\ddot{q}_{max}}} \tag{D.4}$$

The duration Δt is calculated as the maximum between the t_v and the t_a. Then the a, b, c and d are calculated respectively and the form of the third order polynomial is completely calculated. The velocities are given to the robot arm and the robot starts moving following the polynomial.

Appendix E

Calculating the maximum displacement of a link $\lambda_{k,max}$

This appendix repeats shortly the result of the paper [SScL05]. Let $\mathbf{p}$ is the vector of a point of the rigid body of the link A_i. The straight path segment in the c-space between q_a and q_b can be written as $q(t) = (1-t) \cdot q_a + t \cdot q_b$, $t \in [0,1]$.

From the Jacobian definition it is known that the velocity of the vector $\mathbf{p}$ can be computed as:

$$\dot{\mathbf{p}}(t) = \sum_{k=1}^{i} \frac{\partial \mathbf{p}}{\partial t} \dot{q}_k(t) \tag{E.1}$$

The equation E.1 can be bounded as follows:

$$||\dot{\mathbf{p}}(t)|| \leq \sum_{k=1}^{i} ||\frac{\partial \mathbf{p}}{\partial t}|| \cdot |\dot{q}_k(t)| \tag{E.2}$$

where the $|\dot{q}_k(t)|$ is equal to $|q_{b,k} - q_{a,k}|$. According to the definition of the $Rmax_k^i$ given in the chapter 4, we have:

$$||\frac{\partial \mathbf{p}}{\partial t}|| \leq Rmax_k^i \tag{E.3}$$

The length $L_{\mathbf{p}}$ of the curved traced by the $\mathbf{p}$ when t varies from 0 to 1 is equal to:

$$L_{\mathbf{p}} = \int_0^1 ||\dot{\mathbf{p}}(t)||dt \leq \int_0^1 \sum_{k=1}^{i} ||\frac{\partial \mathbf{p}}{\partial t}|| \cdot |\dot{q}_k(t)|dt \tag{E.4}$$

The latter leads to the following result:

$$L_{\mathbf{p}} = \int_0^1 \sum_{k=1}^{i} ||\frac{\partial \mathbf{p}}{\partial t}|| \cdot |\dot{q}_k(t)|dt \leq \sum_{k=1}^{i} Rmax_k^i \cdot \int_0^1 |\dot{q}_k(t)|dt = \sum_{k=1}^{i} Rmax_k^i \cdot |\Delta q_k| = \lambda_{k,max} \tag{E.5}$$