Antti Valli

Sensor-based motion planning for a robotic manipulator

School of Electrical Engineering

Thesis submitted for examination for the degree of Master of Science in Technology.

Espoo August 8, 2013

Thesis supervisor:

Prof. Ville Kyrki

Thesis advisor:

D.Sc. Sami Terho

Aalto University
School of Electrical Engineering

AALTO UNIVERSITY
SCHOOL OF ELECTRICAL ENGINEERING

ABSTRACT OF THE MASTER'S THESIS

Author: Antti Valli

Title: Sensor-based motion planning for a robotic manipulator

Date: August 8, 2013 Language: English Number of pages:13+108

Department of Automation and Systems Technology

Professorship: Automation Technology Code: AS-84

Supervisor: Prof. Ville Kyrki

Advisor: D.Sc. Sami Terho

When teleoperating a manipulator, obtaining sufficient situational awareness is often difficult. The operator's task can be made easier by automatically taking care of supporting tasks, such as collision avoidance, enabling the operator to concentrate on the manipulation task. In this work planning based collision avoidance methods for a robot manipulator in a changing unstructured environment are studied. Sensor based motion planning system is developed. The system detects the environment using a three-dimensional range sensor and produces an occupancy grid map. Three motion planning algorithms based on rapidly exploring random trees (RRT) are compared based on the planning time of the algorithm, the execution time of the planned motion and the end effector movement caused by the motion. For the experiments the motion planning system was implemented using Kinova JACO robot arm as the manipulator and Microsoft Kinect as the sensor. RRT-based algorithms were found to be suitable for this kind of motion planning systems. Of the algorithms compared, RRT-Connect was the fastest, but RRT* produced the best solution paths. The selection of algorithm depends on the relative value of path quality and solution speed and is application dependent.

Keywords: Robot manipulator, Collision detection, Path planning, Sampling-based planning, Rapidly exploring random trees

AALTO-YLIOPISTO
SÄHKOTEKNIIKAN KORKEAKOULU

DIPLOMITYÖN
TIIVISTELMÄ

Tekijä: Antti Valli

Työn nimi: Robottimanipulaattorin anturipohjainen liikesuunnittelu

Päivämäärä: August 8, 2013 Kieli: Englanti Sivumäärä:13+108

Automaatio- ja systeemitekniikanlaitos

Professuuri: Automaatiotekniikka Koodi: AS-84

Valvoja: Prof. Ville Kyrki

Ohjaaja: TkT Sami Terho

Teleoperoitaessa manipulaattoria riittävän tilannetietoisuuden välittäminen operaattorille on haastavaa. Operaattorin työtä voidaan helpottaa hoitamalla automaattisesti manipulaatiotehtävää tukevia tehtäviä, kuten törmäyksen välttämistä. Tässä työssä tutkitaan suunnittelupohjaisia menetelmiä robottimanipulaattorin törmäyksen välttämiseen muuttuvassa ja etukäteen tuntemattomassa ympäristössä. Työssä toteutettu anturipohjainen liikesuunnittelujärjestelmä havainnoi ympäristöä kolmiulotteista etäisyysmittausta hyödyntäen ja muodostaa siitä varauskartan. Kolmea nopeasti tutkiviin satunnaispuihin (RRT) perustuvaa liikesuunnittelualgoritmia vertaillaan suunnitteluun kuluneen ajan, liikkeen suoritusajan ja työkalun kulkeman matkan suhteen. Kokeita varten järjestelmä toteutettiin Kinova JACO manipulaattoria ja Microsoft Kinect anturia hyödyntäen. Kokeissa havaittiin RRT algoritmien soveltuvan tämän tyyppisiin liikesuunnittelujärjestelmiin. Vertailluista algoritmeista RRT-Connect oli nopein, mutta RRT* tuotti parhaan ratkaisun. Algoritmin valinta riippuu ratkaisun laadun ja suunnitteluun käytetyn ajan välisestä arvotuksesta ja siten sovelluskohteesta.

Avainsanat: Manipulaattori, Törmäystarkastus, Liikkeensuunnittelu, Näytteistykseen perustuva liikesuunnittelu, Nopeasti tutkivat satunnaispuut

Preface

This research was conducted in Department of Automation and Systems Technology in Aalto University School of Electrical Engineering. I wish to thank Professor Ville Kyrki and my advisor D.Sc. Sami Terho for their valuable guidance regarding this work. I also wish to express my gratitude to D.Sc. Jarmo Ilonen from Lappeenranta University of Technology who provided the code that was used for calibrating the position of Kinect sensor.

The research was a part of FAMOUS project of the Energy and Life Cycle Cost Efficient Machines (EFFIMA) research program, managed by the Finnish Metals and Engineering Competence Cluster (FIMECC), and funded by the Finnish Funding Agency for Technology and Innovation (TEKES), research institutes and companies. Their support is gratefully acknowledged.

Espoo, August 8, 2013

Antti Valli

Contents

Symbols and abbreviations

Symbols

α	confidence level
α_c	skew coefficient
α_i	the angle between z-axes of successive joints $i-1$ and i
β, γ	depth parameters of the Kinect sensor
δ	offset added to d_i in robot kinematics
$\Delta_{n,m}$	the mean distance between populations n and m
θ_i	joint angle of the ith joint
θ_{JACO}	joint angles as used by the robot
θ_{model}	joint angles as defined by the kinematic model
μ_z	the mean of Fisher transform z_i
ρ_i	the correlation coefficient of population i
σ_i^2	the variance of Fisher transform z_i
a_i	length of the ith link
b	bias parameter of a linear model
$\vec{c}$	translation between camera and world coordinate system
$c_1 \dots c_4$	parameters of a plane
$\mathcal{C}_{free}$	free configuration space
$\mathcal{C}_{obs}$	configuration space obstacles
c_x	x coordinate of the principal point
c_y	y coordinate of the principal point
d_i	joint offset of the ith joint
d	disparity
$\vec{d_x}$	tangential distortion vector
e_i	the mean distance of the ith population from target
f	focal length
f_x	focal length in horizontal pixels
f_y	focal length in vertical pixels
H_0	the null hypothesis
H_1	the alternative hypothesis
i	index
k	slope parameter of a linear model
$k_{1\dots5}$	image distortion coefficients
L_1	the taxicab norm
L_2	Euclidean norm
L_∞	the maximum norm
l_{cell}	side length of a cell
l_{free}	logs-odds of cell being occupied if ray passes through it
l_{occ}	log-odds of cell being occupied if ray reflects within it
$L(n\|m_{1:t})$	log-odds of cell n being occupied given observations $m_1 \dots m_t$

m_t	measurement at time t
n	index
N	normal distribution
n_i	size of the ith sample
P	probability
q	maximum dimension of AABB of an object
$\mathbb{R}$	set of real numbers
r	distance from image center in the normalized image projection
R	rotation matrix
R^2	coefficient of determination
R_i	sum of the ranks of sample i
r_i	the sample correlation coefficient of sample i
r_l	radius of the curved links
s_{est}	worst-case estimate of end effector movement. The s-metric.
s_{kinect}	distance to a wall along its normal, measured with Kinect
s_{real}	real end effector movement caused by execution of a path
s_{ref}	distance to a wall, measured with tape measure
s_U	the standard error of U statistic
${}^{i-1}T_i$	transformation between joints $n-1$ and n
t_{est}	estimated execution time of a movement. The t-metric.
t_{obs}	measured execution time of a movement
T_x	translation in the direction of x axis, e.g. baseline of stereo systems
T_y	translation in the direction of y axis
T_z	translation in the direction of z axis
U, U'	U test statistic
U_E	the expected value of U statistic
v	a test statistic used to compare correlation coefficients
$w(x)$	the weight associated with milestone x
x, y, z	coordinates
x_d, y_d	normalized point coordinates including lens distortion
X_i	transformation matrix of the nth link transform
x_n, y_n	normalized point coordinates (pinhole projection)
x_p, y_p	pixel coordinates
x_{target}	the x coordinate of the target position
y_{target}	the y coordinate of the target position
z_i	Fisher transform of sample correlation coefficient of sample i
Z_n	transformation matrix of the nth joint transform

Abbreviations

AABB	Axis-aligned bounding box
aLRF	Actuated Laser Range Finder
API	Application Programming Interface
BSP-tree	Binary Space Partitioning tree. A spatial data structure
BT-RRT*	RRT* combined with Ball Trees
BV	Bounding volume
BVH	Bounding volume hierarchy
CAD	Computer Aided Design
CSG	Constructive Solid Geometry
DoF	Degrees of Freedom
EPA	Expanding Polytope Algorithm
EST	Expansive Space Tree motion planning algorithm
FCL	Flexible collision detection library
GJK	Gilbert-Johnson-Keerthi collision detection algorithm.
GPU	Graphics Processing Unit. A parallel processor designed for rendering 3D graphics
IR	Infra Red
k-DOP	Discrete Oriented Polytope with k faces
KPIECE	Kinodynamic Planning by Interior-Exterior Cell Exploration. A motion planning algorithm
LADAR	Laser radar
LED	Light Emitting Diode
LIDAR	Light Detection and Ranging
MPR	Minkowski Portal Refinement. A Collision detection algorithm
NDT	Normal distribution transform. A representation used in scan matching and mapping
NDT-OM	NDT combined with Occupancy Map
NURBS	Non-uniform rational basis spline. A mathematical model used to represent curved surfaces
OBB	Oriented Bounding Box
ODE	Open dynamics engine physics simulation library
OMPL	Open motion-planning library
OpenGL	Open Graphics Library. A cross-platform API for 3D and 2D rendering
OpenNI	Open Natural Interaction. Open source SDK for developing 3D sensing applications
OpenRAVE	Open Robotics Automation Virtual Environment
PC	Personal Computer
PCD	Point cloud data file format
PCL	Point cloud library
PRM	Probabilistic roadmaps motion planning algorithm

RANSAC	Random sample consensus. An iterative method of fitting a model to noisy measurements
RGB	Red-Green-Blue. Common way of representing colors with additive mixing
ROS	Robot Operating System
RRT	Rapidly exploring random trees. A technique used in motion planning.
RRT*	Asymptotically optimal variant of RRT motion planner
SBL	Single-query Bidirectional probabilistic roadmap planner with Lazy collision checking
SDK	Software Development Kit
SDL	Simple DirectMedia Layer. A cross-platform multimedia library.
SIFT	Scale-invariant feature transform. Algorithm for detecting and describing local features in images.
SO2	Special Orthogonal group of rotations in two dimensional space
SWIFT	Speedy Walking via Improved Feature Testing. A collision detection algorithm
SyCLoP	Synergistic Combination of Layers of Planning. A framework for motion planning
ToF	Time of Flight
ToI	Time of Impact. Type of collision detection query
USB	Universal Serial Bus
VTK	Visualization Toolkit. Open source library for visualization

List of Figures

List of Tables

List of Algorithms

1 Introduction

Robot manipulators have proven very successful in the industrial setting where they can perform operations with a speed and accuracy that is well beyond human capability. Industrial settings such as warehouses and factories tend to be well structured environments with predictable conditions and a limited set of objects that the robots need to be able to handle. The robots are also kept separate from humans working in the environment either physically or by some security mechanism that stops the robot if a human enters its work area. In these environments the robot's programming can more or less be fixed with some limited ability to adjust to small variations such as differing placement of objects on a conveyor.

The attention of the research has increasingly shifted towards more difficult applications outside the factory where the robot may encounter a dynamic changing environment full of unexpected objects and varying conditions. In what is perhaps the most difficult case the robot needs to be able to work with humans. At the minimum, the robot should not cause humans danger, should they wander on its working area.

The problems posed by unstructured environments can be difficult. Instead of trying to make the machines fully autonomous, a combination of autonomous action and teleoperation by a human operator may be pursued. In such cases the robot performs autonomously actions that it can handle and relies on a human to do some difficult part of the work. This frees the human operator from monotonous parts of the work and allows them to use the saved time to operate multiple machines.

A challenge in teleoperation is how to give the operator enough information that they can safely and efficiently operate the machine. It is also possible for the machine to help them in this task. For example, when operator is controlling a boom in a forwarder-type forestry machine, the machine can take care of the collision free movement of the arm while the operator is focusing on the end effector.

The research questions this thesis answers are:

1. What techniques are available for path planning of a robot manipulator in changing unstructured environments?

2. How can a collision avoidance system be implemented using those techniques?

In this thesis an application capable of environment detection, modelling and path planning in an example case is developed. The approach is intended to be applicable to dynamic environments and to the case of mobile manipulators in that it is able to quickly construct a representation of an unstructured environment from the gathered sensor information and the representation can be updated as new information arrives. Path planning methods suitable for one time path planning are used to account for the changing environment.

The approach that is examined can be used to complement teleoperation of manipulators. The manipulator would automatically move near the target object and back to home position and the operator would take care of the often difficult grasping of target object.

This thesis begins with overview of range sensors (Section 2.1) that can be used for building a three-dimensional representation of the environment. In Section 2.2 mapping techniques capable of representing the environment in a way that is suitable for manipulator collision avoidance are examined. Section 2.3 presents an overview of collision detection methods. A brief look at collision avoidance in Section 2.4 provides a context for overview of path planning techniques. These techniques are presented in Section 2.5. First, the historical background of those techniques is provided by the classic methods and then recent sampling-based techniques suitable for the high-dimensional planning problems arising in manipulator motion planning are examined. The main focus is on two of the most well known families of sampling-based motion planning algorithms: the rapidly exploring random trees and the probabilistic roadmaps. Some of the recent improvements to those algorithms are presented, notably the RRT-Connect, RRT* and BT-RRT* algorithms that are the used in this work. Other sampling-based methods that are part of the OMPL motion planning library that was used in this thesis are also presented. In Section 3 the specified hardware and functionality for the motion planning system is first introduced and then the high-level structure of the system is examined. In Section 4 the software libraries used for implementing the motion planning system are introduced. Section 5 describes the implementation of the system in more detail. In Section 6 the calibration of the Kinect sensor and its position in relation to the robot are explained. Next, experiments comparing different planning algorithms together with two different configuration space metrics and the effects of post-processing the solution paths are performed and the accuracy of the robot and Kinect sensor are investigated. In Section 7, conclusions are drawn and possibile directions for future reasearch discussed.

2 Background

2.1 Sensing the environment

The availability of 3D information is important for avoiding obstacles and detecting objects in three-dimensional workspace. High sensing rate, long range and high spatial resolution are required features of the sensors for robots operating outside industrial settings. Further challenges are posed by the limited available space and power on mobile robots and the possibly difficult environmental conditions such as low lighting, dust or fog. Range sensing techniques include active methods based on lasers or ultrasound and passive techniques such as camera based methods.

Laser-based range sensors measure the distance between object and the sensor using either the time-of-flight(ToF) of the laser pulse, the phase difference between emitted and received laser pulses or triangulation methods. Frequency shift of a frequency modulated laser is also possible source of distance information, but it is rarely used in robotics [1]. In most laser based methods, a single laser beam produces one range measurement at a time and the direction of the beam is varied by either one or two-axis scanning mechanism. The ToF-based scanning sensor is also called a LADAR (Laser radar) or a LIDAR (light detection and ranging). The accuracy of these sensors is limited by minimum observation time and temporal accuracy of the observer as well as the duration of the laser pulse. The ToF sensors are suitable for long range sensing whereas the phase shift method is best suited to medium range (below 50 m) [1]. The phase shift method is also more sensitive to ambient light than ToF method. The advantage of the phase shift method is the higher sampling rate and consequently the ability to provide a higher spatial density of the measurements than a ToF sensor [1].

All laser sensor types (phase shift, frequency shift, triangulation and time-of-flight) have problems with specular reflections on surfaces such as polished metal [2]. Another typical error is caused by laser footprint, when part of a laser beam lies on a nearby surface and part on another one further away. In such cases the sensors may report in-between values for range [2]. In environments with fog, dust or smoke, scattering of the laser pulse may cause problems for laser-based methods [1]. For ToF laser sensors, the problem with scattering can be solved by measuring the last returning pulse.

Most widely used 3D range sensors in mobile robotics have been the actuated laser range finders (aLRF). An actuated laser range finder is constructed by mounting a 2D laser scanner on an actuator that enables getting measurements along the extra dimension. Challenges with the technology are related to the scanning speed, high weight and number of moving parts [3].

LADARs producing 3D point clouds have been available for some time. Although they have been able to produce measurements with good quality they have been too slow for many tasks in mobile robotics. Recently, sensors with better frame rates have become available. One example of such sensor is the Velodyne HDL-64E which provides 360 degree azimuth and 26.5 degree elevation field of view and 5 Hz to 15 Hz frame rate [4]. The sensor has range of 120 meters with range error less than 2 cm.

The unit has 64 emitter receiver pairs in the vertical direction and rotates around the vertical axis so that these sensors sweep across the scene. The rotation speed can be programmatically controlled, but the amount of measurements per second is constant, so this allows for different compromises between horizontal resolution and frame rate. At best the horizontal resolution is 0.05° with frame rate of 5 Hz. The vertical resolution of the sensor is about 0.4° [5]. As with other 3D laser range finders the high price of the Velodyne sensor remains a problem [3].

The performance of the scanning based techniques are often limited by the scanning mechanism. Therefore there has been interest in ranging techniques not depending on scanning. A time-of-flight sensor called flash LADAR has a timing circuit for each pixel of a sensor chip. A laser flash is used to illuminate the whole scene and measurement is then obtained for each pixel simultaneously [2]. Similar devices can be built using amplitude modulated continuous wave instead of short laser pulses. An example of such device is the SwissRanger SR-3000 depth camera. The scene is illuminated with modulated infrared light and each camera pixel records the phase-shift of the received light in relation to the transmitted light. From this phase-shift the depth can be simultaneously measured for each camera pixel. The time-of-flight cameras are a promising alternative in that they do not have any moving parts and are able to produce measurements of the entire scene at once. They do not require environment to have texture and provide their own illumination. Because of these features they can be used to complement stereo cameras. Typical time-of-flight cameras can provide dense range measurements at 50 Hz frame rate [3]. However, the technology is not yet as mature as the scanning LADAR sensors. The sensors have been found to be sensitive to ambient lighting which limits their outdoor use[6]. Sensors based on phase shift measurement have problems with wrap-round errors, where two distances resulting in the same phase-shift can be confused. Also reflections can cause error in range measurement.

Active triangulation is one of the first range imaging techniques used in robotics. A laser beam is swept over the scene and observed with a camera. From the position of the laser in image the depth is solved with triangulation. A laser of known wave length can be used together with a band pass filter to reduce the effect of other bright lights allowing to uniquely identify the laser spot from the image [2, pp. 529]. Although the method is simple to implement, it has several drawbacks when applied to robotics [1]. The accuracy of the system, as with passive triangulation, is dependent on the baseline of the stereo pair. Also, image has to captured by the camera for each position of the scanning mechanism, limiting the speed of the measurement. When projecting a single laser dot or a stripe, multiple images are required to reconstruct the full scene. It is possible to mitigate this latter problem by projecting a structured pattern instead of a single point or a stripe. However, this has the problem that it is difficult to project enough power over long range. The method is also sensitive to ambient light.

Stereo cameras are a passive triangulation based method. The advantages of the method are that the stereo camera system is smaller and cheaper than most laser based ranging sensors. Unlike scanning-based systems, stereo cameras are able to produce measurement for the whole image at a time which enables very

fast measurement. However finding corresponding points is more difficult than in active triangulation methods and requires computational power. A suitable stereo vision can be used for long range measurement. The range resolution of a stereo camera system is determined by its baseline and falls as a quadratic function of the distance [2]. Longer baseline provides better resolution, but causes more issues with occlusions, where some part of the scene is only visible to one of the cameras. This problem can be alleviated by using more than two cameras at the cost of increased computational cost. Other limitations include the dependence on natural texture of the objects and on ambient light. The lack of texture is especially common with man-made surfaces common in indoor settings whereas the problems with ambient lighting affect performance especially in outdoor settings.

In stereo camera system, two or more cameras produce an image of the scene. A measured point in the world and its projection on two of the cameras form a triangle, which allows the position of the point to be calculated when the angle formed by camera rays and the baseline of the stereo system are known (Figure 1). The images are modified to correspond to images that would have been taken by ideal pinhole cameras. Then matching features are searched from images. There are many ways to do this, but two common methods are local patch correlation and matching of discrete features [2]. The feature-based methods are more robust to viewpoint changes and camera differences, but yield only sparse depth image and require potentially slow feature extraction. The correlation method is easier to implement efficiently and produces a dense depth image, but can cause the expansion of nearby objects. After detecting the corresponding features by either the correlation methods or by feature-based methods, the disparity (position difference) between the feature as detected in left and right images $d = x' - x$ is calculated. The disparities for each image point form the disparity image. Disparity values can then be converted to depth using [2]

$$Z = \frac{f - xT_x}{d} \tag{1}$$

where f is the focal length, T_x is the baseline of the stereo pair and x is the x coordinate of the feature in the leftmost camera.

In November 2010 Microsoft released the Kinect sensor as a game console accessory. Since then it has been widely studied as a sensor for mobile robots. Its attractive features include a low price, relatively small size and the ability to get dense range measurements at high frame rate. The Kinect sensor is an integrated off-the-shelf solution and does not require physical modifications to be used as sensor in robotics.

The sensor is based on an active triangulation process where a pseudo-random infra-red speckle pattern is projected on the scene and detected using an infra-red camera. The camera and the projector form a stereo pair [7]. The camera image is searched for pieces matching parts of the pattern by correlation methods. The range to the measured objects is then solved using triangulation. Because the sensor projects its own pattern, the Kinect sensor does not depend on object texture or require ambient illumination. As with a stereo vision system the depth measurement accuracy of this type of a sensor is dependent on the baseline of the

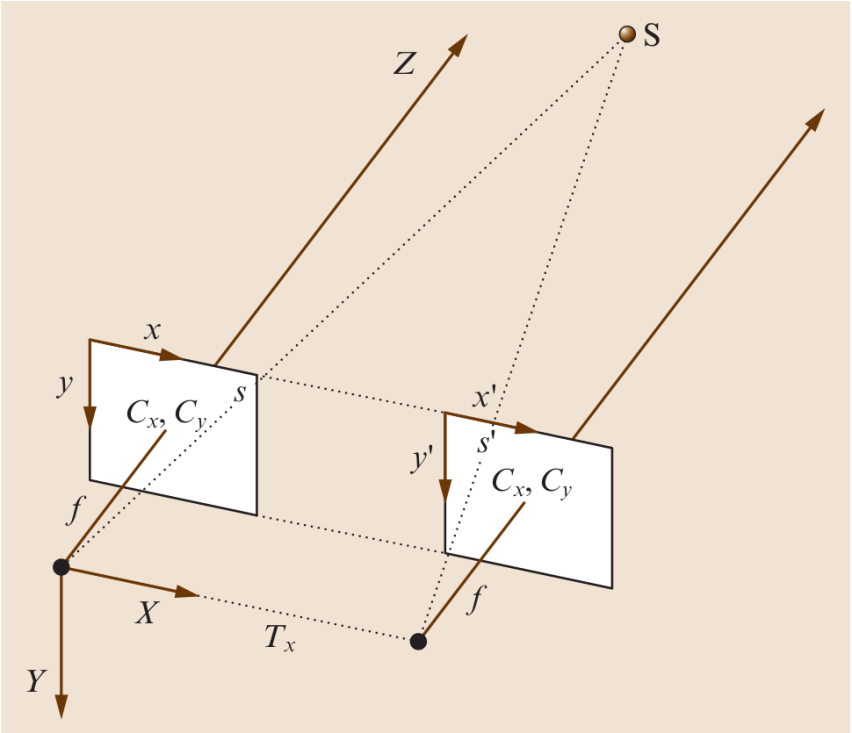


Figure 1: The ideal stereo geometry. The images are lined up horizontally and the translation between the focal points is along X axis. (C_x, C_y) is the point where the principal ray pierces the image plane. [2, p. 523]

sensor and degrades quadratically with respect to distance. In addition, the power of the projector may limit the maximum range.

Comparisons between different sensors can be found in literature. In [3] an actuated laser range finder, two time-of-flight cameras (SwissRanger SR-4000 and Fotonic B70) and Microsoft Kinect were evaluated for indoor mobile robotics and automated logistics use. It was found that an aLRF produces the most accurate measurements, but that for applications where low-range sensing is sufficient the Kinect and Fotonic B70 present a viable alternative to an aLRF. For ranges below 3 m the Kinect was found to outperform the ToF cameras whereas the performance of ToF cameras was better than Kinects for ranges over 3 m. In [6] an aLRF sensor, a ToF camera (SwissRanger SR-3000) and a stereo vision system (Videre STOC) were characterized in the context of response robots. The three sensors were similar in terms of space, payload, power and price. In [8] multiple aLRF type sensors, a flash-LIDAR, Kinect, custom stereo vision and structured light sensors and a frequency modulated laser sensor were compared for mapping of underground cave-like environments. The performance of the sensors was found to correlate with cost with the exception that the Kinect sensor was shown to greatly outperform its price point. Based on sample accuracy and density metrics the sensors fall into three groups with Faro Photon80 frequency modulated phase shift sensor forming the best performing group, all five planar ToF sensors the second group and the rest of the sensors in the third group. The performance of the sensors in the third group was found to be inconsistent.

2.2 Environment representation

Information gathered from the range sensors usually comes in the form of a range image or a point cloud [2, p. 522]. From these forms it can be further processed to form a representation of the environment. The two main categories of environment representation are topological and geometric representations. Topological representations are ones that do not contain metric information such as scale but retain connectivity information about the represented environment. Geometric representations also contain the metric information. The roadmaps constructed by various motion planning algorithms are an example of a topological representation. In this section some of the geometric representations are examined. Many formats are available for describing 3D data and they are suitable for different purposes and environments. For some tasks such as mobile robot navigation less than three dimensions may be enough. Purely two-dimensional maps, although useful in indoor navigation, are not discussed in this context.

Desirable features of the environment representation include updatability, particularly in case of dynamic environments, ability to differentiate between unknown and free space (e.g. sensor occlusions), compactness in terms of memory and storage, and the ability to represent environment in a way that is general enough. For example, line-based representation is general enough to represent indoor environments consisting mostly of straight surfaces but may not be general enough to adequately capture the terrain details and vegetation present in outdoor environments. In addition to these criteria it may be desired that the environment representation is cheap to construct and update, that the collision checking is cheap and that the representation can describe the environment in desired detail.

Although the environment models may be used for other purposes, in this thesis they are mainly utilized in the context of path planning. For some planning algorithms, a particular environment representation may be critical. However, for the sampling based planning any representation that can be used for collision checking with the robot will work [9, p. 89].

Point sets are a simple way to represent 3D data. The points in the set may be detected feature points such as corners where multiple planes meet or direct measurement points such as the point cloud formed from the range sensor measurements. A purely point based map may not be feasible because its storage requirements grow as new points are added to the map [10]. Point sets also do not provide a representation of the free or unknown space [11].

A triangulated surface model represents the world as set of triangles that can be large or small allowing the resolution of the representation to be adapted. For tasks like grasping and navigation, the representation should be constructed so that it encloses the represented objects entirely [2]. The triangle models are especially well suited to visualization as graphics accelerators use triangles as their model representation format [2]. The triangles can be texture mapped with pictures taken from the environment thus allowing for 3D scene reconstruction. Polygonal model has been used for 3D mapping in a mobile robot in e.g. [10].

In a voxel approach the world is represented as a potentially infinite grid of fixed-

size cubical volumes. For each such volume information is stored about whether the volume contains free space or a part of an object. In two dimensional mapping, grid based models are popular. Probably the most common example are the occupancy grid map models [12], where a grid cell stores the occupancy probability of the space it encompasses. These probabilities are then updated to reflect the measurements as they arrive. This approach allows the representation of unknown areas and provides a way of dealing with sensor noise.

For incorporating new information into the occupancy grid map, an inverse sensor model is used [11]. This model is different for each sensor and describes the conditional probability that a voxel is occupied given a measurement. Using a log-likelihood representation simplifies the update into a simple addition [11]

$$L(n|m_{1:t}) = L(n|m_{1:t-1}) + L(n|m_t), \tag{2}$$

where L is the log-likelihood of a volume n being occupied and m_t is the sensor measurement at time t. For laser range finders and similar sensors this update can be done with a ray-casting operation that is a 3D variant of the Bresenham algorithm [13]. Volumes traversed by the ray of the scanner are updated with the inverse sensor model [11]

$$L(n|m_t) = \begin{cases} l_{\text{occ}} & \text{, if ray is reflected within volume} \\ l_{\text{free}} & \text{, if ray traverses volume.} \end{cases} \tag{3}$$

The occupancy grid maps have been applied to 3D mapping as well [14]. The naïve grid representation of such a map suffers from high memory usage and potentially expensive computation if many cells have to be checked for content. In the uniform grid representation the resolution of the whole map is determined by those parts that require the highest resolution. A better way to implement the voxel map is as a hierarchical representation such as an octree [11]. In an octree representation either the volume is entirely occupied or free in which case it is marked as such or the volume is subdivided into two equal parts along each of the three coordinate axes producing eight child volumes. The subdivision is continued until a minimum cell size is reached or the entire subvolume can be represented by same occupancy status. This hierarchical representation allows multiresolution support where the map can be viewed in a lower resolution to improve speed of tasks such as path planning [11]. It also allows to delay the allocation of voxels until they are needed. In such implementation, the unallocated nodes implicitly describe the unknown space. In [15] occupied voxels are stored in a list for more compact representation.

An elevation model or a heightmap is in its most basic form a mapping $f : (x, y) \mapsto z$ that presents for each grid cell on a reference plane the height of the object in that cell along the normal of the plane. Heightmaps can be used in navigation, particularly in indoor and urban environments. As it is essentially a 2D grid, heightmap requires less memory than the 3D maps and it is also relatively cheap to do collision checking with. For instance, in [16] a hierarchy of successively coarser heightmaps was used for efficient collision checking of a planetary manipulator. The

main limitation of this basic form of the map is that it can not represent overhanging structures such as bridges. This limitation is solved in a multilevel surface map [17] by using multiple elevation clusters, each with its own mean and variance. For rough terrain navigation, other kinds of point statistics can be stored in the grid cells to describe the local ground plane and penetrability of vegetation [2, p. 539]. Although individual cells might be marked as unknown, the elevation maps do not store the free or unknown space in a volumetric way [11].

In the Normal Distribution Transform (NDT) representation, the measurement points are divided into bins similar to occupancy grid map and samples falling within a bin are summarized as a normal distribution [18]. The Normal Distribution Transform provides a piece-wise continuous and differentiable probability density, which is useful for scan matching. Although the original application of the NDT was in 2D laser scan matching, it has since been extended to 3D [19]. An alternative to the grid representation, as in the case of occupancy grid maps, is a hierarchical data structure, such as the octree. As an environment representation, the 3D-NDT has been shown to outperform occupancy grid maps in terms of representing the observations [20]. Also, the NDT map is less sensitive to the choice of grid cell size and can use larger grid cell size than occupancy grid without sacrificing accuracy. NDT and occupancy grid mapping can also be combined. NDT Occupancy Map (NDT-OM) [21] is capable of modelling free space similar to occupancy map while being able to accurately represent the distribution of measurement points even when a sparse grid is used. NDT-OM can be updated in real-time, even for large environments and can be used as a multi-resolution map.

2.3 Collision detection

In its simplest form collision detection is intersection testing between two models, returning a boolean value that tells whether the models are intersecting. It may also be valuable to know what part of the objects are touching. Sometimes single intersection point is sufficient, but for some applications such as rigid body simulation, a full contact manifold (set of contacting points) may need to be computed. The shortest distance that the objects need to be moved, so that they are no longer in collision is called the penetration depth. The opposite is the separating distance, which is the distance between the closest points on the two objects. The closest points themselves may also be of interest. For applications such as controlling the time step of a simulation, the time of impact (ToI) query may also be used.

Two different approaches exist for detecting the collisions between moving objects: Discrete collision checking and continuous collision checking [22]. These are also known as static and dynamic collision detection. Continuous collision detection takes the full movement into account and is able to report the exact time of collision and the first colliding points. Discrete collision detection approximates the continuous collision detection by discretizing the motion into a finite set of sample points. At these sample points the objects are tested for collision as if they were static. Discrete collision checking is much cheaper than continuous collision checking, but a short enough discretization step must be used to ensure that collisions

are not missed [23, pp. 16-17].

The collision detection algorithms form a pipeline that can be divided into *broad phase*, *narrow phase* and *exact phase*. This division was first used by Hubbard [24]. It has since been widely used in the literature, although today the exact phase tends to be included in the narrow phase [25]. In many cases, there are more than two objects that need to be checked for collisions. The broad phase algorithms are used to find potentially colliding pairs of objects with as little computational cost as possible, thus reducing the number of pairwise collision checks that need to be done. If all object pairs are tested $O(n^2)$ pairwise checks are required to check collisions between n objects. The narrow phase algorithms implement the pairwise collision checking between objects. Some of the methods such as bounding volumes can be used in both broad and narrow phase [25].

2.3.1 Broad phase

The simplest broad phase algorithm is the brute force approach where each objects bounding volume is collision checked with the bounding volume of every other object. This still results in $O(n^2)$ tests, but the tests may be simpler than the ones in narrow phase. The more advanced broad phase algorithms can be divided into spatial subdivision, topological methods and kinematic methods [26]. The topological methods are ones that are based on objects positions in relation to each other. One of the most well known methods in this category is the sweep-and-prune algorithm [27, 28]. Kinematic methods take into account the movement of the objects. The basic idea is that objects that are moving away from each other can not collide. For example, Vaněček uses the direction of motion to cull away the polygons on the opposite side of the object as those can not collide [29]. The idea of the spatial subdivision is that objects situated in distant parts of the space have no chance of colliding. Spatial subdivision techniques include quad-trees, oct-trees, BSP-trees, grids and k-d trees shown in Figure 2.

The grid-based subdivision methods work by dividing the space into unit cells. Objects are placed in the cells that they at least partly occupy. Then it suffices to test the collisions of object pairs that share at least one cell. For objects of uniform size this is quite ideal algorithm and is also suitable for parallelization [31]. However, it is difficult to set an optimal cell size and storing a large regular grid can require

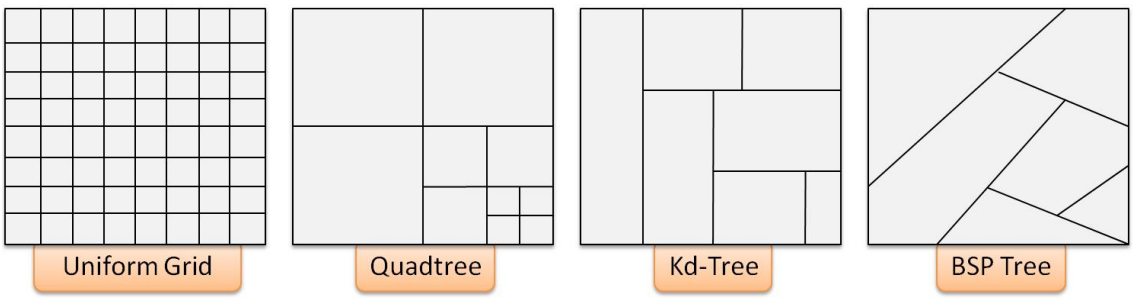


Figure 2: Different spatial subdivision schemes. [30]

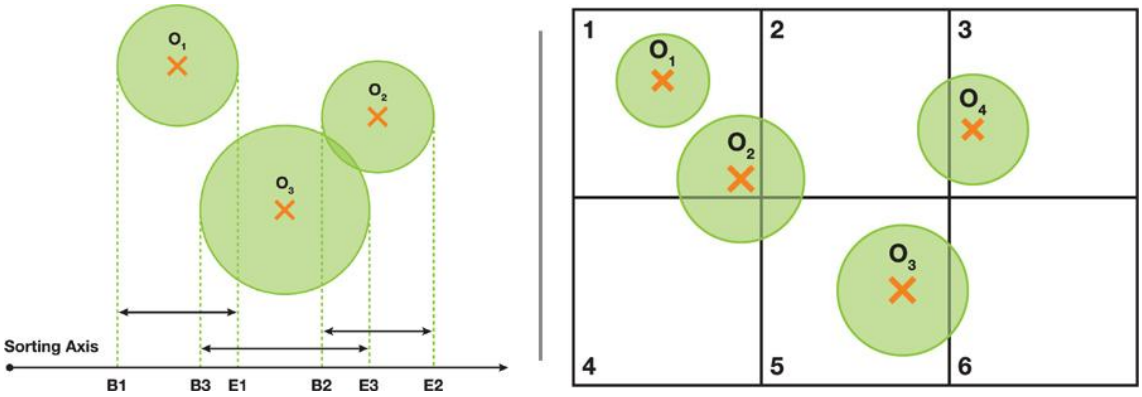


Figure 3: Left: Sweep and prune. Right: grid-based spatial subdivision [34]

lots of memory. A regular grid can not grow, so the dimensions and resolution of the space need to be known in advance. Using a hash function to map the grid to a hash table allows the representation of potentially infinite grid and is also more memory efficient [22]. Uniform grids can not easily deal with objects of varying sizes, but a hierarchy of grids with different cell sizes can be used [23, p. 300].

The other alternative is to use tree structures for spatial subdivision. Quadtree[32] is a data structure that divides a planar area in four parts at each tree level. These parts may be arbitrarily shaped, but squares are commonly used. Octree is a similar three-dimensional structure that partitions space into eight octants at each interior node of the tree. A binary space partitioning tree (BSP-tree) is a binary tree that uses hyperplanes to split the space into two parts at each level of the tree. In k-dimensional tree (k-d tree)[33] the normals of the splitting hyperplanes are aligned with the coordinate axes.

The sweep-and-prune algorithm [27, 28] shown in Figure 3 is based on the fact that a collision between objects can not occur unless the projections of their bounding volumes on coordinate axes overlap. The algorithm works by determining the axis-aligned bounding boxes (AABBs) of the objects and projecting those on each coordinate axis resulting in two points: the beginning and end of the interval that is projection of objects AABB. The projected endpoints of all objects are then sorted. When the sorted list is iterated, the objects that are candidates for collision are such that object B's beginning point is encountered after object A's beginning point and before its end point. Sorting the lists in general case take $O(n \log n)$ time, but the algorithm can be improved to exploit the temporal coherence between subsequent queries by making the sorting persistent between successive time steps. In environments where the objects move relatively little between two tests the list will already be nearly in order. After updating the coordinates of the endpoints the list can be sorted in expected $O(n)$ time with algorithms such as the insertion sort. For each object pair there is a boolean flag per dimension describing the overlap status of objects. The overlap status of the projections can only change when the insertion sort performs a swap. When the flags for all dimensions are set, the object pair is in the active list. Only the collision pairs in this list need to be examined.

2.3.2 Narrow phase

In the narrow phase the potentially colliding objects are checked pairwise. In a CSG representation the check is between the elementary geometrical objects comprising the object. In implicit function or parametric surface representation the tests involve membership functions and in polygonal models the test is done between the polygonal primitives. For a more complete survey of different methods used in the narrow phase collision checking see [25, 31].

The narrow phase algorithms can be categorized into feature-based, simplex-based, volume-based and image-space based algorithms and spatial data structures such as bounding volume hierarchies[25]. The feature-based algorithms use the geometric features (vertices, edges or faces) of the objects. The first such algorithm is the Lin-Canny algorithm [35], which tracks the closest features of two polyhedra. Temporal coherence between simulation time steps can be exploited by remembering the pair between time steps, as closest features are unlikely to change. Subsequently, improved algorithms such as V-clip[36] and SWIFT[37] have been based on Lin-Canny. The Gilbert-Johnson-Keerthi(GJK) algorithm[38] is the ancestor of Simplex-based algorithms. It has been generalized to be applied to arbitrary convex point-sets [39]. In image-space-based methods, image-based occlusion queries are used. These methods are suitable for implementing on GPUs, but can be implemented on CPUs as well. The volume based methods are conceptually similar to image-space methods, but also utilize techniques such as layered depth images and distance fields [25].

The implementation of narrow phase collision checking is influenced by the selected representation of objects (Figure 4). The models can be divided into polygonal and non-polygonal models [31]. Of the non-polygonal models, the most generic type is a polygon soup, which basically means just a collection of polygons with no definite structure. If the polygonal model defines a closed manifold, it is a proper solid with well defined inside and outside. These solids can be either convex or concave and many algorithms are developed for convex polytopes. The non-polygonal models form a more varied set. Objects can be described using constructive solid geometry (CSG) that describes objects as combinations of primitives such as spheres, blocks and tori using set theoretic operations. The produced objects are always valid, without any cracks and it is quick to test whether a given point is inside a CSG object[23, p. 10]. However, some shapes are hard to describe as CSG and the rendering of a CSG object may be difficult. The implicit surface model describes the object using a function $f : \mathbb{R}^3 \mapsto \mathbb{R}$ so that at the objects surface $f(x, y, z) = 0$ and on the inside $f(x, y, z) < 0$. This representation may allow fast intersection testing and can also be used as bounding volumes to accelerate collision queries [23, p. 9]. Unlike implicit surfaces, the parametric surfaces do not generally describe closed manifolds but they are easier to render than implicit surfaces. A parametric surface model is defined by mapping $f : \mathbb{R}^2 \mapsto \mathbb{R}^3$. One example of such a model is the non-uniform rational B-spline (NURBS). Point clouds are a primitive that has become increasingly popular with the development of range sensors capable of producing 3D measurements. However, the collision detection of these models has not yet been widely studied [40, 41].

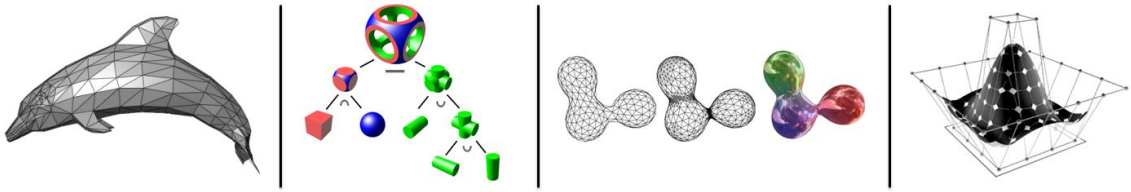


Figure 4: Different object representations. From left to right: polygonal mesh, constructive solid geometry, implicit surface and parametric surface (NURBS). [30]

2.3.3 Bounding volumes

Bounding volumes (Figure 5) provide a way to accelerate collision checking by enclosing the object geometry within a simpler object. The original objects can not collide unless the enclosing volumes also collide. For some applications, the collision check between BVs may even be sufficient and the exact collision checking can be skipped [23, p. 75].

In order to be efficient, the bounding volumes must be tight-fitting, so that they eliminate as many expensive collision checks between the contained objects as possible [23, pp. 76–77]. However, at the same time, the collision checking between the bounding volumes themselves should be fast. The bounding volumes should be inexpensive to fit to the underlying geometry. By realigning the bounding volume into the object when it moves, the possibly expensive refitting can be skipped. The bounding volumes should therefore be easy to rotate and transform. Finally, the bounding volume should use as little memory as possible.

Most common bounding volumes are the axis-aligned bounding boxes(AABBs) and spheres [23, p. 123]. Both have the advantage that the overlap testing is really fast. The AABB is easier to fit, but being rotationally invariant, the spheres are trivial to realign. The spheres are the most memory efficient bounding volume [23, p. 88]. Oriented bounding boxes (OBB) are hard to fit tightly and the difference between tight-fitting OBB and a poorly fitting OBB can be large [23, p. 107]. However, OBBs can be made to fit the geometry tighter than AABBs, thus leading to better pruning [23, p. 101–107]. Storing the OBB requires more memory than AABBs or spheres and the intersection testing is more expensive. However, the OBB can be rotated along with the underlying geometry.

Sphere-swept volumes are formed from a primitive shape such as a line or rectangle, by "sweeping" it with a sphere producing the Minkowski sum between the primitive and a sphere. The collision checking is done by calculating a distance between the inner primitives and subtracting the radii of the volumes. One of the sphere swept volumes is the capsule or sphere capped cylinder that is formed by sphere sweeping a line segment. Collision checking between capsules is faster than between cylinders [23, p. 113]. Realigning a capsule is also easy. The capsules require slightly more memory than AABBs.

Discrete oriented polytopes (k-DOPs) are another interesting bounding volume type. They are a convex region of space limited by planes with fixed set of normals, thus only minimum and maximum along these axes need to be stored. For example,

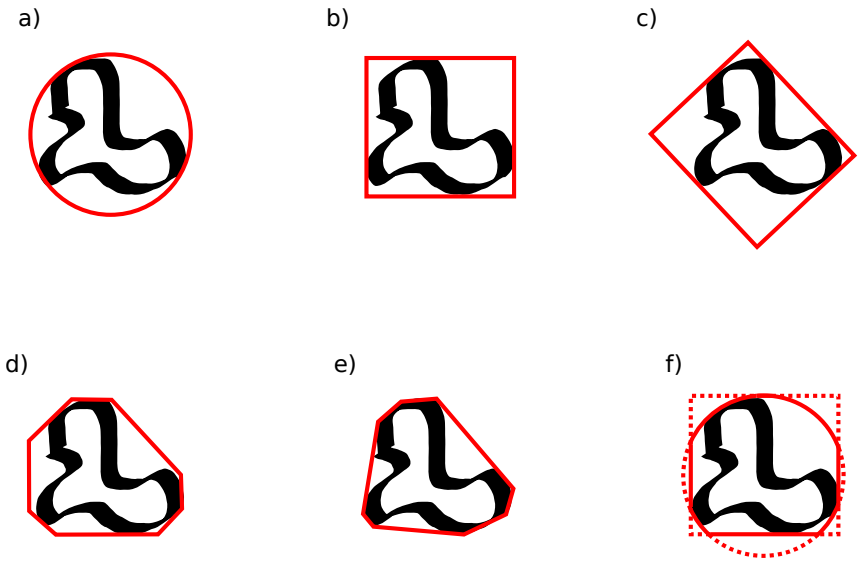


Figure 5: Some bounding volumes enclosing an object. The presented shapes are in 2D for clarity. a) sphere b) axis-aligned bounding box c) oriented bounding box d) discrete oriented polytope e) convex hull f) the intersection of sphere and axis-aligned bounding box

the 6-DOP is a three dimensional AABB and 8-DOP additionally has the corners cut. k-DOPs approach the convex hull as k grows, thus for high enough k they are better fitting than OBBs [23, pp. 117–118]. At the same time the collision test between k-DOPs is much faster than for OBBs, even when k is large. The updating of k-DOPs can be costly as they can not simply be rotated with the object.

2.3.4 Bounding volume hierarchies

A bounding volume hierarchy (BVH) is a spatial data structure that represents the objects as a tree hierarchy, where each node corresponds to a bounding volume [23, p. 235]. The bounding volumes of the object primitives form the leafs of the tree and the bounding volume of the entire represented object forms the root. The intermediate levels of the tree provide progressively less coarse approximation of the underlying geometry (Figure 6). While trees are often constructed so that the parent bounding volume encloses the bounding volumes of its children, this is not required. The parent's bounding volume should, however, contain the object primitives contained by the children. Compared with the spatial partitioning schemes, the difference is that bounding volume hierarchies partition the object primitives instead of space: where in spatial partitioning the partitions are disjoint so that each point in space belongs to a single partition, but the objects may belong to more than one partition, the bounding volume hierarchies allow the partitions to overlap, but place every object in just one of them.

Different bounding volumes have been used, such as cylinders, spheres [24], AABBs[42], OBBs[43], discrete oriented polytopes (k-DOP)[44] and convex hulls. In some cases intersection of multiple different BVs can be used [45]. Some of the alternatives are presented in Figure 5.

As different bounding volumes have different trade-offs between the desirable properties, it is necessary to consider the application. With rigid objects the goal may simply be to construct a BVH that can answer queries as fast as possible. For deformable objects, also the cost of refitting and updating the BVH has to be taken into account. For this reason, in these cases bounding volumes with simple geometry

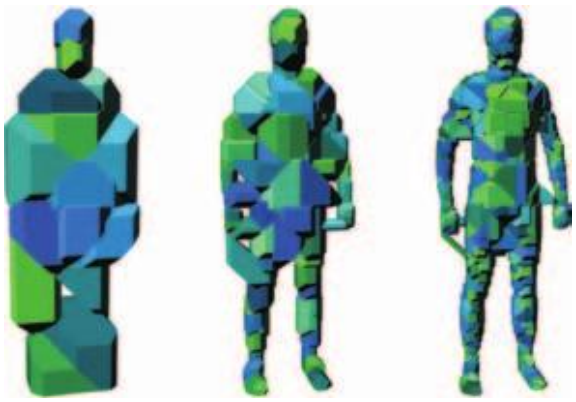


Figure 6: A bounding volume hierarchy of 18-DOPs provides progressively finer approximation of the underlying geometry. [22]

such as AABB may perform better than more tight fitting complex volume such as OBB [22]. In the case of robotic manipulators with rotatory joints, the ability to realign bounding volumes as the manipulator moves instead of refitting is a beneficial property.

Another design feature of the BVH is the number of children per tree node, that is, the *arity* of the tree. For rigid objects binary trees are commonly used, but for deformable objects 4-ary or 8-ary trees have been shown to exhibit better performance [22]. As the trees with high arity are shallower, the update costs of the BVH are smaller. The shallowness of the tree also reduces the memory requirements for the stack when traversing the hierarchy. However, more work has to be done in each visited node if the arity is high [23, p. 238].

Finally, there are many strategies for building, updating and traversing the trees. The trees may be constructed bottom-up, top-down or with insertion construction [23, pp. 235–259]. The top-down strategy is most commonly used, although it may not result in the best tree. In the top-down strategy, the objects are divided into subsets at each level and a BV is estimated for those subsets. In the bottom-up strategy proceeds by merging smaller sets of objects and estimating the BVs for the produced sets. In insertion construction, object primitives are added to the tree one at a time.

When collision checking between two BVHs, the descending strategy determines the order in which the two trees are traversed. The trees can be traversed simultaneously, alternatingly, breadth-first, depth-first or according to some heuristic [23, pp. 254–255]. It is possible to optimize the traversal by remembering the set of intersection tests that prove the two BVHs distinct [23, p. 282] or by grouping together multiple queries that traverse the same branches of the BVH [23, p. 278].

2.4 Collision avoidance

At a basic level two kinds of approaches exist for collision avoidance: *global* and *local* techniques. The global approach is motion planning that takes into account the entire known environment of the robot. The motion of the robot is planned from an initial point to the target point before executing it. The main limitation of the global techniques is that they are computationally demanding, limiting the ability to replan in a case where the environment is continuously changing. The global techniques are therefore not appropriate for fast obstacle avoidance. Challenges may also be posed if the world model is inaccurate or incomplete [46]. The local techniques are control based and more reactive than the global techniques. These techniques only take into account obstacles that are near the robot. The main advantage of the local techniques comes from being computationally light and therefore able to react to changes in the environment at a high rate. By themselves the local techniques can result in globally suboptimal behavior and even get stuck in parts of the environment where the locally optimal motion does not lead to progress towards the goal [46]. An example of such a case is a robot that is stuck in the bottom of a U-shaped obstacle. If progressing toward the bottom brings the robot closer to its goal, it may not be able to find its way out without global information about connectivity of the space.

Instead of replacing the global planning, the local methods may be used to augment it.

For mobile robots many different local collision avoidance methods have been developed such as nearness diagrams[47], vector field histograms[48], the dynamic window method[46] and the artificial potential field methods[49, 50]. Out of these, the artificial potential fields have also been popular with manipulators [49, 51, 52]. In the potential field approach, a virtual repulsive force is calculated between the robot and obstacles similar to the one between two electric charges of same polarity. A method for evaluating the distance between robot and obstacles is required to calculate the potential field. After the repulsive force is known, a method that calculates the appropriate reactive movement by the manipulator is applied. One of the difficulties of potential field method when applied to articulate structures is obtaining a smooth repulsion force [53].

There are also methods that combine features of planning based approaches and reactive approaches. In the elastic band framework[54], the path is first planned globally and then represented as an elastic band, a curve in configuration space that is affected by various virtual forces repelling it from obstacles while maintaining the tautness of the elastic band. The reaction to dynamic obstacles is achieved by deforming the elastic band. However the performance of the elastic band method may degrade in high dimensional configuration spaces [55]. In redundant manipulators there are more degrees of freedom than is necessary for performing the manipulation task. The extra degrees of freedom can be used to perform a secondary task of collision avoidance. One of the approaches capable of utilizing this *self-motion* of the manipulator is the elastic strip framework[55]. The elastic strip framework allows the execution of previously planned motion in dynamic environment for robots with many degrees of freedom. It combines real-time obstacle avoidance based on a potential field with a desired posture behavior. The planned path is incrementally modified in a way that maintains the topological properties of the path as well as the constraints imposed by the task. Therefore the modified path remains valid.

Simpler collision avoidance schemes have also been proposed. Instead of modifying the path of the robot to avoid collision it may be sufficient to control the speed of the robot along the path [56]. This is the case when the obstacles will move out of the path of the robot by themselves.

2.5 Path planning

Path planning refers to the geometric specification of a robot's movement [57]. A related term is *trajectory planning* which also takes into account timing of the movement. The common term for the two is *motion planning*. Motion planning is either static or dynamic depending on whether all information about obstacles is available a priori. *Constrained motion planning* deals with additional constraints to robot's movements such as the limits on velocity and acceleration or the curvature of the trajectory whereas *unconstrained motion planning* only deals with obstacles. *Kinodynamic motion planning* problems pose limits on velocities, forces and torques to be satisfied in addition to the kinematic constraints such as obstacles [58].

The motion planning algorithms can be classified based on their completeness into exact and heuristic methods [57]. The *exact methods* always either find a solution or prove that none exists. Because the motion planning problem is PSPACE-hard [59] exact methods are computationally expensive. That is why *heuristic algorithms* are often used. These algorithms may fail to produce an answer for certain hard cases even though they would be solvable. Two milder forms of completeness can be defined: *resolution complete* algorithm works in a discretized way and thus returns approximate results. It can, however, be made to approximate the continuous case to an arbitrary precision by making the discretization step smaller. A *probabilistically complete* algorithm finds a solution with a probability which can be made arbitrarily close to 1, typically by devoting more time to the search.

Single-query planning algorithms are algorithms optimized for planning a single path as efficiently as possible, whereas *multi-query* algorithms use a potentially expensive preprocessing step to be able to solve subsequent planning tasks in the same environment as efficiently as possible [9, p. 186].

World space is the physical space in which robots and obstacles exist. In addition a description called *configuration space* is used in motion planning. A configuration of an object is a set of parameters that specifies the location of every point of that object. For a solid object in three dimensional space, there are three parameters specifying the position of the object and three parameters specifying its rotation. A configuration space for a robotic manipulator is formed by the positions of its joints. The configuration space can be divided into free configuration space ($\mathcal{C}_{free}$) and the configuration space obstacles ($\mathcal{C}_{obs}$). Configuration space obstacles are the set of those configurations that cause the robot to collide with itself or an external obstacle. The dimensions of the configuration space of a robot are referred to as the degrees of freedom of the robot.

The configuration space (C-space) plays a significant role in motion planning. All path planning problems are essentially equivalent when formulated in configuration space and can be reduced to finding a connected sequence of points between a starting configuration and a goal configuration [57]. There are multiple ways of computing the configuration space obstacles from the world space. For example, consider the robot a rigid object without rotation. The configuration space obstacles can be given as the union of the Minkowski differences between area occupied by robot and the area occupied by each obstacle. The Minkowski difference means the difference of two sets as given by [57]

$$Mdiff(A, B) = \{a - b | a \in A, b \in B\}. \tag{4}$$

Many motion planning algorithms rely on representing the configuration space obstacles explicitly as a set of occupied cells or polygonal objects. The explicit representation of the configuration space of a robot manipulator is problematic because it typically has high dimensionality and the shapes of the configuration space obstacles are complex [57]. For this reason planning methods that avoid explicit description of the configuration space are required. One category of such algorithms that has been successfully applied to wide variety of problems are the sampling based planning algorithms. They probe the configuration space only at a set of discrete

sample points. To determine whether some configuration belongs to an obstacle or free C-space the planning algorithm calls the collision detection algorithm, which is treated as a black box [9, p. 185]. Thus the motion planning algorithm does not need to concern itself with details about how objects are represented and how the collision checking is implemented.

Two much used sampling based planning methods are probabilistic roadmaps (PRM)[60] and rapidly exploring random trees (RRT)[61]. The probabilistic roadmap is a multi-query planning algorithm whereas the RRT is a single-query algorithm. In a relatively static environment the PRM allows solving subsequent path planning problems faster while the first path carries the overhead of preprocessing. Both of these have been further improved to form a large group of related path planning methods. The basic versions of the algorithms are probabilistically complete but almost surely suboptimal [62]. For both methods, an asymptotically optimal variant exists that is slower by a constant factor [62].

2.5.1 Classic methods

The classic methods are variants of a few general approaches: roadmaps, cell decomposition, potential fields and mathematical programming [57]. In roadmap methods the free C-space is reduced to a network of one dimensional lines. The motion planning problem then becomes a graph search problem. First start and end states are connected to points on the roadmap, then the path between those points is composed from lines belonging to the roadmap. For the method to be complete, the roadmap must represent all topologically distinct paths within $\mathcal{C}_{free}$.

In *visibility graph* method the roadmap is composed of lines joining features of the objects, such as the vertices of the polygonal obstacles [57]. The visibility graph can be constructed in $O(n^2)$ time, where n is the number of features on the objects. In the *Voronoi diagram* method, the Voronoi diagram of the obstacle features is used. The Voronoi diagram is the set of points that are equidistant from two or more object features. This method is good if the robot has to stay away from obstacles. Voronoi diagram can be constructed in $O(n \log n)$ time, where n is the number of features.

Voronoi diagrams and visibility graphs are used mostly for 2D motion planning. For dimensions higher than two, both have higher complexity [57]. It is not obvious what to select as the features as the Voronoi diagram of polyhedra is not a 1D roadmap, but is composed of 2D faces. In the two-dimensional case, the visibility graph contains the shortest path, if start and goal configuration are included. For higher dimensions the vertices of the polyhedra can be chosen as features, but the resulting roadmap no longer contains the shortest path.

Two other roadmap methods are the *silhouette* and *subgoal network*. In silhouette method the objects of a high dimensional space are first projected into lower dimensions and then the boundary curves of the projection are traced. Trough recursive projection, 1D lines are finally obtained. The subgoal network is unlike the other methods in that it does not build an explicit representation of the C-space obstacles. Instead it maintains a list of configurations reachable from the start where the reachability is tested by some simple local motion planning algorithm such as

moving in straight line.

In *cell decomposition* methods the free C-space is decomposed into cells. A sequence of connected cells between starting configuration and the goal configuration forms the solution path. The decomposing can be done by either taking into account the shape of the obstacles or ignoring it [57]. It is more difficult to build an object dependent decomposition, but it results in a smaller number of cells.

The *potential field* method[63, 49] describes the robot as a particle moving in artificial potential field that resembles the electric field. The structure of the field is so that the obstacles have high potential that push the robot away and the goal has a low potential that attracts the robot. The potential field method can also be used to build real time control based obstacle avoidance or as a component in other path planning algorithms. Its biggest problem is the local minima of the potential field where the robot can get stuck. If there are many concave obstacles, the expression of the potential field may become cumbersome.

In the *mathematical programming* based approaches the path is presented as a curve between the start and goal configurations. The obstacles are represented as constraints in the problem and because the resulting problem is nonlinear some numerical method is used to solve it. Heuristic optimization methods such as *genetic algorithms*, *particle swarm optimization* or *simulated annealing* can sometimes be used to solve the optimization problem [64].

2.5.2 Probabilistic roadmaps

The probabilistic roadmap approach [60] is related to the classical roadmap techniques such as the visibility graph. It is a probabilistically complete heuristic algorithm that is suited for motion planning in high dimensional C-spaces. The algorithm consists of two phases: in the *learning phase* a roadmap is constructed and in the *query phase* a path is found using the roadmap. The aim is to make the query phase almost instantaneous through preprocessing done in the learning phase. The algorithm may use any amount of time building the roadmap and the time should be long enough so that $\mathcal{C}_{free}$ is adequately represented. However it is difficult to know when this point has been achieved. It is possible to interleave learning and query steps so that a more complete roadmap can be built if the query phase fails. Because of this two-phase nature the probabilistic roadmap method is suited to static planning scenarios where multiple motion queries are to be planned in same environment.

In the learning phase the C-space is first sampled randomly. If the random configuration is collision free it becomes a *milestone*. For each new milestone a connection to nearest neighbors that are already part of the roadmap is attempted. The connection is attempted to nearby nodes first and only if they are part of a different connected component of the roadmap. Whether connection is possible or not is determined using a simple local planner. These steps are repeated until a sufficiently large roadmap has been built. It is not necessary to store the paths generated by the local planner as those can be easily recomputed later.

If after this construction step the roadmap is still not connected, an attempt

is made to connect it by sampling configurations near the difficult places. These can be determined by, for example, examining the amount of neighbors a node has within some radius or examining the distance (using some distance measure) to a nearest disconnected component.

In the query phase, the start and goal configurations are first connected to the roadmap using the local planner. After that, a graph search algorithm can be used to quickly find the path between start and goal. The path of the robot can then be reconstructed by using the local planner to plan the paths between the nodes of the path that were returned by the graph search.

2.5.3 The rapidly exploring random trees

Rapidly exploring random tree (RRT) [61] is a randomized data structure which can be used to solve high dimensional motion planning problems. It is also able to handle non-holonomic constraints such as dynamics. Unlike probabilistic roadmap methods the rapidly exploring random trees allow to construct path planners without requiring the ability to steer the system between two prescribed states. Also, the method does not use preprocessing in the same way as PRM, which makes it fast for single query planning. RRT is probabilistically complete under very general conditions. The produced path is not optimal but merely a feasible one [62]. It has been experimentally determined that in plane, the distance of a randomly chosen vertex of the RRT to the root is 1.3 – 1.7 times the Euclidean distance of the points [65].

In the case of RRT the planning space is the *state space* of the system. For standard problems this can be simply the configuration space of the robot, but for kinodynamic planning the state space also includes the velocities. Many other interpretations of the state space are also possible.

The basic RRT planner works by growing a tree in free state space starting from a start state. A random state from the state space is selected and the node of the tree that is closest to that random state is selected. The required control input to bring this state towards the selected random state is then determined and a new leaf of the tree is formed by simulating the behavior of the robot over some time interval assuming this control is applied. If this leaf obeys all the restrictions of the problem and is collision free, it is added to the tree. When a node is close enough to the goal state or the allotted time is exhausted, the process terminates.

The sampling procedure of RRT ensures that the tree growth is biased towards the largest unexplored regions of the state space. Although the algorithm is simple, it is not easy to find a method with this rapid exploration property. As an example, a naïve approach was tested that just grows the tree by selecting random control inputs [61]. Such a method ends up re-exploring areas close to the starting state.

Later RRT methods have been improved in many ways including the use of biased distributions of random points, modification of the way the new vertices are added and the use of more than one RRT [65]. It is possible to make the search more greedy by growing the tree with biased random distribution like selecting the goal position with some fixed probability instead of the random state or selecting a

state near goal state so that the distribution is controlled with the distance between tree and goal.

In RRT-Connect [66] one tree is grown from the starting configuration and the other starting from the goal position. First, one of the trees is expanded as in original single tree RRT planner then an attempt is made to connect the nearest vertex of the other tree to the vertex that was added to the first one. The trees take turns in these two roles. When applicable the bidirectional search approach is much more efficient than the single RRT approach. However in the bidirectional approach there is a need to join the two trees with some method and in non-holonomic cases this may be problematic [65].

The other idea presented in [66] was to repeatedly grow the tree towards the selected random state instead of just adding a single vertex. This makes the tree growth more greedy in searching a connection with the other tree and it reduces the need for nearest neighbor queries. It was found that this approach works best for relatively uncluttered spaces and for holonomic planning problems [65].

Recently the suboptimality of RRT regarding path length was proved and an asymptotically optimal variant RRT* was formulated in [62]. When adding a new vertex the RRT* algorithm considers the set of vertices that are close to the new vertex, connects the new vertex to the neighbor that allows for minimum cost path and rewires the tree in a way that checks if a path with lower cost from the root to some of the neighboring nodes is possible through the new vertex. It has been proved that this algorithm is both probabilistically complete and asymptotically optimal. In addition it is computationally effective, being within a constant factor from the running time of RRT. It is possible to use different additive cost functions, such as specifying low or high cost regions.

In [67] a ball tree algorithm is described. It is a modification of RRT where, instead of storing sample points of free configuration space, hyper-spheres are used. In the exact version of the algorithm the radius of these hyper-spheres is set to the distance of nearest obstacle. The sampling step is modified to reject samples that are inside a sphere, which is already known to be free space. This allows the sampling to focus on more difficult areas of the configuration space and results in trees that are much sparser than those of the plain RRT algorithm. In the inexact variant of the algorithm the radii are first set optimistically, but reduced if collision is detected during the connection of a new state to the tree.

In [68] the RRT* algorithm is combined with a ball tree based approach and memoized collision detection. The advantage of the ball tree algorithm is that it quickly finds the initial path whereas RRT* is able to improve the solution by using extra planning time. It is found that the proposed BT-RRT* algorithm finds initial solution quickly (2.51 seconds compared to RRT*'s 7.92 seconds) but is able to spend the extra computation time to improve the solution like RRT* and reaches similar cost for the final solution. The algorithm is compared to both RRT and RRT* on single and dual arm motion planning scenarios (7 and 12 DOF).

Anytime planning algorithm based on RRT* is presented in [69]. This algorithm divides the planning to *initial planning phase*, which is continued until the robot has to start moving, and *iterative planning phase* in which the robot keeps refining the

path while executing it. Two improvements to RRT* are presented. The first is the notion of *committed path*. The robot commits to following the current plan to some future time instant and the vertex in RRT corresponding to this future time is then made the new root. This locks the path until that moment against modifications and allows the algorithm to focus on improving the rest. The other improvement is a *branch-and-bound* approach that uses an admissible heuristic similar to the one used in A* to periodically prune away parts of the tree that can not improve over the best currently known path.

One problem with the RRT* algorithm is that the vertices forming the path to a goal may be surrounded by other vertices thus making their Voronoi regions small. This means that the path improvement is slow as these vertices are unlikely to be selected. In [70] two methods are presented for biasing the sampling of RRT* so that path improvement would be faster. One is a local biasing method and it was found to be useful. The other is based on rejecting the new node if the current best path length is less than the cost of going directly from start state to the goal state via the new node. In addition the paper presents a bidirectional RRT* which is a combination of bidirectional RRT and RRT*.

2.5.4 Other sampling based methods

Expansive Space Tree (EST) algorithm [71] is a single-query bi-directional tree-based planner. The problem with PRM planners that the EST tries to address is that they sample connected components of the configuration space that are irrelevant to the path that is being planned. One tree is grown starting from the goal configuration and another from the start configuration. When the trees are close enough they are connected if the connecting edge is collision free. To each milestone a weight $w(x)$ is associated that is the number of other milestones within some constant radius from it. The samples are generated near a selected milestone where the probability of a node getting selected is $1/w(x)$. K samples are generated and each sample y is retained with probability $1/w(y)$.

In [72] a Single-query bi-directional probabilistic roadmap planner with lazy collision checking (SBL) was presented. The algorithm is motivated by the fact that PRM planner spends more than 90% of its running time doing collision checking. Also in their tests the authors note that if two configurations are collision free and close enough to each other, then it is likely that the edge connecting those two milestones is also collision free. The algorithm grows two trees in a similar manner to the EST algorithm. The *lazy collision checking* means that the collision checking is postponed until its absolutely necessary. All milestones in the tree are initially assumed to be connected by collision free paths. When the trees meet and form a path from start to goal configuration this path is collision checked by recursively subdividing the edges and checking the collision of midpoint until a desired resolution is reached. Once a collision is found, the respective edge is removed which may cause parts of the graph migrating across the trees. In the tests performed the lazy collision checking provides a speed advantage of a factor 4.4–40.

The Kinodynamic Planning by Interior-Exterior Cell Exploration (KPIECE) al-

gorithm [73] is a planning algorithm designed for complex systems where integration backwards in time is not possible. It maintains a tree of motions where each motion is obtained by simulating the system forward in time from a state while applying random control input. These motions are formed from states that correspond to the integration steps, but these intermediate states are not explicitly saved. The planner keeps track of the coverage of the state space by performing a projection to a Euclidean space of lower dimension. This can be for example the space of the coordinates of the end effector of a manipulator or the center point of a car-like robot. If the user does not specify the projection, random projections are used. The lower dimensional space is then discretized into cells and the motions are split so that each motion belongs to a single cell. A set of heuristics is used to select the motion for expansion. Firstly, there is a fixed probability of selecting a motion from an exterior cell (a cell that is at the boundary of the explored and unexplored region) versus an interior cell. Then a cell is deterministically selected based on the number of states inside the cell, the last time the cell was selected, the iteration where the cell was first encountered, the number of explored neighbor cells and the prior success of the exploration when starting from a given cell. Out of the motions contained within the cell, one is selected at random, biasing towards the recently added motions. From the selected motion, a start state is picked with uniform probability. In the tests performed the KPIECE algorithm was compared with RRT and EST algorithms. A speedup of two orders of magnitude was observed in the complex cases as well as a reduction of memory requirements by a factor of 40. In the simpler cases both RRT algorithm and EST outperformed KPIECE.

SyCLoP [74] is a hybrid planner that uses a discrete planner operating on a decomposition of the work space to guide the exploration of a sampling-based continuous planner operating in the state space. The discrete planner computes a *lead*, a sequence of workspace segments connecting the start and the goal regions. The lead directs the sampling of the continuous planner towards the goal. As there are many possible leads, the selection of a lead is biased based on a cost that is in turn updated based on the success of the sampling-based planner. Many different planners can be used as a part of SyCLoP, such as RRT, EST and SBL. Experiments with these planners in high-dimensional dynamic planning problems show significant speed improvement of up to two orders of magnitude when used as part of SyCLoP.

3 A sensor-based motion-planning system

In this thesis a collision avoidance method for a robot manipulator is implemented. The system used for the experiments is implemented indoors in a tabletop environment, but the methods that are investigated are intended to be applicable to a wide variety of sensors, manipulators and environments. In this section, the functional specification and the available hardware for the implemented system are presented. After covering the specifications, the system level design is presented.

3.1 Functional specification

The system needs to be able to observe the environment and based on the observation plan movement that brings the manipulator to a desired goal position without collision. The planning and execution times should be reasonable to allow interactive use as part of teleoperation. The system needs to be able to adapt to changes in environment, but fast changes such as those caused by humans entering the working area do not have to be accounted for.

3.2 Hardware specifications

3.2.1 Kinova JACO-arm

The robot used in this work is JACO research edition from Kinova [75]. It is a lightweight robot manipulator with 6 degrees of freedom and a three finger gripper. The structure and dimensions of the arm are presented in Figure 7. The arm has a maximum reach of 90 cm and a maximum payload of 1.5 kg. The absolute position accuracy of the joints is $\pm 0.5°$ and the maximum velocity 8 RPM for the large joints and 10 RPM for the small ones. In Cartesian mode the maximum linear speed is 15 cm/s and absolute position tolerance 8 mm. The arm has sensors for current, Hall effect, absolute position and temperature in each actuator. In addition there is an accelerometer at the base.

The arm can be controlled with the included joystick or through a USB 2.0 interface. Both speed and position control is supported in both angular and Cartesian modes. The driver supports both Windows and Linux operating systems. At the time of this work, the API of the robot [76] was only provided as a set of .NET assemblies. Later a native C++ version has been released by the manufacturer. The API consists of three "managers" each acting as a facade to different methods. The control manager can be used to send a trajectory to the robot for execution and for querying the positions of the arm joints and the end effector pose. The other managers provide functions such as reading the sensors, configuring the joystick interface and diagnostics.

The angular control is rather simple and consists of specifying the pose of the robot joint by joint. A path composed of these configurations can be sent to the robot for execution. There are some limitations on the angles specified [76]. The joints 1,4,5 and 6 have a range of $[-10000°, 10000°]$. Joint number 2 has a range of $[42°, 318°]$ and joint 3 has a range of $[17°, 343°]$.

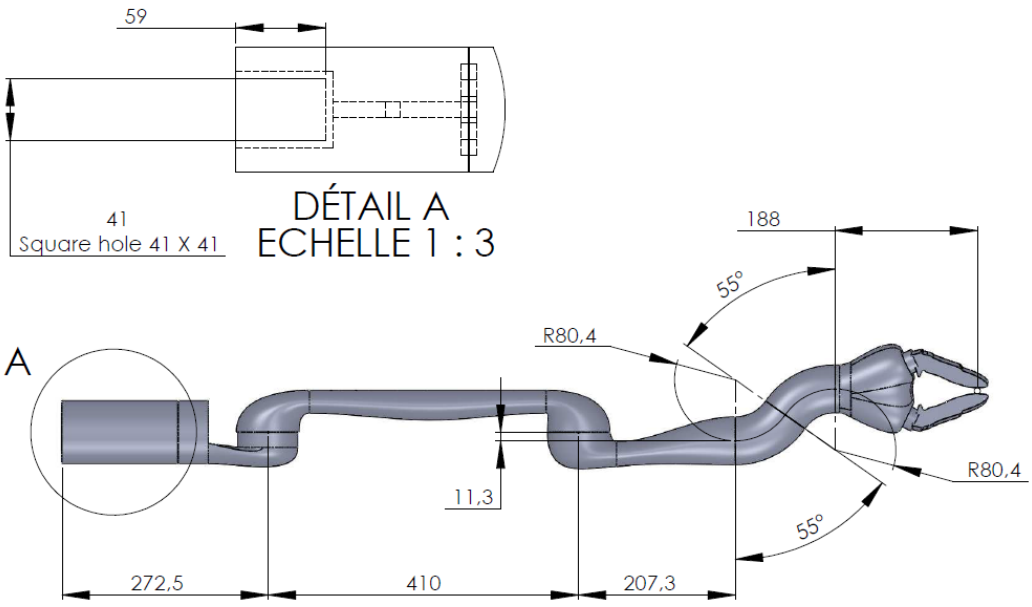


Figure 7: The dimensions of the JACO manipulator.[75]

3.2.2 Kinect

Microsoft Kinect is a range sensor based on active triangulation that is marketed as a controller for the XBox 360 console. Partly due to its low cost it has attracted the attention of robotics researchers. However, because it is a closed product, the details of its inner structure and functions are not publicly available. The details that are known are mainly due to reverse engineering efforts by third parties. There are two open source drivers for Kinect: libfreenect[78] and OpenNI[79].

Kinect sensor is based on structured light. It contains an infrared projector in which coherent light is diffracted by an object to form a pseudo-random pattern (Figure 9). The projector forms a stereo pair with an infrared camera. In addition to the IR camera-projector-pair the Kinect contains a separate RGB camera which can be used to detect the color information (Figure 8). The infrared image is searched for pattern corresponding to the one that was projected using a small correlation window [80]. From the disparity between the projected and detected images the distance of an object can be determined. Due to this correlation window, there is a small shift between the depth image and the IR-image. The correlation is estimated with sub-pixel accuracy of 1/8th of a pixel.

In [80] the reprojection errors of both Kinects cameras were examined. It was found that for IR camera the reprojection error was 0.34 pixels before calibration, which went down to 0.17 after calibration. Similarly for RGB camera the reprojection error was 0.53 but went down to 0.16 after calibration. From this they draw the conclusion that the cameras are unlikely to be calibrated for distortion correction out-of-the-box, but that the quality of the lenses is high compared to usual webcams. They note that while it is important to calibrate the RGB camera if it is used for machine vision purposes it might not be so critical for the IR camera since the depth image is already smoothed over neighborhood of pixels by the correlation window.

Figure 8: Kinect sensor with cameras and projector annotated. Picture adapted from [77]

Figure 9: a) RGB image produced by Kinect b) IR image of pseudorandom IR pattern projected on JACO arm and its environment by Kinect (with contrast and brightness added) c) The computed depth image

The equations

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = R(\vec{x} - \vec{c}) \tag{5}$$

$$\begin{bmatrix} x_n \\ y_n \end{bmatrix} = \begin{bmatrix} x/z \\ y/z \end{bmatrix} \tag{6}$$

$$\begin{bmatrix} x_p \\ y_p \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & \alpha_c f_x & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_d \\ y_d \\ 1 \end{bmatrix} \tag{7}$$

$$\begin{bmatrix} x_d \\ y_d \\ 1 \end{bmatrix} = (1 + k_1 r^2 + k_2 r^4 + k_5 r^6) \begin{bmatrix} x_n \\ y_n \\ 0 \end{bmatrix} + \vec{d_x}, \text{ where } r^2 = x_n^2 + y_n^2 \tag{8}$$

$$\vec{d_x} = \begin{bmatrix} 2k_3 pq + k_4(r^2 + 2p^2) \\ 2k_4 pq + k_3(r^2 + 2q^2) \end{bmatrix} \tag{9}$$

describe a geometrical model for the cameras [81]. Equation 5 describes the transformation of point $\vec{x}$ from world coordinates to camera coordinate system. $\vec{c}$ is the position of the camera and R is a rotation matrix describing its orientation. These are the extrinsic camera parameters. The transformed coordinates are then projected to image space using Equations 6 and 7. f_x and f_y are the focal lengths in horizontal and vertical pixels. These may be different when the camera has non-square pixels. α_c describes the skewness of the pixels. This is usually zero, as cameras tend to have rectangular pixels. (c_x, c_y) is the principal point of the camera. Equations 8 and 9 describe the lens distortions, that is the deviation from the ideal pinhole camera model. In these equations k_n are the distortion parameters. The distortion is composed of two parts: radial and tangential. The tangential distortion d_x is often left out, because most lenses do not have error in centering. A model with only radial distortion is easily invertible [81]. Smisek et al. [82] calibrated the camera parameters of Kinect using this model. Parameters determined by them are presented in Table 1. These are likely to vary between individual units, but their magnitude should remain comparable.

In [7] the accuracy of Kinect's depth data is examined. It is noted that a properly calibrated Kinect does not contain large systematic errors. The resolution of depth measurement is shown to decrease and the random error increase as a function of the measurement distance. The sensor is estimated to have an error ranging from few millimeters at the distance of 0.5 m to about 4 cm at the sensor's maximum range of 5 m. The error increase is quadratic with respect to the distance.

Similar to the work done in this thesis, Kinect is used to capture the environment of robot manipulator in [83]. They calibrate the depth parameters of the Kinect and the the position of the camera in relation to the robot. For the calibration of camera position they use a marker attached to the end effector of the manipulator, move the manipulator to different poses and use an iterative least squares approach to solve for the transformation. Then they filter the robot from the point cloud generated by transforming the robot to camera coordinates and removing the points within its bounding volume.

Table 1: Parameters of the Kinect cameras according to [82].

Camera	IR	RGB
Sensor resolution	1280×1024	1280×1024
Field of view	$57° \times 45°$	$57° \times 45°$
Focal length	6.1 mm	2.9 mm
	585.6 px	524 px
Pixel size	5.2 µm	2.8 µm
Principal point x	316 px	316.7 px
Principal point y	247.6 px	238.5 px
Distortion coefficients		
k_1	-0.1296	0.2402
k_2	0.45	-0.6861
k_3	-0.0005	-0.0015
k_4	-0.002	0.0003
k_5	—	—

3.3 High-level design

The sensor-based motion planning system consists of several subsystems. The subsystems and the information flow between them are illustrated in Figure 10. Software and hardware tools that are used to implement the subsystems listed in parentheses inside the block representing each subsystem. The arrows represent the information flow between the subsystems. The arrows are color coded to differentiate them by function.

The red flow describes how the environment representation is built by capturing a depth image, converting it to a point cloud, removing the points belonging to the robot, integrating the point cloud into an occupancy grid map which is then converted into a heightmap for collision detection.

The green flow is the state validity checking. The planning algorithm uses a callback function which converts the joint space description of a state into a set of geometric primitives using the direct kinematics and the bounding volume model of the robot. The collisions between these primitives and the heightmap are determined and a boolean value returned to the planner.

The purple flow is information about the current state of the robot which is required by the path planner as a starting state and by the self-filter that removes points belonging to the robot from the point cloud.

The black flow is the movement command originating from the user in the form of target end effector position. This end effector position is converted into a set of goal states in joint space by the inverse kinematic model of the robot. Finally the motion planner produces a path bringing the robot from the initial state to one of the goal states. This trajectory is then sent to the robot.

The blue flows are information that is sent to the user interface for visualization. The visualized pieces of information are the point clouds of the robot and the world from self-filter, the produced heightmap, the current state of the robot, the inverse kinematics solutions corresponding to the selected target and the solution trajectory.

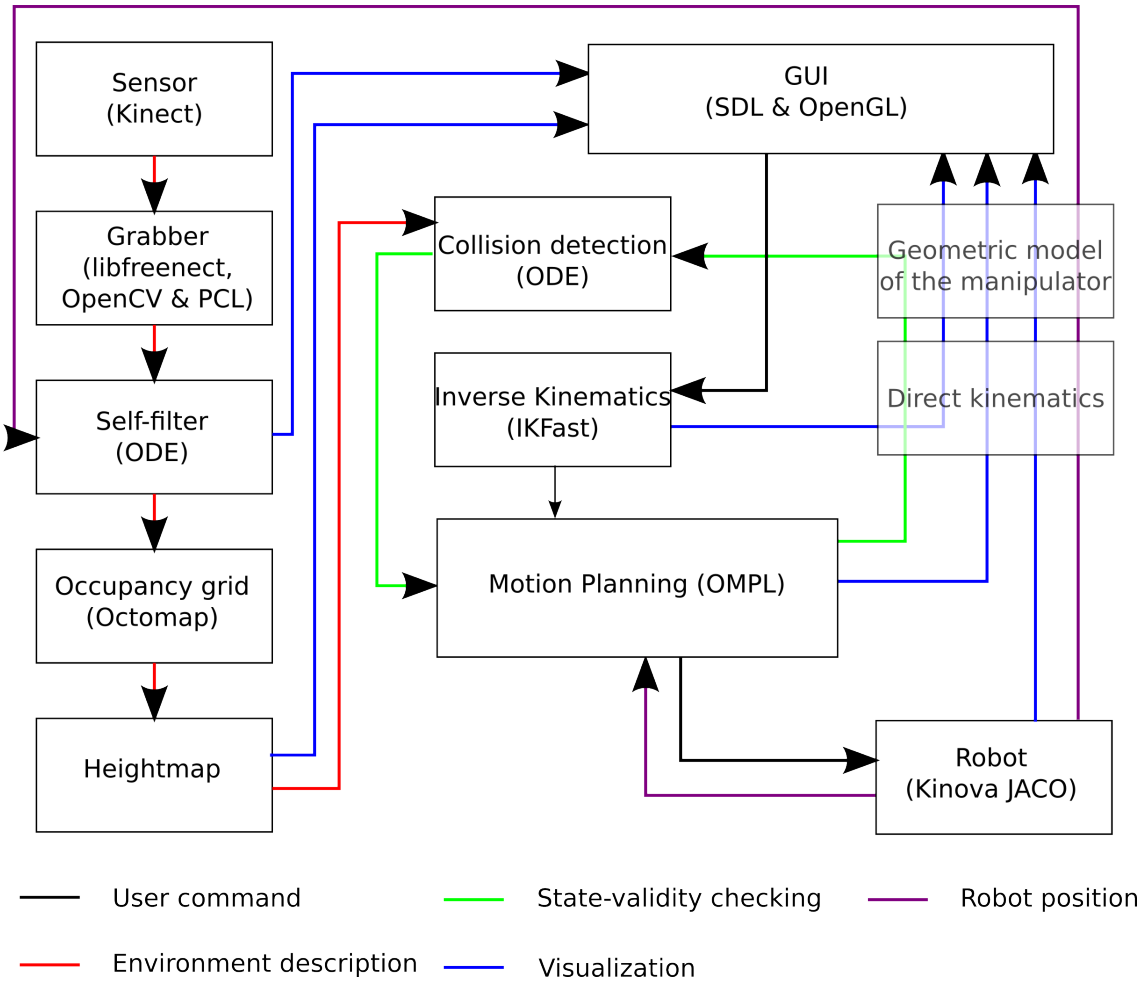


Figure 10: Overview of the sensor-based motion planning system. The boxes represent different subsystems and the arrows the flow of information between them. In parentheses are the tools used to implement a subsystem. The geometric model an direct kinematics are used to process many information streams and are presented as transparent boxes with the streams passing through them.

4 Software libraries

Several open-source libraries were used in implementing the motion planning system. In this section the chosen libraries are presented.

4.1 Motion planning

The Open Motion Planning Library (OMPL) [84] was selected for implementing the motion planning functionality. The advantages of the library include its clean and well documented API, continuing development, and a variety of different algorithms available as almost drop-in replacements.The library also provides the RRT* and BT-RRT* algorithms which were considered the main candidates for planning algorithm in this work because of their single-query nature and asymptotic optimality.

OMPL is an open-source library providing C++ implementations of many sampling-based motion planners. Compared to other motion planning software such as OpenRAVE, OMPL has a relatively tight focus in that it does not provide features such as collision checking or visualization, but rather implements motion planning algorithms using an abstract interface which makes it easy to integrate it with other software packages. OMPL's API is designed so that different components map clearly concepts in sampling based motion planning. It does not include a representation of work spaces or of robots and is therefore applicable to much wider variety of motion planning problems than just those arising in robotics. The library is designed in a way that facilitates contributions from other researchers and makes it easier to compare different motion planning algorithms with each other. Sharing of low level data structures and subroutines between different planners makes it possible to measure the difference in high-level algorithms.

The library includes planners for both geometric and control based planning. If an algorithm is capable of both kinds of planning, two implementations are provided. This is to avoid forcing the implementations to be so generic that they become needlessly complicated or inefficient. Implemented geometric planning algorithms include KPIECE, bidirectional KPIECE, lazy bidirectional KPIECE, RRT, RRT-connect, lazy RRT, SBL, EST and PRM. Control based planners include KPIECE, SyCLoP, EST and RRT. Asymptotically optimal RRT* and BT-RRT* algorithms have also been contributed to the library [85]. In addition to the motion planning algorithms, path smoothing methods based on shortcutting and B-splines are available in the library as well as numerical inverse kinematics methods using hill climbing or genetic algorithms [85].

Core concepts for geometric planning in OMPL include state spaces, samplers, state validity checkers and a goal representation. The state space is a generic representation of the space in which the planning happens. For example, state space of a robot manipulator can be the space formed by the angles of its joints. These angles can be represented in OMPL as SO(2) state spaces (the space of rotations in a plane) if the joints can rotate freely or as RealVectorSpace if the joint has limits. These component state spaces can be combined to form a compound state space. Instances of CompoundStateSpace can be constructed at run-time, which

allows the generation of state spaces from input files describing the robot. The state space encapsulates functionality such as distance evaluation, equality checking and interpolation between its member states. It is also possible to define projections to Euclidean spaces. These projections can be used by planner algorithms such as KPIECE to keep track of which areas of the state space have been sufficiently explored.

Sampling forms the core of sampling-based motion planning algorithms. The OMPL represents four kinds of samplers. Samplers which sample the state space uniformly, valid state samplers which generate only valid samples, control samplers and directed control samplers. The last two are related to generating controls for control based planning. Because the sampling of the state spaces needs to take into account the structure of the space, the state space samplers are implemented as a part of the space.

The state validity checker determines whether a state of the state space is valid or not. This often means some kind of collision checking, but any kind of criteria on the states may be used. For example, the pose of the end effector may be required to be upright for carrying liquids. For validating movements, a motion validator is used. The default motion validator simply performs discrete collision checking by calling the state validity checker at interpolated sample points on the trajectory, but more advanced schemes such as continuous collision detection can be used by implementing them as motion validators.

The goals can be described in many ways, different representations forming a hierarchy from generic to specific. At the most generic level, a goal is described by a simple routine that returns a boolean value telling whether a given state is a goal state. While this can be used to represent almost any kind of goal, most motion planning algorithms can benefit from extra information regarding the nature of the goal. At the next level, the subroutine can return also a heuristic describing the distance to a goal state from the given state. A goal region is a goal representation that has a distance evaluation function and a limit value so that all distances less than the limit are considered part of the goal region. These representations allow to check if a given state satisfies the goal condition, but they do not provide a method for producing states that do. For this, a samplable goal region is defined. Two special cases of samplable goal region are a goal state and a set of goal states. If the goal sampling is possible but slow a lazy goal can be used which uses a separate thread to generate the samples.

For control based planning a control space is used to represent the parametrization of controls and a state propagator is used to describe the evolution of the states given certain controls. The state propagator can be anything from a simple wrapper around a numerical integrator to a full physics engine. The OMPL has built-in support for the ODE physics engine.

The OMPL also provides a simple interface for setting up the planning problems. It only requires the specification of planning space, a state validity checker and the specification of start and goal states. The planner and its parameters, along with other details can be automatically determined. It is also possible to specify every detail manually.

4.2 Collision detection

Open Dynamics Engine (ODE) [86] is an open-source library for simulating rigid body dynamics. It is designed to be used in interactive or real-time simulation. It focuses on the stability of the simulation instead of accuracy and has implicit integrators for preventing the growth of simulation errors. It also supports the simulation of articulated models. ODE is written in C++ but has a native C interface. For this work the ODE is used for its built in collision detection system.

ODE supports the collision detection between many different primitives (Table 2). Among these primitives are many that are useful as bounding volumes of the mobile manipulator. It is also possible to detect collisions with triangular meshes if a more detailed description of the geometry is desired or create user defined object classes. There are dedicated functions to do the collision checking between each pair of primitive classes. This makes it possible for the library to take advantage of the simple implicit surface description of objects such as spheres and capsules. ODE can not do proximity queries, only collision checking is supported. However some of the primitives may be queried for the penetration depth or proximity of a given point. When a collision is detected a set of contact points is generated to support the physics model. Each of these contact points contains a position vector of the point where the contact has happened, in global coordinates, the normal of the contact surface, the penetration depth and the colliding objects. To speed up collision detection it is possible to tell the library that the quality of contact information is not that important.

Some of ODEs collision detection capabilities are implemented using libccd[88]. Libccd is a library for collision detection between two convex shapes. It implements the Gilbert-Johnson-Keerthi(GJK) algorithm[38], Expanding polytope algorithm (EPA) [89] and Minkowski Portal Refinement algorithm (MPR)[90]. From the library, ODE uses MPR algorithm to implement its collision detection[91].

ODE supports broad phase collision checking through "spaces". The spaces can form a hierarchy by containing other spaces. When a collision between objects AABBs is detected by the space, a callback function is called which can then decide to do a more detailed collision check or if one of the colliding objects is a space itself it can recursively detect the collision with that space. The ODE has functions for detecting collisions both between two spaces and within a single space. There are

Table 2: Primitives supported by ODE and the collision checks that can be done between them. The asterisk indicates that feature is available through libccd[87].

	Ray	Plane	Sphere	Box	Capsule	Cylinder	Trimesh	Convex	Heightfield
Ray	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Plane	-	No	Yes	Yes	Yes	Yes	Yes	Yes	No
Sphere	-	-	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Box	-	-	-	Yes	Yes	Yes	Yes	Yes*	Yes
Capsule	-	-	-	-	Yes	Yes*	Yes	Yes*	Yes
Cylinder	-	-	-	-	-	Yes*	Yes	Yes*	Yes
Trimesh	-	-	-	-	-	-	Yes	No	Yes
Convex	-	-	-	-	-	-	-	Yes	Yes
Heightfield	-	-	-	-	-	-	-	-	No

three broad phase collision checking methods available: hash spaces, quadtree spaces and simple spaces. Simple spaces merely group the geometries together without doing any collision culling. Collision checking inside a simple space requires $O(n^2)$ operations as all the objects are checked against each other.

The hash space contains multiple 3D grids of different resolution where the objects are stored [92]. These grids are stored as hash tables to conserve space. Each grid has cubical cells with side length halved at each resolution level. The correct level for storing an object is selected so that an object can never occupy more than eight cells regardless of its placement. So the used cell size is given by

$$\frac{l_{\text{cell}}}{2} < q \leq l_{\text{cell}}, \tag{10}$$

where l_{cell} is the side length of the cell and q is the maximum dimension of the AABB of the object. When doing the collision detection for the objects, candidates for collision are found within the cells occupied by the object in the grid and also from intersecting higher level cells. The time required to do collision checking for n objects in hash space is $O(n)$ as long as the objects are not clustered too tightly together. However, there is a maximum and a minimum level for the resolution. Objects that are too large for the maximum level are stored as a simple list and intersected with all other objects. Because the grid cells are cubical, a space containing lots of long, thin objects may thus degenerate into a simple space with the added overhead of maintaining the hash tables.

The quad tree space is especially effective for large amounts of objects in a landscape-shaped world [86]. The objects are stored in a quad tree structure that partitions the space along two dimensions only. The objects are stored at the node that contains the AABB of the object. The collision checking of an object requires examining the objects stored at the node containing objects AABB and its parents and children. When constructing a space its extents and maximum resolution need to be specified as ODEs implementation of quad tree space preallocates the storage for the tree at construction time and the structure can not grow [92]. If an objects AABB lies outside the AABB of the root node, it is stored at the root none the less.

ODE has been used in many projects, such as Robot Operating System (ROS)[93] to provide collision checking capabilities. Recently, a new library called Flexible Collision Detection Library (FCL) was developed to replace it [41]. The time required by the libraries for collision checking a robot against environment consisting of different types of obstacles was evaluated, but in that test neither of the libraries was consistently faster for all classes of obstacles. The FCL library is much more versatile and extensible than ODE, being capable of handling different queries such as discrete collision detection, continuous collision detection, separation distance computation and penetration depth estimation on wide variety of different primitives, including collision checking with point clouds. Given these advantages, FCL was the first choice evaluated for this work. However, being a relatively new library, it has not yet reached the same level of documentation and stability as ODE.

4.3 Point cloud capture

PCL[94] is an open-source C++ library for 3D point cloud processing. It provides a wide variety of algorithms that operate on point cloud data. Each set of algorithms is defined via base classes that attempt to incorporate the common functionality. Single algorithms or processing steps can be chained to form a pipeline or a processing graph.

The point cloud library is further split into smaller software libraries.

1. *pcl_filters* contains filters such as downsampling and outlier removal.
2. *pcl_features* implements 3D features such as surface normals and curvatures, integral images, scale-invariant feature transform (SIFT) descriptor, etc.
3. *pcl_io* implements io operations such as reading and writing PCD data files. It also contains a data grabber for Kinect [95]
4. *pcl_segmentation* implements cluster extraction, model fitting, etc.
5. *pcl_surface* implements operations such as surface reconstruction and meshing.
6. *pcl_registration* implements point cloud registration methods
7. *pcl_keypoints* implements keypoint extraction methods
8. *pcl_range_image* supports generating range images from point cloud data sets
9. *pcl_visualization* is a visualization library for quickly prototyping and visualizing the results of different algorithms. It is based on the visualization toolkit (VTK) library[96].

In this work PCL was mainly used to provide a grabber for Kinect and a container data structure for point clouds and to segment a plane that is used in calibration of Kinect's depth parameters. The PCD file reading and writing capabilities and visualization library were utilized for debugging purposes.

The OpenNI grabber included in PCL was initially used to capture the point cloud from Kinect, but it was not possible to alter the calibration parameters of the grabber. For this reason the point cloud is captured using a grabber developed in the Lappeenranta University of Technology. This grabber is based on libfreenect[78] that allows capturing of RGB and disparity images from Kinect and also provides access to the motors and the accelerometer of the device. OpenCV[97] is used to correct the lens distortions from these images before converting them to a point cloud. This point cloud is converted to a PCL data structure for compatibility with the OpenNI grabber.

4.4 Mapping

Octomap[11] is an open-source C++ library for 3D mapping that provides a map representation that is updateable, flexible and compact. It represents the environment as an octree structure. For this work Octomap provides a compact in-memory representation that allows the modelling of areas that are occluded from the sensor as obstacles and also supports incorporating multiple scans into a single 3D world model. The ability to update this representation as new sensor information arrives is valuable for operating in a dynamic environment.

An octree containing only boolean occupancy values can be efficiently compressed in memory by combining leaf nodes so that if all children of a node have the same occupancy label the node itself is made a leaf by discarding its children and storing the occupancy label in it. This helps to keep the map compact both in memory and on disk. For robotic systems which have to deal with sensory noise and changing environment such simple occupancy labels are insufficient. The map needs to be able to represent the occupancy probability and can no longer be compressed by this method.

When the environment changes, new information from sensors contradicts the information stored in the map. In a traditional log-likelihood update, as much evidence needs to be accumulated in favor of the new state as is stored in the current map. In octomap, a clamped log-likelihood update is used where upper and lower limits for the occupancy probabilities are specified that can not be exceeded. This keeps the map responsive to new evidence. Also, nodes that are at either of the limits can be combined similar to the way that was used to compress trees with occupancy labels. If new evidence then arrives which contradicts the value of the node, its children will be automatically re-generated.

In addition to this lossless in-memory compression, the Octomap library pays further attention to the memory effectiveness in its implementation by storing the child pointers of a tree node in a separate table that is only allocated if the node actually has its children initialized. The storage of maximum likelihood trees is optimized also on disk with a compact bitstream representation where each node is represented by 16 bits that contains a 2 bit label for each of its children. This label describes the nodes as either free, occupied, unknown or inner node. The leaves of the tree need not be stored as they can be reconstructed from this representation.

4.5 Kinematics

Open Robotics Automation Virtual Environment (OpenRAVE) [98] provides an environment for testing, developing and deploying planning algorithms. It includes simulation, visualization, planning, scripting and control. The aim is to make it possible to apply algorithms to any scenario with little modification as robots or target objects change. The users should not have to explicitly manage details of robot kinematics or dynamics, collision detection, world updates and robot control. In this work a single module of OpenRAVE, the IKFast kinematics compiler, was used. The visualization provided a useful debugging tool when writing the robot

description in the required *robot.xml* format. OpenRAVE also supports files in OpenCOLLADA format[99].

The inverse kinematics of a robot may be solved either analytically or with numerical methods. The numerical solution is often easier for complex mechanisms, but the closed-form analytical solution is preferred, if possible, because of two reasons: Closed-form code is faster than the numeric solution of inverse kinematics and it allows all different solutions to be calculated. However, creating hand optimized inverse kinematics code that takes into account all the degenerate cases is not trivial. The IKFast module is a Python program that analytically solves the inverse kinematics equations and generates a C++ solver optimized for the problem. It is part of OpenRAVE, but the resulting C++ code is independent and the only library that may be required is *lapack*[100]. The solver can automatically detect the degenerate cases where axes of the robot align. While the IKFast compiler is in principle able to handle robots with arbitrary joint complexity like non-intersecting axes, in practice there are still mechanisms it can not solve 6D inverse kinematics for.

The joints of the robot are divided to *solve joints* and *free joints*. If the robot has more degrees of freedom than the solution requires the values of the free joints may be set at runtime to desired values. The inverse kinematics solver then solves the values of solve joints. The program supports generating different kinds of solvers that use different amounts of degrees-of-freedom to specify the goal:

1. *Transform6D* solves for joint configurations reaching a desired 6D transform of the end effector.

2. *Rotation3D* can be used when only the desired rotation of the end effector is specified.

3. *Translation3D* is a solver for cases where only the translation of the end effector is important.

4. *TranslationXY2D* is used when the Z part of the translation is not important.

5. *TranslationXAxisAngle4D* ,

6. *TranslationYAxisAngle4D* and

7. *TranslationZAxisAngle4D* are used when the translation and a desired angle about a single axis is specified.

8. *TranslationLocalGlobal6D* is used to solve for configurations that bring a given point in end effector coordinate system to another point that is given in manipulators base coordinate system.

9. *Direction3D* brings a direction in the end effector coordinate system to desired direction in global coordinate system.

10. *Ray4D* is used to have a ray in the end effector coordinate system coincide with a ray in global coordinate system.

11. *TranslationDirection5D* is used when in addition to Ray4D the origins of the rays must coincide.

12. *LookAt3D* has the end effector point towards a given 3D position.

5 Implementation

In this work an application was developed that allowed the planning of collision-free paths for robot manipulator. The high-level design of the application was presented in Section 3. In this section the implementation of the application is covered in more detail. The application contains two simultaneous threads of execution, one is constantly reading the measurement data from Kinect and integrating those measurements into a map of the environment while the other manages the interaction with the user and the planning and execution of paths. *The main components of the application are the interface to the robot, direct and inverse kinematic models, motion planner, collision detection, the interface to the Kinect sensor, mapping of the environment and the user interface.*

The goal in this work was to be able to plan a collision free path for the robot manipulator in a changing environment. For this end methods were selected such that the building and updating of the environment representation would be possible from sensor data. A path planning method that is suitable for high dimensional planning was required and because of the changing environment a single-query planner was selected. Because the RRT* provides asymptotic optimality as well as probabilistic completeness, it was preferred over the non-optimal RRT. Another interesting algorithm is the BT-RRT*, which promises faster initial planning times than the RRT*. The chosen algorithm can be easily replaced with others implemented in the OMPL library, which makes it possible to compare the suitability of different algorithms for a given task.

5.1 Robot interface

At the time of this work the library used to communicate with the robot was only available as a .NET assembly. To interface this library with the rest of the code two main approaches were considered: either to write as much of the code in C# as possible and make wrappers for the rest of the code as needed or to create a wrapper around the robot interface and write the rest of the code in C++. The latter approach was pursued. A class was written in C# that acted as a facade to the robot API and provided a simplified interface. This class was compiled as an assembly. A wrapper class was written in C++ that executed the methods of the C# class using embedded Mono runtime[101]. The wrapper is a singleton class that takes care of the initialization of the runtime, loading of assembly and initialization of the arm as well as the eventual deinitialization and freeing of resources automatically. The class provides methods for reading the arm position and sending trajectories to the arm. In addition, the arm status can be queried and the control of the arm claimed. The trajectories can be sent in either Cartesian coordinates or as a vector of joint angles. The Cartesian coordinates are converted to angular commands using the robot's built-in kinematic model. In the application developed in this thesis, only the angular control mode is used.

5.2 Kinematic model

The manipulator was modeled by determining the Denavit-Hartenberg parameters from the CAD drawing provided by the manufacturer (Figure 7). The manufacturer provided a different set of parameters but those were in conflict with the CAD drawing. By visual inspection of the physical robot and measurements done from it, it was determined that the CAD drawing corresponds more closely to reality. In the Denavit-Hartenberg convention, each link is associated with a coordinate system which are related to each other via screw displacements along the joint axis and along the common normal of successive joint axes as

$$^{i-1}T_i = Z_{i-1}X_i. \tag{11}$$

where Z_{i-1} is the transformation associated with the joint $i-1$ and X_i is the transformation associated with link i. The total transformation from end effector to base coordinate system is the product of individual transformations between the joint coordinate systems given by

$$^0T_6 = {}^0T_1\,{}^1T_2\,{}^2T_3\,{}^3T_4\,{}^4T_5\,{}^5T_6. \tag{12}$$

Each joint has an associated homogeneous transformation matrix

$$Z_i = \begin{bmatrix} \cos\theta_i & -\sin\theta_i & 0 & 0 \\ \sin\theta_i & \cos\theta_i & 0 & 0 \\ 0 & 0 & 1 & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{13}$$

that defines the displacement and rotation along the joint axis and each link has an associated homogeneous transformation matrix

$$X_i = \begin{bmatrix} 1 & 0 & 0 & a_i \\ 0 & \cos\alpha_i & -\sin\alpha_i & 0 \\ 0 & \sin\alpha_i & \cos\alpha_i & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{14}$$

that describes a displacement and rotation along the link axis.

The coordinate systems are selected so that the joint axis is always z-axis of the coordinate system and the x-axis is parallel to the common normal of the z-axes of the adjacent joints. The y-axis is chosen so that it results in a right-handed coordinate system. The Denavit-Hartenberg parameters are for each link-joint pair the offset d_i along previous z-axis (joint offset), the angle about the z-axis θ_i (joint angle), the link length a_i which is the radius about the previous z-axis and link twist α_i which is the angle between successive z-axes. If joint axes are parallel, such as the second and third joint of the JACO-arm, d_i is a free parameter. The Denavit-Hartenberg parameters for the JACO-manipulator are collected in Table 3. The curved links are treated by dividing the translation between the z-axes of the preceding joint and following joint (Figure 12). From the trigonometry presented

Table 3: The Denavit-Hartenberg parameters for JACO arm.

i	θ_i	d_i	α_i	a_i
1	0°	0.2725 m	90°	0.00 m
2	90°	0.0350 m	180°	0.41 m
3	−90°	0.0463 m	−90°	0.00 m
4	0°	0.2492 m	−55°	0.00 m
5	0°	0.0837 m	55°	0.00 m
6	0°	0.2299 m	0°	0.00 m

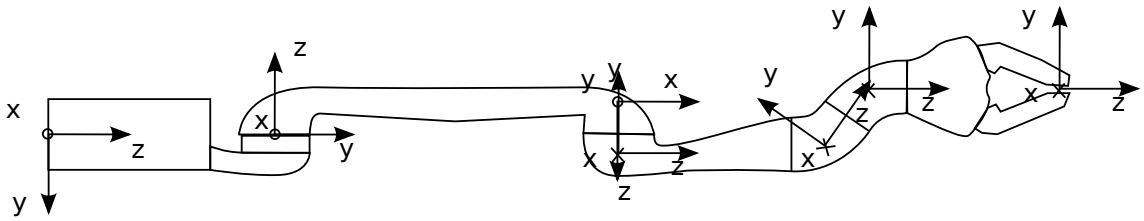


Figure 11: The placement of link coordinate systems.

in the picture, we can calculate the offset δ added to previous and successive joint offsets d_i and d_{i+1} with

$$\delta = r_l \tan \alpha_i. \tag{15}$$

Otherwise the parameters can be read straightforwardly from the measurements in CAD drawing. The free parameter d_i for the parallel axis case is selected so that when the bounding volume capsule of the link is transformed using the joint transform the points belonging to the manipulator are in the middle of the bounding volume. The coordinate systems after every link transform are presented in figure 11. The angular commands sent to JACO arm are related to this model so that $\theta_{JACO} = 360° - \theta_{model}$.

The robot's direct kinematic model was written by hand in C++ so that the positions of each individual bounding volume could be solved. The structure of JACO manipulator is such that solving the inverse kinematic model by hand would have been time consuming. Instead of solving the inverse kinematics by hand, automatic solver tool called IKFast was used. The IKfast compiler requires the description of the robot to be provided as a *robot.xml* file. The inverse kinematics compiler can generate various different inverse kinematics solvers from this description. The ready solver is a C++ file with no external dependencies. For this application an inverse kinematics solver for 6D transform (x, y, z, pan, tilt and roll) was compiled. The produced inverse kinematics code was then tested against the hand crafted direct kinematics to ensure that it functioned correctly.

5.3 Collision detection

The collision detection can be divided into self-collision detection between the links of the robot and collision detection between the robot and the world. The motion

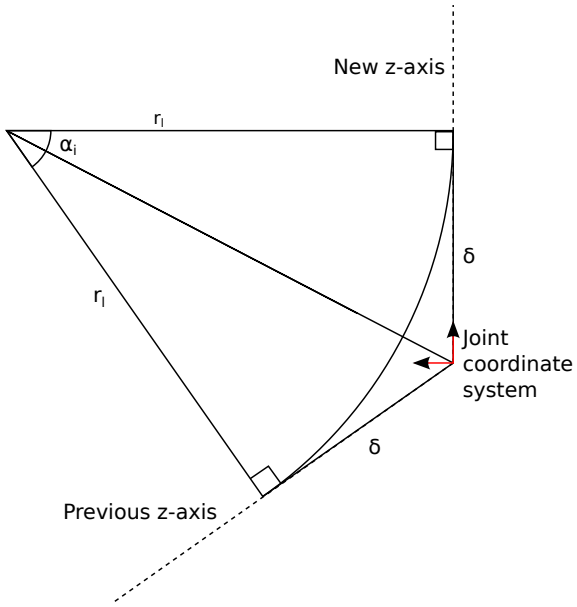


Figure 12: The treatment of the curved links.

planner queries the collision detection code for the collision state of a configuration. The direct kinematics for each such configuration is evaluated and used to transform the links of the manipulator model accordingly. Then the manipulator model is evaluated for self-collisions as this check is rather cheap, involving only few objects. If the robot does not collide with itself collisions against the environment are checked. The self-collision checking is done only between link pairs that have a possibility of colliding and in the order of approximate likelihood of collision to gain a small speed advantage in the case where collision is detected. As the adjacent links of the manipulator are constantly touching at the joint their collision checking is implemented as simple joint limits instead of the bounding volume model. The algorithm is presented in Algorithm 1.

Capsule-shaped bounding volumes were used for modeling the manipulator. These bounding volumes have many desirable features. They are easy to transform as the robot moves, relatively memory efficient and the collision checking is computationally cheap. They also resemble the shape of the links of the JACO -manipulator, which allows them to fit tightly around the actual geometry. As grabbing the target object autonomously was not part of the objective, the arm can maintain a certain distance to the detected objects. In fact some extra safety margin may even be preferred. A detailed model is therefore not required as long as the collision checking is pessimistic, that is it never misses a collision but may detect collisions even when none exist. The dimensions of the capsules were set using Kinect data so that none of the Kinect's measurement points corresponding to the manipulator fall outside the manipulator bounding volume.

ODE provides broad-phase collision checking through *spaces* which are objects that contain other objects. There are two algorithms that can be used. One is based on hierarchical grid that is stored as a hash table and the other is based on

Algorithm 1: Collision checking of a configuration of the robot.

```
input  : A robot configuration and a world model
output : Is the robot in collision?

Function collision_check(configuration, world): in_collision is
    if violatesJointLimits(configuration) then
        return true
    end
    robot ← poseManipulator(configuration)
    /* The joint numbering starts at robot's base with 1 meaning
       robot's base and 6 being the bounding volume of the end
       effector.                                                  */
    if checkLinkPairCollisions(robot,1,6) then
        return true
    end
    if checkLinkPairCollisions(robot,2,6) then
        return true
    end
    if checkLinkPairCollisions(robot,2,4) then
        return true
    end
    if checkLinkPairCollisions(robot,2,5) then
        return true
    end
    if checkWorldCollisions(robot,world) then
        return true
    end
    return false
end
```

quad-trees. The quad-tree is better suited to storing landscape-shaped environments such as the heightmap whereas the hierarchical grid could be used if the objects had more even aspect ratio and were also distributed along the z-axis. In the application, boxes stored in a quad-tree space are used to represent the heightmap. Instead, the heightmap geometry provided by the library could have been used, but more control was desired over how the neighboring values are interpolated. Using custom object classes that provide an extension mechanism in ODE, the performance of collision detection between the robot and the heightmap could later be optimized.

5.4 Motion planning

The trajectory planning of the manipulator is done with the RRT* algorithm from the Open Motion Planning Library (OMPL). For the planner the manipulator is presented as a compound state space where the limited joints are presented as a real vector state space and the other joints as a SO2 state space. A state in the compound space is a combination of states of its subspaces. For each planning

Algorithm 2: A function for planning a path to the target position

input : Target position in robot's workspace
output: A path from robot's current position to the target position

Function *PlanPath(*target_position*):* solution_path **is**
    IkSolutions ← `SolveIK(target_position)`
    if IkSolutions ∈ ∅ **then**
        return
    end
    Start ← `GetRobotPose()`
    while $(t_{\text{used}} < t_{\min}) \vee (t_{\text{used}} < t_{\max} \wedge \text{solution} \in \emptyset)$ **do**
        solution_path ← `RunPlanner` (Start,IkSolutions)
    end
    if *post processing enabled* **then**
        solution_path ← `PostProcess` (solution_path)
    end
    `SetUIState(`*visualize solution*`)`
    return solution_path
end

problem, the planner is initialized with the starting state and some definition of the set of goal states. The planner is then run until a termination condition becomes true. The process that handles the path planning is described in Algorithm 2.

The planning algorithms have various configurable parameters. Their meaning was documented at the header file of BT-RRT* implementation of the OMPL library version 0.12.2. Two of the parameters *ball_ radius_ constant* and *max_ ball_ radius* affect the radius of the neighborhood where the RRT* and BT-RRT* search for samples when attempting to rewire the tree. Ideally *max_ ball_ radius* should be set to the length of the longest straight line from initial state to any other state in the state space. According to the comments, the *ball_ radius_ constant* should be set to at least the side length of the bounding box encompassing the state space. In addition to setting these values, the *delay_ cc* was enabled. This parameter allows the planners to delay collision checking, improving the performance in cases where collision checking is expensive. Other parameters were left to their default values. The *goal_ bias* parameter determines the probability of selecting one of the goal states as the target state instead of a random state. It has a default value of 0.05 and in the documentation it is noted that the optimal value is usually close to this. The *range* parameter sets the maximum length of a motion to be added to a tree. It is the only parameter used by RRT-Connect. The value of this parameter can be automatically selected as constant fraction of the extent of the configuration space. The planner parameters that were used have been summarized in Table 4

As there is no single correct metric for measuring the distance between two robot configurations, two candidate metrics were designed and evaluated. The first metric (from here on called the *s-metric*) approximates the potential end effector movement in Cartesian space caused by moving between the configurations. The

Table 4: Planner parameters

Parameter	Value
delay_cc	1
max_ball_radius	15.39
ball_radius_constant	6.283
goal_bias	0.05
range(t-metric)	3.5813
range(s-metric)	2.7261

Table 5: Joint weights used for the metrics

Joint #	t-metric	s-metric
1	0.2618	1.01400
2	0.2618	0.95380
3	0.2618	0.54380
4	0.2618	0.33650
5	0.2618	0.26225
6	0.2618	0.18800

approximation is like a worst-case scenario. Every joint is assumed to cause an end effector movement that can be obtained by multiplying the difference in joint angles by the length of the rest of the arm. This corresponds to a situation where the arm is extended straight to the side. The cumulative sums of individual link lengths starting from the end effector are used to weight the individual subspaces in the compound space (Table 5). Thus, the total end effector movement is assumed to be the sum of movements caused by the individual joint movements, which is the upper limit on end effector movement as given by the triangle inequality. The curved links are estimated as having a length equal to the length of the chord of the curve.

For the second metric, a subclass was inherited from the compound space. This subspace overrides the method used to combine individual subspace distances. Instead of L_1-norm L_∞-norm is used, where the total distance is the maximum absolute distance between the corresponding substates. For this space, the inverse angular velocities of the individual joints were used as weights. Thus this metric (*t-metric*) corresponds to the execution time of the movement between the two configurations, assuming all the joints move simultaneously at their maximum velocities. In the JACO robot manipulator, there are many different velocity limitations, some for the angular velocities, some for linear. In this case the joint velocity limit limits the velocities of all of the joints. Thus, all the joint weights are equal for this metric (Table 5).

The description of goal state can be given implicitly as a decision rule, which when evaluated for a state tells whether it belongs to a goal region. Initially the goal was implemented by computing the direct kinematics for a candidate state and determining whether the end effector point was within some tolerance of the target. This implementation required no inverse kinematic model, but the planner

was unable to bias the tree growth toward the target and so the algorithm took a long time to find a solution. The algorithm was reimplemented so that the inverse kinematics is first solved using IKFast generated solver and all the inverse kinematics solutions are added as goal states.

The probabilistically complete planning algorithms are guaranteed to find the solution, if one exists, with probability approaching one as time approaches infinity. Often the solution can be found in a practical amount of time. However, it is difficult to determine exactly how much time the algorithm is going to need. The asymptotically optimal algorithms are also capable of continuing the planning after the initial solution has been found. The extra time is used to improve the solution path, with path cost approaching the optimum as time goes to infinity. Whether the cost of the current path is optimal or could still be improved and by how much are again difficult questions to answer. In OMPL, a planner termination condition determines when the planner is terminated, whether it has found an answer or not. In the application developed, a time limit was used as the termination condition. In addition to this upper limit, a lower limit on time usage was set so that even if the planner were to find the initial solution faster, the rest of this minimum time would be used to improve the solution.

Before being sent to the robot for execution the trajectory can optionally be post-processed. This post-processing is a common operation to improve the quality of the paths generated by sampling based planners. Particularly the algorithms that are not asymptotically optimal often produce jagged paths and are unable to improve the initial paths. The post-processing operation attempts to shorten the path by making shortcuts, that attempt to connect two points on the path directly with a straight line. The post processing can also involve smoothing the path using Bézier splines. OMPL uses both methods as indicated in Algorithm 3.

Algorithm 3: Path post-processing

input : A path from planner
output : A post-processed path

Function *reduceVertices(path): path* **is**
    Attempt to make shortcut between two non-consecutive vertices on path
    If successful, return path with in-between vertices removed
end

Function *collapseCloseVertices(path): path* **is**
    Attempt to make shortcut between two non-consecutive vertices on path, that are near each other
    If successful, return path with in-between vertices removed
end

Function *shortcutPath(path): path* **is**
    Attempt to shortcut path by connecting two interpolated points on the path
    If successful, return path with in-between vertices removed
end

Function *smoothBSpline(path): path* **is**
    Attempt to smooth path while keeping it valid.
end

begin
    path ← `reduceVertices(path)`
    path ← `collapseCloseVertices(path)`
    i ← 0
    while `reduceVertices()` *helps* & i ≤ 5 **do**
        path ← `reduceVertices(path)`
        i ← i + 1
    end
    path ← `shortcutPath(path)`
    i ← 0
    while `shortcutPath()` *helps* & i ≤ 5 **do**
        path ← `shortcutPath(path)`
        i ← i + 1
    end
    path ← `smoothBSpline(path)`
end

5.5 Interfacing with Kinect

The environment is captured using Kinect sensor and processed into a point cloud. This was initially done with the PCL library and the OpenNI grabber included in it. The OpenNI grabber does not support calibrating camera parameters or Kinect depth parameters. For this reason the OpenNI grabber was substituted with libfreenect[78] -based grabber that was obtained as part of the calibration code developed by Ilonen and Kyrki[102]. The processing of Kinect data into a map of the environment is represented as pseudocode in Algorithm 4.

The grabber obtains the disparity image using libfreenect, undistorts it with OpenCV [97] and then converts it to point cloud using the mapping

$$x = \frac{(x_i - c_x)z}{f_x}, \tag{16}$$

$$y = \frac{(y_i - c_y)z}{f_y}, \tag{17}$$

$$z = \frac{1}{\beta d + \gamma}, \tag{18}$$

where (x_i, y_i) are the image coordinates, (c_x, c_y) is the principal point, f_x and f_y are the focal lengths, d is the disparity and β and γ are the Kinect depth parameters.

Next the point cloud is transformed to the robot coordinate system using the transform described by the extrinsic parameters. The points belonging to the robot are then removed from the cloud. This filter first checks whether the points lie inside the robot's axis-aligned bounding box. Only if they do are they compared against the bounding volume model of the manipulator. This collision query is done using ODE. Because the robot is described by a set of implicit surfaces, this query is relatively cheap to compute.

Algorithm 4: Process that generates a map from Kinect measurements

```
while true do
    RawDepth ← GetDepthImage()
    UndistortedDepth ← UndistortDepth(RawDepth, CameraParams)
    MetricDepth ← DisparityToDepth(UndistortedDepth, DepthParams)
    PointCloud ← MakePointCloud(MetricDepth, CameraParams)
    TransformedCloud ← TransformPointCloud(PointCloud,
    CameraTransform)
    RobotPose ← GetRobotPose()
    FilteredCloud ← RemoveRobot(TransformedCloud, RobotPose)
    OccupancyGridMap ← InsertScan(OccupancyGridMap, FilteredCloud)
    HeightMap ← ConvertToHeightMap(OccupancyGridMap)
end
```

5.6 Mapping the environment

The mapping is done with Octomap library. The part of the point cloud produced by Kinect that makes it through the self-filter is integrated into a 3D occupancy grid map. To handle the occluded areas of the scene, a ray casting approach is used where the rays between a point in the cloud and the Kinect origin are considered to have a high probability of being free space and the voxel where the measurement point lies as being occupied with some probability. The measurement is then integrated into the current belief about the environment. This voxel model of the world is stored as an octree structure.

The voxel representation is converted to a heightmap where the unknown areas are considered as impassable by the manipulator. The unknown areas directly around the robot are marked as passable to prevent the area occluded by the robot itself from causing false collisions. The algorithm that was used is presented as Algorithm 5. The processing is done simply by iterating through the voxels in each column and recording the height where the tallest obstacle is found. If a voxel of unknown space does not have a voxel with free space below it, it is treated as obstacle. Obstacles that are high enough that their top can not be seen are assumed to be higher than the manipulator so that the manipulator would not try to go over them. The heightmap is represented as a collection of boxes in the Open Dynamic Engine(ODE) collision space. Compared to the voxel based representation the heightmap requires less collision primitives, but can not represent overhanging structures. Also the process of building the heightmap has some computational overhead and is not suited to mapping of larger environments as is.

Algorithm 5: Conversion of voxel map to heightmap

input : The voxel map, desired resolution for the map, manipulator position, amount of clear space around manipulators base, a probability limit for considering cell occupied

output : A heightmap describing the robot's environment

```
Function
ConvertToHeightMap( GridMap,res,manipulator,BaseClearance,P_limit):
Heightmap is
    [Xmin, Xmax, Ymin, Ymax, Zmin, Zmax ] ← Dimensions(GridMap)
    Imax ← ⌊(Xmax − Xmin)/res ⌋
    Jmax ← ⌊(Ymax − Ymin)/res ⌋
    Kmax ← ⌊(Zmax − Zmin)/res ⌋

    for i ← 0 to Imax −1 do
        for j ← 0 to Jmax −1 do
            free[i,j] ← false
            shadow[i,j] ← true
            for k ← Imax −1 . . . 0 do
                x ← (i *res)+Xmin
                y ← (j *res)+Ymin
                z ← (k *res)+Zmin
                P ← QueryMap(GridMap, x, y, z)
                if P ∉ ∅∧P < P_limit then
                    free[i,j] ← true
                    shadow[i,j] ← false
                end
                if free[i,j] then
                    height[i,j] ← z
                    if P ∈ ∅ then
                        shadow[i,j] ← true
                        break
                    end
                end
                if P ∉ ∅∧P ≥ P_limit then
                    shadow[i,j] ← false
                    break
                end
            end
        end
    end
```

Continued...

Algorithm 6: Conversion of voxel map to heightmap continues

Continued...

```
    Heightmap ← ∅
    for i ← 0 to Imax −1 do
        for j ← 0 to Jmax −1 do
            /* Do not add columns with no measurements, unless they
               have neighbor with measurements                    */
            ShouldAdd ←free[i,j]∨(i > 0∧free[i-1,j]) ∨ (j > 0∧free[i,j-1]) ∨ ((i
            +1) <Imax ∧free[i +1,j]) ∨ ((j +1) <Jmax ∧free[i,j +1])
            if ShouldAdd then
                x ← (i *res)+Xmin
                y ← (j *res)+Ymin
                z ← height[i,j]
                if length([x, y])>BaseClearance ∨¬shadow[i,j] then
                    repeat
                        Box ← MakeBox(x,y,z,res,Zmin)
                        z ← z − res
                    until ¬InCollision(manipulator,Box);
                    Heightmap ← Heightmap ∪{ Box }
                end
            end
        end
    end
    return Heightmap
end
```

5.7 The user interface

The user interface (Figure 13) was implemented using Simple DirectMedia Layer (SDL)[103] and OpenGL[104]. The user interacts with the application by a set of commands invoked from the keyboard and can point a target location from the screen using a mouse. The pointing is done on a map reconstructed from measurements. Another option would have been to use the camera image provided by Kinect. The chosen approach has the advantage that points can be selected even if they are currently occluded by the robot. The interface contains a 3D-model of the robot which is the same model that is used for collision checking rendered as a triangle mesh. This allows the visual verification of the collision status of the manipulator. In the default mode, the pose of the 3D-model reflects the angles measured from the physical robot arm in real time. The model can also be used to examine a solution path before executing it, in which case the model repeatedly performs the planned motion or to examine the possible inverse kinematic solutions of the goal position, in which case the model cycles through the possible solution poses. The visualization can also show either the current map or the point cloud acquired from the Kinect. In the point cloud view, the manipulator model is shown as translucent and the measurement points falling inside the model are separated from the environment by color. This allows the verification of the extrinsic calibration parameters and the radii of the bounding volumes.

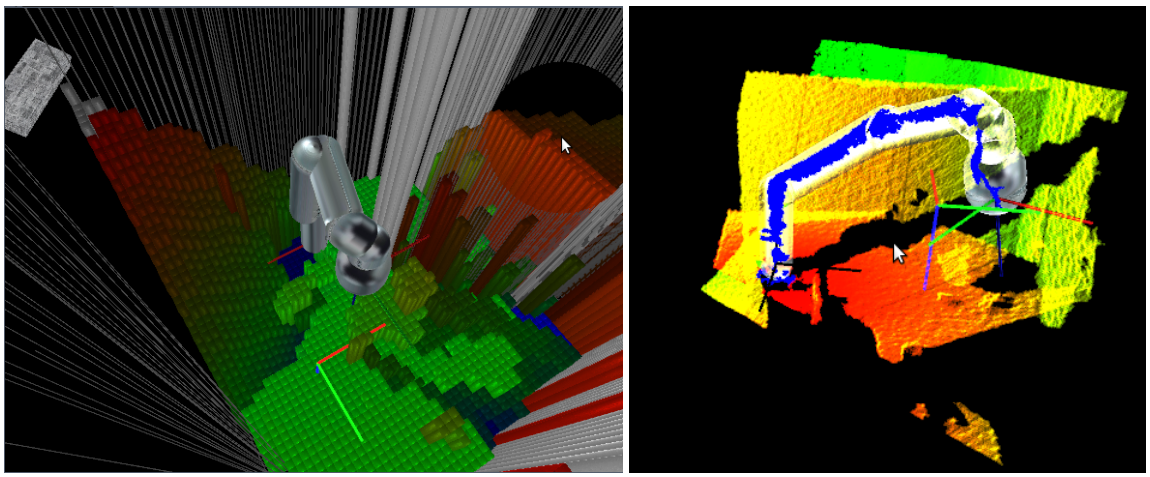


Figure 13: User interface. *left:* Visualization of the manipulator, heightmap and target. Color indicates height, unknown areas are rendered in darker color or in wireframe if entire column is unknown. *right:* Visualization of the point cloud. Points colored blue are considered to lie within the manipulator model and filtered away.

6 Experiments and results

Before the motion planning system can be used and the experiments performed, the system needs to be calibrated. The calibrated parameters are the parameters of the Kinect sensor and its position in the base coordinate system of the robot arm. This section begins with description of the calibration procedure.

The accuracy of the motion planning system is investigated. The accuracy of the whole system from given target position to reached goal position is measured in the xy-plane. This accuracy is mostly affected by the accuracy of the robot and its kinematic model. To investigate the cause of errors, the accuracy and repeatability of the movements of the robot arm in joint space are measured. The accuracy of the calibrated Kinect sensor is also evaluated. The accuracy of sensor and robot form a limit to to the accuracy of reaching the target and to the amount of clearance that should be kept around the robot in order to avoid collisions.

Experiments were performed to study the performance of the planning algorithms used. Three different planning algorithms (RRT-Connect, RRT* and BT-RRT*) were compared using two different configuration space metrics. This experiment is intended to help in selecting a suitable algorithm for the implemented system and to investigate what effect the choice of configuration space metric has on the planning algorithms in terms of planning times and the quality of the produced paths. In order to be suitable for this application the planner has to produce solution reasonably fast so that the application remains responsive, but the speed of the manipulator determines what is the ideal trade-off between path quality and solution time. The benefit of giving the algorithms extra time to improve solutions is evaluated and the need for path post-processing is investigated.

The two configuration space metrics that were developed are intended to be related to the end effector movement and execution time respectively. The actual relationships between the metrics and their related quantities are investigated. In the case of the t-metric, comparison with actual execution times is performed. The s-metric is compared to end effector movement using simulated kinematics.

6.1 Calibration

Before the Kinect can be used for measurement it needs to be calibrated. The calibration is divided into determining the intrinsic and the extrinsic camera parameters. In addition, two parameters are associated with mapping from disparity image to depth image (Equation 18). The intrinsic and depth parameters remain relatively constant and thus only need to be calibrated once. On the other hand, the extrinsic parameters express the pose of the camera relative to the robot's coordinate frame and need to be recalibrated each time there is a possibility that the camera has moved relative to the robot.

The intrinsic parameters of the infrared camera were calibrated by taking pictures of a camera calibration pattern from various directions (Figure 14). In order to detect the pattern the infrared projector was covered with a piece of paper and a halogen lamp used as an external infrared source. Matlab Camera Calibration Toolbox [81]

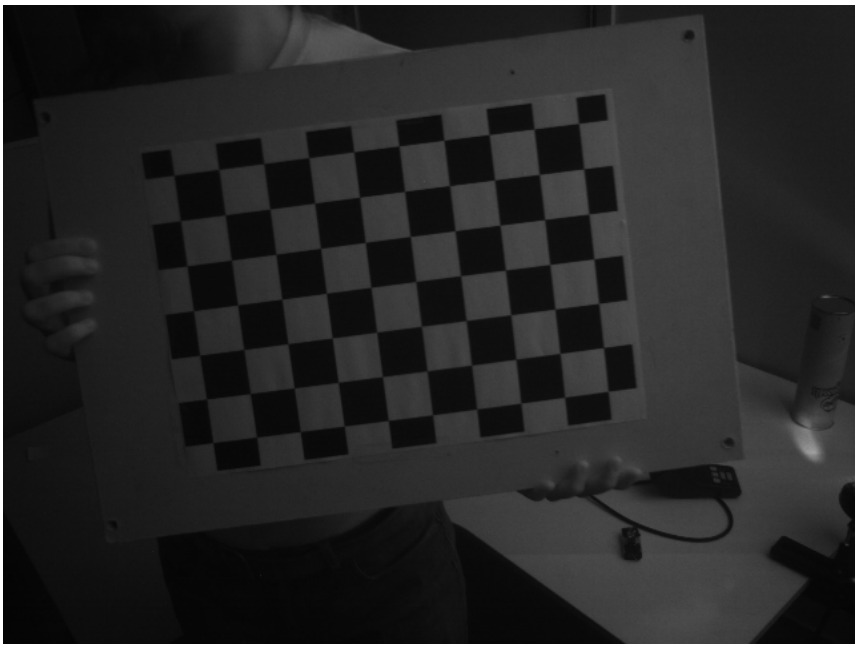


Figure 14: Calibration pattern as seen by the Kinect IR camera.

was used to calculate the intrinsic parameters based on the captured images. The tangential distortion was found to be small and it was left out from the model. The pixels of the camera were assumed to be square. The estimated parameters were the focal length, principal point and the three radial distortion parameters. With these modifications the Equations 7 and 8 simplify to

$$\begin{bmatrix} x_p \\ y_p \\ 1 \end{bmatrix} = \begin{bmatrix} f & 0 & c_x \\ 0 & f & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_d \\ y_d \\ 1 \end{bmatrix} \tag{19}$$

and

$$\begin{bmatrix} x_d \\ y_d \\ 1 \end{bmatrix} = (1 + k_1 r^2 + k_2 r^4 + k_5 r^6) \begin{bmatrix} x_n \\ y_n \\ 0 \end{bmatrix}, \text{where } r^2 = x_n^2 + y_n^2. \tag{20}$$

The parameters in Table 6 were estimated based on 147 images of the calibration grid in various poses. Their values are similar to ones determined by Smisek et al. [82] (Table 1), but still different.

The Kinect depth parameters were calibrated by placing the Kinect sensor so that it is facing a wall. The sensor was placed at different distances from the wall and RANSAC algorithm from the PCL library was used to fit a plane to the point cloud captured at those positions. Initial values for the depth parameters corresponding to another Kinect were used for the conversion of disparities. From the equation

$$c_1 x + c_2 y + c_3 z + c_4 = 0 \tag{21}$$

describing the plane, its distance d_n to the origin, that is the sensor, can be easily

Table 6: Estimated intrinsic parameters for the Kinect IR camera

Parameter	Value	Uncertainty
Focal length	588.79405	0.63828
Principal point x	316.89830	0.28803
Principal point y	254.56166	0.33633
k_1	-0.13173	0.00352
k_2	0.52141	0.01861
k_3	—	—
k_4	—	—
k_5	-0.66806	0.02961

computed using

$$d_n = \frac{|c_4|}{\sqrt{c_1^2 + c_2^2 + c_3^2}}. \tag{22}$$

Measuring the distance in the direction of the normal of the wall allows to ignore small errors in the orientation of the sensor. The method also averages the distance estimates that would be given by individual points in the image, thus leading to improved precision. An average of the distances measured in 20 successive frames was used to produce one distance measurement. The whole series of distance measurements was repeated three times.

A line $s_{\text{kinect}} = ks_{\text{ref}} + b$ was fitted to the observations, with the distance measured with tape measure as s_{ref} and the distance measured with Kinect as described above as s_{kinect}. It was originally assumed that the slope of the line would be close enough to 1, but this turned out not to be the case. The estimated parameters were $k = 1.0150 \pm 0.000088$ and $b = -0.0258 \pm 0.0001$ (with $\alpha = 0.05$). The coefficient of determination being $R^2 = 0.99995$. The slope $k = 1$ does not fall within the confidence interval. Both depth parameters β and γ (Equation 18) were scaled by the slope. The constant term of the regression line was ignored. By not using the constant, the measurement is independent of the exact position of the origin within Kinect sensor in relation to the tape measure measurement points. The obtained values for the depth parameters were $\beta = -0.002857, \gamma = 3.121712$.

After determining the intrinsic and depth parameters, the extrinsic parameters were calibrated using a method presented in [102]. The code that was used for calibration is based on the code developed by Ilonen and Kyrki [102]. This method aims to provide a fast, accurate and reliable means to calibrate the relative pose of camera and robot. These are all highly desirable properties as the extrinsic parameters may have to be recalibrated if the configuration changes. The calibration works by placing a marker in the manipulator's end effector. In this work, an infrared blinker LED was used as the marker. The manipulator then moves into a series of random poses and the camera takes a picture of each of these poses. The marker is located from those images where it is visible. Based on the marker location in the image and the known transformation between robot's base and end effector, the translation between the robot's base and camera and the position of the marker in

end effector coordinate system are estimated using numerical optimization. Robust M-estimators are used to reduce the effect of outliers that may be caused by, for example, the camera detecting the reflection of the marker on robot's surface. The uncertainty of calibrated parameters can be measured with the back-propagated covariance of the residual error in the positions of marker images.

For calibrating the extrinsic parameters in relation to the JACO arm, a series of 120 poses were used. These poses were selected so that they would cover a wide variety of different end effector positions and orientations while keeping the LED visible for most of the time.

6.2 Comparison of planning algorithms

6.2.1 Experiments

Different planning algorithms were tested for suitability by simulating the robot arm in two different environments. The environments were constructed on top of a table and captured using Kinect sensor in the same way as in the interactive user interface. The first environment is an uncluttered one with relatively few obstacles. The second environment (Figure 15) has some more obstacles added, but is otherwise the same. In the picture, the target is behind the box on the left, the other target is the bottle to the right. First scene is similar, but without the box and the rightmost pillar. There is an occlusion caused by the leftmost pillar and the stack of objects in the middle that is treated as obstacle. The objects added for the second scene do not have occlusions, because the area behind them was previously observed to be empty. The environments remained static during the entire test.

Each different algorithm was executed 500 times and was given 20 seconds to produce a solution. If a planner is capable of optimizing the produced path, it was given the rest of the planning time for path improvement. The maximum time available for path post-processing in cases where it was used was 2 seconds. The post-processing times were not systematically tested, but were observed to range from almost zero to around 0.3 seconds. This time is not included in the solution times used to compare algorithms.

During the test runs the RRT* planner ran into a deadlock from time to time and for that reason the data sets are shorter than for other planners. The problem is apparently due to the implementation and not a flaw in the algorithm itself.

In addition to comparing the algorithms, two metrics of the configuration space were investigated. These metrics, called s-metric and t-metric, are presented in Section 5. The t-metric aims to describe the execution time of the movement and s-metric the end effector movement caused by the motion. When these metrics are used together with a planning algorithm capable of optimizing the solutions, the resulting path should have the measured quality minimized.

Finally, the effect of path post-processing operations were investigated. The post-processing is often used on the paths generated by sampling based planning techniques to reduce the jaggedness of the paths. Methods such as shortcutting (removing some intermediate configurations from the path) and path smoothing by

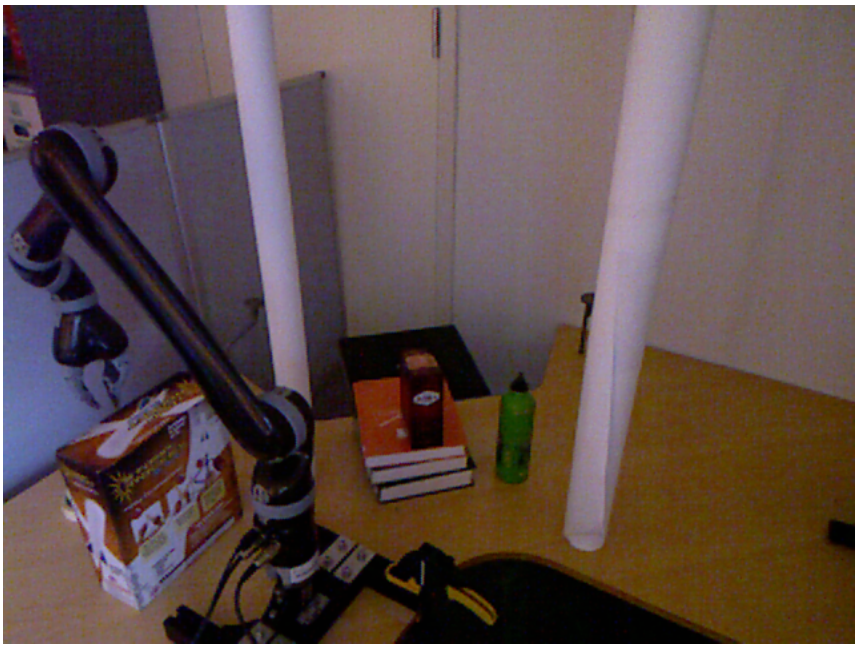


Figure 15: The second test scene as seen by the Kinect.

fitting Bézier curve to the configurations can be used. Because the asymptotically optimal variants of the RRT algorithm use extra computation time to straighten the path,the benefit of post-processing was thought to be smaller than for non-optimizing planners. Also, the goal was to determine whether the optimization performed by RRT* has additional benefit in this case when compared to merely post processing an existing, suboptimal path.

The different combinations of above variables were compared by the quality of initial paths and optimized paths and the time to produce the first solution. The path quality was measured by the execution time that was estimated in the same way that was used in the t-metric and by estimating the distance traveled by the end effector by simulating the direct kinematics of the robot arm.

6.2.2 Results

The planning algorithms were executed 500 times each with a time limit of 20 seconds. Some planners were able to consistently produce a solution within the allotted time, but others were not. The locking problems encountered with RRT* sometimes resulted in less than 500 samples. These results are summarized in Table 7.

It is evident from visual inspection of the recorded measurements that they can not be assumed to be normally distributed. For example, the distribution of planning times of BT-RRT* (Figure 16) is visibly skewed. The exact form of the distributions arising from a complex probabilistic algorithm like BT-RRT* are difficult to deduce, so non-parametric statistics was used to describe and compare the results. The comparison was done with Mann-Whitney U-test (Appendix C) with the single-tailed alternative hypothesis ($H_0 : P(X > Y) = P(Y < X)$ and $H_1 : P(X > Y) > P(X < Y)$). The test was repeated both ways, so it is effectively a two-tailed test and p-values corresponding to two-tailed tests were used.

The conclusions presented below can be drawn with significance level better than or equal to $\alpha = 0.01$ except where otherwise noted. The full list of tested hypotheses and their associated p-values are provided in Appendix B. Figures in Appendix A present comparison of different algorithms in a visual format.

Different metrics of the configuration space seem to be suited to different planners. RRT-Connect works better with the s-metric. It is both faster and produces better paths. BT-RRT* is both faster and produces better solutions with t-metric. In scene 1, the difference is only observed in optimized paths and in the execution time of the unoptimized solutions when post-processing is not used. In scene 2, the difference is also observed in the end effector movement of the initial solutions prior to post processing. Post processing removes the difference in the quality of the initial paths and actually turn the situation around for execution time so that the s-metric produces post-processed initial solution with smaller execution time.

The RRT* is a more complicated case. For scene 1, t-metric results in smaller execution times of the solution paths, but no difference in end effector movement or planning times can be noticed. In scene 2, the t-metric produces consistently better paths, but also requires longer time for the initial solution to be found.

Table 7: Number of sample paths in each test. Valid sample paths are those that were found within the time limit. Sample sizes for RRT* are smaller because of locking problems

Scene	Algorithm	Samples	Valid
1	BTRRT* t	500	496
1	BTRRT* t pp	500	497
1	BTRRT* s	500	464
1	BTRRT* s pp	500	455
1	BTRRT* s pp	500	455
1	RRT* t	500	497
1	RRT* t pp	500	497
1	RRT* s	500	500
1	RRT* s pp	500	500
1	RRTC t	500	500
1	RRTC t pp	500	500
1	RRTC s	500	500
1	RRTC s pp	500	500
2	BTRRT* t	500	499
2	BTRRT* t pp	500	498
2	BTRRT* s	500	497
2	BTRRT* s pp	500	497
2	RRT* t	267	263
2	RRT* t pp	428	418
2	RRT* s	69	69
2	RRT* s pp	404	404
2	RRTC t	500	500
2	RRTC t pp	500	500
2	RRTC s	500	500
2	RRTC s pp	500	500

RRT-Connect is the fastest planner to produce the initial solution. However the quality of the initial solution produced is not as good as the optimized solution of the other planners. When using the s-metric the initial paths produced by RRT Connect are sometimes better than the initial paths produced by other planners. In scene 2, the quality of the initial paths is as good as the other algorithms when post processing is not used and even better than the other algorithms when post processing is used. In scene 1, the initial post-processed paths produced by RRT-Connect have shorter execution time than those produced by BT-RRT*. When t-metric is used, the paths produced by RRT-Connect are not as good as those produced by the other algorithms, except for end effector movement in scene 1 which is similar for all three algorithms.

RRT* is faster than BT-RRT*. The optimized paths produced by RRT* tend to be of better quality. In scene 1, this is true except for the end effector movement when using t-metric. In scene 2, BT-RRT* produces better paths than RRT* when

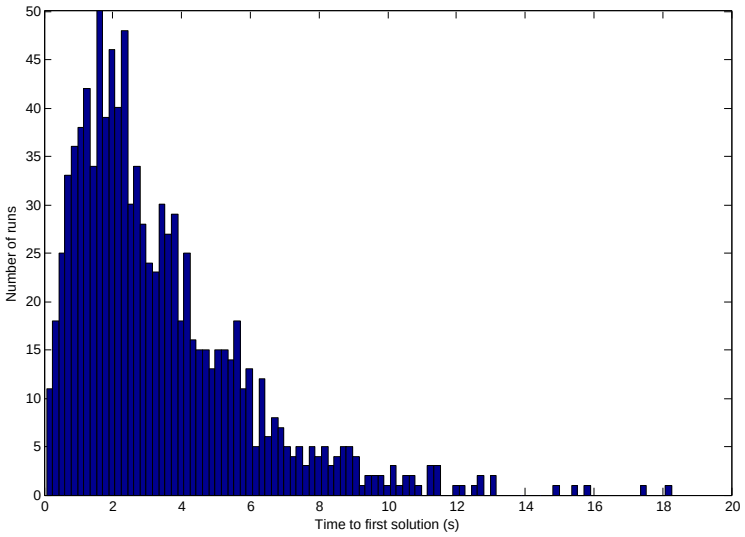


Figure 16: The recorded planning times times of the BT-RRT* algorithm using the s-metric.

using the s-metric without post-processing ($\alpha = 0.02$ for EE movement), but after post-processing RRT* performs better. For the initial paths, similar pattern holds. When using the t-metric the RRT* produces better paths in both scenes except that there is no noticeable difference in end effector movement after post processing and in scene 1 post processing also eliminates the difference in execution times. When using s-metric, in scene 1 the RRT* still performs better except for end effector movement after post processing. In scene 2, the BT-RRT* produces better solutions, but the difference disappears when post processing is applied.

Visual inspection of the distribution of planning times of various algorithms (Appendix A, Figures 24 and 24). It can be noticed that the planning times of RRT-Connect do not have as heavy right tail as the distribution of planning times of other planners. This means that the planning times of the RRT-Connect are more predictable. On the other hand, the variance in execution times is larger.

The post-processing reduces both execution time and end effector movement for both scenes and all planners and metrics. This is true for both the initial and the optimized paths.

Whether the optimizing planners should be used depends on the environment and on the manipulator. The extra time that is used in planning should be compensated by the shorter execution time. In scene 2, the difference in execution time between optimized solutions of the RRT* (with t-metric and post processing) and the solutions of RRT-Connect (with s-metric and post processing) is 6.45 s in the median case. In this case it is probably not worth it to use 20 seconds optimizing the solution. In the simpler environment of scene 1, the difference is only 4.39 s. If the manipulator is slow in relation to its size or the environment is complex, the

improvements to the optimality of the motion can be enough to justify the extra planning time.

6.3 The accuracy of the robot arm

6.3.1 Experiments

The accuracy of the system was tested by repeatedly planning a path to target end effector position starting from various initial configurations. The final positions were measured by installing a laser pointer to the end effector of the manipulator and recording the point where the laser hits the table on a graphing paper. The position of the graphing paper in relation to the robot coordinate system was determined using a tape measure. The errors caused by the angle between the end effector z-axis and the laser ray were estimated by placing the end effector on target height and rotating the wrist so that the laser point draws a circle on the graphing paper. The error is considered to be the radius of the circle, 60 mm in the 1st and 2nd test, 7.5 mm in 3rd test and 43 mm in 4th and 5th test. This error is constant within a test as the hand orientation should remain the same.

The arm accuracy test was performed for four different target positions. The first position was tested twice. In addition, an attempt was made to repeat the measurement corresponding to the last target position using a plummet instead of a laser pointer. Only four positions were measured until the robot stopped, probably due to overheating.

The accuracy and repeatability of the motions of the JACO arm was examined by comparing the target configurations used in calibration to the realized configurations and comparing the realized configurations between two calibration runs. The calibration dataset consisted of 120 positions that were obtained by moving the arm with joystick and storing the position of the robot's joints as reported by the robot's API. Since the configurations are always entered in the same order, the trajectory leading to each one is similar between runs.

6.3.2 Results

The accuracy of the JACO arm was tested by logging points where the ray of a laser held by the robot hit a graphing paper placed on the table in front of it. The robot was then moved over a series of target positions by the planner. In all but the first target position, the measurements of the end effector positions split into two populations (Figures 17 and 18). This split is not visible in the end effector positions that were estimated using the direct kinematic model based on the encoder readings from the robot arm. From the photographs of poses corresponding to samples in different populations (Figure 19) it can be observed that the two populations correspond to two different solutions for the kinematics of the robot. Most notably the wrist joints are turned the other way round. The distances different populations from each other as well as their distances from the goal and their internal variance are summarized for all the tests in Table 8.

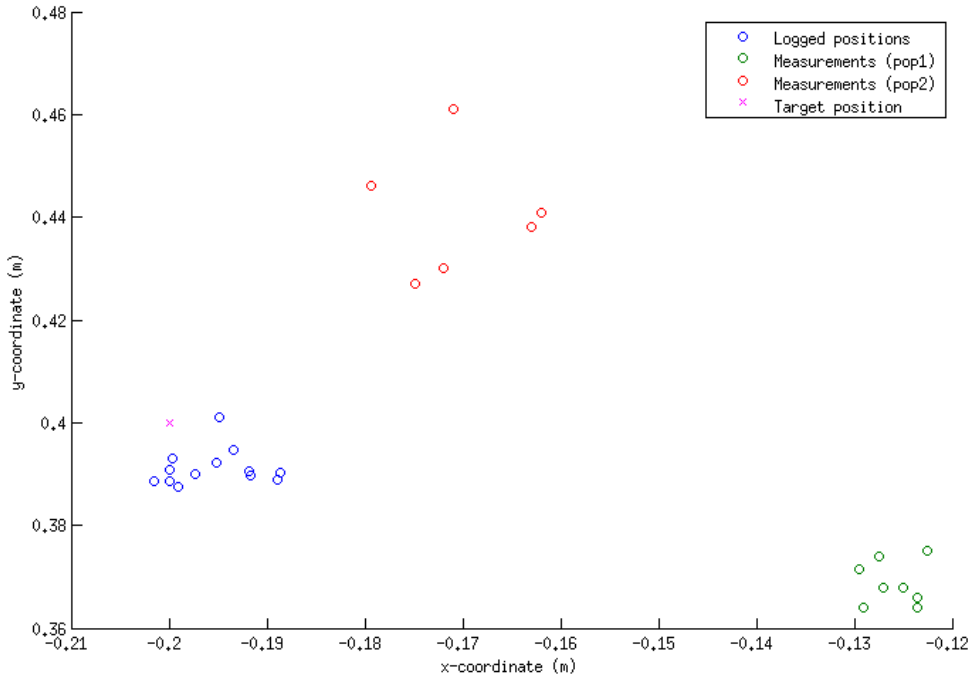


Figure 17: The target position of test number 3, the measured positions of the end effector and the positions logged from the kinematic model based on the readings from robot's encoders.

Table 8: The target positions, sizes of populations 1 and 2 in the measurement, distances of measurement population means from target position (e_n), the distance between the means of populations ($\Delta_{1,2}$) and the standard deviations of the populations in x and y direction. The measurement number 6 was made with plummet instead of laser. The unit is millimeter.

#	x_{target}	y_{target}	n_1	n_2	e_1	e_2	$\Delta_{1,2}$	$s_{x,1}$	$s_{y,1}$	$s_{x,2}$	$s_{y,2}$
1	107.5	514.1	7	0	12.334	—	—	6.91	6.29	—	—
2	107.5	514.1	10	0	17.051	—	—	6.71	6.41	—	—
3	-200	400	8	6	78.711	50.154	84.365	2.68	4.28	6.81	12.24
4	80	350	3	6	10.896	63.691	74.502	1.26	5.62	4.73	10.58
5	-120	490	10	3	17.687	53.751	63.367	6.41	10.93	4.49	11.03
6	-120	490	4	0	76.452	—	—	5.45	4.25	—	—

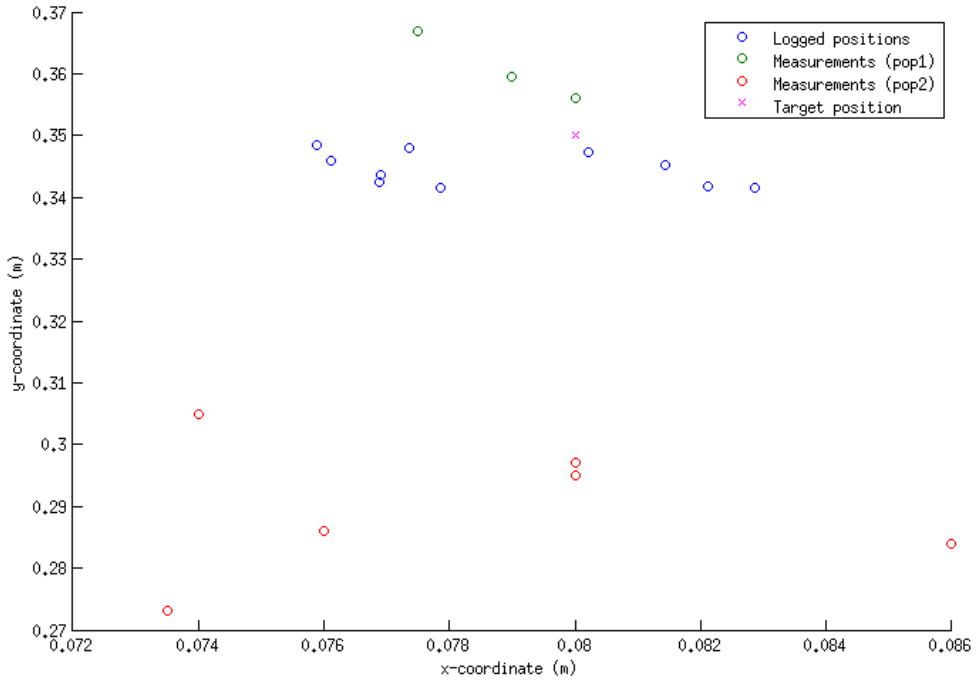


Figure 18: The target position of test number 4, the measured positions of the end effector and the positions logged from the kinematic model based on the readings from robot's encoders.

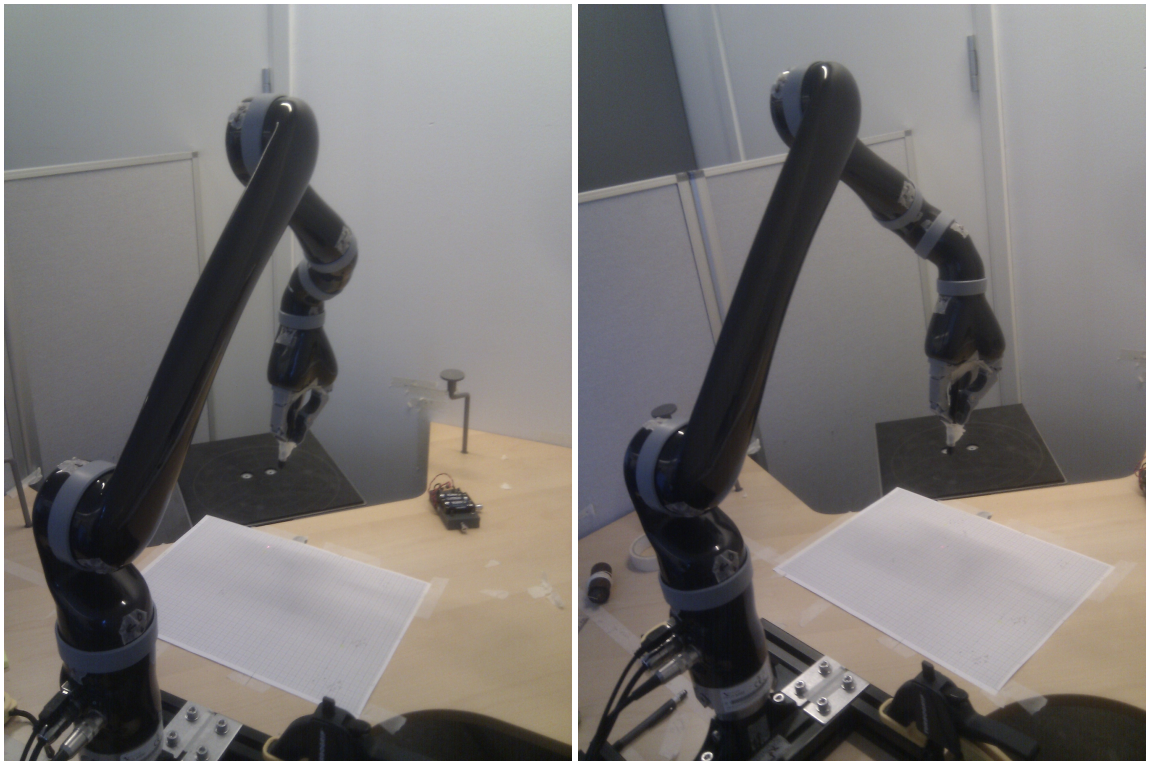


Figure 19: Robot poses corresponding to populations 1 (left) and 2 (right) in test number 5

In order to estimate the accuracy of the JACO arm independent of the kinematic model and planner, the arm was commanded to move through 120 pseudorandom positions. The errors in joint positions is presented in Table 9. The accuracy is measured in this case as the difference in commanded position and reached position as reported by the robot's internal sensors. In addition, the table shows the difference between positions achieved in two different calibration runs. This measures the repeatability of the motions, which is much better than the accuracy. The worst-case accuracy for the shoulder joint (joint number 2) is notably bad, 0.0592 radians of error is 3.39 degrees, which is much larger than the ± 0.5 deg given in the spec sheet [75]. The distribution of errors for the shoulder joint is illustrated in Figure 20. At least with the calibration poses, the error is in over 90% of the cases to the same direction. The distribution looks relatively normal, except for the higher-than-expected frequency of the really large errors in the leftmost bin. The samples in the leftmost bin could represent a different population where some kind of error has occurred but the amount of observations is small and could be due to chance. For the most part the same error source affects all the motions and the expected error is non-zero which means that there is a bias in the position of the shoulder and this bias is also detected by the sensors of the robot. The bias may be due to gravity or some asymmetry in the robot's controller.

The effect of these accuracy errors on the Cartesian position of the robot's end effector and on the laser point location in the first test was examined by simulation. The robot's kinematic model was placed in 4 different configurations corresponding to the 4 possible inverse kinematics solutions for reaching the target position in the 5th test position of the first accuracy test. All the combinations of either adding an error to the nominal angles, subtracting it or leaving the angle to its nominal value were then enumerated and the direct kinematics solved for each one. The test was done with the median errors and maximum errors presented in Table 9. The results have been summarized in Table 10. The errors tend to be larger in the direction of the y-axis. The error in the attitude of the end effector results in the laser point deviating more from the desired position than the end effector point does. In the case of median errors there is around 46 mm spread in the laser points.

Table 9: The absolute errors between commanded and realized calibration positions (accuracy) and between realized calibration positions in two different calibration runs (repeatability). The repeatability is computed from different data sets than accuracy. The angle unit is radian.

Error type	**Accuracy**					
Joint #	1	2	3	4	5	6
Min	0.0007	0.0001	0.0012	0.0000	0.0001	0.0020
Q1	0.0066	0.0123	0.0064	0.0048	0.0029	0.0059
Median	0.0084	0.0209	0.0134	0.0070	0.0079	0.0084
Q3	0.0115	0.0288	0.0256	0.0107	0.0105	0.0099
Max	0.0146	0.0592	0.0320	0.0186	0.0158	0.0126
Error type	**Repeatability**					
Joint #	1	2	3	4	5	6
Min	0.0008	0.0008	0.0006	0	0.0013	0.0008
Q1	0.0008	0.0008	0.0006	0	0.0013	0.0008
Median	0.0008	0.0009	0.0006	0	0.0013	0.0008
Q3	0.0010	0.0009	0.0011	0	0.0013	0.0019
Max	0.0027	0.0044	0.0029	0.0026	0.0039	0.0019

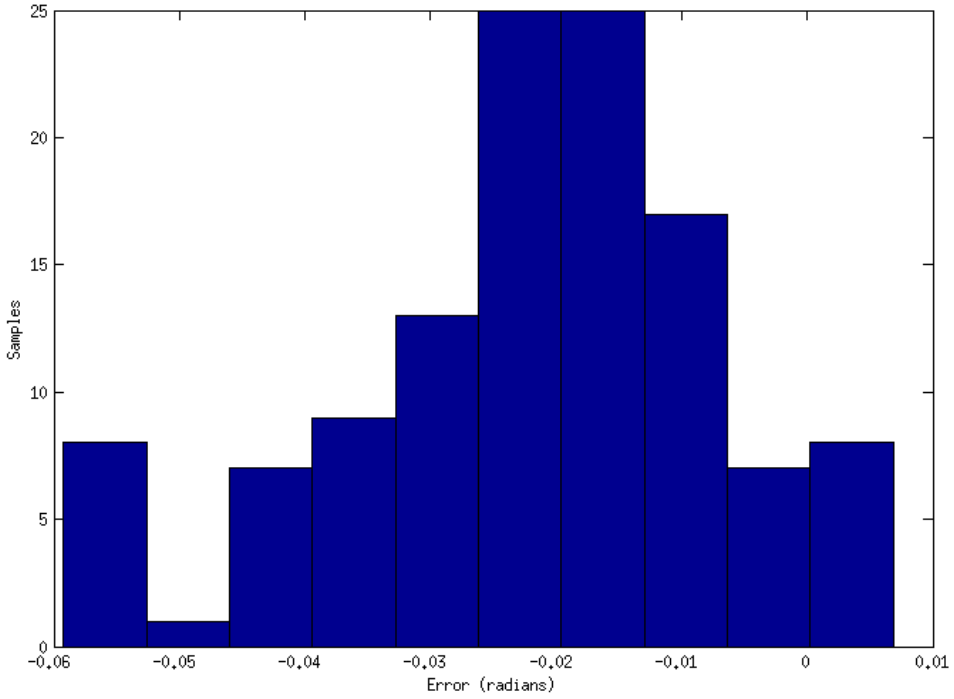


Figure 20: The distribution of errors in shoulder joint (joint 2) position while executing the calibration poses.

Table 10: The length of the range of reached positions for various kinematic solutions of test position 5 when errors according to median or maximum measured errors were added to joint angles. Values are in millimeters.

	Median error					**Maximum error**				
	End effector			Laser point		End effector			Laser point	
Solution	x	y	z	x	y	x	y	z	x	y
1	18.3	21.3	27.8	29.0	41.1	39.1	51.2	76.1	66.4	102.5
2	19.1	21.1	27.6	30.5	40.6	40.8	50.7	75.7	70.1	101.4
3	14.1	21.4	27.8	20.0	41.9	29.1	50.9	76.1	44.5	103.7
4	13.5	21.7	27.6	18.76	41.8	27.4	51.5	75.7	41.0	103.6

6.4 Comparison of t-metric with execution time

6.4.1 Experiments

The estimated execution times of various paths were recorded and the actual execution times by the JACO-manipulator were measured with a stopwatch. This was done to ensure that the t-metric discussed above corresponds to reality.

6.4.2 Results

In order to compare the t-metric to real execution time, 17 different trajectories were executed on the manipulator. The measured and estimated execution times for the trajectories are listed in Table 11. Although the measurement with stopwatch contains small errors, the estimated execution time (t-metric) explains observed execution times well. A line $t_{\text{obs}} = kt_{\text{est}} + b$ was fitted to the data using least-squares regression method, where t_{obs} is the observed execution time and t_{est} is the estimated execution time (Figure 21). The k parameter was estimated to be 0.975 ± 0.059 and the b parameter to be $0.346s \pm 0.585s$ ($\alpha = 0.05$ for the confidence intervals). The values $k = 1, b = 0$ lie within the confidence intervals. The coefficient of determination is $R^2 = 0.9878$. This means that the t-metric is essentially the same as the measured execution time.

Table 11: Measured and estimated execution times for 17 different trajectories

Estimate(s)	Measurement(s)
10.30	10.7
5.00	4.4
4.32	4.8
9.84	9.8
10.60	10.0
11.27	10.8
11.16	11.4
4.22	4.4
11.63	12.0
11.10	11.4
9.56	10.4
10.82	10.9
13.90	13.9
14.79	14.3
12.27	12.8
2.19	2.6
3.13	3.5

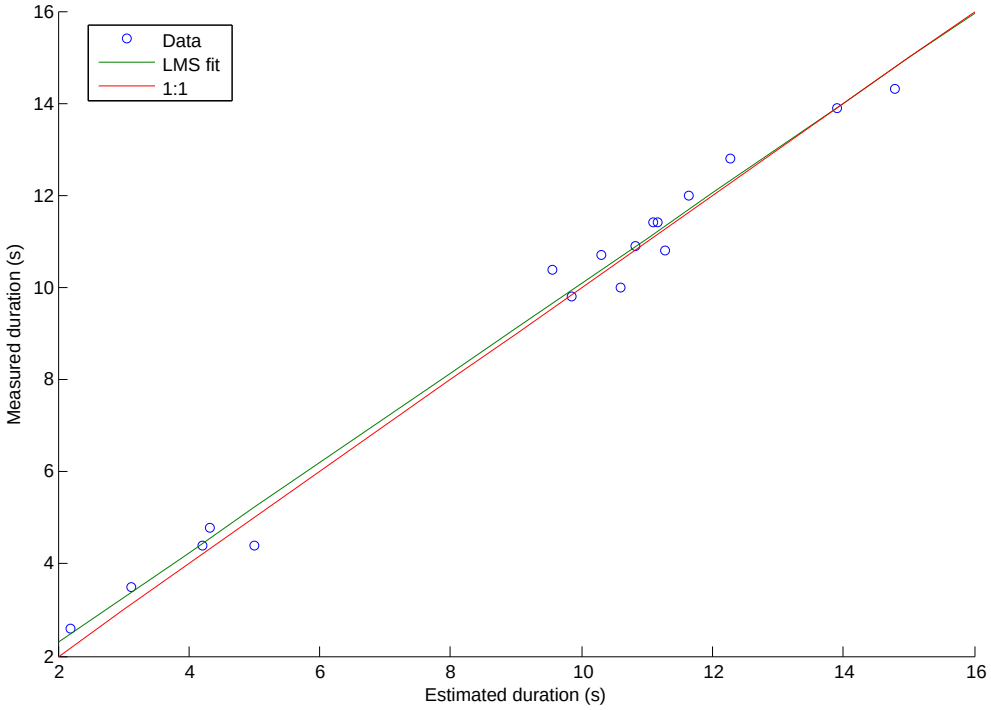


Figure 21: Measured execution time as a function of estimated execution time. Measurements, the regression line and line $y = x$ are shown.

6.5 Comparison of s-metric with end effector movement

6.5.1 Experiments

The t-metric is the estimated execution time of the path. The s-metric however is not the same as the end effector movement but its worst case estimate. The path lengths as measured by the s-metric were compared with the amount of end effector movement caused by executing path. This comparison was done by simulating the direct kinematics of the robot and by computing sum of Euclidean distances between successive end effector positions. The path was first linearly interpolated in the configuration space so that new states were inserted between path nodes at equal spaces. The paths that were used for this tests were ones that were produced by RRT-Connect, using s-metric in scene 2. No post-processing was done. This algorithm results in greater variance of path length than the optimizing algorithms (Figure 30 in Appendix A) and thus it is better suited to examination of the dependency between the metric and end effector movement. It is assumed that the method by which these paths have been generated does not affect this comparison. The data set initially consisted of 500 paths, but the test was repeated with larger data set of 10000 sample paths.

6.5.2 Results

A regression line $s_{\text{est}} = ks_{\text{real}}+b$ was fit to the data using least-squares method, where s_{real} is the end effector movement and s_{est} is the value of s-metric. The resulting model is $k = 3.0432 \pm 0.2029, b = 1.9808 \pm 0.7110$ (with $\alpha = 0.05$). The regression line is shown along with the original data in Figure 22. The value at $x = 0$ is greater than 0 because it is possible for the s-metric to be over 0 while the end effector point is not moving, but it is not possible for the s-metric to be 0 while the end effector point is moving. This is because the arm could be moving in a way that holds the end effector point static and such motion would still be measured by the s-metric which is a weighted sum of the motions performed by each joint.

The correlation between s-metric and end effector movement in the data set was 0.7972 and the correlation between t-metric and end effector movement 0.7643. The data set was split randomly ($n_1 = 240, n_2 = 260$), correlation coefficients were determined for s-metric and end effector movement in the first half and t-metric and end effector movement in the second half. The correlation coefficients were then tested for equality ($H_0 : \rho_1 = \rho_2, H_1 : \rho_1 > \rho_2$. See Appendix C). The test shows that the correlation with s-metric is almost significantly better than with t-metric (p=0.0236). The test was repeated for a larger data set: 10000 sample paths were generated by running RRT-Connect with s-metric. Proceeding as before, with $n_1 = 5042, n_2 = 4958$, the correlation between s-metric and end effector movement (r_1) is 0.7810 and the correlation between t-metric and end effector movement (r_2) is 0.7443. The p-value for the single-tailed test is 5.61×10^{-6}, which confirms the previous conclusion.

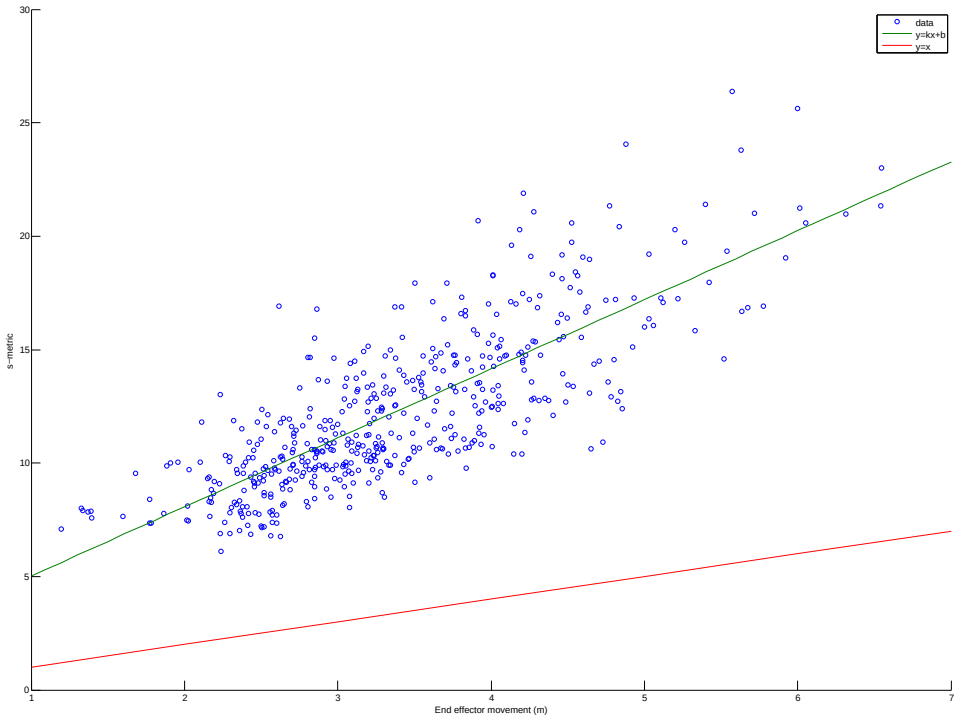


Figure 22: s-metric as a function of end effector movement. Measurements, the regression line and line $y = x$ are shown.

Table 12: Calibration parameters describing the transformation from Kinect coordinate system to robot coordinate system and their standard deviations. The rotation vector $[\theta_1, \theta_2, \theta_3]$ specifies the rotation axis and its magnitude the amount of rotation

Parameter	Value	std
θ_1	-3.4989	0.0028
θ_2	-1.7673	0.0048
θ_3	0.9838	0.0055
T_x	-0.0874m	0.0020m
T_y	0.3044m	0.0015m
T_z	1.1270m	0.0039m

6.6 Kinect calibration and accuracy

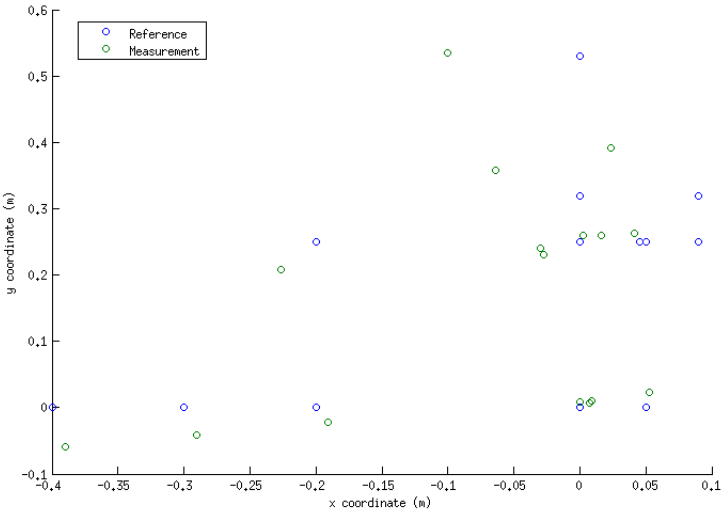
6.6.1 Experiments

The position and attitude of the Kinect sensor was calibrated in relation to the robot base coordinate system. A marker object was placed on measured positions in relation to the robot origin. Their coordinates were then measured by picking the target object from Kinect depth image. By comparing the measured coordinates to true positions of the marker, accuracy of the calibrated Kinect can be estimated.

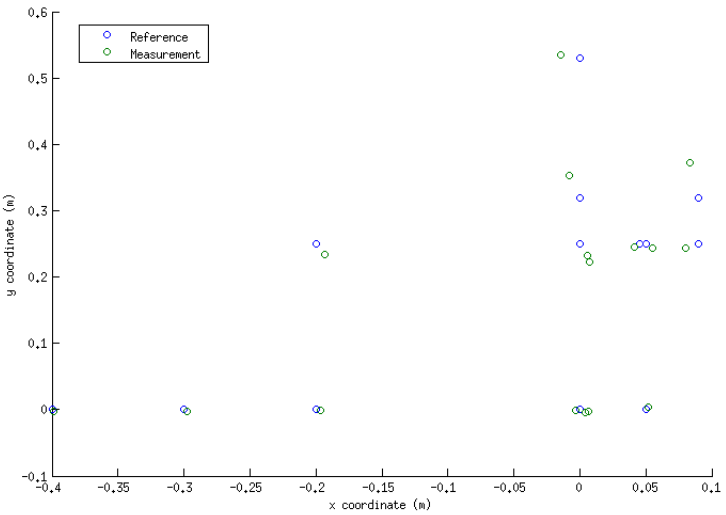
6.6.2 Results

Kinect calibration was performed and the standard deviation of the calibrated parameters was estimated (Table 12). The standard deviation for the location of robot's origin is ± 2 mm in the x direction and ± 1.5 mm in the y direction.

Positions of a target object were measured by selecting them from Kinect depth image and also with tape measure. The transformation matrix calculated from the calibrated extrinsic parameters was used to transform the selected points from Kinect coordinate system to robot coordinate system. The resulting measurement points and the reference points are presented in Figure 23a. It was noticed that the points appear to be rotated with respect to each other. By minimizing sum of squared error, the rotation between the two coordinates was estimated to be 0.1664 radians (9.534 degrees). After correcting the rotation the shift between the two coordinate systems was estimated by using the sample mean of the residual. The computed shift is 4.2 mm $\pm$ 3.2 mm in the x direction and 9.9 mm $\pm$ 8.3 mm in the y direction ($\alpha = 0.05$). The standard deviation of the residual is 7.3 mm in the x direction and 19 mm in the y direction. This includes the inaccuracy caused by Kinect as well as the inaccuracy of pointing a desired measurement point from the image. The measurements after correcting the rotation and translation are presented in Figure 23b.



(a) before correcting transformation



(b) after correcting transformation

Figure 23: The measurement points transformed into robot coordinates and the reference measurements.

7 Discussion

The goal of this thesis was to study methods for collision free path planning of robot manipulators in changing environments. First an overview of different sensors, environment representations, methods for collision detection and path planning algorithms were presented. A program capable of detecting the environment, forming a map of it and planning the movements of a robotic arm was developed. Calibration of intrinsic, extrinsic and depth parameters of the Kinect sensor was performed. Then three sampling-based planning algorithms based on rapidly exploring random trees were compared in an example case. Effect of the choice of configuration space metric for the performance of the planning algorithms was studied. The methods that were used are intended to be used as a support function for teleoperation. For this reason automatic grasping of the object was not pursued. Another goal was that the methods should be applicable to other kinds of manipulators as well.

The objectives were reached in the sense that the application developed was able to plan the trajectories and adapt its environment representation, both in a reasonable amount of time. For highly dynamic environments, an environment representation that can be updated even faster would be useful. The execution of the planned motions should be monitored based on the sensor data so that if a dynamic obstacle suddenly appears in the robot's path, the robot could be stopped and a replanning or path modification could be performed. This approach could be combined with reducing the robot's speed along its path to give the obstacle time to move out of robot's path while the new path is being planned.

The approach that was developed can be used to complement teleoperation of manipulators. The operator would point the desired target either from camera image or some reconstructed model or the target object might be automatically detected from sensor information. The motion planner would then design a collision free trajectory that brings the manipulator close to the desired object. The operator would take care of the often difficult manipulation of the target object after which the automatic motion planner could plan a trajectory back to home position or to store the object for transportation.

Were the developed program to be adapted to another application some further development would be needed. Because the kinematics and collision model of the robot as well as its state space description would have to be changed, it should be possible to read these from a configuration file. The planning algorithms are able to adapt to robots with different kinematics and the collision detection library is able to represent many different bounding volumes, in case the capsules are not suitable to describing the robot's structure. The method that was used to calibrate the location of the Kinect sensor can be adapted to robots with different kinematics and by using a different marker it could be used for other kinds of sensors, such as stereo cameras. Although the interface for the robot and the Kinect are currently implemented within the code, it should be relatively easy to develop an interface for any sensor providing point cloud data and any robot that can be sent a path represented in its configuration space. One part of the code that is more difficult to implement in a generic way is the inverse kinematics solver. A separate inverse

kinematics solver has to be compiled with IKFast for each new robot.

In principle, the planning algorithms could work without knowing the inverse kinematics of the robot, but this severely slows the algorithms as it becomes hard to guide the tree growth towards target states. The inverse kinematics compiler (IKFast) can be used to produce inverse kinematics solvers for many different robot structures. It has the advantage that all inverse kinematics solutions are produced and can be used as goal states for the sampling based planners. The solvers produced tend to be faster than optimization-based inverse kinematics. However, the compilation times can vary based on the structure of the robot and times up to hours were observed while for other structures the compilation times are below one minute. There may even be seemingly equivalent ways of representing the robot in *robot.xml* format that still have very different compilation times. The other alternative is to use optimization based inverse kinematics solvers such as the ones provided in the OMPL library.

The chosen planning algorithms do not depend on a particular environment representation and treat the collision detection as a black box. This allows the environment representation to be chosen freely to fit the needs of the application. In this application the environment was represented as a heightmap. This representation allows for easy collision checking and could adequately represent the test environment. However, in environments where overhanging structures limit robot's motion this representation has to be augmented to take such obstacles in to account or different representation has to be used. Other limitation with the heightmap as it was implemented in this work is the function performing the conversion from octomap. Although it works well for small environments such as the test cases it is not efficient enough to be used for large environments. A function could be developed to convert the voxels of octomap into box geometries of ODE, but that would result in huge amount of redundant information in the orientations and positions of the boxes in addition to requiring time for the conversion.

The collision detection library of ODE allows custom classes to be implemented. The whole octomap could be presented as a single ODE geometry removing the need for converting the map to ODE collision geometries. This would both save memory as the data does not have to be duplicated and be faster because the conversion step is not needed and the hierarchical structure of the octree can be used to accelerate collision queries.

The planners tested presented different trade-offs between the speed of the planning and the quality of the produced paths. The choice of planner may then depend on the execution speed of the planned motions in relation to the extra planning time. The RRT-Connect algorithm is able to produce a solution in under a second for both test cases. The median planning time for the RRT* algorithm was also under a second, but the right tail of the distribution is fatter. For some cases the algorithm could take an order of magnitude more time. The RRT* was also plagued by a strange thread locking issue that is most likely due to a bug in the implementation of the library. The third algorithm that was compared is the BT-RRT*. This is a combination of the ball trees and rapidly exploring random trees. It is supposed to be able to produce the initial solution quickly and then improve upon it. In the tests

it surprisingly tends to be slower than the other algorithms. The median solution time is between 2 s and 5 s in the test cases and the right tail of the distribution is fat. The poor performance of the algorithm may be due to a wrong configuration value or a bug in its implementation and does not necessarily represent a problem with the algorithm itself. Based on the experiments performed the RRT-Connect was the most reliable planner in that its planning time was most predictable. The quality of the solutions obtained with RRT-Connect was not as good as the quality of optimized paths produced by the other planners. The solutions produced by RRT* tended to be better than the ones produced by BT-RRT*.

Although the configuration space metrics that were developed showed good correlation with the property that they were intended to approximate (execution time and end effector movement) this did not affect the performance of the planners in a way that was predicted. The hypothesis was that a metric measuring execution time would lead to paths with smaller execution time and similarly metric estimating the end effector movement would lead to less end effector movement. Although in test scene 1, the RRT* algorithm using t-metric produced paths with shorter execution times than the one using the s-metric while the end effector movement remained the same, this pattern did not appear in other cases. Instead different algorithms seemed to benefit from different choice of metrics. The RRT-Connect algorithm worked faster and produced better paths with s-metric and the other algorithms produced better paths with t-metric. This may be due to the fact that the area searched when rewiring the configuration space is a ball induced by the configuration space metrics. The same is true for the balls used by BT-RRT* to reduce the amount of collision checking. The balls induced by different metrics have different volumes and that may affect the performance of the algorithms. This effect could perhaps be compensated by altering the parameter values that the planning algorithms use to calculate the radius of these configuration space volumes.

The accuracy of the JACO arm was tested and some rather significant issues were found. In the accuracy test of the planner the positions of the end effector were distributed into two different populations in most test positions. The populations seemed to correspond to different kinematic solutions of the arm mechanism. In the positions of the end effector that were calculated using the kinematic model based on the angular positions of the joints of the robot as reported by its sensors did not show division into two populations. However, those points do not have the error in the attitude of the arm included, which would cause the laser points to shift accordingly. Based on these results, it would seem either that the kinematic model is not accurate, that the error is mostly due to attitude of the arm or that the sensors of the robot do not provide the correct reading. The kinematic model was based on the CAD drawing provided by the manufacturer, but this drawing is in conflict with the kinematic parameters that they provided. Brief tests with the manufacturer provided values also showed two populations of laser points, but more comprehensive testing would need to be performed in order to show which of the parameters provide better results. From visual inspection and measurements of the robot arm, the CAD drawing appears to be closer to reality.

The arm was sometimes observed to droop a little below the target position. This

drooping was noticeable both by visual inspection of the robot and in the sensor readings from the robot. When attempting to correct for the droop by repeatedly commanding the robot to the target position, the arm would fall further. Even when reading the position of the robot's joints and feeding those same joint values back to the droop would increase. This is probably due to the controller of the arm. To determine the arms accuracy independent of any planning algorithms or kinematic models, the set of poses used for the calibration of the extrinsic parameters of the Kinect was used. In that test a worst case error of almost 3.4° was observed for one of the joints. This is the same shoulder joint whose position is mostly responsible for the drooping behavior. This error does not depend on the kinematic model used an is thus somehow caused by the robot itself. The repeatability of the motions is an order of magnitude better, so this error is systematic. The effect of errors observed in the joint positions on the accuracy of the Cartesian position of the arm was examined. The median case error is not large enough to explain the division of the points into two populations, but the worst case error would be. However, the division into two populations that was observed in the measured laser positions is not noticed in the error distribution of the shoulder joint. Therefore it is unlikely that the drooping of the shoulder joint is by itself the cause of the error in laser measurements. The likely conclusion is that there is some inaccuracy in the kinematic model. This is also supported by the observation that the two populations correspond to different kinematic solutions. In the end, it seems that the likely causes of the inaccuracies would be the robot's motion controller and possibly inaccurate kinematic model. Further testing might help answering this question more conclusively. Finally, the angle of the laser in relation to the z-axis of the end effector coordinate system as well as its position in the end effector coordinate system add a constant displacement to the accuracy measurements that could perhaps be eliminated with a better calibration procedure.

The calibration of the position of the Kinect sensor is based on the recorded positions of a marker attached to the robot's end effector. Therefore the accuracy of the robot's motions can also affect the calibration. The calibration was done several times, but unfortunately its accuracy was only tested once as the test mainly focused on estimating the accuracy of measurements done with Kinect after the calibration. More complete tests of the calibration procedure itself are provided in the original paper describing the method [102]. An error of 9.5° was noticed in the rotation around the z-axis of the robot's base. The standard error of the measurements after compensating for this rotation is slightly larger than estimate given in [7].

The immediate future work on this topic could include applying the methods that were used to a different robot manipulator. Robot operating system (ROS) could be utilized to implement the code in a more modular way. A function should be implemented that guards the execution of paths so that if an obstacle appears on the robot's route during execution, the robot is stopped and replanning is started. The reasons for the poor accuracy obtained with JACO arm should be further investigated. The effect of different parameter values on the performance of the planning algorithms should be studied to find the optimal values. The configuration space metrics could also be studied further by compensating for the change of metrics by

adapting the planner parameters. The factors that cause the BT-RRT* to perform worse than RRT* should be investigated.

On a slightly longer term, future work on this subject could include investigating the suitability of NDT-OM as environment representation for manipulation tasks. The parallelization of sampling based motion planning algorithms is another interesting topic. The planning based approaches are limited in their ability to react to quick changes in the environment because of the computation time of the algorithms remains high. A parallel implementation of PRM motion planner on a GPU has been demonstrated to yield 1–2 orders of magnitude speedup [105]. Such a dramatic speedup would allow the algorithms to be utilized in increasingly dynamic environments.

References

[1] Martial Hebert. Active and Passive Range Sensing for Robotics. In *2000 IEEE International Conference on Robotics and Automation*, pages 102–110, San Francisco, CA, 2000.

[2] Robert B. Fisher and Kurt Konolige. Range sensors. In *Springer Handbook of Robotics*, chapter 22, pages 521–542. Springer, Berlin Heidelberg, 2008.

[3] Todor Stoyanov, Rasoul Mojtahedzadeh, Henrik Andreasson, and Achim J. Lilienthal. Comparative evaluation of range sensor accuracy for indoor mobile robotics and automated logistics applications. *Robotics and Autonomous Systems*, (DOI: 10.1016/j.robot.2012.08.011), October 2012.

[4] Velodyne. HDL-64E S2 datasheet, 2010. http://velodynelidar.com/lidar/products/brochure/HDL-64ES2datasheet_2010_lowres.pdf, Checked: 29.5.2013.

[5] Velodyne. Velodyne's HDL-64E: A high definition lidar sensor for 3-D applications, 2007. http://velodynelidar.com/lidar/products/white_paper/HDLwhitepaper_OCT2007_web.pdf, Checked: 29.5.2013.

[6] Jann Poppinga, Andreas Birk, and Kaustubh Pathak. A characterization of 3D sensors for response robots. In *RoboCup 2009: Robot Soccer World Cup*, Lecture Notes in Computer Science, pages 264–275. Springer Berlin Heidelberg, 2010.

[7] K Khoshelham. Accuracy analysis of kinect depth data. In *International Society for Photogrammetry and Remote Sensing Workshop*, volume XXXVIII, pages 133–138, Calgary, 2011.

[8] Uland Wong, Aaron Morris, Colin Lea, James Lee, Chuck Whittaker, Ben Garney, and Red Whittaker. Comparative evaluation of range sensing technologies for underground void modeling. In *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3816–3823. Ieee, September 2011.

[9] Steven M. LaValle. *Planning Algorithms*. Cambridge University Press, Cambridge, 2006.

[10] Benoit Morisset, Radu Bogdan Rusu, Aravind Sundaresan, Kris Hauser, Motilal Agrawal, Jean-Claude Latombe, and Michael Beetz. Leaving Flatland: Toward real-time 3D navigation. In *2009 IEEE International Conference on Robotics and Automation*, pages 3786–3793. Ieee, May 2009.

[11] Kai M. Wurm, Armin Hornung, Maren Bennewitz, Cyrill Stachniss, and Wolfram Burgard. OctoMap: A probabilistic, flexible, and compact 3D map representation for robotic systems. In *2010 IEEE International Conference on Robotics and Automation Workshop*, 2010.

[12] Hans Moravec and Alberto Elfes. High resolution maps from wide angle sonar. In *1985 IEEE International Conference on Robotics and Automation*, pages 116–121. IEEE, 1985.

[13] John Amanatides and A Woo. A fast voxel traversal algorithm for ray tracing. In *EUROGRAPHICS 1987*, Amsterdam, The Netherlands, 1987.

[14] H. P. Moravec. Robot Spatial Perception by Stereoscopic Vision and 3D Evidence Grids. Technical report, The Robotics Institute, Carnegie Mellon University, 1996.

[15] Julian Ryde and Huosheng Hu. 3D mapping with multi-resolution occupied voxel lists. *Autonomous Robots*, 28(2):169–185, September 2009.

[16] Chris Leger. Efficient sensor/model based on-line collision detection for planetary manipulators. In *2002 IEEE International Conference on Robotics and Automation*, volume 2, pages 1697–1703. IEEE, 2002.

[17] R. Triebel, P. Pfaff, and W. Burgard. Multi-level surface maps for outdoor terrain mapping and loop closing. In *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2006.

[18] Peter Biber and W. Strasser. The normal distributions transform: a new approach to laser scan matching. In *2003 IEEE/RSJ International Conference on Intelligent Robots and Systems*, volume 3, pages 2743–2748. IEEE, 2003.

[19] M. Magnusson, T. Duckett, and A. J. Lilienthal. 3D scan registration for autonomous mining vehicles. *Journal of Field Robotics*, 24(10):803–827, 2007.

[20] Todor Stoyanov, Martin Magnusson, Hakan Almqvist, and Achim J. Lilienthal. On the accuracy of the 3D Normal Distributions Transform as a tool for spatial representation. In *2011 IEEE International Conference on Robotics and Automation*, pages 4080–4085. Ieee, May 2011.

[21] Jari Saarinen, Henrik Andreasson, Todor Stoyanov, Juha Ala-luhtala, and Achim J Lilienthal. Normal Distributions Transform Occupancy Maps : Application to Large-Scale Online 3D Mapping. In *IEEE International Conference on Robotics and Automation*, pages 2225–2230, Karlsruhe, Germany, 2013.

[22] M Teschner, S. Kimmerle, B. Heidelberger, G. Zachmann, L. Raghupathi, A. Fuhrmann, M.-P. Cani, F. Faure, N. Magnenat-Thalmann, W. Strasser, and P. Volino. Collision Detection for Deformable Objects. *Computer Graphics Forum*, 24(1):61–81, March 2005.

[23] C. Ericson. *Real-Time Collision Detection*. Morgan Kaufmann, 2005.

[24] P. M. Hubbard. Collision detection for interactive graphics applications. *IEEE Transactions on Visualization and Computer Graphics*, 1(3):218–230, 1995.

[25] S Kockara, T Halic, K. Iqbal, C. Bayrak, and Richard Rowe. Collision detection: A survey. In *2007 IEEE International Conference on Systems, Man and Cybernetics*, pages 4046–4051, October 2007.

[26] Quentin Avril, Valérie Gouranton, and Bruno Arnaldi. New trends in collision detection performance. In *2009 Virtual Reality International Conference*, page 53, Laval, France, 2009.

[27] David Baraf. *Dynamic Simulation of Non-Penetrating Rigid Bodies*. Ph. d. thesis, Cornell University, 1992.

[28] Jonathan D. Cohen, Ming C. Lin, Dinesh Manocha, and Madhav Ponamgi. I-COLLIDE: An interactive and exact collision detection system for large-scale environments. In *1995 Symposium on Interactive 3D graphics*, pages 189–ff., New York, USA, 1995. ACM Press.

[29] George Vaněkčkek. Back-face culling applied to collision detection of polyhedra. *The Journal of Visualization and Computer Animation*, 5(1):55–63, January 1994.

[30] Quentin Avril. *Détection de Collision pour Environnements Large Échelle: Modèle Unifié et Adaptatif pour Architectures Multi-coeur et Multi-GPU*. PhD thesis, INSA de Rennes, 2011.

[31] M. C. Lin and Stefan Gottschalk. Collision detection between geometric models: A survey. In *8th IMA Conference on the Mathematics of Surfaces*, pages 37–56, 1998.

[32] R. A. Finkel and J. L. Bentley. Quad trees a data structure for retrieval on composite keys. *Acta informatica*, 4(1):1–9, 1974.

[33] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, September 1975.

[34] Scott Le Grand. Broad-Phase Collision Detection with CUDA. In *GPU Gems 3*, chapter 32, page 1008. Addison-Wesley Professional, 2007.

[35] M. C. Lin and J.F. Canny. A fast algorithm for incremental distance calculation. In *1991 IEEE International Conference on Robotics and Automation*, pages 1008–1014, Sacramento, California, 1991.

[36] Brian Mirtich. V-Clip: fast and robust polyhedral collision detection. *ACM Transactions on Graphics*, 17(3):177–208, July 1998.

[37] S. A. Ehmann and M. C. Lin. Accelerated proximity queries between convex polyhedra by multi-level Voronoi marching. In *2000 IEEE/RSJ International Conference on Intelligent Robots and Systems*, volume 3, pages 2101–2106. IEEE, 2000.

[38] E. G. Gilbert, D. W. Johnson, and S. S. Keerthi. A fast procedure for computing the distance between complex objects in three-dimensional space. *IEEE Journal on Robotics and Automation*, 4(2):193–203, April 1988.

[39] E. G. Gilbert and C.-P. Foo. Computing the distance between general convex objects in three-dimensional space. *IEEE Transactions on Robotics and Automation*, 6(1):53–61, 1990.

[40] Mauro Figueiredo and J Oliveira. An efficient collision detection algorithm for point cloud models. In *20th International Conference on Computer Graphics and Vision*, St.Petersburg, Russia, 2010.

[41] Jia Pan, Sachin Chitta, and Dinesh Manocha. FCL: A general purpose library for collision and proximity queries. *2012 IEEE International Conference on Robotics and Automation*, pages 3859–3866, May 2012.

[42] Gino van den Bergen. Efficient collision detection of complex deformable models using AABB trees. *Journal of Graphics Tools*, 2(4):1–13, 1998.

[43] S Gottschalk, M. C. Lin, and D. Manocha. OBBTree. In *23rd Annual conference on Computer graphics and interactive techniques*, pages 171–180, New York, USA, 1996. ACM Press.

[44] G. Zachmann. Rapid collision detection by dynamically aligned DOP-trees. In *IEEE 1998 Virtual Reality Annual International Symposium*, pages 90–97, Atlanta, GA, 1998.

[45] Jung-Woo Chang, Wenping Wang, and Myung-Soo Kim. Efficient collision detection using a dual OBB-sphere bounding volume hierarchy. *Computer-Aided Design*, 42(1):50–57, January 2010.

[46] D Fox, W Burgard, and S Thrun. The dynamic window approach to collision avoidance. *IEEE Robotics & Automation Magazine*, 4(1):23–33, March 1997.

[47] Javier Minguez and Luis Montano. Nearness Diagram (ND) Navigation: Collision Avoidance in Troublesome Scenarios. *IEEE Transactions on Robotics and Automation*, 20(1):45–59, February 2004.

[48] Johann Borenstein and Yoram Koren. The vector field histogram — fast obstacle avoidance for mobile robots. *IEEE Transactions on Robotics and Automation*, 7(3):278–288, June 1991.

[49] O. Khatib. Real-time obstacle avoidance for manipulators and mobile robots. In *1985 IEEE International Conference on Robotics and Automation*, volume 2, pages 500–505, 1985.

[50] R.B. Tilove. Local obstacle avoidance for mobile robots based on the method of artificial potentials. In *1990 IEEE International Conference on Robotics and Automation*, pages 566–571, 1990.

[51] Winkler Alexander and Jozef Suchý. Vision Based Collision Avoidance of Industrial Robots. In *18th IFAC World Congress*, pages 9452–9457, Milano, Italy, August 2011.

[52] Fabrizio Flacco and Alessandro De Luca. A Depth Space Approach to Human-Robot Collision Avoidance. In *2012 IEEE International Conference on Robotics and Automation*, pages 338–345, Saint Paul, Minnesota, USA, 2012.

[53] Agostino De Santis, Alin Albu-Schaffer, Christian Ott, Bruno Siciliano, and Gerd Hirzinger. The skeleton algorithm for self-collision avoidance of a humanoid manipulator. In *2007 IEEE/ASME international conference on advanced intelligent mechatronics*, pages 1–6. IEEE, 2007.

[54] S. Quinlan and O. Khatib. Elastic bands: connecting path planning and control. *1993 IEEE International Conference on Robotics and Automation*, pages 802–807.

[55] Oliver Brock and O. Khatib. Elastic Strips: A Framework for Motion Generation in Human Environments. *The International Journal of Robotics Research*, 21(12):1031–1052, December 2002.

[56] Andrew Spencer, Mitch Pryor, Chetan Kapoor, and Delbert Tesar. Collision avoidance techniques for tele-operated and autonomous manipulators in overlapping workspaces. In *2008 IEEE International Conference on Robotics and Automation*, pages 2910–2915. IEEE, May 2008.

[57] YK Hwang and N Ahuja. Gross motion planning — a survey. *ACM Computing Surveys*, 24(3), 1992.

[58] Bruce Donald, Patrick Xavier, John Canny, and John Reif. Kinodynamic motion planning. *Journal of the ACM*, 40(5):1048–1066, November 1993.

[59] John H Reif. Complexity of the mover's problem and generalizations. In *20th Annual Symposium on Foundations of Computer Science*, pages 421–427. IEEE, 1979.

[60] LE Kavraki, Petr Svestka, JC Latombe, and M.H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 12(4):566–580, 1996.

[61] S. M. LaValle. Rapidly-Exploring Random Trees: A New Tool for Path Planning. Technical report, Computer Science Department, Iowa State University, 1998.

[62] Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning. *The International Journal of Robotics Research*, 30(7):846–894, June 2011.

[63] O Khatib and L M Mampey. Fonction decision-commande d un robot manipulateur, Rep. 2/7156. *DERA/CERT, Toulouse, France*, 1978.

[64] Ellips Masehian and Davoud Sedighizadeh. Classic and heuristic approaches in robot motion planning — a chronological review. *World Academy of Science, Engineering and technology*, 29:101–106, 2007.

[65] Steven M. LaValle and James J. Jr. Kuffner. Rapidly-exploring random trees: Progress and prospects. In *The Fourth International Workshop on Algorithmic Foundations of Robotics*, pages 293–308, Hanover, New Hampshire, 2000.

[66] J.J. Kuffner and S.M. LaValle. RRT-connect: An efficient approach to single-query path planning. In *2000 IEEE International Conference on Robotics and Automation*, volume 2, pages 995–1001, San Francisco, CA, 2000. IEEE.

[67] Alexander Shkolnik and Russ Tedrake. Sample-Based Planning with Volumes in Configuration Space, September 2011. http://arxiv.org/abs/1109.3145, Checked: 3.7.2013.

[68] 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. In *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4307–4313, September 2011.

[69] Sertac Karaman, Matthew R. Walter, Alejandro Perez, Emilio Frazzoli, and Seth Teller. Anytime Motion Planning using the RRT*. In *2011 IEEE International Conference on Robotics and Automation*, pages 1478–1483, May 2011.

[70] Baris Akgun and Mike Stilman. Sampling heuristics for optimal motion planning in high dimensions. In *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2640–2645, September 2011.

[71] David Hsu, JC Latombe, and R Motwani. Path planning in expansive configuration spaces. In *1997 International Conference on Robotics and Automation*, volume 3, pages 2719–2726. IEEE, 1997.

[72] G Sánchez and JC Latombe. A single-query bi-directional probabilistic roadmap planner with lazy collision checking. In *Robotics Research*, Springer Tracts in Advanced Robotics, pages 403–417. Springer, Berlin Heidelberg, 2003.

[73] I. Sucan and L. E. Kavraki. A Sampling-Based Tree Planner for Systems With Complex Dynamics. *IEEE Transactions on Robotics*, 28(1):116–131, February 2012.

[74] Erion Plaku, Lydia E. Kavraki, and Moshe Y. Vardi. Motion Planning With Dynamics by a Synergistic Combination of Layers of Planning. *IEEE Transactions on Robotics*, 26(3):469–482, June 2010.

[75] Kinova. JACO Spec Sheet 2012, 2012. http://kinovarobotics.com/wp-content/uploads/2012/04/kinova-jaco-spec-sheet-2012.pdf, Checked: 29.5.2013.

[76] Kinova. Jaco API programming guide R4.0.5, 2012. http://kinovarobotics.com/wp-content/uploads/2012/04/JacoAPI_ProgrammingGuide.pdf, Checked: 29.5.2013.

[77] Evan-Amos. Wikimedia Commons, Picture of Kinect, 2011. http://commons.wikimedia.org/wiki/File:Xbox-360-Kinect-Standalone.png, Checked: 15.4.2013.

[78] libfreenect 0.0.1. https://github.com/OpenKinect/libfreenect, Checked: 22.5.2013.

[79] OpenNI. http://www.openni.org/, Checked: 2.7.2013.

[80] Kurt Konolige and Patrick Mihelich. ROS wiki, Techincal details of Kinect calibration, 2010. http://www.ros.org/wiki/kinect_calibration/technical, Checked: 5.12.2012.

[81] Camera calibration toolbox. http://www.vision.caltech.edu/bouguetj/calib_doc/, Checked: 7.12.2012.

[82] Jan Smisek, Michal Jancosek, and Tomas Pajdla. 3D with Kinect. *2011 IEEE International Conference on Computer Vision Workshops*, pages 1154–1160, November 2011.

[83] Panjawee Rakprayoon, Miti Ruchanurucks, and Ada Coundoul. Kinect-based obstacle detection for manipulator. *2011 IEEE/SICE International Symposium on System Integration*, pages 68–73, December 2011.

[84] IA Sucan, Mark Moll, and LE Kavraki. The open motion planning library (OMPL). *IEEE Robotics & Automation Magazine*, (December):1–10, 2012.

[85] OMPL website, Available planners, 2013. http://ompl.kavrakilab.org/planners.html, Checked: 17.5.2013.

[86] Russell Smith. Open Dynamics Engine 0.5 User Guide, 2006. http://www.ode.org/ode-latest-userguide.html, Checked: 22.5.2013.

[87] ODE wiki, Manual, Collision detection. http://ode-wiki.org/wiki/index.php?title=Manual:_All#Collision_Detection, Checked: 30.11.2012.

[88] Daniel Fiser. libccd 1.4. http://libccd.danfis.cz/, Checked: 22.5.2013.

[89] Gino Van Den Bergen. Proximity queries and penetration depth computation on 3D game objects. *Game developers conference*, 2001.

[90] Gary Snethen. Xenocollide: Complex Collision Made Simple. In *Game Programming Gems 7*, pages 165–178. Cengage Learning, 2008.

[91] ODE 0.12 Source, collision_libccd.cpp, 2012. http://sourceforge.net/p/opende/code/HEAD/tree/tags/0.12/ode/src/collision_libccd.cpp, Checked: 29.4.2013.

[92] Russel et al. Smith. Open Dynamics Engine 0.12, 2012. http://sourceforge.net/projects/opende/, Checked: 22.5.2013.

[93] ROS. http://www.ros.org/wiki/, Checked: 2.7.2013.

[94] Radu Bogdan Rusu and Steve Cousins. 3D is here: Point Cloud Library (PCL). *2011 IEEE International Conference on Robotics and Automation*, pages 1–4, May 2011.

[95] PCL Kinect Grabber, 2013. http://docs.pointclouds.org/trunk/classpcl_1_1_open_n_i_grabber.html, Checked: 13.4.2013.

[96] Visualization Toolkit, 2013. www.vtk.org, Checked: 18.4.2013.

[97] OpenCV 2.3.1. http://opencv.org/, Checked: 22.5.2013.

[98] Rosen Diankov. *Automated construction of robotic manipulation programs*. PhD thesis, Carnegie Mellon University, Robotics Institute, 2010.

[99] OpenRAVE Documentation, 2013. http://openrave.org/docs/latest_stable/overview/, Checked: 29.4.2013.

[100] IKFast Documentation, 2013. http://openrave.org/docs/latest_stable/openravepy/ikfast/, Checked: 29.4.2013.

[101] Mono. http://www.mono-project.com/Main_Page, Checked: 2.7.2013.

[102] J. Ilonen and V. Kyrki. Robust robot-camera calibration. *2011 15th International Conference on Advanced Robotics*, pages 67–74, June 2011.

[103] Simple DirectMedia Layer. http://www.libsdl.org/, Checked: 2.7.2013.

[104] OpenGL. http://www.opengl.org/, Checked: 2.7.2013.

[105] Jia Pan and Christian Lauterbach. g-Planner: Real-time motion planning and global navigation using GPUs. In *24th AAAI Conference on Artificial Intelligence*, pages 1245–1251, 2010.

[106] Richard P. Runyon and Audrey Haber. *Fundamentals of Behavioral Statistics*. McGraw-Hill Inc, 6th edition, 1988.

[107] W. J. Connover. *Practical Nonparametric Statistics.* Wiley, 3rd edition, 1999.

[108] George W. Snedecor and William G. Cochran. *Statistical Methods.* Iowa State University Press, Ames, Iowa, USA, 6th edition, 1971.

A Comparison of planners: figures

The planning algorithms were compared according to the time it takes to produce first solution, the end effector movement of the initial solution path, the execution time of the initial solution path, the end effector movement of the optimized path (after 20 seconds) and the execution time of the optimized path. The results are presented in Figures 24–31 for the two test scenes. The name of the algorithm (either 'RRTC', 'BTRRT*' or 'RRT*') is followed by the name of the metric which is followed by 'pp' if post processing was used.

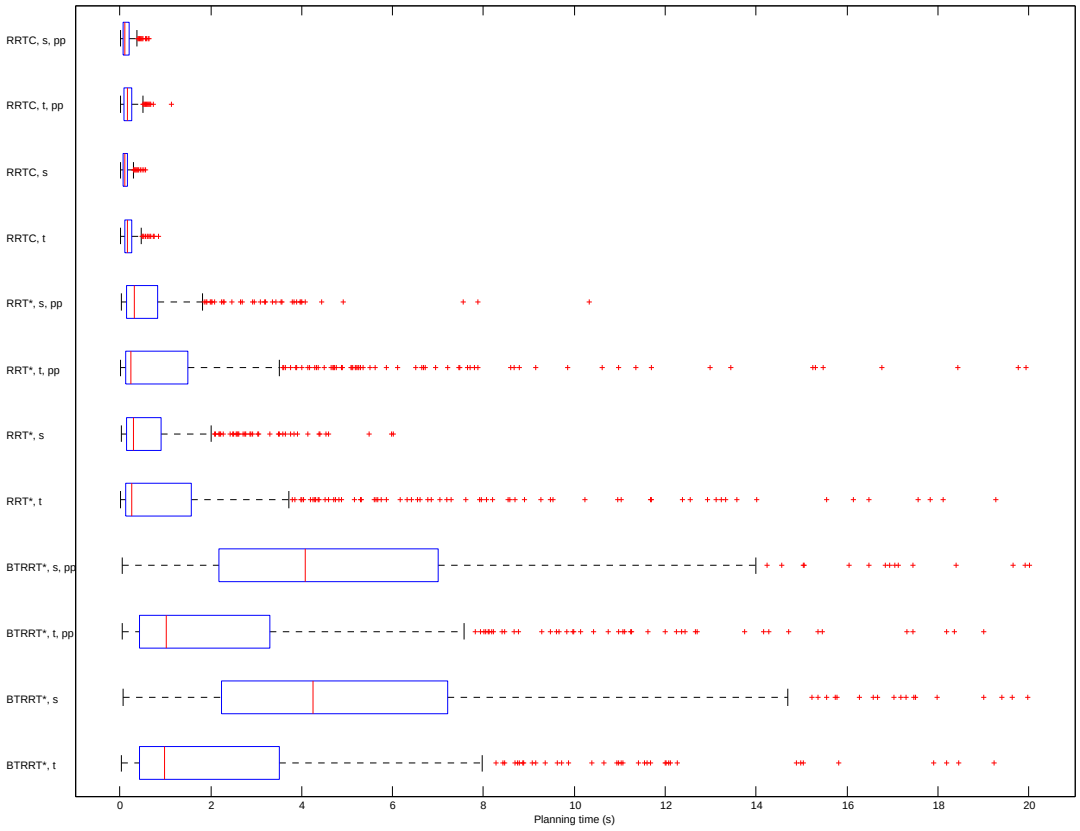


Figure 24: The time required to produce the first solution for all planners in scene 1.

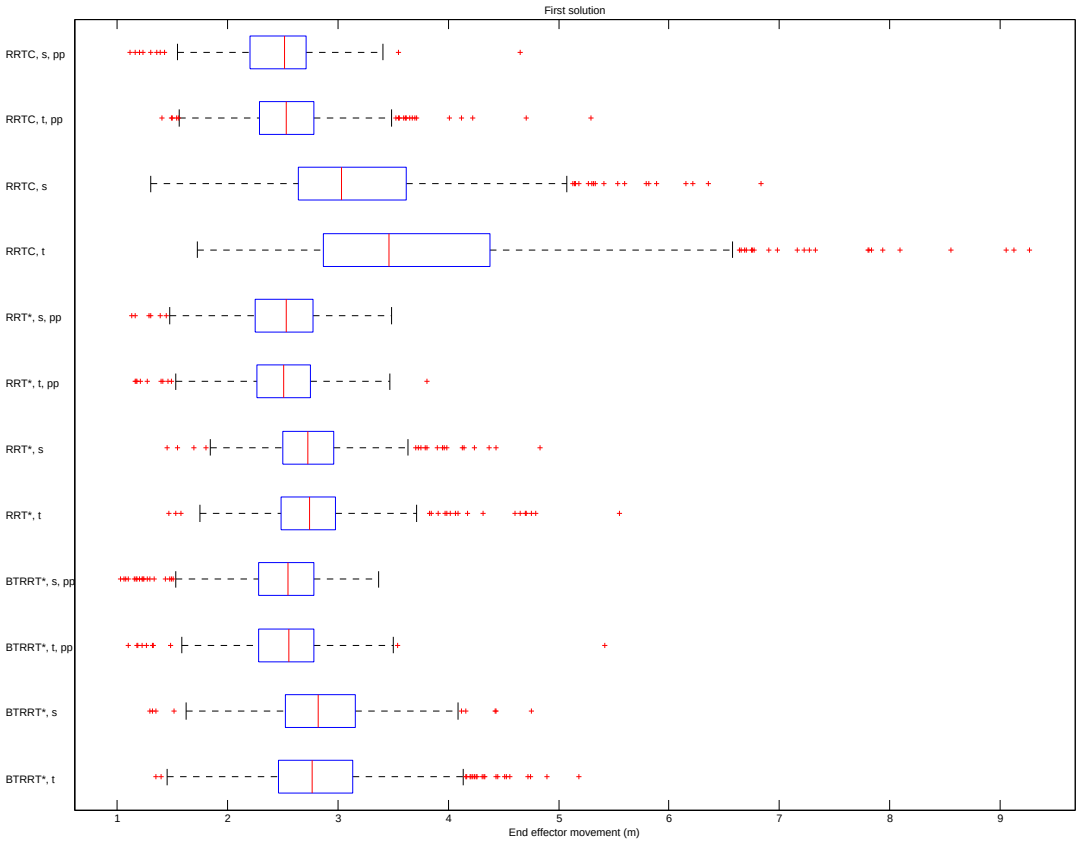


Figure 25: The end effector movements for the initial paths produced by different planners in scene 1.

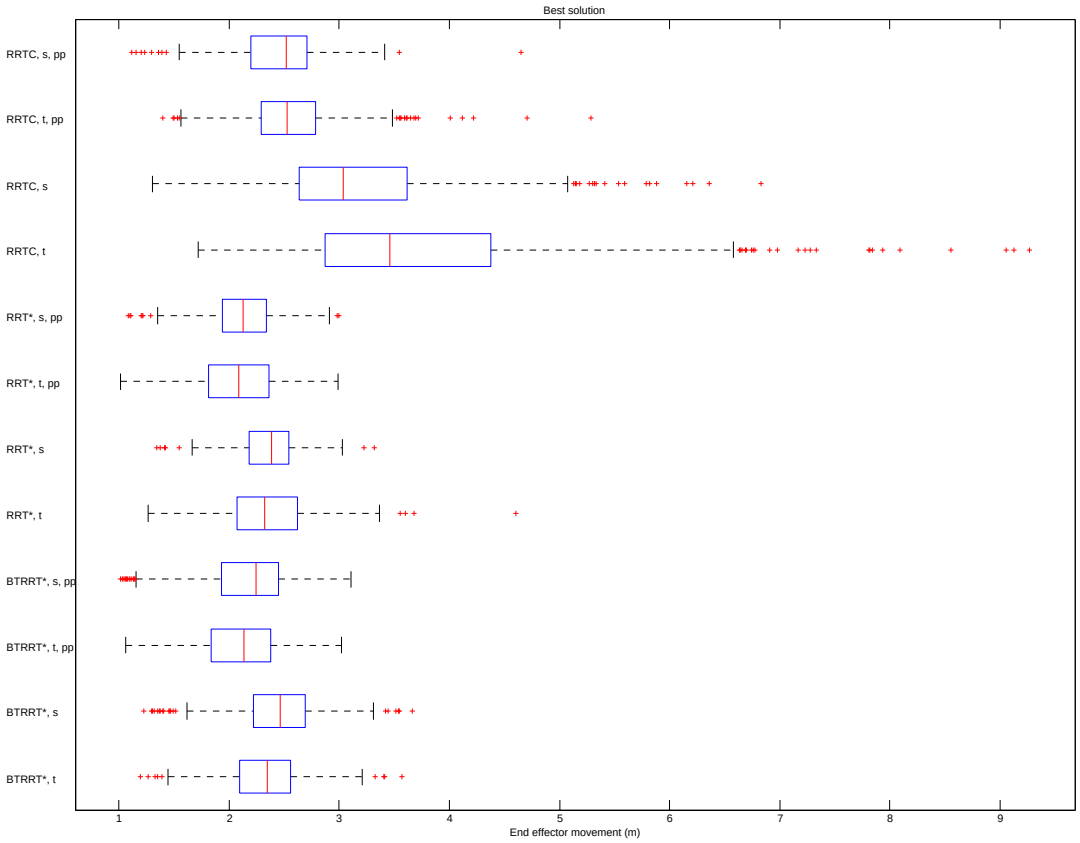


Figure 26: The end effector movements for the optimized paths produced by different planners in scene 1.

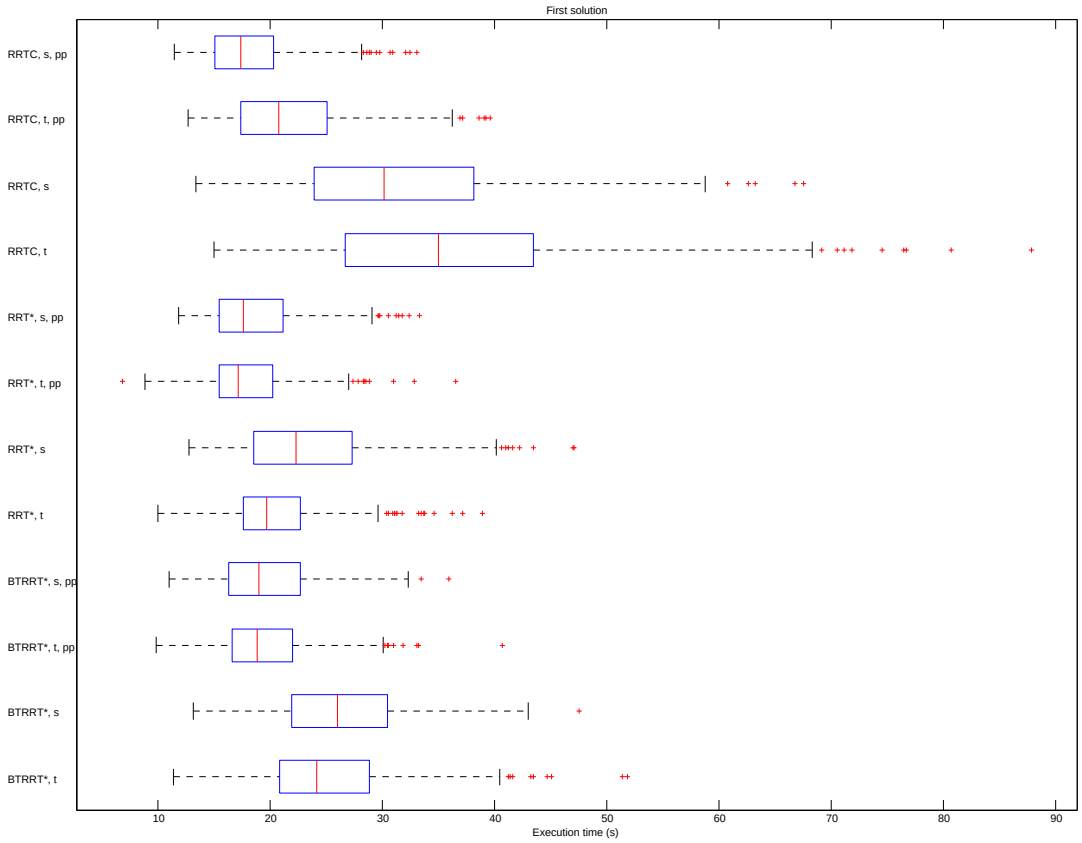


Figure 27: The execution times for the initial paths produced by different planners in scene 1.

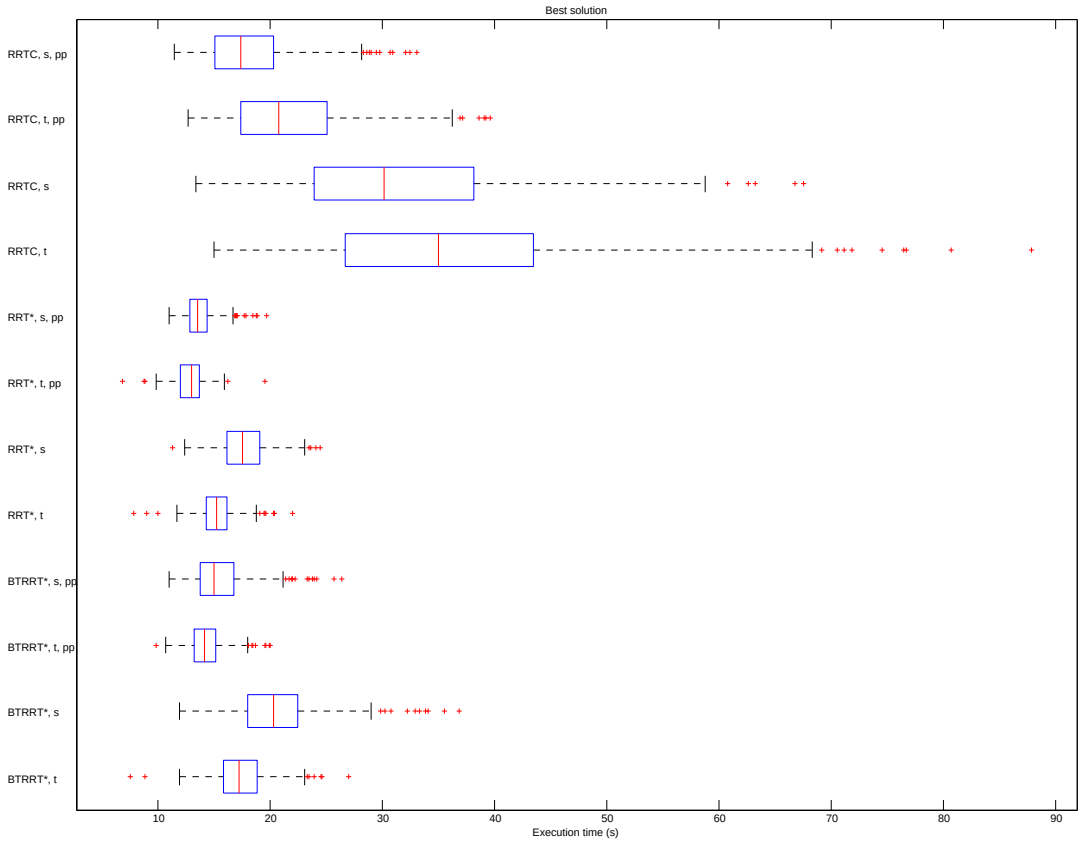


Figure 28: The execution times for the optimized paths produced by different planners in scene 1.

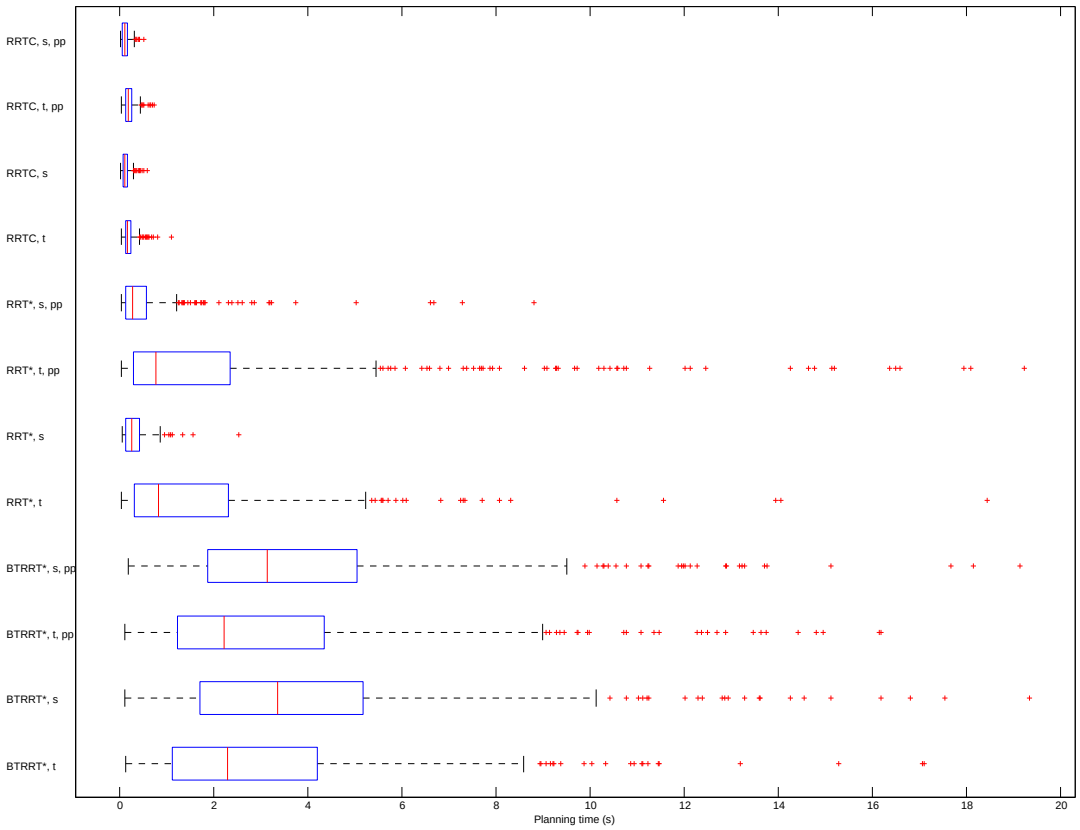


Figure 29: The time required to produce the first solution for all planners in scene 2.

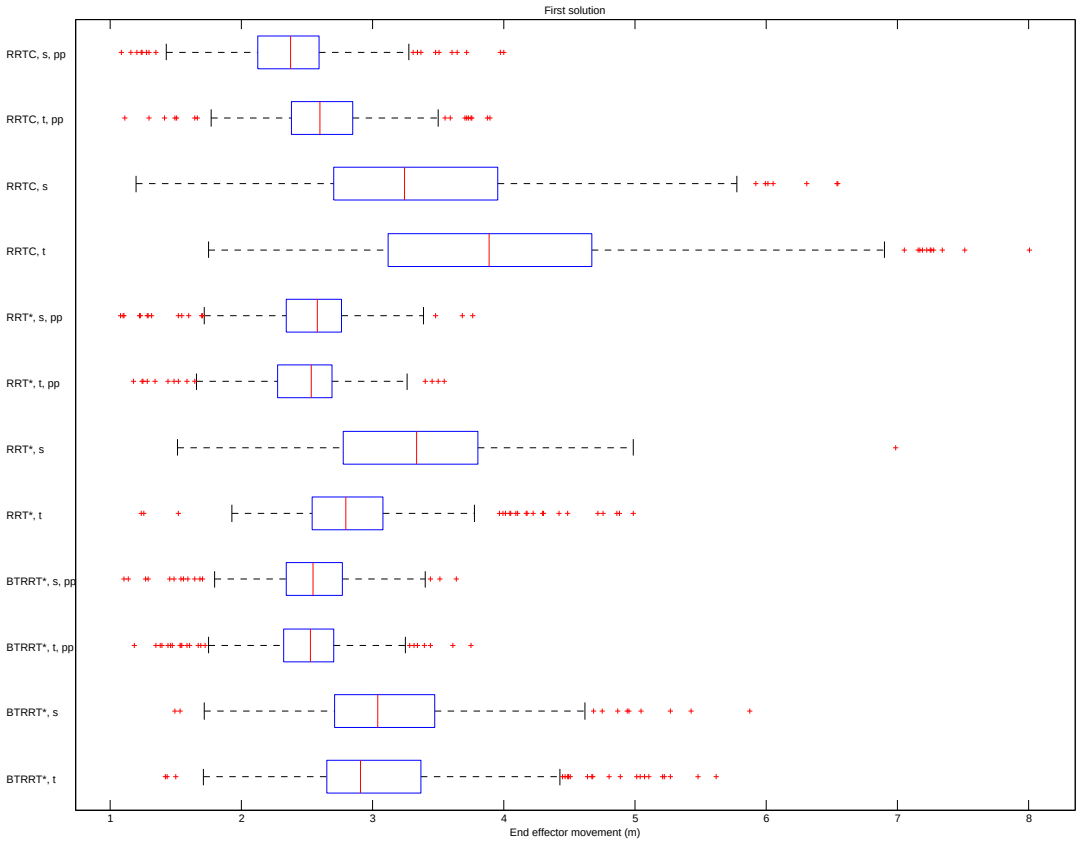


Figure 30: The end effector movements for the initial paths produced by different planners in scene 2.

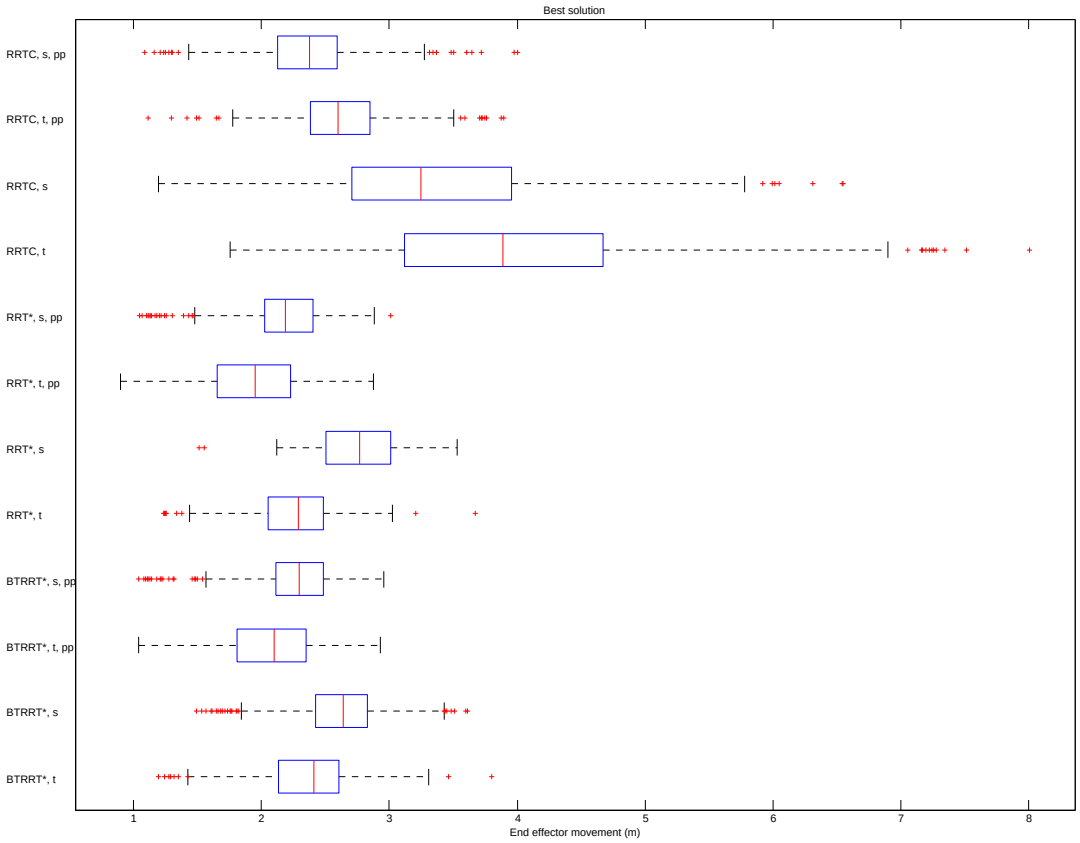


Figure 31: The end effector movements for the optimized paths produced by different planners in scene 2.

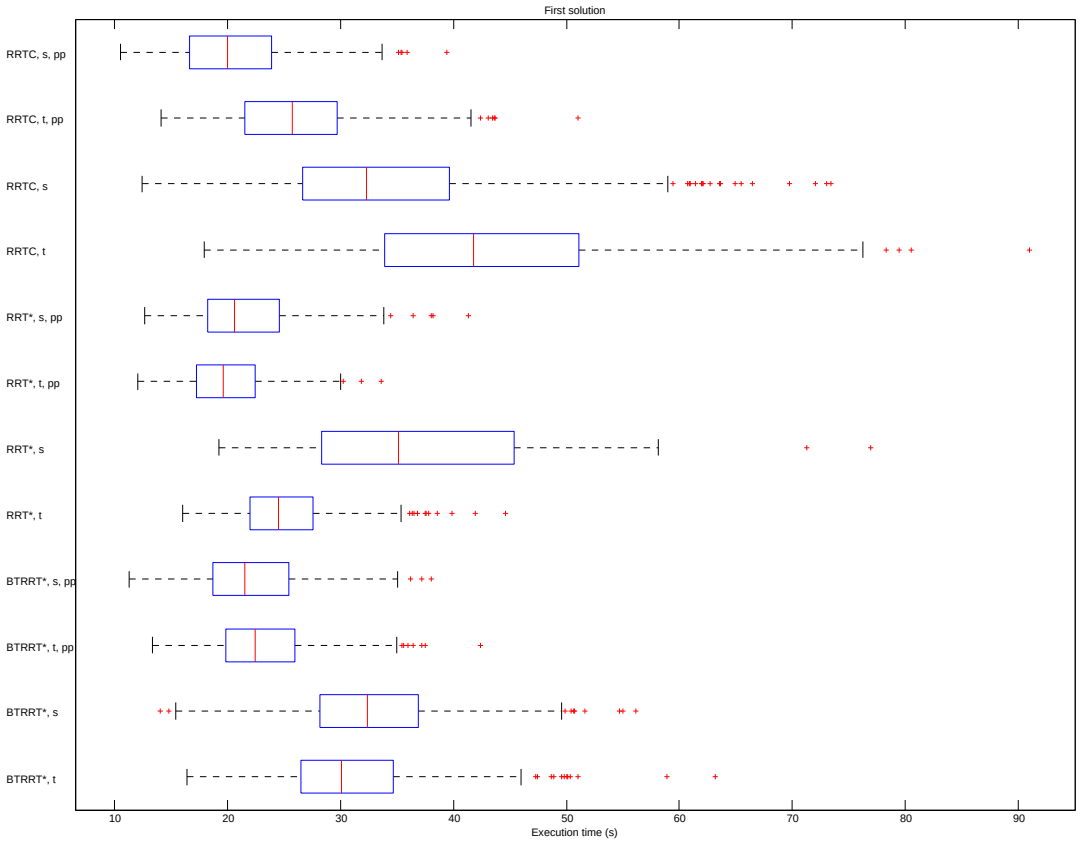


Figure 32: The execution times for the initial paths produced by different planners in scene 2.

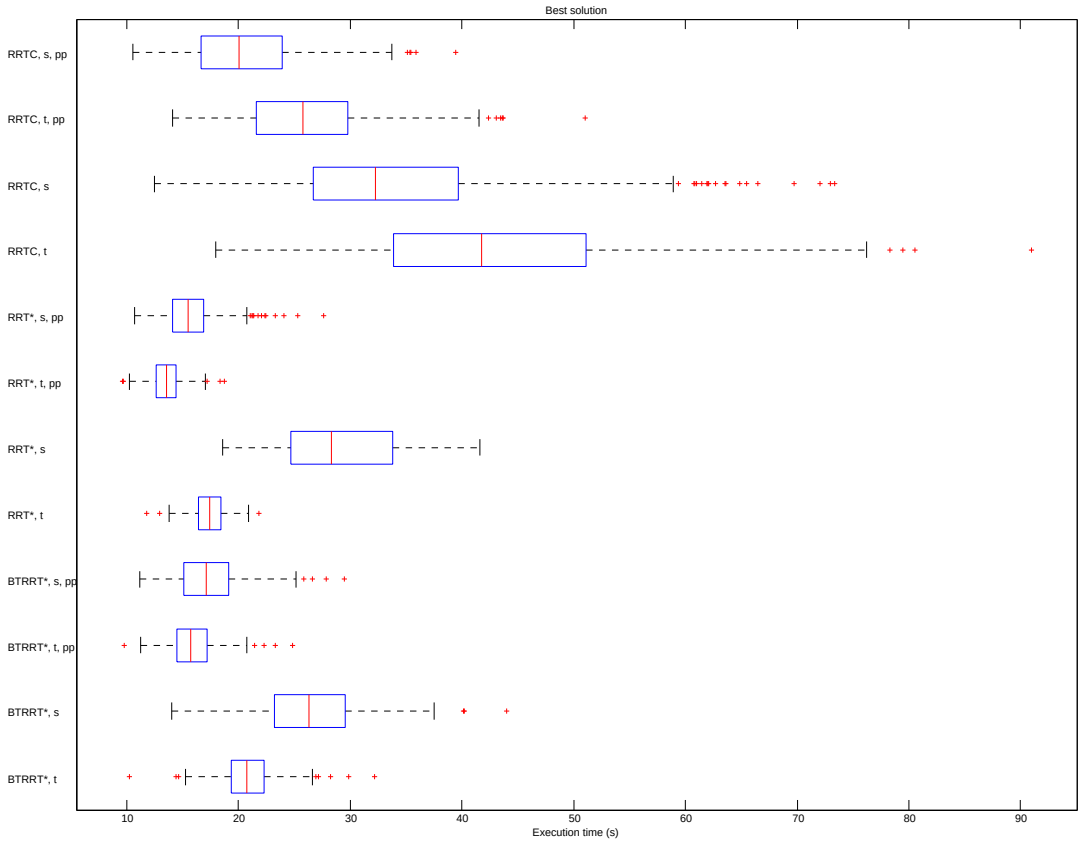


Figure 33: The execution times for the optimized paths produced by different planners in scene 2.

B Comparison of planners: tables

The planning algorithms were compared according to the time it takes to produce first solution, the end effector movement of the initial solution path, the execution time of the initial solution path, the end effector movement of the optimized path (after 20 seconds) and the execution time of the optimized path. Five number summaries (sample minimum, quartiles, sample maximum) of the results are presented in Table 13 and Table 14. The tested hypotheses and their associated p-values are presented in Table 15 and Table 16. The name of the algorithm (either 'RRTC', 'BTRRT*' or 'RRT*') is followed by the name of the metric which is followed by 'pp' if post processing was used.

Table 13: Results obtained from simulations on scene 1. Five number summaries of execution times and end effector movement for initial and optimized paths and the time taken to find first solution.

Planner	Measurement	Sample min	Q1	Median	Q2	Sample max
BTRRT* t	Solution time (s)	0.025678	0.43311	0.97895	3.5122	19.2464
	EE movement – init. (m)	1.3517	2.4587	2.7635	3.1321	5.1845
	EE movement – best (m)	1.1939	2.0953	2.3415	2.5586	3.5681
	Execution time – init. (s)	11.3684	20.8503	24.1569	28.8049	51.8445
	Execution time – best (s)	7.4959	15.8355	17.2071	18.7974	26.9741
BTRRT* t pp	Solution time (s)	0.043694	0.43946	1.0089	3.2618	19.02
	EE movement – init. (m)	1.1019	2.2828	2.5581	2.7795	5.4177
	EE movement – best (m)	1.0606	1.8391	2.1367	2.3789	3.0179
	Execution time – init. (s)	9.8467	16.5823	18.8096	21.9575	40.7025
	Execution time – best (s)	9.8467	13.2195	14.1498	15.1354	20.023
BTRRT* s	Solution time (s)	0.068832	2.223	4.246	7.2068	19.9711
	EE movement – init. (m)	1.2945	2.5247	2.8174	3.1542	4.7561
	EE movement – best (m)	1.2278	2.2239	2.4672	2.6913	3.6667
	Execution time – init. (s)	13.1718	21.8601	26.01	30.4169	47.5016
	Execution time – best (s)	11.8795	17.9391	20.2573	22.4261	36.8101
BTRRT* s pp	Solution time (s)	0.061124	2.1669	4.0831	6.9924	20.015
	EE movement – init. (m)	1.0302	2.2783	2.5474	2.7821	3.3694
	EE movement – best (m)	1.0165	1.9278	2.2409	2.4461	3.1097
	Execution time – init. (s)	10.9726	16.2639	18.9612	22.6777	35.9386
	Execution time – best (s)	10.9726	13.7502	15.023	16.7178	26.3968
RRT* t	Solution time (s)	0.01764	0.11975	0.25507	1.5677	19.272
	EE movement – init. (m)	1.4675	2.482	2.7395	2.9772	5.5544
	EE movement – best (m)	1.2628	2.0758	2.3251	2.624	4.608
	Execution time – init. (s)	10.0193	17.6018	19.6673	22.5935	38.8741
	Execution time – best (s)	7.8276	14.2835	15.2206	16.0985	21.9708
RRT* t pp	Solution time (s)	0.020543	0.12158	0.2504	1.4936	19.9364
	EE movement – init. (m)	1.1624	2.2623	2.511	2.7523	3.8082
	EE movement – best (m)	1.0191	1.8134	2.0846	2.3644	2.9867
	Execution time – init. (s)	6.8449	15.4192	17.0977	20.1806	36.5254
	Execution time – best (s)	6.8449	11.9923	12.9579	13.6452	19.5582
RRT* s	Solution time (s)	0.029077	0.14459	0.30619	0.89213	6.0266
	EE movement – init. (m)	1.4486	2.5025	2.7284	2.9631	4.8293
	EE movement – best (m)	1.3464	2.1808	2.3892	2.5415	3.3201
	Execution time – init. (s)	12.7918	18.5253	22.2877	27.3051	47.0435
	Execution time – best (s)	11.2635	16.1599	17.5268	19.0324	24.4849
RRT* s pp	Solution time (s)	0.02949	0.15139	0.31202	0.83138	10.3284
	EE movement – init. (m)	1.1345	2.2476	2.5312	2.7712	3.4879
	EE movement – best (m)	1.0855	1.9368	2.1298	2.3394	2.9998
	Execution time – init. (s)	11.8348	15.4599	17.6073	21.1079	33.262
	Execution time – best (s)	11.0133	12.8	13.5536	14.3639	19.6666
RRTC t	Solution time (s)	0.018794	0.10201	0.1667	0.25444	0.84841
	EE movement – init. (m)	1.7241	2.8703	3.4584	4.3674	9.2655
	EE movement – best (m)	1.7241	2.8703	3.4584	4.3674	9.2655
	Execution time – init. (s)	15.0231	26.6005	34.9568	43.4275	87.8178
	Execution time – best (s)	15.0231	26.6005	34.9568	43.4275	87.8178
RRTC t pp	Solution time (s)	0.018085	0.10014	0.16486	0.26324	1.1275
	EE movement – init. (m)	1.4013	2.2905	2.5308	2.7819	5.2898
	EE movement – best (m)	1.4013	2.2905	2.5308	2.7819	5.2898
	Execution time – init. (s)	12.6971	17.3432	20.749	25.0578	39.6289
	Execution time – best (s)	12.6971	17.3432	20.749	25.0578	39.6289
RRTC s	Solution time (s)	0.017401	0.064792	0.10974	0.15853	0.56032
	EE movement – init. (m)	1.3061	2.6377	3.034	3.6132	6.8314
	EE movement – best (m)	1.3061	2.6377	3.034	3.6132	6.8314
	Execution time – init. (s)	13.3368	23.8691	30.1572	38.1255	67.5321
	Execution time – best (s)	13.3368	23.8691	30.1572	38.1255	67.5321
RRTC s pp	Solution time (s)	0.018063	0.066884	0.11521	0.19577	0.63497
	EE movement – init. (m)	1.1161	2.2005	2.5165	2.7074	4.6494
	EE movement – best (m)	1.1161	2.2005	2.5165	2.7074	4.6494
	Execution time – init. (s)	11.4467	15.0582	17.3402	20.3175	33.0937
	Execution time – best (s)	11.4467	15.0582	17.3402	20.3175	33.0937

Table 14: Results obtained from simulations on scene 2. Five number summaries of execution times and end effector movement for initial and optimized paths and the time taken to find first solution.

Planner	Measurement	Sample min	Q1	Median	Q3	Sample max
BTRRT* t	Solution time (s)	0.1333	1.1037	2.2839	4.1904	17.1039
	EE movement – init. (m)	1.4217	2.6531	2.9105	3.3688	5.6167
	EE movement – best (m)	1.192	2.131	2.4115	2.6062	3.8032
	Execution time – init. (s)	16.4146	26.4562	30.0906	34.6534	63.1342
	Execution time – best (s)	10.2569	19.3772	20.7767	22.2863	32.1983
BTRRT* t pp	Solution time (s)	0.11243	1.2162	2.2125	4.3048	16.1731
	EE movement – init. (m)	1.1861	2.3205	2.529	2.7019	3.7514
	EE movement – best (m)	1.0375	1.8038	2.0969	2.3492	2.9268
	Execution time – init. (s)	13.402	19.8444	22.4632	25.8986	42.4046
	Execution time – best (s)	9.7587	14.443	15.701	17.1818	24.8507
BTRRT* s	Solution time (s)	0.1154	1.7101	3.3478	5.1408	19.3359
	EE movement – init. (m)	1.4924	2.7108	3.0383	3.474	5.8791
	EE movement – best (m)	1.4924	2.4258	2.6399	2.8248	3.61
	Execution time – init. (s)	14.028	28.1746	32.3011	36.832	56.122
	Execution time – best (s)	14.028	23.2501	26.2496	29.5613	44.023
BTRRT* s pp	Solution time (s)	0.17756	1.8665	3.1141	5.0253	19.1409
	EE movement – init. (m)	1.1054	2.3459	2.5482	2.7695	3.6422
	EE movement – best (m)	1.0393	2.1089	2.2953	2.4818	2.954
	Execution time – init. (s)	11.2785	18.7402	21.5301	25.4153	38.0607
	Execution time – best (s)	11.1575	15.0852	17.0778	19.123	29.4657
RRT* t	Solution time (s)	0.031305	0.30462	0.81603	2.3051	18.4321
	EE movement – init. (m)	1.2351	2.5427	2.7959	3.0794	4.99
	EE movement – best (m)	1.2351	2.0491	2.2841	2.4802	3.6734
	Execution time – init. (s)	16.0808	21.9821	24.4801	27.5903	44.6364
	Execution time – best (s)	11.7626	16.4049	17.4263	18.4085	21.8045
RRT* t pp	Solution time (s)	0.038532	0.28502	0.77034	2.3336	19.2215
	EE movement – init. (m)	1.1808	2.2715	2.5314	2.6905	3.5491
	EE movement – best (m)	0.8974	1.653	1.9449	2.2239	2.8731
	Execution time – init. (s)	12.0958	17.2788	19.652	22.4493	33.6316
	Execution time – best (s)	9.634	12.6315	13.5496	14.4185	18.7343
RRT* s	Solution time (s)	0.042875	0.11958	0.24293	0.40559	2.5342
	EE movement – init. (m)	1.5112	2.7725	3.3286	3.7946	6.9864
	EE movement – best (m)	1.5112	2.5001	2.7554	2.9991	3.5313
	Execution time – init. (s)	19.28	28.3584	34.989	44.8662	76.9032
	Execution time – best (s)	18.5561	24.6004	28.3334	33.7525	41.6385
RRT* s pp	Solution time (s)	0.027442	0.13073	0.27066	0.56634	8.8073
	EE movement – init. (m)	1.0807	2.3411	2.5751	2.7625	3.7618
	EE movement – best (m)	1.0446	2.0256	2.1829	2.3989	3.0096
	Execution time – init. (s)	12.6849	18.2397	20.6567	24.5958	41.2967
	Execution time – best (s)	10.7248	14.1173	15.47	16.8426	27.607
RRTC t	Solution time (s)	0.034323	0.11572	0.16	0.23905	1.1041
	EE movement – init. (m)	1.7535	3.1156	3.8826	4.668	8.0075
	EE movement – best (m)	1.7535	3.1156	3.8826	4.668	8.0075
	Execution time – init. (s)	17.9666	33.9025	41.7407	51.1169	90.9615
	Execution time – best (s)	17.9666	33.9025	41.7407	51.1169	90.9615
RRTC t pp	Solution time (s)	0.036235	0.11821	0.17842	0.24428	0.72894
	EE movement – init. (m)	1.1138	2.379	2.5968	2.8473	3.8963
	EE movement – best (m)	1.1138	2.379	2.5968	2.8473	3.8963
	Execution time – init. (s)	14.134	21.5221	25.7767	29.7302	51.0164
	Execution time – best (s)	14.134	21.5221	25.7767	29.7302	51.0164
RRTC s	Solution time (s)	0.01718	0.061411	0.099027	0.15399	0.57818
	EE movement – init. (m)	1.1958	2.7014	3.2462	3.9529	6.5463
	EE movement – best (m)	1.1958	2.7014	3.2462	3.9529	6.5463
	Execution time – init. (s)	12.472	26.6767	32.25	39.6622	73.3805
	Execution time – best (s)	12.472	26.6767	32.25	39.6622	73.3805
RRTC s pp	Solution time (s)	0.023321	0.059442	0.10248	0.1578	0.51294
	EE movement – init. (m)	1.0836	2.125	2.3754	2.5906	4.0012
	EE movement – best (m)	1.0836	2.125	2.3754	2.5906	4.0012
	Execution time – init. (s)	10.5548	16.6503	20.0053	23.8866	39.419
	Execution time – best (s)	10.5548	16.6503	20.0053	23.8866	39.419

Table 15: Conclusions drawn from planner tests in scene 1 and their two-tailed p values. The comparison symbols are shorthand "$A > B$" $\mapsto P(A > B) > P(B > A)$ and " = " means that such conclusion can not be drawn.

Compared measurement	Conclusion			p value
Test that RRT* variants improve solutions				
Execution Time	BTRRT* t init.	>	BTRRT* t best	1.687e-110
EE Movement	BTRRT* t init.	>	BTRRT* t best	1.8386e-50
Execution Time	BTRRT* t pp init.	>	BTRRT* t pp best	8.0313e-116
EE Movement	BTRRT* t pp init.	>	BTRRT* t pp best	3.1638e-47
Execution Time	RRT* t init.	>	RRT* t best	9.6123e-111
EE Movement	RRT* t init.	>	RRT* t best	1.9164e-40
Execution Time	RRT* t pp init.	>	RRT* t pp best	3.7308e-132
EE Movement	RRT* t pp init.	>	RRT* t pp best	1.2327e-45
Testing the effect of metrics				
BTRRT*				
Solution Time	BTRRT* t init.	<	BTRRT* s init.	1.06e-42
Execution Time	BTRRT* t init.	<	BTRRT* s init.	0.00029888
EE Movement	BTRRT* t init.	=	BTRRT* s init.	0.36525
Execution Time	BTRRT* t best	<	BTRRT* s best	1.2685e-46
EE Movement	BTRRT* t best	<	BTRRT* s best	1.9882e-07
Solution Time	BTRRT* t pp init.	<	BTRRT* s pp init.	6.7079e-40
Execution Time	BTRRT* t pp init.	=	BTRRT* s pp init.	0.8343
EE Movement	BTRRT* t pp init.	=	BTRRT* s pp init.	0.865
Execution Time	BTRRT* t pp best	<	BTRRT* s pp best	4.3244e-16
EE Movement	BTRRT* t pp best	<	BTRRT* s pp best	0.021926
RRT*				
Solution Time	RRT* t init.	=	RRT* s init.	0.85358
Execution Time	RRT* t init.	<	RRT* s init.	4.521e-13
EE Movement	RRT* t init.	=	RRT* s init.	0.98175
Execution Time	RRT* t best	<	RRT* s best	1.3952e-67
EE Movement	RRT* t best	=	RRT* s best	0.49655
Solution Time	RRT* t pp init.	=	RRT* s pp init.	0.71516
Execution Time	RRT* t pp init.	=	RRT* s pp init.	0.059382
EE Movement	RRT* t pp init.	=	RRT* s pp init.	0.64381
Execution Time	RRT* t pp best	<	RRT* s pp best	2.3943e-20
EE Movement	RRT* t pp best	=	RRT* s pp best	0.053144
RRT Connect				
Solution Time	RRTc t init.	>	RRTc s init.	2.5928e-20
Execution Time	RRTc t init.	>	RRTc s init.	8.7334e-10
EE Movement	RRTc t init.	>	RRTc s init.	9.3495e-12
Solution Time	RRTc t pp init.	>	RRTc s pp init.	2.4364e-12
Execution Time	RRTc t pp init.	>	RRTc s pp init.	5.5833e-29
EE Movement	RRTc t pp init.	>	RRTc s pp init.	0.013583
Testing the effect of post-processing				
BTRRT*				
Solution Time	BTRRT* t init.	=	BTRRT* t pp init.	0.98817
Execution Time	BTRRT* t init.	>	BTRRT* t pp init.	9.0266e-50
EE Movement	BTRRT* t init.	>	BTRRT* t pp init.	5.0361e-19
Execution Time	BTRRT* t best	>	BTRRT* t pp best	8.8722e-96
EE Movement	BTRRT* t best	>	BTRRT* t pp best	4.0397e-16
Solution Time	BTRRT* s init.	=	BTRRT* s pp init.	0.50596
Execution Time	BTRRT* s init.	>	BTRRT* s pp init.	1.3879e-61
EE Movement	BTRRT* s init.	>	BTRRT* s pp init.	1.706e-24
Execution Time	BTRRT* s best	>	BTRRT* s pp best	6.248e-89
EE Movement	BTRRT* s best	>	BTRRT* s pp best	6.0595e-23
RRT*				
Solution Time	RRT* t init.	=	RRT* t pp init.	0.81661
Execution Time	RRT* t init.	>	RRT* t pp init.	6.2856e-25
EE Movement	RRT* t init.	>	RRT* t pp init.	1.5453e-18
Execution Time	RRT* t best	>	RRT* t pp best	6.0718e-110
EE Movement	RRT* t best	>	RRT* t pp best	1.4842e-22
Solution Time	RRT* s init.	=	RRT* s pp init.	0.78952
Execution Time	RRT* s init.	>	RRT* s pp init.	1.1056e-39
EE Movement	RRT* s init.	>	RRT* s pp init.	3.1064e-18
Execution Time	RRT* s best	>	RRT* s pp best	3.8188e-127
EE Movement	RRT* s best	>	RRT* s pp best	3.7209e-30
RRT Connect				
Solution Time	RRTc t init.	=	RRTc t pp init.	0.9403
Execution Time	RRTc t init.	>	RRTc t pp init.	2.0334e-94
EE Movement	RRTc t init.	>	RRTc t pp init.	4.667e-77
Solution Time	RRTc s init.	=	RRTc s pp init.	0.086819
Execution Time	RRTc s init.	>	RRTc s pp init.	2.9094e-117
EE Movement	RRTc s init.	>	RRTc s pp init.	1.8142e-65
Comparing different algorithms				
Solution time				
Solution Time	RRTc t init.	<	RRT* t init.	5.1617e-16
Solution Time	RRT* t init.	<	BTRRT* t init.	2.0973e-27
Solution Time	RRTc t init.	<	BTRRT* t init.	1.5883e-118

Solution Time	RRTc s init.	<	RRT* s init.	2.8666e-67
Solution Time	RRT* s init.	<	BTRRT* s init.	2.004e-119
Solution Time	RRTc s init.	<	BTRRT* s init.	4.3629e-156
Best solution				
Execution Time	BTRRT* s best	<	RRTc s init.	7.53e-86
Execution Time	BTRRT* s best	>	RRT* s best	1.5321e-38
Execution Time	RRTc s init.	>	RRT* s best	3.4288e-137
EE Movement	BTRRT* s best	<	RRTc s init.	3.1065e-63
EE Movement	BTRRT* s best	>	RRT* s best	1.0362e-05
EE Movement	RRTc s init.	>	RRT* s best	8.2227e-92
Execution Time	BTRRT* s pp best	<	RRTc s pp init.	8.2569e-27
Execution Time	BTRRT* s pp best	>	RRT* s pp best	1.6798e-39
Execution Time	RRTc s pp init.	>	RRT* s pp best	2.1723e-95
EE Movement	BTRRT* s pp best	<	RRTc s pp init.	4.2109e-25
EE Movement	BTRRT* s pp best	>	RRT* s pp best	0.0060387
EE Movement	RRTc s pp init.	>	RRT* s pp best	3.3885e-42
Execution Time	BTRRT* t best	<	RRTc t init.	7.1518e-150
Execution Time	BTRRT* t best	>	RRT* t best	3.4326e-57
Execution Time	RRTc t init.	>	RRT* t best	9.5943e-161
EE Movement	BTRRT* t best	<	RRTc t init.	2.6649e-106
EE Movement	BTRRT* t best	=	RRT* t best	0.44994
EE Movement	RRTc t init.	>	RRT* t best	9.5268e-101
Execution Time	BTRRT* t pp best	<	RRTc t pp init.	1.0672e-129
Execution Time	BTRRT* t pp best	>	RRT* t pp best	9.0331e-48
Execution Time	RRTc t pp init.	>	RRT* t pp best	9.7003e-156
EE Movement	BTRRT* t pp best	<	RRTc t pp init.	4.2561e-50
EE Movement	BTRRT* t pp best	=	RRT* t pp best	0.13378
EE Movement	RRTc t pp init.	>	RRT* t pp best	6.3633e-55
First solution				
Execution Time	BTRRT* s init.	<	RRTc s init.	1.6851e-15
Execution Time	BTRRT* s init.	>	RRT* s init.	3.4465e-16
Execution Time	RRTc s init.	>	RRT* s init.	1.52e-45
EE Movement	BTRRT* s init.	<	RRTc s init.	1.7941e-11
EE Movement	BTRRT* s init.	>	RRT* s init.	0.00095087
EE Movement	RRTc s init.	>	RRT* s init.	3.0762e-22
Execution Time	BTRRT* s pp init.	>	RRTc s pp init.	1.7894e-09
Execution Time	BTRRT* s pp init.	>	RRT* s pp init.	2.5966e-05
Execution Time	RRTc s pp init.	<	RRT* s pp init.	0.041066
EE Movement	BTRRT* s pp init.	>	RRTc s pp init.	0.024566
EE Movement	BTRRT* s pp init.	=	RRT* s pp init.	0.62547
EE Movement	RRTc s pp init.	=	RRT* s pp init.	0.069438
Execution Time	BTRRT* t best	<	RRTc t init.	7.1518e-150
Execution Time	BTRRT* t init.	>	RRT* t init.	1.1062e-39
Execution Time	RRTc t init.	>	RRT* t init.	1.952e-112
EE Movement	BTRRT* t init.	<	RRTc t init.	2.8905e-37
EE Movement	BTRRT* t init.	=	RRT* t init.	0.059787
EE Movement	RRTc t init.	>	RRT* t init.	3.0432e-48
Execution Time	BTRRT* t pp init.	<	RRTc t pp init.	3.7531e-08
Execution Time	BTRRT* t pp init.	>	RRT* t pp init.	1.6553e-12
Execution Time	RRTc t pp init.	>	RRT* t pp init.	7.4884e-32
EE Movement	BTRRT* t pp init.	=	RRTc t pp init.	0.94143
EE Movement	BTRRT* t pp init.	=	RRT* t pp init.	0.24347
EE Movement	RRTc t pp init.	=	RRT* t pp init.	0.20799

Table 16: Conclusions drawn from planner tests in scene 2 and their p values. The comparison symbols are shorthand "$A > B$" $\mapsto P(A > B) > P(B > A)$ and " = " means that such conclusion can not be drawn.

Compared measurement	Conclusion			p value
Test that RRT* variants improve solutions				
Execution Time	BTRRT* t init.	>	BTRRT* t best	1.2316e-139
EE Movement	BTRRT* t init.	>	BTRRT* t best	1.1576e-81
Execution Time	BTRRT* t pp init.	>	BTRRT* t pp best	2.6932e-126
EE Movement	BTRRT* t pp init.	>	BTRRT* t pp best	1.8655e-68
Execution Time	RRT* t init.	>	RRT* t best	2.4207e-78
EE Movement	RRT* t init.	>	RRT* t best	2.7183e-44
Execution Time	RRT* t pp init.	>	RRT* t pp best	1.5878e-119
EE Movement	RRT* t pp init.	>	RRT* t pp best	6.984e-69
Testing the effect of metrics				
BTRRT*				
Solution Time	BTRRT* t init.	<	BTRRT* s init.	4.3101e-08
Execution Time	BTRRT* t init.	<	BTRRT* s init.	2.5507e-05
EE Movement	BTRRT* t init.	<	BTRRT* s init.	0.0063026
Execution Time	BTRRT* t best	<	BTRRT* s best	1.0253e-84
EE Movement	BTRRT* t best	<	BTRRT* s best	2.79e-28
Solution Time	BTRRT* t pp init.	<	BTRRT* s pp init.	1.7415e-07
Execution Time	BTRRT* t pp init.	>	BTRRT* s pp init.	0.0014774
EE Movement	BTRRT* t pp init.	=	BTRRT* s pp init.	0.05815
Execution Time	BTRRT* t pp best	<	BTRRT* s pp best	7.2067e-15
EE Movement	BTRRT* t pp best	<	BTRRT* s pp best	1.1859e-19
RRT*				
Solution Time	RRT* t init.	>	RRT* s init.	8.9123e-11
Execution Time	RRT* t init.	<	RRT* s init.	1.143e-19
EE Movement	RRT* t init.	<	RRT* s init.	2.4734e-07
Execution Time	RRT* t best	<	RRT* s best	1.3471e-36
EE Movement	RRT* t best	<	RRT* s best	1.2747e-17
Solution Time	RRT* t pp init.	>	RRT* s pp init.	2.0912e-27
Execution Time	RRT* t pp init.	<	RRT* s pp init.	2.2555e-06
EE Movement	RRT* t pp init.	<	RRT* s pp init.	0.0040486
Execution Time	RRT* t pp best	<	RRT* s pp best	1.4496e-48
EE Movement	RRT* t pp best	<	RRT* s pp best	1.1851e-18
RRT Connect				
Solution Time	RRTc t init.	>	RRTc s init.	1.0404e-36
Execution Time	RRTc t init.	>	RRTc s init.	1.3352e-29
EE Movement	RRTc t init.	>	RRTc s init.	7.9139e-19
Solution Time	RRTc t pp init.	>	RRTc s pp init.	7.9188e-39
Execution Time	RRTc t pp init.	>	RRTc s pp init.	1.9261e-45
EE Movement	RRTc t pp init.	>	RRTc s pp init.	4.3395e-24
Testing the effect of post-processing				
BTRRT*				
Solution Time	BTRRT* t init.	=	BTRRT* t pp init.	0.75293
Execution Time	BTRRT* t init.	>	BTRRT* t pp init.	1.0736e-82
EE Movement	BTRRT* t init.	>	BTRRT* t pp init.	3.6374e-57
Execution Time	BTRRT* t best	>	BTRRT* t pp best	3.6869e-129
EE Movement	BTRRT* t best	>	BTRRT* t pp best	6.5656e-34
Solution Time	BTRRT* s init.	=	BTRRT* s pp init.	0.86794
Execution Time	BTRRT* s init.	>	BTRRT* s pp init.	4.8086e-102
EE Movement	BTRRT* s init.	>	BTRRT* s pp init.	2.7667e-65
Execution Time	BTRRT* s best	>	BTRRT* s pp best	6.025e-131
EE Movement	BTRRT* s best	>	BTRRT* s pp best	2.057e-53
RRT*				
Solution Time	RRT* t init.	=	RRT* t pp init.	0.78405
Execution Time	RRT* t init.	>	RRT* t pp init.	3.2494e-40
EE Movement	RRT* t init.	>	RRT* t pp init.	1.0613e-25
Execution Time	RRT* t best	>	RRT* t pp best	3.1952e-94
EE Movement	RRT* t best	>	RRT* t pp best	3.2348e-22
Solution Time	RRT* s init.	=	RRT* s pp init.	0.42011
Execution Time	RRT* s init.	>	RRT* s pp init.	8.972e-29
EE Movement	RRT* s init.	>	RRT* s pp init.	4.4617e-20
Execution Time	RRT* s best	>	RRT* s pp best	3.304e-39
EE Movement	RRT* s best	>	RRT* s pp best	9.494e-25
RRT Connect				
Solution Time	RRTc t init.	=	RRTc t pp init.	0.3119
Execution Time	RRTc t init.	>	RRTc t pp init.	1.7958e-98
EE Movement	RRTc t init.	>	RRTc t pp init.	9.8058e-100
Solution Time	RRTc s init.	=	RRTc s pp init.	0.9057
Execution Time	RRTc s init.	>	RRTc s pp init.	2.5781e-99
EE Movement	RRTc s init.	>	RRTc s pp init.	3.9898e-87
Comparing different algorithms				
Solution time				
Solution Time	RRTc t init.	<	RRT* t init.	1.8548e-55
Solution Time	RRT* t init.	<	BTRRT* t init.	3.6173e-23
Solution Time	RRTc t init.	<	BTRRT* t init.	2.9988e-158

Solution Time	RRTc s init.	<	RRT* s init.	2.524e-15
Solution Time	RRT* s init.	<	BTRRT* s init.	3.8869e-36
Solution Time	RRTc s init.	<	BTRRT* s init.	3.7635e-162
Best solution				
Execution Time	BTRRT* s best	<	RRTc s init.	1.8458e-35
Execution Time	BTRRT* s best	<	RRT* s best	0.00030413
Execution Time	RRTc s init.	>	RRT* s best	0.00047556
EE Movement	BTRRT* s best	<	RRTc s init.	4.1867e-50
EE Movement	BTRRT* s best	<	RRT* s best	0.013982
EE Movement	RRTc s init.	>	RRT* s best	7.4429e-10
Execution Time	BTRRT* s pp best	<	RRTc s pp init.	4.0339e-25
Execution Time	BTRRT* s pp best	>	RRT* s pp best	2.2177e-17
Execution Time	RRTc s pp init.	>	RRT* s pp best	4.0245e-51
EE Movement	BTRRT* s pp best	<	RRTc s pp init.	0.00054526
EE Movement	BTRRT* s pp best	>	RRT* s pp best	1.7959e-06
EE Movement	RRTc s pp init.	>	RRT* s pp best	9.1558e-14
Execution Time	BTRRT* t best	<	RRTc t init.	2.4299e-151
Execution Time	BTRRT* t best	>	RRT* t best	1.1254e-73
Execution Time	RRTc t init.	>	RRT* t best	1.353e-113
EE Movement	BTRRT* t best	<	RRTc t init.	1.1162e-126
EE Movement	BTRRT* t best	>	RRT* t best	1.7617e-05
EE Movement	RRTc t init.	>	RRT* t best	5.5914e-95
Execution Time	BTRRT* t pp best	<	RRTc t pp init.	2.8888e-140
Execution Time	BTRRT* t pp best	>	RRT* t pp best	2.3624e-68
Execution Time	RRTc t pp init.	>	RRT* t pp best	4.6362e-148
EE Movement	BTRRT* t pp best	<	RRTc t pp init.	1.2932e-85
EE Movement	BTRRT* t pp best	>	RRT* t pp best	2.5056e-06
EE Movement	RRTc t pp init.	>	RRT* t pp best	5.7059e-95
First solution				
Execution Time	BTRRT* s init.	=	RRTc s init.	0.57021
Execution Time	BTRRT* s init.	<	RRT* s init.	0.0015067
Execution Time	RRTc s init.	<	RRT* s init.	0.01348
EE Movement	BTRRT* s init.	<	RRTc s init.	2.4157e-05
EE Movement	BTRRT* s init.	<	RRT* s init.	0.008584
EE Movement	RRTc s init.	=	RRT* s init.	0.92966
Execution Time	BTRRT* s pp init.	>	RRTc s pp init.	1.7799e-07
Execution Time	BTRRT* s pp init.	=	RRT* s pp init.	0.14401
Execution Time	RRTc s pp init.	<	RRT* s pp init.	0.00023521
EE Movement	BTRRT* s pp init.	>	RRTc s pp init.	1.7334e-17
EE Movement	BTRRT* s pp init.	=	RRT* s pp init.	0.87178
EE Movement	RRTc s pp init.	<	RRT* s pp init.	3.8717e-16
Execution Time	BTRRT* t best	<	RRTc t init.	2.4299e-151
Execution Time	BTRRT* t init.	>	RRT* t init.	3.8269e-37
Execution Time	RRTc t init.	>	RRT* t init.	6.1396e-77
EE Movement	BTRRT* t init.	<	RRTc t init.	3.3723e-48
EE Movement	BTRRT* t init.	>	RRT* t init.	0.00014704
EE Movement	RRTc t init.	>	RRT* t init.	6.3664e-46
Execution Time	BTRRT* t pp init.	<	RRTc t pp init.	4.9824e-17
Execution Time	BTRRT* t pp init.	>	RRT* t pp init.	1.0198e-22
Execution Time	RRTc t pp init.	>	RRT* t pp init.	1.7846e-53
EE Movement	BTRRT* t pp init.	<	RRTc t pp init.	3.3015e-05
EE Movement	BTRRT* t pp init.	=	RRT* t pp init.	0.33201
EE Movement	RRTc t pp init.	>	RRT* t pp init.	5.5192e-07

C Statistical methods

This appendix briefly explains some of the statistic methods used. Explanation of these methods can also be found in statistics textbooks, but for the convenience of a reader unfamiliar with the techniques they are also summarized here. Familiarity with statistical testing in general is assumed.

C.1 Mann-Whitney U-test

The Mann-Whitney U-test can be used instead of Student's t-ratio when the assumption of normality of the measurements is questionable or the measurements fail to achieve interval scaling [106, pp. 424–429]. It is applicable to cases where the two samples have been independently drawn. The test statistic U can be computed by taking samples A and B and sorting $A \cup B$ in ascending order. The U statistic is the number of measurement pairs where measurement from A precedes measurement from B. The variable U' corresponds to the number of pairs where measurement from B precedes measurement from A. Under the null hypothesis of the test, it is expected that measurements from A precede measurements from B as many times as measurements from B precede measurements from A, that is $P(A < B) = P(B < A)$ and $U = U'$.

If the null hypothesis does not hold, the U statistic will deviate from its expected value. For small sample sizes, the distribution of the U statistic has been tabulated and for large sample sizes it can be approximated using normal distribution. In one-tailed test, the values of either U or $U\prime$ are compared to the lower α quantile of the distribution. In two-tailed test the minimum of U and U' is compared to the $\alpha/2$ quantile.[107, p. 274]

As it is inconvenient to calculate the U statistic as described above, an equivalent method is used. The ranks of the first sample are summed and the U statistic is computed using

$$U = N_1 N_2 + \frac{N_1(N_1 + 1)}{2} - R_1 \tag{23}$$

where N_i are the sample sizes and R_i is the sum of the ranks of sample i. The value of U' can be computed using

$$U + U' = N_1 N_2. \tag{24}$$

If there are measurements that are tied for the same rank, the U statistic can be modified to take this into account. Not correcting for the presence of ties leads to more conservative test.

When the sample sizes are large enough, the U statistic is normally distributed with the expected value

$$U_E = \frac{N_1(N_1 + N_2 + 1)}{2} \tag{25}$$

and standard error

$$s_U = \sqrt{\frac{N_1 N_2(N_1 + N_2 + 1)}{12}}. \tag{26}$$

The variable can be normalized and the tables for standard normal distribution used to provide the critical values.

C.2 Test for comparison of correlation coefficients

Let there be two independent samples from two-dimensional normal distributions with sizes n_1 and n_2. Pearson's correlation coefficients for the two populations (ρ_1 and ρ_2) can be compared using Fisher transformation of the sample correlation coefficients [108, p. 185]. Let r_i be the correlation coefficient for sample i. Under the null hypothesis $H_0 : \rho_1 = \rho_2$ the random variable

$$z_k = \frac{1}{2} \log \left(\frac{1 + r_i}{1 - r_i} \right) \tag{27}$$

is approximately normally distributed with mean μ_z and variance

$$\sigma_k^2 = \frac{1}{n_i - 3} \tag{28}$$

provided the sample sizes n_i are large enough.

Because of the independence of z_1 and z_2 the test statistic

$$v = \frac{z_1 - z_2}{\sqrt{\frac{1}{n_1 - 3} + \frac{1}{n_2 - 3}}} \tag{29}$$

is approximately normally distributed with $v \sim N(0, 1)$. By comparing the computed value of this test statistic to the critical values obtained from the cumulative distribution function of the standardized normal distribution, either one or two-tailed tests can be performed.