AN INFORMATION THEORETICAL INCREMENTAL APPROACH TO SENSOR-BASED MOTION PLANNING FOR EYE-IN-HAND SYSTEMS

by

Yong Yu

B.Sc., Xi'an Jiaotong University, China, 1984
M.Sc., Northeast University, China, 1987

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY
in the School of
Engineering Science

SIMON FRASER UNIVERSITY
September 2000

APPROVAL

Name: Yong Yu

Degree: Doctor of Philosophy

Title of thesis: An information theoretical incremental approach to sensor-based motion planning for eye-in-hand systems

Examining Committee:

Dr. W. Gruver, Chair
Professor, Engineering Science
Simon Fraser University

Dr. K. Gupta, Senior Supervisor
Professor, Engineering Science
Simon Fraser University

Dr. B. Bhattacharya, Supervisor
Professor, Computer Science
Simon Fraser University

Dr. Z. Li, Supervisor,
Professor, Computer Science,
Simon Fraser University

Dr. S. Payandeh, SFU Examiner
Associate Professor, Engineering Science
Simon Fraser University

Dr. S. Hutchinson, External Examiner,
Associate Professor,
Electrical and Computer Engineering,
University of Illinois, Urbana-Champaign

Date Approved: _____

Abstract

Our research is concerned with sensor-based motion planning (MP) for articulated robot arms with many degrees of freedom. The articulated robot arm (called robot from here on), equipped with a range sensor on the end-effector of the robot that gives the distances of the objects from the sensor, is required to plan and execute collision-free motions in an environment initially unknown to the robot.

In our view, sensor-based MP can be decomposed into two key sub-problems: (i) *view planning*, i.e. to determine the next sensing action, including which region to scan and from where to scan this region, and (ii) *C-space expansion*, i.e. to incrementally expand the C-space (configuration space) representation. These two sub-problems are essentially repeatedly solved in an interleaved fashion in the sensor-based MP.

We approach the view planning problem from an information theoretical point of view, introduce and develop a novel concept of C-space entropy which characterizes the uncertainty of unknown C-space. Sensing actions reduce the entropy of the C-space. In each iteration of the planning process, the next scanning action is determined by maximizing the entropy reduction, or equivalently, gain in information.

Because of the high dimensionality of the C-space of a typical robot manipulator, a roadmap (Probabilistic Roadmap specifically) is adopted for C-space representation. We call our approach *sensor-based incremental construction of probabilistic roadmap* (SBIC-PRM). The crux of C-space expansion is to *incrementally* construct a roadmap as the sensor senses new free space in the physical environment. The evolving roadmap reflects the connectivity of known free C-space, $\mathcal{C}_{free}$, within which the robot carries out its motion to further sense the physical environment. The $\mathcal{C}_{free}$ and the roadmap expand as the sensor senses new free regions in the physical space. This process is repeated until certain conditions are met. For example, a final goal is reachable from (one of the landmarks in) the roadmap, i. e. a

simple planner in the algorithm finds a collision-free path from one of the landmarks to the goal, or the goal is declared unreachable. We ensure that the new landmarks lie in the new free regions in the C-space, $\Delta\mathcal{C}_{free}$, that corresponds to the additional free regions in the physical space, $\Delta\mathcal{P}_{free}$, obtained in the scan. This guarantees that the landmarks in the roadmap remains uniformly distributed in the C-space.

Our Eye-in-Hand system is implemented on a PUMA 560 robot with a range scanner mounted on its wrist. We present extensive experimental results with our sensor-based planner running on this real test-bed. The robot is started in unknown and cluttered environments. Typically, the planner is able to reach (planning as it senses) the goal configurations in about 7 - 25 scans (depending on the scene complexity), while avoiding collisions with the obstacles throughout. The experimental results show the efficacy of the C-space entropy concept in exploring the environment efficiently, and the ability of the incremental roadmap construction to keep the roadmap from degenerating.

Acknowledgments

I would first like to thank my supervisor, Dr. Kamal Gupta. He has spent many an hour to discuss this entire project with me. I can not convey how thankful I am to him.

I would also like to thank my supervisory committee, Dr. B. Bhattacharya and Dr. Z. Li. Discussions with them were extremely valuable and improved my project a many fold!

I would also like to thank Dr. J. Ahuactzin, Dr. M. Greenspan and Dr. M. Mehrandezh for their stimulating discussions.

I would also like to thank Gilbert Soucy for the scanner development, Dr. J. Llyod for prompt response of my questions on RCCL.

I would also like to thank both faculty and staff in the School of Engineering Science. Everyone has always been quite happy to answer any questions that I have had.

I would also like to thank all of the graduate students that I have come to know throughout my at SFU. In particular, they include Maria Amezquita, Ian Gipson, Wuilbert Jaramillo, Derek Jung, Pengpeng Wang and Ruiquan Zhang. Every one of these people have helped me in my studies.

I would also like to thank Mrs. Calvert to proof-read my thesis.

Last, but certainly not least, I wish to thank my wife, my parents and my in-laws. Their encouragement, support and patience are extremely valuable.

Contents

List of Tables

List of Figures

Nomenclature

$\mathcal{A}$:	the robot (including all the physical parts).
$\mathcal{AS}$:	the combined robot and sensor system.
$\mathcal{A}(q)$:	the physical space occupied by the entire robot (including the body of the sensor) at configuration q.
$\mathcal{A}_{unk}(q)$:	the unknown part of the physical space occupied by the entire robot (including the body of the sensor) at configuration q.
$\mathcal{C}_{\mathcal{A}}$:	the robot configuration spanned by $n_{\mathcal{A}}$.
$\mathcal{C}_{\mathcal{AS}}$:	the configuration space (C-space) of the entire robot and sensor system.
$\mathcal{C}_{\mathcal{S}}$:	the sensor configuration space.
$\mathcal{C}_{\tilde{\mathcal{S}}_{int}}$:	the configuration sub-space spanned by quasi-internal sensing *dofs*.
$\mathcal{C}_{free}$:	the entire free C-space.
$\mathcal{C}_{free}^{(i)}$:	the corresponding free C-space (an open set) at a certain stage i of the planning process.
$\Delta\mathcal{C}_{free}^{(i)}$:	the *additional* increment in free C-space due to the ith scan.
$\mathcal{C}_{free+}(m)$:	the region which landmarks are placed, which can be viewed as free C-space enlarged by a layer.
$\mathcal{C}_{obs}$:	the obstacles in C-space.
$\mathcal{C}_{obs}^{(i)}$:	the obstacles (a closed set) in C-space at a certain stage i of the planning process.
$\mathcal{C}_{unk}^{(i)}$:	the unknown C-space at a certain stage i of the planning.
$C(q)$:	the status (obstacle or free) of configuration q.
$\mathcal{E}$:	the set of edges of the roadmap.
$E(x)$:	the expected value of random variable x.

$G_e(x)$:	the exploratory component of an objective function for searching the next place to scan.
$G_g(x)$:	the goal (greedy) component of an objective function for searching the next place to scan.
$g_q(x)$:	an intermediate function, which is the expected contribution of a configuration q to the C-space entropy reduction when point x is scanned.
$H()$:	the entropy.
$\underset{A \Rightarrow 0}{\Delta} H(\mathcal{C})$:	the difference in C-space entropy before sensing point A (or region A) and after point A (or region A) becomes free.
$\underset{A \Rightarrow 1}{\Delta} H(\mathcal{C})$:	the difference in C-space entropy before sensing point A (or region A) and point A (or region A) because obstacle.
$\widetilde{H}(\mathcal{C})$:	the approximated entropy of C-space.
$n_{\mathcal{A}}$:	the degrees of freedom (*dofs*) of the robot.
$n_{\mathcal{S}_{ext}}$:	the external sensing *dofs*.
$n_{\mathcal{S}_{int}}$:	the internal sensing *dofs*.
$n_{\mathring{\mathcal{S}}_{int}}$:	the quasi-internal *dofs*.
$n_{\mathcal{S}}$:	the sensing dofs.
$\mathcal{P}$:	the physical space (the 3-D Euclidean space).
$\mathcal{P}_{free}$:	the entire free physical space.
$\mathcal{P}_{free}^{(0)}$:	the initial free region for the robot.
$\mathcal{P}_{obs}$:	the obstacles in physical space.
$\mathcal{P}_{free}^{(i)}$:	the cumulative free physical space acquired by the sensor, at a certain stage i of the planning.
$\mathcal{P}_{obs}^{(i)}$:	the obstacles seen by the sensor, at stage i of the planning.
$\mathcal{P}_{unk}^{(i)}$:	the physical space remaining unknown, at stage i of the planning.
$\Delta\mathcal{P}_{free}^{(i)}$:	the *additional* increment in free physical space due to the ith scan of physical space.
$q_{\mathcal{A}}$:	a robot configuration.
$q_{\mathcal{AS}}$:	a configuration for the entire (robot+sensor) system.
$q_{\mathcal{S}}$:	a sensor configuration.
q_l:	the configuration of landmark.
q_0:	the initial robot configuration.
$\mathcal{R}$:	the roadmap.
$\mathcal{R}(q_0)$:	the roadmap containing the initial robot configuration.

$\mathcal{S}$:	the sensor (the reference frame only).
$\mathcal{L}$:	the set of landmarks.
$\mathcal{L}_b$:	the set of black landmarks.
$\mathcal{L}_g$:	the set of gray landmarks.
$\mathcal{L}_w$:	the set of white landmarks.
l:	a landmark.
l_G:	a gray landmark which is closest to the goal and more than m_g fraction of the robot is in free space when the robot takes this configuration.
m_g:	see l_G.
$\mathcal{V}_{\mathcal{S}}(q_{\mathcal{S}})$:	the sensed free region (an open set) when the robot scans at configuration $q_{\mathcal{S}}$.
$\partial\mathcal{V}_{\mathcal{S}}(q_{\mathcal{S}})$:	the boundary of the region $\mathcal{V}_{\mathcal{S}}(q_{\mathcal{S}})$.
$\partial\mathcal{V}_{\mathcal{S}_{obs}}(q_{\mathcal{S}})$:	the obstacle surface in $\partial\mathcal{V}_{\mathcal{S}}(q_{\mathcal{S}})$.
x_{max}:	the point in physical space at which the information gain density function is maximized.
$\phi(\mathcal{B})$:	the number of points in a set $\mathcal{B}$ in a Poisson point process.
λ:	the degree of cluttering of the obstacles in a Poisson point process.
$\pi_{[q_0,q_1]}$:	a path from q_0 to q_1, including q_0 and q_1.
$\pi_{[q_0,q_1)}$:	a path from q_0 to q_1, including q_0 but excluding q_1.
ρ:	the density of the landmarks.
$\mathcal{X}(x)$:	C-zone of x.

Glossary

Configuration: A configuration of an object is a specification of the position of every point in this object relative to a fixed reference frame. For an articulated robot, the configuration consists of all the joint values of the robot.

C-space: The space of all the configurations.

Complete algorithm: An algorithm is complete for a problem if it is guaranteed, for all instances of the problem, to find a solution when one exists or to return failure other wise.

C-space entropy: The entropy of an unknown C-space.

C-zone of x: A set in C-space. If the robot takes a configuration in this set, then the robot intersects x in the physical space.

Explorable Configuration: A configuration q is explorable if $\mathcal{A}(q) \subset \bigcup_{q \in \mathcal{C}_{reach}} \mathcal{V}_{\mathcal{S}}(q)$ ($q \in$ known $\mathcal{C}_{free}$) or $\mathcal{A}(q)$ intersects any part of a known obstacle ($q \in$ known $\mathcal{C}_{obs}$).

External sensing *dofs*: the *dofs* correspond to those *dofs* that, when the position or orientation of the sensor changes, the body of the robot moves.

Information Gain density function (IGDF): The information gain per unit volume when a small region of the physical space is scanned.

Internal sensing *dof*: The *dofs* belong to the sensor and are "independent" of $n_{\mathcal{A}}$, i.e. the robot does not need to move.

Landmark: A configuration in C-space. In this thesis, the landmarks are randomly selected in C-space with uniform distribution. When a landmark is in free C-space, it is a white landmark. When it is in an obstacle, it is a black landmark. When it is in unknown C-space, it is a gray landmark.

Observable Physical Space: Let $\mathcal{C}_{reach} = \{$all s-reachable configurations$\}$. Given

an environment, $\mathcal{P}_{observ} = \bigcup_{q \in \mathcal{C}_{reach}} (\mathcal{V}_{\mathcal{S}}(q) \bigcup \partial \mathcal{V}_{\mathcal{S}obs}(q)) \subset \mathcal{P}$ is called the observable subset of this environment.

Probabilistically complete: a planner is called probabilistically complete if, given a solvable problem and any probability p_0, the probability of solving this problem with the planner is greater than p_0 as the running time is greater than a finite time $T(p_0)$. Such a planner is guaranteed to solve any solvable problem within finite time (in a probabilistic sense).

Projection Map: a data structure to store the information of a range image.

Quasi-internal *dofs*: $(n_{\mathcal{S}_{int}})$ *dofs* (mostly distal) affect robot motions on a much smaller scale than the other *dofs* (mostly proximal).

s-completeness: An s-complete algorithm for the start–goal problem should return an s-feasible path to a given goal if such a path exists, for any given environment and any initial free region $\mathcal{P}_{free}^{(0)}$. Otherwise it should report that the goal is not s-reachable.

Sensing dofs: the *dofs* that are used for moving the sensor.

s-feasible Path: A collision free path which is feasible for sensor-based robot motion planning. On this path, the robot is always within the known free region obtained from the sensor.

s-reachable Configuration: A configuration q_1 is called s-reachable from q_0 if there exists an s-feasible path for $\mathcal{AS}$ to move from q_0 to q_1. Otherwise, we say q_1 is not s-reachable from q_0.

Unoccupiable region: A set in free physical space, with which the robot can never intersect either because the kinematics of the robot or obstacles in the physical space.

View node: A robot configuration from which the sensor takes a scan.

Acronyms

ACA: Ariadne's Clew Algorithm.
IGDF: Information Gain Density Function
MER: Maximal Entropy Reduction
MP: Motion Planning
OBPRM: Obstacle Based Probabilistic Roadmap
PRM: Probabilistic Road Map.
RRT: Rapidly-exploration Random Tree
SBIC-PRM: Sensor-based Incremental Construction of Probabilistic Roadmap
TCP: Transmission Control Protocol
UDP: User Datagram Protocol

Chapter 1

Introduction

1.1 Introduction

This research is motivated mainly by the need for future applications of autonomous robot manipulators in unknown environments. For example, robot manipulators may be used to inspect and maintain hazardous nuclear waste sites [25]. Very often the exact environmental models are not available. Therefore, a certain degree of autonomy is essential for the safety of the robot and the working environment. In another example, Honda has recently developed a humanoid robot for future home use [26]. Eventually, this robot will be able to move through rooms filled with furniture and manipulate objects we encounter in daily life. All these applications require robots with the ability to sense and move without colliding with obstacles in the environment.

Robot motion planning (MP) is a discipline dealing with such problems. In a narrow sense, the basic robot MP problem consists of finding collision-free movements for the robots in known environments [40]. The seminal work of Lozano-Perez [44] introduced the concept of configuration space in which the robot is treated as a single point and the motion planning problem is converted to that of finding a one dimensional "path" in the free configuration space. The configuration space (C-space) of a robot is composed of all the possible robot poses. For manipulator robots, the dimension of the C-space equals the degrees of freedom of the robot. For example, the two joint values of a two link robot span the C-space of the robot (see Figure 1.1).

Since the entire model of the environment and the robot is known, this type of MP is referred to as model-based MP (see preface of [21]).

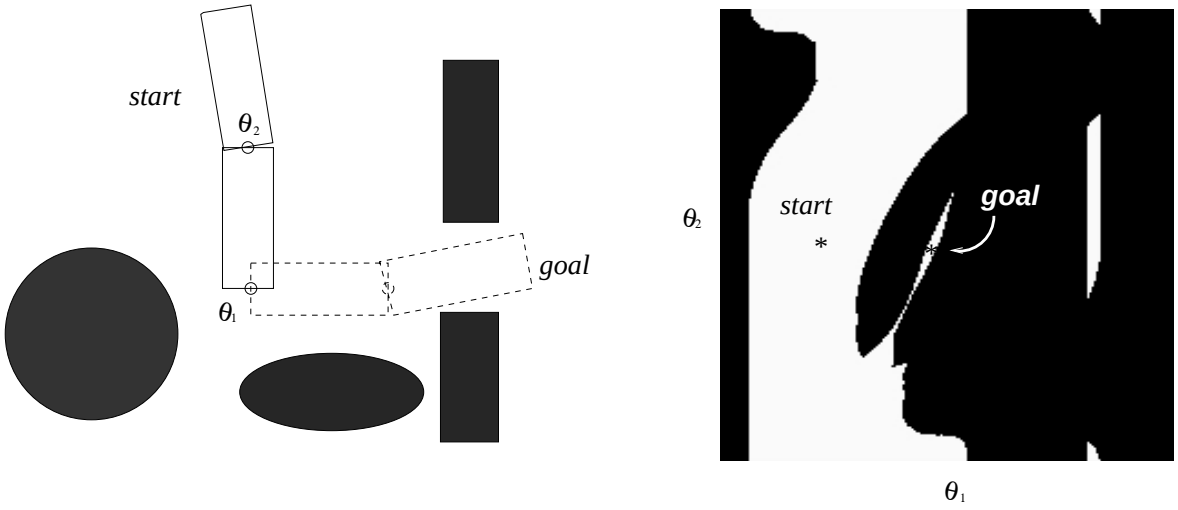


Figure 1.1: An example of a two-link robot and its C-space. In the physical space (left), white regions are free space and black regions are obstacles. The figure on the right side is the C-space of the robot. A point in C-space corresponds to a robot pose. When a point in C-space is in the black region, the corresponding pose of the robot intersects the obstacles. Otherwise, the robot is collision free. Planning collision-free movement for the robot is equivalent to finding a path in the free region of C-space.

A critical problem is that for most manipulator robots the dimension is high, and explicit computation of C-obstacles is intractable [5, 40]. A recent roadmap-based paradigm effectively solves this problem [1, 4, 27, 36, 42, 52]. The basic idea behind this class of algorithms (see later Section 1.5.1) is to use a graph structure to capture the connectivity of free C-space of a robot. The robot can move without collision on paths corresponding to edges in the graph. Figure 1.2 illustrates a roadmap constructed from the C-space of the previous example.

A limitation of the model-based motion planning algorithm is that a complete model of the environment is needed before the planner can proceed. An answer to this problem is sensor-based MP. Sensor-based MP allows robots to work autonomously in unknown environments. Specifically, the robot is able to move to a given configuration

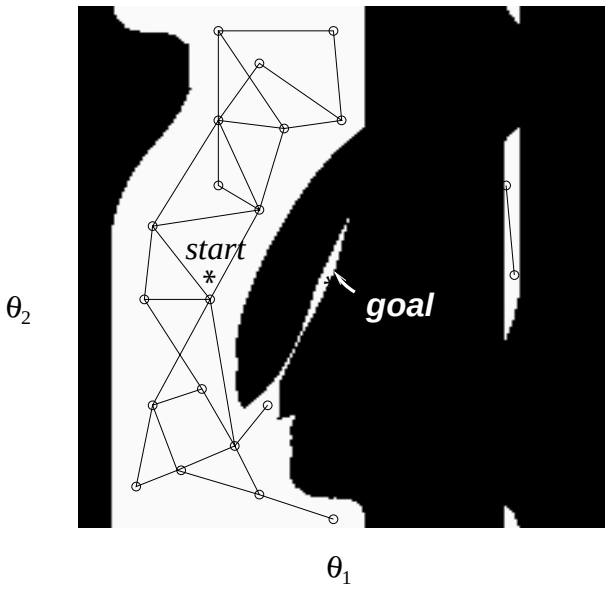


Figure 1.2: A roadmap constructed in the free C-space of the previous example.

without prior knowledge of the environment. The lack of *a priori* knowledge (at least complete knowledge) about the environment is the fundamental difference between sensor-based MP and model-based MP. Sensor-based MP is therefore not an off-line process, as is the case with model-based MP. Instead, motion is generated step by step while more and more knowledge about the environment is accumulated incrementally. This changes the nature of the problem in a fundamental way.

A great variety of sensors can be used for robots, including tactile sensor, vision sensor, range sensor and etc. Our research is concerned with sensor-based MP for articulated robot arms with many degrees of freedom (see Section 1.5.2 for a review of other sensor-based planning research). The arm (called robot from here on), equipped with an "eye" sensor on the end-effector of the robot (see Figure 1.3) that gives distances of the objects from the sensor, is required to plan and execute collision-free motions in an environment initially unknown to the robot. This type sensor-robot structure is often referred as "eye-in-hand" [64]. The sensor is a discrete time sensor, i.e. the eye sensor only takes range images at discrete times[20].

Figure 1.3: The experimental testbed for sensor-based MP. The robot is a PUMA 560 with 6 degrees of freedom. The "eye" sensor, a triangulation based area scan laser range finder, is mounted on the PUMA wrist. The inset shows an enlarged view of the sensor with the camera on the left and the laser striper on the right. The sensor gives distances of the objects (within a certain field of view and range) from the sensor.

The decision to mount the sensor on the robot end-effector, thus providing a six-dof scanning ability, was motivated by the additional maneuverability for sensing. The sensor can scan from any point in the reachable workspace of the robot and in almost any direction (subject to joint limits). The maneuverability is important because the observable region of the sensor in physical space is closely related to it.

As a first step towards building a robot capable of dealing with real unknown environments (as with Honda's humanoid robot) or unknown hazardous environments, this research focuses on a robot in a static environment. In this environment, the robot is the only moving entity. Our results are valid for robots with non-trivial geometry/kinematics (see Section 1.2) equipped with an "eye" sensor although the specific system considered in this thesis is an Eye-in-Hand system.

1.2 Distinction from Mobile Robot Literature

The planning space (the C-space) and the sensor space (physical environment) are very different for robot manipulators (see also [22]). This is a crucial distinction from the assumption in most of the current approaches to sensor-based planning for mobile robots [9, 7, 8, 39, 45, 69, 57]. In order to simplify the planning problem, they all treated the mobile robot as a single point (or a circular cross section) in 2-d or 3-d physical space, i.e. they have trivial geometry/kinematics. The configuration spaces for these mobile robots are basically the same as the physical space. In these approaches, the sensing and planning basically take place in the same (physical) space. Therefore, these sensor-based approaches for point mobile robots can not be directly extended to the case of Eye-in-Hand systems via the standard "convert to C-space" argument, since the sensor senses in physical space, and the sensed data (a region in physical space) do not correspond to "sensing" if a given configuration (in C-space) is free or not. These approaches to sensor-based motion planning for mobile robots can not be directly applied to non-trivial geometry/kinematics mobile robots either. To our knowledge, the only two papers on sensor-based motion planning for non-point (and non-circle) mobile robots was presented by Choset [9] and by Ghosh [17] respectively. The first paper [9] proposed (without implementation or simulation) an

algorithm which extended the same author's work for point robots to a 2-d rod robot (only a line segment in physical space). Extension to real 3-d solid model robots is unknown. The second paper [17] studied the problem of a convex mobile robot in 2-d physical space with a sensor. The assumed sensor model is that of entire visibility polygon from a given point on the robot [22]. The robot is only allowed to explore the environment under pure translation. The extension of their algorithms to robots under both translation and rotation is unknown.

1.3 Overview

We view that our sensor-based MP framework consists of four major components as shown in Figure 1.4. The algorithm starts with invoking a model-based motion planner to reach a given goal at the beginning of each iteration (block 1 in Figure 1.4). Sensor-based MP uses this underlying model-based MP algorithm to plan paths within the known environment. If this underlying MP algorithm returns a success, the algorithm finds the path to the given goal and terminates. Otherwise, a view planner is invoked to plan a view (block 2). The robot moves to the new view position and takes a scan of the environment (block 3). This new scan increases the knowledge of the environment. The physical space and C-space are then updated and expanded (block 4). This process is repeated until the algorithm finds a path or no new information can be obtained from the environment.

Since the status of the work space (physical space) of the robot gradually changes as the sensor senses the environment, from unknown to free or obstacle space, this model-based planner should be able to adapt to the changing status of the environment. Therefore, an incremental version of the model-based planner is needed for efficiency.

The two key sub-problems that sensor-based MP needs to solve are (see Figures 1.4) (i) *view planning*, i.e. to determine the next sensing action, including which region to scan and from where to scan, and (ii) *C-space expansion*, i.e. to incrementally expand the C-space representation. These two sub-problems are repeatedly solved in an interleaved fashion in the sensor-based MP.

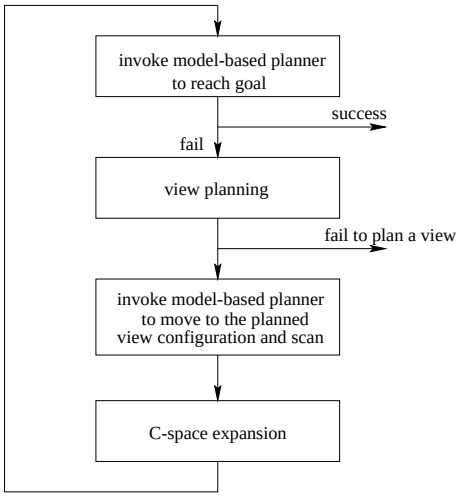


Figure 1.4: A flow chart and decomposition of sensor-based MP.

We approach the view planning problem from an information theoretical point of view, and introduce and develop a novel concept of C-space entropy characterizing the uncertainty of unknown C-space (a direct result of the unknown physical space). Each scanning action reduces the entropy of C-space. In each iteration, the next scanning action is determined by maximizing the entropy reduction (MER), or equivalently, gain in information.

View planing algorithm (the view planner) takes a small fraction of the time ($<$ 3-5 second) of an entire scan-plan-move iteration (1 - 2 minutes) (see Section 7.1). But the advantage is a much faster information gain (entropy decrease) of C-space compared with randomly chosen views.

Because of the high dimensionality of the C-space of a typical manipulator, a roadmap is adopted for C-space representation. The crux of the C-space expansion is to *incrementally* construct a roadmap as the sensor senses new free region ($\Delta\mathcal{P}_{free}$) in the physical environment. The (evolving) roadmap reflects the connectivity of known free C-space ($\mathcal{C}_{free}$) within which the robot carries out its motion to further sense the physical environment. The $\mathcal{C}_{free}$ (and the roadmap) expand as the sensor senses new free region in the physical space. This process is repeated until certain

conditions are met, for example, when a final goal is reachable from (one of the nodes in) the roadmap, or when the goal is declared unreachable. We ensure that the new landmarks lie in the new free region in the C-space ($\Delta\mathcal{C}_{free}$) that corresponds to the additional free physical space ($\Delta\mathcal{P}_{free}$) obtained in each scan. This guarantees that the roadmap does not "degenerate".

In the motion planning literature for model-based MP, several variants of roadmap exist (see Section 1.5.1). We adapt the probabilistic roadmap (PRM) approach of [36], and call our algorithm sensor-based incremental construction of probabilistic roadmap (SBIC-PRM).

Our physical space representation uses spatial occupancy octrees. Octrees are a memory efficient hierarchical structure for representing three dimensional spatial occupancy data [60]. It usually needs only a small fraction of memory that a voxel map would need to represent the same environment.

Octrees are particularly useful in representing spatial occupancy in physical space for sensor-based robot motion planning. It is significantly easier to obtain from sensors than more abstract CAD models, and it needs less memory than voxel maps do. In addition, some augmented octrees, such as octree based distance maps (with extra distance fields in the nodes of the octree), have been proposed for further improving the collision detection efficiency [32].

1.4 Algorithm Outline

We can now give an overview explanation of our algorithm SBIC-PRM (see Figure 1.5 for illustration). The precise implementation for this algorithm is explained in Chapter 4.

Figure 1.5 shows a schematic illustration of the process at different stages – view planning and C-space (roadmap) expansion. The left side of each block shows the physical space and the right side shows the corresponding C-space. In the left side of each block, the gray region indicates unknown, the white region indicates known free space and the black region indicates known obstacles. The same code applies on the right. Note however, that only the corresponding landmarks are computed in the

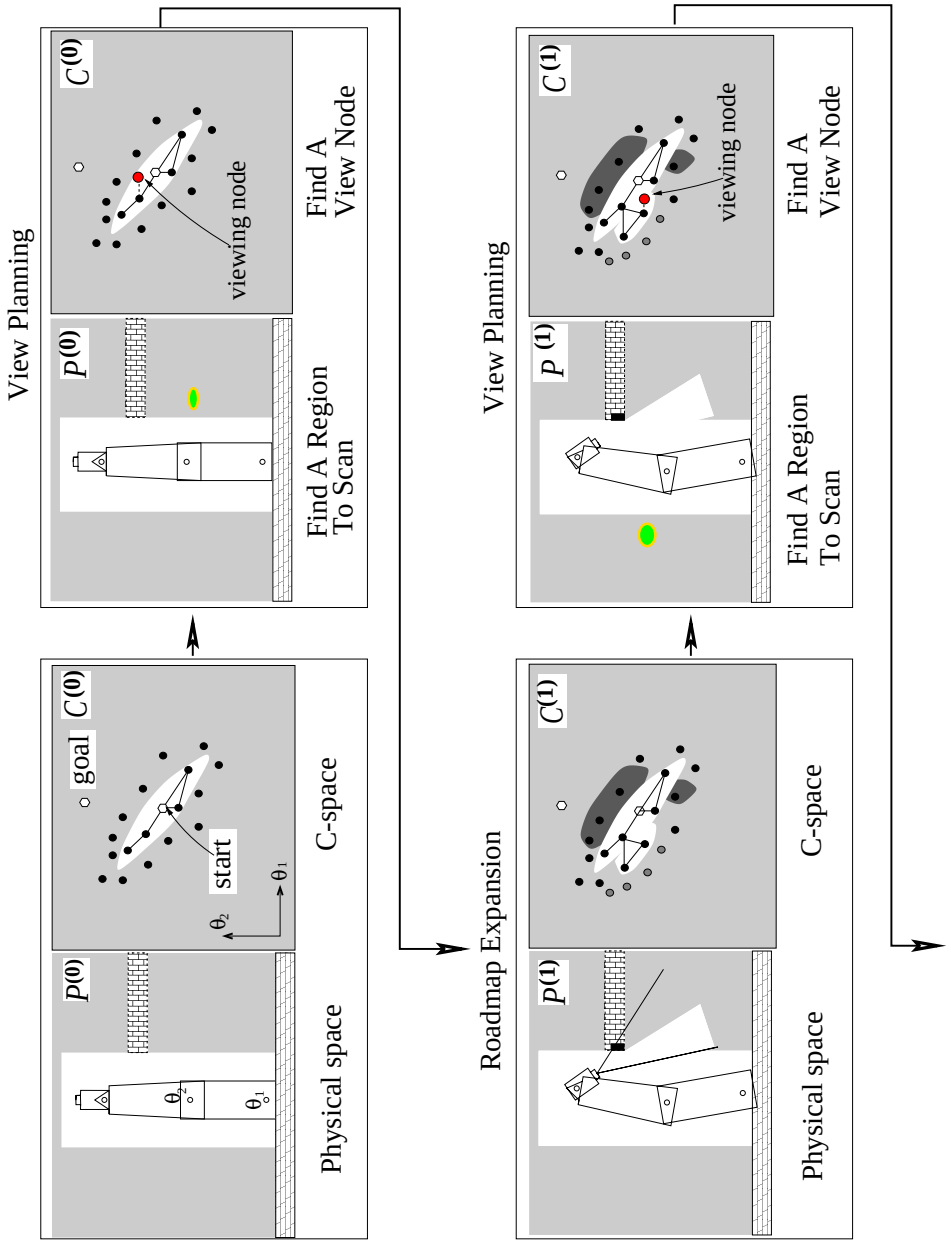


Figure 1.5: A schematic illustration of SBIC-PRM. In both C-space and physical space, white regions represent free space; gray regions are unknown to the robot and black regions are known obstacles. Brick-patterned regions are obstacles in physical space which are unknown to the robot, but are shown here for the reader's understanding.

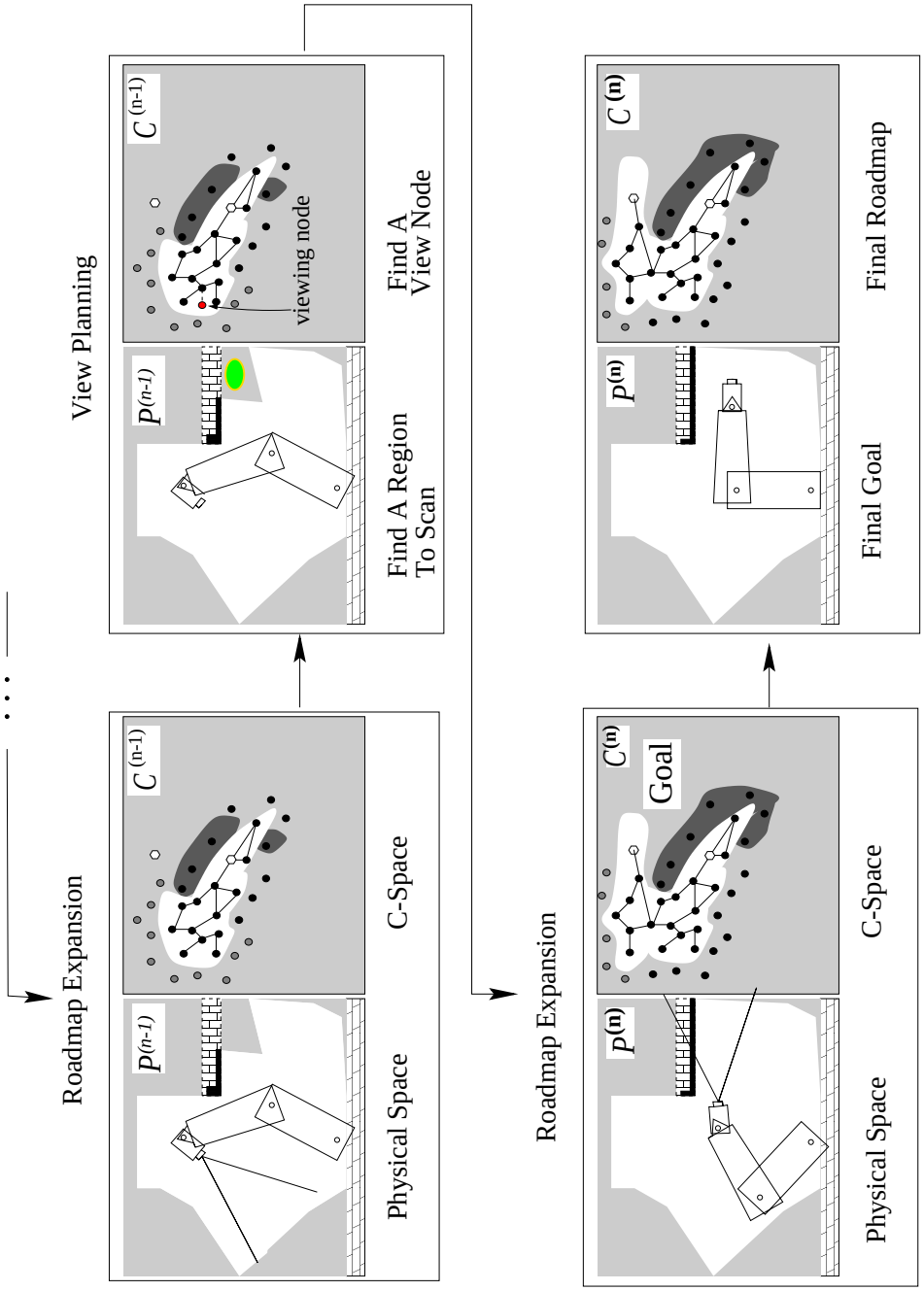


Figure 1.6: A schematic illustration of SBIC-PRM (continued from the last page).

C-space, and not the precise regions, which are shown only for visualization purposes.

The robot starts off in an initial configuration lying in a given initial free physical space, $\mathcal{P}_{free}^{(0)}$ (in top left sub-figure), shown as a white rectangle. The corresponding set of landmarks and the roadmap are shown on the right side. The ellipse shaped white region is the initial free C-space, $\mathcal{C}_{free}^{(0)}$. The robot is at an initial landmark l_0.

At any given iteration, say i, the planner first attempts to reach the goal configuration by invoking the underlying model-based planner. If the model-based planner returns a success, the sensor-based planner has found a collision-free path within currently known free physical space. If it does not succeed, the view planner determines the next view.

View planning is done by first determining a region to be sensed in physical space. This region is chosen so that scanning it results in maximal "information gain" in C-space, or equivalently maximal entropy reduction (MER). Having determined the region to be scanned, the view planner then determines a (reachable) view node (shown with a small circle in the sub-figure on the right side of each block in Figure 1.5) from which the region can be best scanned.

The robot (and the sensor) then moves to this view node. A scan is then taken (see the left side of each block). The sensor returns a range image and a program managing physical space derives $\Delta\mathcal{P}_{free}$ from it and integrates with existing $\mathcal{P}_{free}$ to get updated $\mathcal{P}_{free}$.

Corresponding to $\Delta\mathcal{P}_{free}$, there is new $\Delta\ \mathcal{C}_{free}$. However, note that it is not practical to compute $\Delta\mathcal{C}_{free}$ explicitly because the C-space dimension in our system is high (we deal with 6 dimensional C-space). Instead, the roadmap is expanded in a manner so that it "expands" into $\Delta\mathcal{C}_{free}$. New landmarks are (randomly) generated and selected in a manner to expand the roadmap into $\Delta\mathcal{C}_{free}$ (see Chapter 4.4). The entire process can be thought of as the wavefront of $\mathcal{C}_{free}$ expanding toward the goal.

Having updated the roadmap, the underlying model-based planner is invoked again to reach the goal and the entire process is repeated, until either the goal is reached or the entire algorithm exits after a certain maximum number of iterations.

1.5 Related Work

1.5.1 Model-based Motion Planning

Motion planning problems for high dimension C-space had been very successful until a recent paradigm, randomized approaches, for solving MP for robots with many degrees of freedom was introduced [70]. This set of algorithms capture the connectivity of robot's free C-space ($\mathcal{C}_{free}$) in a finite graph structure, such as Ariadne's Clew Algorithm (ACA) [1, 6], Probabilistic Road Map (PRM) [4, 27, 36, 52] and Rapidly-exploration Random Tree (RRT) [42]. In the graph structure, each node represents a free robot configuration, and an edge between two nodes represents that a simple local planner can find a collision-free path between the corresponding robot configurations. These algorithms adopt different strategies for placing landmarks (nodes) and for connecting these landmarks (nodes). We briefly outline these algorithms.

ACA [1] iteratively selects the landmark(s) from a set of embryos which are generated from current landmarks by random walks. The random walks proceed always in $\mathcal{C}_{free}$ such that all the embryos are connected by the route of the random walk. The new landmark is the embryo which is the furthest from all the current landmarks in order to explore $\mathcal{C}_{free}$ efficiently. The route of the random walk is the edge connecting this new landmark to the existing roadmap. ACA explores the reachable component of $\mathcal{C}_{free}$ and forms a tree to characterize the connectivity of $\mathcal{C}_{free}$.

Original PRM [36] randomly generates nodes in $\mathcal{C}_{free}$ and connects them with straight lines in C-space. Later, different strategies for selecting nodes, such as Obstacle Based Probabilistic Roadmap (OBPRM) [4] and Hsu's [27] algorithm, have been investigated to improve the efficiency of covering the entire C-space. Better strategies for connecting nodes, such as "rotate-at-s"[3], are also proposed.

RRT [42] iteratively generates nodes in a rather complex approach. The growth of their tree structure is based on two configurations. One is a randomly sampled in the C-space at each iteration, called x_{rand}. The other configuration x_{near} is the configuration in the tree which is closest to x_{rand}. In each iteration, a set of configurations U at a fixed distance from x_{near} are generated. The (free) configuration in U, which is

closest to x_{rand} and can be connected to x_{near} with a straight line planner, is selected and added to the tree. The graph structure, as in ACA, is a tree because the new nodes are connected to only one node.

Note that these approaches do not enumerate the C-space at a fixed resolution as is done by a grid-based search[1]. Grid-based search algorithms obviously can not be extended to very high dimensional spaces, since the computational complexity of MP is known to be exponential to the degrees of freedom of the robot [40]. The graph (or tree) in these roadmap-based approaches is constructed without explicitly representing the C-obstacles, and is therefore particularly useful as a representation for high dimensional C-spaces.

We choose PRM as our underlying model-based motion planning algorithm. The two major reasons for choosing PRM are: (1) PRM has the ability to conveniently control the density of landmarks or nodes (in a probabilistic sense). Once a new $\Delta\mathcal{C}_{free}$ is obtained, landmarks with a *constant* density (in a probabilistic sense) should be placed in this region. Both ACA and RRT iteratively add landmarks (nodes) to the existing roadmap. It is difficult to add an adequate number of landmarks and maintain a constant landmark density because $\Delta\mathcal{C}_{free}$ is not explicitly computed. (2) PRM generates landmarks in the temporarily disconnected $\mathcal{C}_{free}$ before they become reachable. Once a new region in C-space becomes free because of new regions of free physical space, landmarks are placed immediately in this new free C-space region (no matter if this new C-space region is connected). Both ACA and RRT will have to generate many landmarks to fill these (formerly disconnected) regions *after* they become connected. The number of landmarks for the new connected region is difficult to determine directly for ACA and RRT.

[1]Grid-based approaches are related to "cell decomposition" approaches. It consists of first decomposing the set of free configurations of the robot into a finite collection of cells and then searching a connectivity graph representing adjacency relation among the cells. See reference [40].

1.5.2 Sensor-based Motion Planning

A great deal of work has been done on sensor-based planning for robots [34, 41, 39, 51, 47, 45, 58, 59] and can be grouped into two broad categories [22]: (1) robots with an "eye" sensor (camera or range), (2) robots with a "skin" sensor, i.e. sensors that are whole-arm sensitive and sense along the entire robot geometry. As mentioned earlier, category (1) mostly consists of systems with mobile robots, generally modeled as a point robot. Hence, the sensing space (physical space) and the planning space (C-space) are the same in these systems. This body of work is, therefore, not directly applicable to Eye-in-Hand systems. Even otherwise, most of these algorithms are applicable to 2-*dof* or 3-*dof* systems.

In category (2), articulated arms have been considered. The skin sensor can consist of proximity or range sensors distributed along the entire manipulator geometry [9, 47]. The sensed data, with this type of sensor, is directly "transformed" to C-space, since a sensing action can directly determine if a given configuration is free (or in contact with a C-obstacle). The point robot approaches (similar to those in category 1) can now be applied. For instance, Lumelsky et al. have applied BUG algorithms to 2-*dof* manipulators, and to the very specific case of a 3-*dof* arm with prismatic joints with such "skin" sensors [46, 47]. From a practical perspective, such "skin" sensors are likely to be more complex than the commercially available range-scanners for manipulator robot applications. Choset and Burdick [7, 8] developed a generalized Voronoi diagram approach to (point) mobile robots. Later, this research was extended to a planar rod robot whose generalized Voronoi diagram was built in C-space with the assumption that range sensors can be distributed over the robot's entire geometry. Although the generalization from a point robot to a rod robot is only the first step towards real robots, the planning problem became rather complicated. In addition to these two major categories, Rimon [59] assumes abstract sensors that provide distances in C-space.

The only system with similar hardware to ours was first presented by Renton et al., and is called Plan-N-Scan system. It comprises a PUMA robot with a wrist mounted range sensor [58]. The task was to scan a given target voxel. The motion planning

was accomplished with an A^* algorithm. A brief description of their algorithm is as follows: the algorithm keeps a stack to store the goal and sub-goals (voxels to be scanned). A given final goal voxel is pushed into the stack during initialization. At each iteration, the algorithm pops a goal/sub-goal voxel from the top of the stack. The planner first selects a viewing position to scan the voxel. The viewing position has to satisfy two criteria: (1) the line of sight from the sensor head to the viewing target is not obstructed; and (2) the manipulator is collision free when in the view configuration. If the robot moves to the viewing position successfully and scans the goal/sub-goal voxel, the sub-goal is removed from the stack. If the robot intersects an unknown region on a path to the scanning position, this unknown region is pushed into the stack as a new sub-goal voxel to be scanned. The algorithm stops when the stack is empty and the given goal voxel is scanned.

The approach in this thesis has three main advantages over Plan-N-Scan System (i) from C-space exploration point of view, our entropy-based view planning is more efficient than their algorithm, which, in each iteration, scans sub-goals determined in the previous iteration; (ii) the roadmap, once built, can be used to quickly answer further path planning queries [6, 36], and (iii) the computational expense of A^* algorithm becomes prohibitive as the degrees of freedom increase, whereas the roadmap approaches, by incorporating randomized techniques, have resulted in practical algorithms for robots with many degrees of freedom [1, 36].

Kruse et al. describe a similar (but simulated) sensor-based system [38]. Although their planning algorithm used a roadmap, it simply repeatedly calls a roadmap based planner as developed in [36]. In their approach, the expansion of this roadmap as the robot senses more of its environment, is ad hoc and the roadmap may easily (and in fact does) degenerate. Their view planning algorithm is based on the following 4 constraints: (a) The next view configuration is collision free; (b) The next view configuration can be reached by a collision free path; (c) The sensing action at the next view configuration should obtain a lot of new information about unknown area. This constraint is expressed by introducing a rating function $R_{inf}(q)$, which is the fraction of unknown volume in the total sensing volume. (d) The next view configuration should not be far from the current one. This constraint is regarded by

$R_{dist}(q) = 1 - d(q, q_{cur})/d_{max}$, where q_{cur} is the current configuration of the robot, $d()$ is a distance function and d_{max} is the maximum distance in the C-space. The next view configuration is chosen such that function $R(q) = R_{inf}(q)(wR_{dist}(q) + 1 - w)$ is maximized over discretized C-space, subject to the first two constraints, where $w \in [0, 1]$ is a weighting parameter. The rationale behind the first two constraints is obvious. The last one, R_{dist}, is only critical when the time used to move to each new sensing configuration takes a very large portion of each scan-plan-move iteration. This is not the situation in our Eye-in-Hand system. The third constraint, R_{inf}, seems to be very reasonable intuitively. However, only maximizing the unknown physical space in each scan is not enough. For example, some regions are frequently occupied by the robot (i.e. the regions which have large C-zones, see Section 3.4.3). These regions should be given a higher priority to be scanned. Many other important characteristics, e.g. if the region is even occupiable by the robot, are also critical.

1.5.3 View Planning in Related Disciplines

A variety of view planning problems have also been considered in computer vision area. These works do not include kinematics constraint of the robot (or the mechanism carrying the sensor) and the geometric constraints due to obstacles in the environment. They usually assume that the sensor can move to arbitrary positions and directions and there are no obstacles in the environment. Their objective is object recognition, which is different from ours – exploration and reaching a given goal for the robot. We mention some typical work only, since it is only peripheral to the problem at hand.

Kovacic et al. [37] presented a view planning method for object recognition. This approach analyses the transformation of feature vectors of templates. These vectors often cluster into many groups. The clustering of these feature vectors (in feature space) causes ambiguity. This approach plans the next view by maximizing the distance of the vectors in a feature space in order to identify objects.

Hutchinson and Kak's work [28] in view planning for object recognition is in spirit closer to our view planning algorithm. They define an ambiguity function which is the entropy of the probabilities of matching models with real objects. A view maximizing

the reduction of the ambiguity function is chosen as the next one. The ambiguity function is based on the probabilities of finite number known object model features.

1.5.4 Physical Space Representations

In the area of environment modeling for robots, especially for mobile robots, when the positioning error of robots is significant, certain probabilistic models (for example 0th order Markov field) are used to integrate sensor data from different sensing iterations. Elfes [16] assigned a probability (of being part of an obstacle) to each point (at a certain discretization level) and updated this probability distribution after new data from the sensor was obtained. Payeur [54] further applied this approach to three dimensional space and incorporated this approach with octree models.

Because our work is mainly gross motion planning, we adopted a simple way. We enlarged obstacles by a security distance to assure that the real robot was collision free. This safe distance is based on the measure of our robot error (1-3 cm on the end-effector). Therefore, the positioning error of the sensor was neglected.

In Greenspan's work [19] (and later used in [58]), they used discretized distance maps to represent the physical space. A field in each voxel records the distance of the closest obstacle from it. This distance can be used to check if a sphere robot collides with obstacles quickly. However, it is needed to decompose the robot into many spheres. As the robot model becomes complex, large number (for example, a few hundreds to more than one thousand for a Puma robot) of spheres may reduce the efficiency of the algorithm.

In Harvey's work [23], they presented a Rubber Band Paradigm for storing sensor data collected by mobile robots. This rubber band method represents the world as a single border between travelable space and obstacle. As octree representations, Rubber Band Paradigm avoids taking up a lot of computer memory in most realistic situations. However, this paradigm has not been extended to 3-d space.

1.5.5 Spatial Occupancy Octree Construction

As we mentioned, we use spatial occupancy octrees to represent the physical space. However, building octrees is expensive in terms of both memory space and computing time. Li and Grebbin [43] proposed an algorithm constructing octree from range images. The aim of this algorithm is to construct an octree for an object, assuming space surrounding the object is free. This algorithm has two steps. First, it constructs an initial octree, only from the object silhouettes in different range images by thresholding out background in the images. Then, it determines concavities from depth information in the range images. The prominent benefit of this algorithm is that it does not require a large amount of temporary memory. However, this algorithm's computation is expensive, as stated in the conclusion of this paper. Most of all, the purpose of this algorithm is very different from ours. This algorithm can not be used in our system. Shu and Kankanhalli [63] developed a faster algorithm generating octree from a voxel map from CT images. The efficiency is basically from its bottom up octree constructing strategy. This algorithm needs large memory space $O(n^3)$ to store the voxel map. Since the source of the information of our system (range images) is different from theirs, this algorithm can not applied in our system either. In another example, Norborio et al. [51] constructed octrees of objects from multiple views. This work represents a single convex object by an octree. However, we need to construct octrees which represent the entire spatial occupancy of the environment.

We present a time and space efficient algorithm which constructs octree directly from range images. Most of the efficiency derives from a key visibility property of most range sensors. All the voxels on object surfaces in an image can be "seen" from the view point, i.e. there is no object between the view point and any visible surfaces (detailed in Chapter 5).

1.5.6 Real Implemented Systems

The only real system similar to ours that we are aware of is the Plan-N-Scan system developed by Renton et al. [58]. It uses sockets to create stream data flow for the processes on different computers. Our software architecture is more modular and

therefore more suitable for developing and testing different planners.

1.6 Contributions

This dissertation presents a real implemented sensor-based motion planning system for an Eye-in-Hand system. The scope of the dissertation is comprehensive, and includes development of a theoretical framework and a planning algorithm, integration of system hardware, and design and development of a software architecture. This thesis makes contributions in each of these domains. We briefly outline them.

1.6.1 Theoretical Framework

- Our view – that sensor-based motion planning for a robot with non-trivial geometry/kinematics with an eye sensor, can be decomposed into two major components: (1) view planning. (2) incremental construction of C-space representation – presents a framework to develop algorithms for such systems.

- The completeness issues for sensor-based motion planning algorithms are different from model based motion planning. This is because of an extra constraint – a robot needs to sense and then move in the free region discovered by the sensor. We introduce the novel concepts of s-feasible path and s-reachable configuration (reachable for sensor-based motion planning). Based on these concepts, we then augment the definition of completeness for sensor-based motion planning [22].

- We introduce and develop a novel concept of C-space entropy which characterizes the uncertainty of unknown C-space. Reducing the entropy of C-space is the purpose of robot exploration. We also introduce a novel concept of information gain density function defined in unknown physical space. Scanning the region where this function is maximized results in the maximal entropy reduction (MER) in each iteration.

1.6.2 Algorithmic Contributions

A novel algorithm, sensor-based incremental construction of probabilistic roadmap (SBIC-PRM), is presented. To represent the unknown C-space, we give a novel concept of gray landmarks. Our roadmap expansion mechanism ensures that the gray landmarks enclose $\mathcal{C}_{free}$ and essentially represent a boundary of $\mathcal{C}_{free}$. In addition, the expansion process maintains the nature of uniform distribution of landmarks throughout the planning process.

A set of spatial occupancy octree algorithms, suitable for our "on-line" type of motion planning and sensor model is developed. These octree algorithms construct octrees of free space and obstacles from range images, and update physical space representation when a new scan is made.

1.6.3 An Eye-in-Hand System

We set up a distributed system composed of COM objects (for sensor) on Windows and a multiple protocol (Transmission Control Protocol (TCP)/ (User Datagram Protocol) UDP) robot server on Unix.

We designed a multiple thread program which best utilizes the computing resource. When the robot is in motion, the planner does not have an intense workload. A background thread which processes non-time-crucial work, runs during this period. This character does not exist in model-based motion planning and is unique to sensor-based motion planning.

We studied different types of servers and found that a connection-oriented(TCP) stateless server is suitable for our sensor server for its reliable data transmission, and a connection-less(UDP) server is suitable for our robot server because of its fast response.

1.7 Outline

The rest of this thesis is organized as follows: Chapter 2 discusses the formal problem statement for sensor-based MP and necessary notation. Concepts such as s-feasible

path, s-reachability and explorability are defined in this chapter. Chapter 3 introduces the concept of C-space entropy and the related information gain density function. The view planning problem is then posed in terms of maximizing the C-space entropy reduction (MER). In Chapter 4, we present our complete algorithm SBIC-PRM. Chapter 5 discusses the representation of physical space with spatial occupancy octrees and presents the algorithms for constructing octrees from range images. Chapter 6 is concerned with our test-bed system, both hardware and software architecture. The experimental results of SBIC-PRM are presented, with both examples and experimental data in Chapter 7. Finally, the conclusion and a discussion of future work is presented in Chapter 8.

Chapter 2

Formal Problem Statement and Completeness Issues

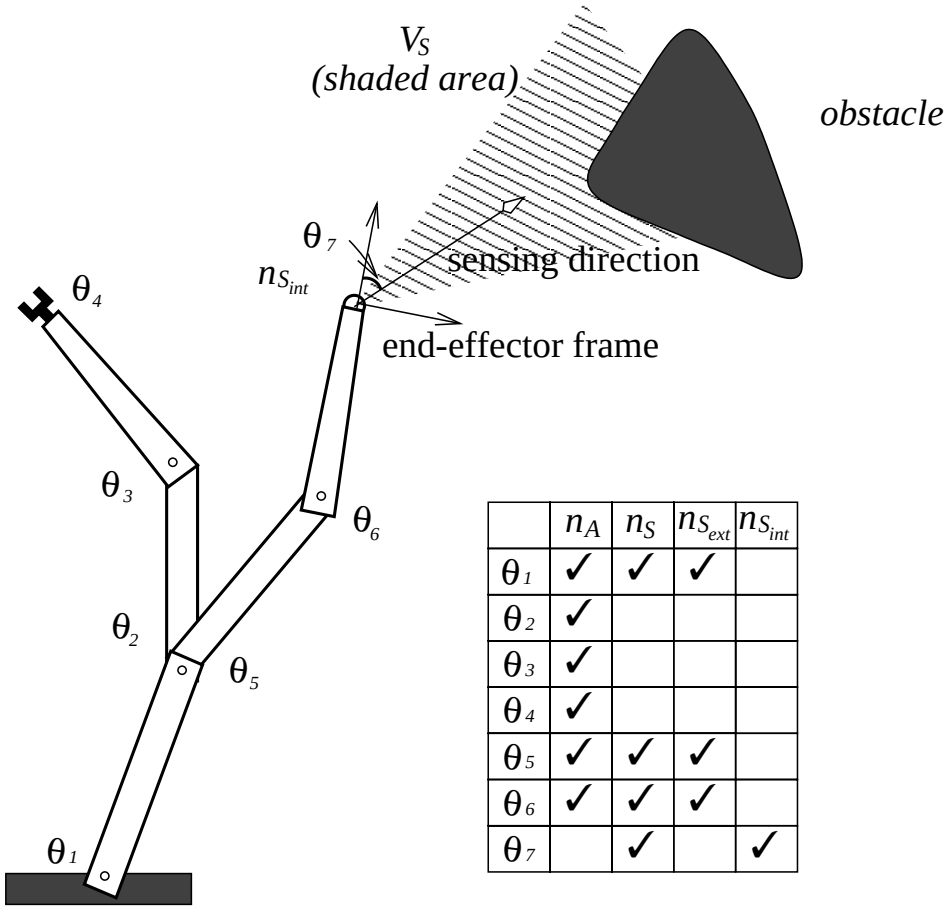
2.1 General Notation

2.1.1 Classification of Degrees of Freedom

Let $\mathcal{A}$ denote the robot (including all the physical parts), $\mathcal{S}$ denote the sensor (the reference frame only), and $\mathcal{AS}$ denote the combined robot and sensor system. Let $n_{\mathcal{A}}$ denote the degrees of freedom (*dofs*) of the robot. Let $n_{\mathcal{S}}$ denote the *sensing dofs*, the *dofs* that are used for moving the sensor. Moving $n_{\mathcal{A}}$ affects motion of (parts or the entire physical body of) the robot. Moving $n_{\mathcal{S}}$ affects motion of the sensor, which results in changing its position and sensing direction. Figure 2.1 illustrates this for a planar branched kinematic chain.

$n_{\mathcal{S}}$ can be further differentiated into (i) internal *dofs*, $n_{\mathcal{S}_{int}}$, and (ii) external *dofs*, $n_{\mathcal{S}_{ext}}$. The criterion to distinguish $n_{\mathcal{S}_{int}}$ and $n_{\mathcal{S}_{ext}}$ is whether the physical position of the robot is changed or not when these *dofs* change. The internal *dofs* belong to the sensor and are "independent" of $n_{\mathcal{A}}$, i.e. the robot does not need to move; no collision detection (or planning) needs to be done in exercising the internal *dofs* to scan the scene. For example, the eye ball movement *dofs* can be considered as the internal sensing *dofs* for a human. The sensing *dofs* which also cause robot body movement

is external sensing *dofs*. The external *dofs* correspond to those *dofs* that the sensor "borrows" from the robot in order to move (to a new sensing location). The external sensing *dofs* is the intersection of $n_{\mathcal{A}}$ degrees of freedom of the robot and $n_{\mathcal{S}}$ degrees of freedom of the sensor (see Figure 2.1).



	n_A	n_S	ns_{ext}	ns_{int}
θ_1	✓	✓	✓	
θ_2	✓			
θ_3	✓			
θ_4	✓			
θ_5	✓	✓	✓	
θ_6	✓	✓	✓	
θ_7		✓		✓

Figure 2.1: An illustration of robot degrees of freedom: $n_{\mathcal{A}}$, $n_{\mathcal{S}}$, $n_{\mathcal{S}_{ext}}$ and $n_{\mathcal{S}_{int}}$. θ_7 is the sensing direction in (one of) the end-effector frame. Changing θ_7 does not move any part of the robot physically, but moves the sensing direction only. Therefore, θ_7 is an internal sensing degree. In this example, $n_{\mathcal{A}}$= 6, $n_{\mathcal{S}} = 4, n_{\mathcal{S}_{ext}} = 3$ and $n_{\mathcal{S}_{int}} = 1$.

Often, robot arms are designed in such a way that some *dofs* (mostly distal) affect motions on a much smaller scale than the other *dofs* (mostly proximal). Path planning for the former subset of *dofs* is much easier (little collision checking is needed) than for the latter subset. When they are part of $n_{\mathcal{S}}$, they affect the pose of the sensor.

We call this subset quasi-internal *dofs* of the sensor. We use $n_{\tilde{\mathcal{S}}_{int}}$ to denote them. For example, if the sensor is mounted on the wrist of the PUMA robot (as in our system), moving the sensor with the last 3 wrist *dofs* causes very small displacement (assuming the sensor itself is small) of the robot+sensor system. See Figure 2.2 for the model of our system. The wrist *dofs* are therefore $n_{\tilde{\mathcal{S}}_{int}}$.

In our Eye-in-Hand system, $n_{\mathcal{S}} = n_{\mathcal{A}} = 6$, i.e. all *dofs* of the robot affect sensor movement. Moreover, the sensor has no internal degree of freedom, i.e., $n_{\mathcal{S}_{int}} = 0$. It further follows, $n_{\mathcal{S}_{ext}} = n_{\mathcal{S}} = 6$. As explained earlier, the wrist *dofs* of the robot can be considered quasi-internal *dofs* for the sensor, i.e., $n_{\tilde{\mathcal{S}}_{int}} = 3$.

2.1.2 Configuration Space and Physical Space

$\mathcal{C}_{\mathcal{AS}}$ represents the configuration space (C-space) of the entire robot and sensor system. It is spanned by all the degrees of freedom. In general, it is the space within which sensor-based motion planning should be performed. Let $\mathcal{C}_{\mathcal{A}}$ denote the robot configuration spanned by $n_{\mathcal{A}}$. $\mathcal{C}_{\mathcal{S}}$ denotes the sensor configuration space. $\mathcal{C}_{\mathcal{S}} (= \mathcal{C}_{\mathcal{S}_{ext}} \times \mathcal{C}_{\mathcal{S}_{int}})$ is a sub-configuration space spanned by $n_{\mathcal{S}}$. As we explain later in Section 4.1, this sub-space decomposition of the composite $\mathcal{C}_{\mathcal{AS}}$ is suitable for solving view planning sub-problem arising in our sensor-based planning framework. In addition, a useful sub-space is $\mathcal{C}_{\tilde{\mathcal{S}}_{int}}$. $\mathcal{C}_{\tilde{\mathcal{S}}_{int}}$ is the configuration sub-space spanned by quasi-internal sensing *dofs*, $n_{\tilde{\mathcal{S}}_{int}}$.

Let $q_{\mathcal{A}} \in \mathcal{C}_{\mathcal{A}}$ denote a robot configuration, $q_{\mathcal{S}} \in \mathcal{C}_{\mathcal{S}}$ denote a sensor configuration, and $q_{\mathcal{AS}} \in \mathcal{C}_{\mathcal{AS}}$ denote a configuration for the entire (robot+sensor) system. Let $\mathcal{P}$ denote the physical space (the 3-D Euclidean space). $\mathcal{A}(q_{\mathcal{A}}) \subset \mathcal{P}$ denotes the physical space occupied by the entire robot (including the body of the sensor) at configuration $q_{\mathcal{A}}$. We assume that the sensor body is part of the robot $\mathcal{A}$. Hence, we can treat the physical sensor as an abstract reference frame without any physical body.

The range sensor takes range images at discrete times (called scans). The free space obtained from each scan is modeled as a cone, whose apex is at the center of the sensor. The base of this cone is composed of visible obstacle surface or the sensing limits. The range image is a discretized array with each element representing

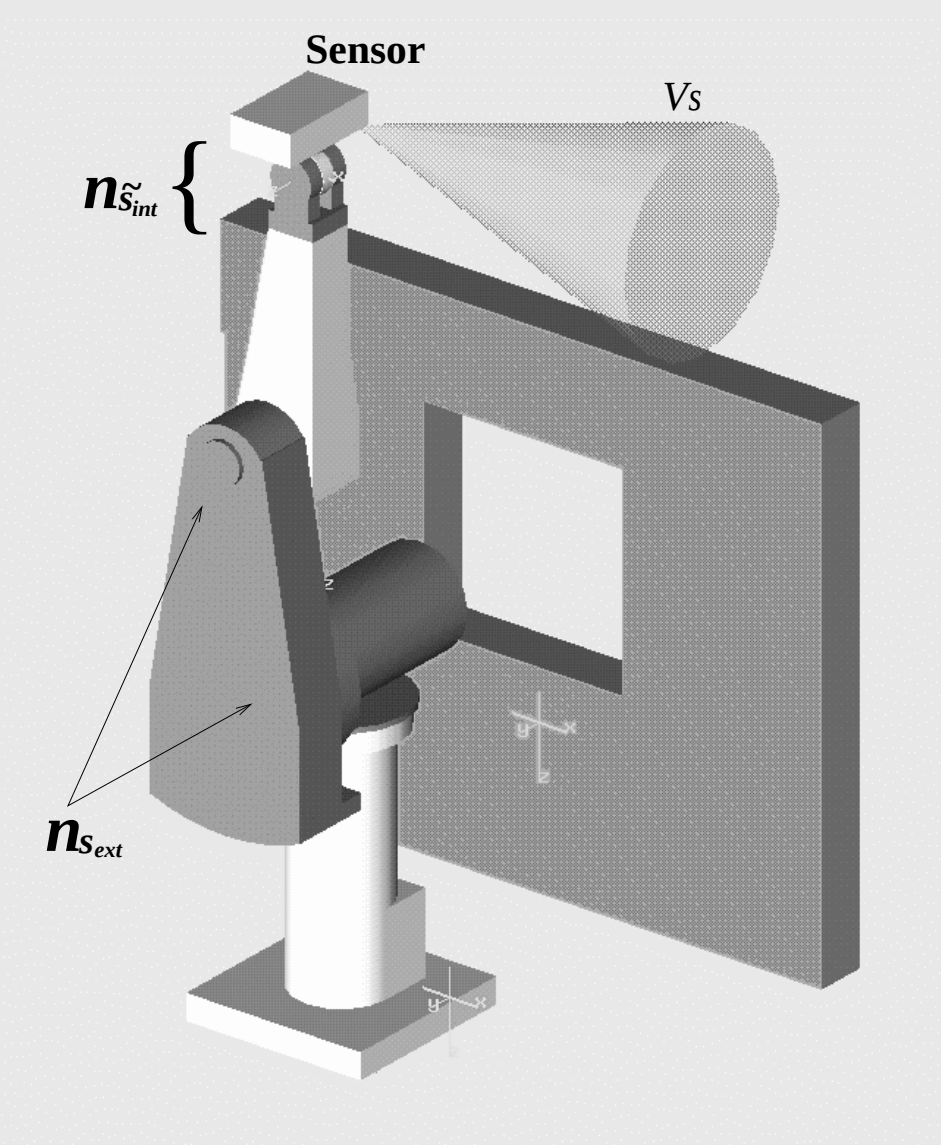


Figure 2.2: Model of the robot in our system. The wrist degrees are quasi-internal sensing *dofs* $n_{\tilde{S}_{int}}$. The other three *dofs* are external sensing *dofs*.

a point in physical space (see Figure 2.3). Let $\mathcal{V}_{\mathcal{S}}(q_{\mathcal{S}}) \subset \mathcal{P}$ denote the sensed free region (an open set) when robot scans at configuration $q_{\mathcal{S}}$. The actual sensed entity is the boundary of the region $\mathcal{V}_{\mathcal{S}}(q_{\mathcal{S}})$, denoted by $\partial\mathcal{V}_{\mathcal{S}}(q_{\mathcal{S}})$. Let $\partial\mathcal{V}_{\mathcal{S}_{obs}}(q_{\mathcal{S}})$ denote the obstacle surface in $\partial\mathcal{V}_{\mathcal{S}}(q_{\mathcal{S}})$. Then, $\mathcal{V}_{\mathcal{S}}(q_{\mathcal{S}}) \bigcup \partial\mathcal{V}_{\mathcal{S}_{obs}}(q_{\mathcal{S}})$ is the new information obtained in each scan.

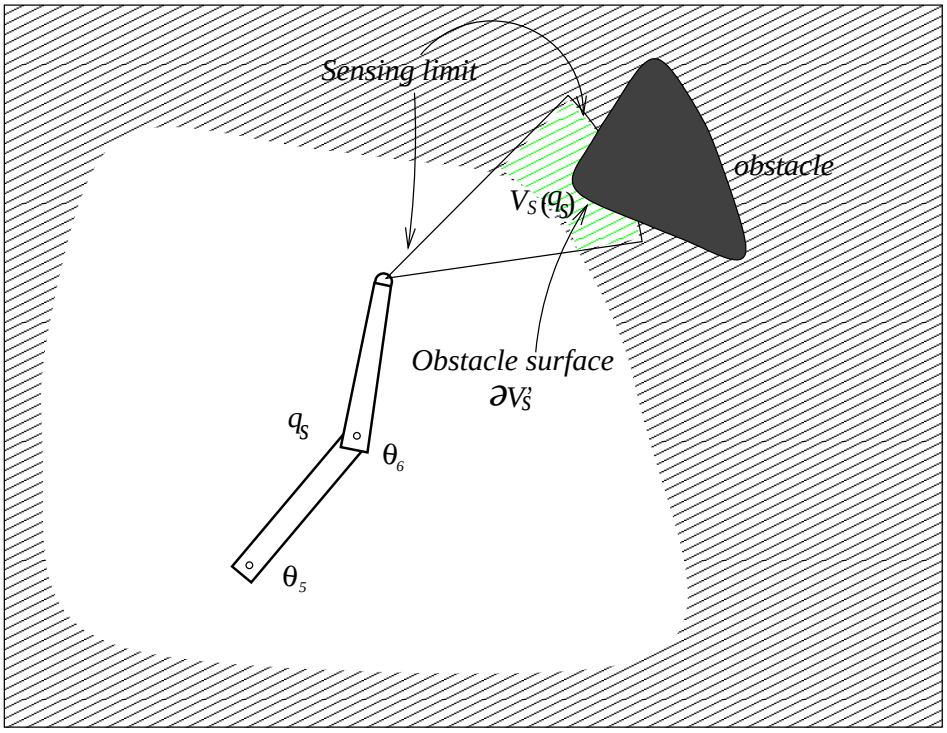


Figure 2.3: An illustration of sensed free region ($\mathcal{V}_{\mathcal{S}}$). Its entire boundary consists of the sensing limits and (part of) sensed obstacle surface $\mathcal{V}_{\mathcal{S}_{obs}}(q_{\mathcal{S}})$.

$\mathcal{P}^{(i)}_{free}, (\mathcal{P}^{(i)}_{obs}) \subset \mathcal{P}$ is the cumulative free physical (obstacle) space acquired (or seen) by the sensor, and $\mathcal{P}^{(i)}_{unk} \subset \mathcal{P}$ is the physical space remaining unknown, at a certain stage i of the planning (and simultaneously sensing) process (after the ith scan has been taken and processed). $\mathcal{P}^{(i)}_{free}$ is an open set. $\mathcal{P}^{(i)}_{obs}$ is a closed set. $\mathcal{C}^{(i)}_{free}, \mathcal{C}^{(i)}_{obs}, \mathcal{C}^{(i)}_{unk}$ denote the corresponding free C-space (an open set), obstacles (a closed set) in C-space, and still unknown C-space. $\mathcal{P}_{free}$ is the entire free physical space and $\mathcal{C}_{free}$ is the entire free C-space. Similarly, $\mathcal{P}_{obs}$ and $\mathcal{C}_{obs}$ are obstacles in physical space and C-space.

The *additional* increment in free physical space due to the ith scan of physical space is denoted by $\Delta\mathcal{P}_{free}^{(i)}$. Note there is overlap between $\mathcal{V}_{\mathcal{S}}$ and $\mathcal{P}_{free}^{(i-1)}$. $\Delta\mathcal{P}_{free}^{(i)}$ is the net increment (shown as light shaded area in Figure 2.3). Corresponding to $\Delta\mathcal{P}_{free}^{(i)}$, $\mathcal{C}_{free}^{(i-1)}$ is augmented by a net increment, $\Delta\mathcal{C}_{free}^{(i)}$.

Since $\mathcal{C}_{\mathcal{AS}} = \mathcal{C}_{\mathcal{A}} = \mathcal{C}_{\mathcal{S}}$ in our Eye-in-Hand system, for simplicity, we omit the use of subscript $\mathcal{A}$ and $\mathcal{S}$, and use $\mathcal{C}$ to denote $\mathcal{C}_{\mathcal{A}} = \mathcal{C}_{\mathcal{S}}$, and q to denote $q_{\mathcal{A}} = q_{\mathcal{S}}$ in subsequent sections when we discuss our algorithm for our Eye-in-Hand system in Chapters 3 and 4.

2.2 S-Feasible Path, S-Reachability, Observability and Explorability

With the notation introduced in the above section, we now introduce some new concepts for a sensor-based motion planning problem. We present these concepts in the continuous approach for easier theoretical analysis in this section. In practice, our sensor is a discrete time sensor, i.e. it only senses at discrete times. The concepts introduced in this section are general and applicable to both types of sensors.

Since a robot must have sensed a region (in any of the previous scans) before physically moving into it, the paths that the robot takes to a given configuration is, in general, different from model-based ones. The robot can only move into those regions already discovered to be free by the sensor. Such paths are fundamental in analysis of sensor-based MP and we call them s-Feasible paths.

s-feasible Path: Suppose a robot+sensor $\mathcal{AS}$ is in a physical environment $\mathcal{P}$. Robot has an initial free space $\mathcal{P}_{free}^{(0)}$. The initial configuration of the robot is q_0, and $\mathcal{A}(q_0) \subset \mathcal{P}_{free}^{(0)}$. Let $\pi_{[q_0,q_1]} \subset \mathcal{C}_{\mathcal{AS}}$ be a path from q_0 to q_1. q_a is a point on $\pi_{[q_0,q_1]}$ and let a one dimensional half-open set $\pi_{[q_0,q_a)}$ represent the segment of path $\pi_{[q_0,q_1]}$ from q_0 to q_a (include q_0, but exclude q_a), and let closed set $\pi_{[q_0,q_a]}$ represent the segment of path $\pi_{[q_0,q_1]}$ from q_0 to q_a (include both q_0 and q_a). Then, $\bigcup_{q \in \pi_{[q_0,q_a]}} \mathcal{A}(q)$ is the volume swept by the robot as it moves along the path segment, and $\bigcup_{q \in \pi_{[q_0,q_a)}} \mathcal{V}_{\mathcal{S}}(q)$ is the total sensing volume sensed all along the path segment before the robot reaches q_a. If $\forall q$

along the path, the volume swept by the robot is a subset of the total sensing volume previously sensed, then this path is a feasible path. Formally, $\forall q_a \in \pi_{[q_0,q_1]}$, if

$$\bigcup_{q \in \pi_{[q_0,q_a]}} \mathcal{A}(q) \subset (\bigcup_{q \in \pi_{[q_0,q_a)}} \mathcal{V}_{\mathcal{S}}(q)) \bigcup \mathcal{P}_{free}^{(0)}, \tag{2.1}$$

then path $\pi_{[q_0,q_1]}$ is an s-feasible path. If the above inclusion does not hold, the path is non-feasible.

If a path is s-feasible, there exists a sequence of iterations composed of sensing, planning and moving for $\mathcal{AS}$ to move to q_1, starting from q_0. The robot will never intersect $\mathcal{P}_{unk}$ along the path. On the other hand, if a robot $\mathcal{A}(q)$ can move along a path and sense in any subsets of the path without intersecting $\mathcal{P}_{unk}$, this path π must be an s-feasible path.

An s-feasible path can trace back on itself. This back-trace could be essential for the path to be an s-feasible path. Figure 2.4 shows an example of an s-feasible path with back-trace for a rectangular robot. This robot has a sensor which can sense within a cone with a fixed direction toward the right of the robot (shown as dotted lines). In order to sense region A, so that the robot can move to point e, the robot has to move to point c to sense. Path $a \rightarrow b \rightarrow c \rightarrow d \rightarrow e$ is an s-feasible path. Without the back-trace segment $b \rightarrow c \rightarrow d$, path $a \rightarrow b \rightarrow e$ is not an s-feasible path because region A can not be sensed on any point on path $a \rightarrow b \rightarrow e$.

For most robots with non-trivial geometry/kinematics (point or circular robot), what the sensor can sense and where the robot can go are intricately tied. We now introduce some definitions to characterize these concepts.

s-reachable Configuration: A configuration q_1 called s-reachable from q_0 if there exists an s-feasible path for $\mathcal{AS}$ to move from q_0 to q_1. Otherwise, we say q_1 is not s-reachable from q_0.

Observable Physical Space: Let $\mathcal{C}_{\mathcal{S}reach} = \{\text{all s-reachable configurations}\}$. Given an environment, $\bigcup_{q \in \mathcal{C}_{reach}} (\mathcal{V}_{\mathcal{S}}(q) \bigcup \partial \mathcal{V}_{\mathcal{S}obs}(q)) \subset \mathcal{P}$ is called the observable subset of this environment. We use $\mathcal{P}_{observ}$ to denote observable physical space.

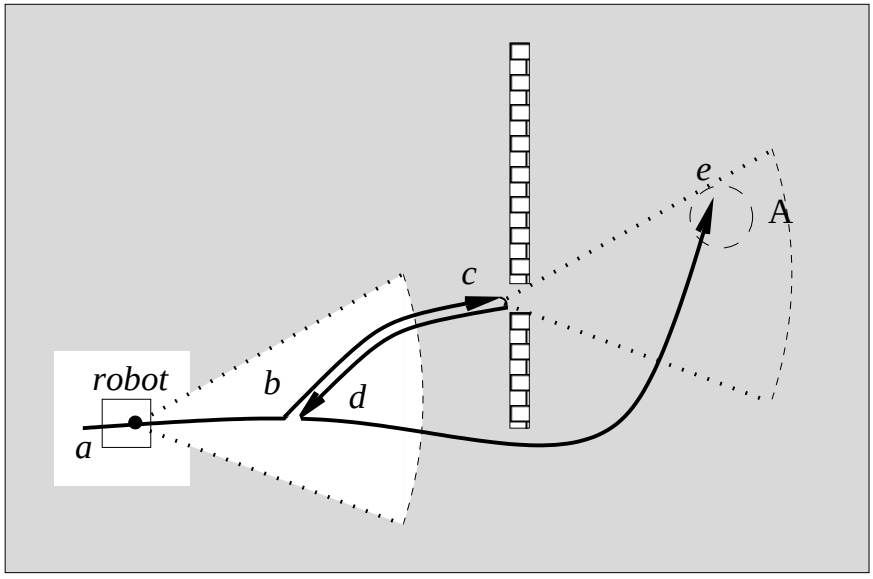


Figure 2.4: An example of an s-feasible with back tracing.

Explorable Configuration[1]**:** A configuration q is explorable if $\mathcal{A}(q) \subset \bigcup_{q \in \mathcal{C}_{reach}} \mathcal{V}_{\mathcal{S}}(q)$ ($q \in$ known $\mathcal{C}_{free}$) or $\mathcal{A}(q)$ intersects any part of a known obstacle ($q \in$ known $\mathcal{C}_{obs}$).

The existence of configurations which are not s-reachable but would be reachable if the environment were known *a priori* are the combination of the following three reasons:

(1) The extra inclusion condition (Equation 2.1) required for sensor-based motion planning, i.e., the robot can only move to and scan from those regions already discovered to be free by the sensor.

(2) Some obstacles occlude the view of the sensor or prevent the sensor from moving to desired places to scan. For example, in Figure 2.5, the robot can never scan the back of itself (gray region) and will never be able to move to q_{goal} although

[1]Strictly speaking, there exist two types of *explorable configurations*: *weak explorable configuration* and *strong explorable configuration*. If the entire $\mathcal{A}(q)$ can be "seen" by the sensor, i.e., $\mathcal{A}(q) \subset \mathcal{P}_{observ}$, q is a strong explorable configuration. The weak explorable configuration and strong explorable configuration are different when $q \in \mathcal{C}_{obs}$. The former requires to determine the status of q only while the latter requires that the entire $\mathcal{A}(q)$ is observable. We use explorable in the weak sense.

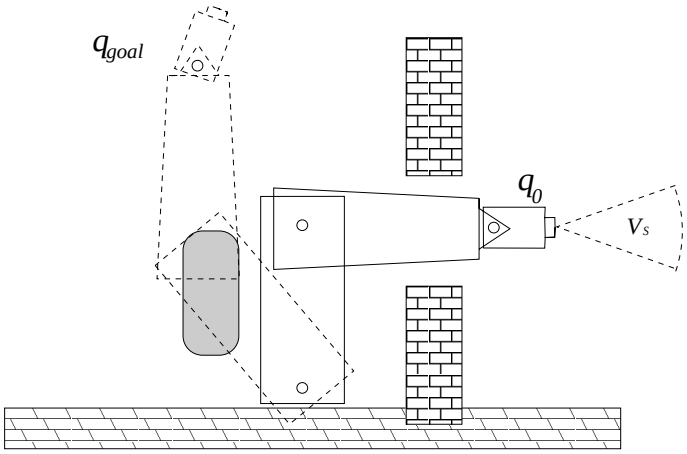


Figure 2.5: An example of configurations which is not s-reachable, but it would be reachable if the entire physical space were known *a priori*. The dash-line robot is not an s-reachable goal configuration from q_0. The gray region intersecting $\mathcal{A}(q_{goal})$ is free, but unknown to the planner. Although q_{goal} is reachable from a model-based motion planning point of view, the sensor-based planner can not find a path because the unknown region can not be sensed. None of the paths from q_0 to q_{goal} are s-feasible paths.

free paths exist, if the environment were known *a priori*.

(3) In general, each individual sensing action can not determine the status of a configuration. The eye sensor in our system is "partial" sensor in C-space. Certain types of robot+sensor may avoid the existence of such configurations and non-s-feasible paths. When a robot can sense an adjacent region around $\mathcal{A}(q)$, for example when a robot is completely covered with skin sensors with a certain sensing range, the robot can always sense a small region in C-space around q. Mobile robots, when considered as a single point, have identical physical and configuration space. Their sensors can be viewed as having the capability to sense around q. But, the sensor in this system does not have such capability – sensing an adjacent region around the current configuration q.

Our conjecture is that this type (robot can sense an adjacent region of q) is the only class of robot+sensor system for which the set of s-reachable configurations is the same as the set of reachable configurations in model-based motion planning, and all the free paths are s-feasible for *any* physical environment settings.

2.3 The Problems

2.3.1 Assumptions

Our two key assumptions are: (i) the robot environment is static, with $\mathcal{AS}$ being the only moving entity, and (ii) a small region of physical space, $\mathcal{P}_{free}^{(0)}$, surrounding the robot and sensor in its initial configuration is free. This latter requirement is important because otherwise even the very first move may not be possible after the initial scan assuming the first scan can be made.

The robot $\mathcal{A}$ with a range sensor $\mathcal{S}$ starts off at configuration q_0. The sensor takes scans at discrete times. Each time it senses a free region $\mathcal{V}_{\mathcal{S}}$ and obstacle $\partial\mathcal{V}_{\mathcal{S}_{obs}}$.

2.3.2 Problem Statement

Different motion planning problems can be formulated. We state three typical ones:

Problem 1 – Start – goal problem: Given a robot+sensor, an initial known free region $\mathcal{P}_{free}^{(0)}$, an initial configuration $q_{\mathcal{AS}}^{(0)}$ ($\mathcal{A}(q_{\mathcal{AS}}^{(0)}) \subset \mathcal{P}_{free}^{(0)}$), and a goal configuration $q_{\mathcal{AS}}^{(goal)} \in \mathcal{C}_{\mathcal{AS}}$, determine an s-feasible path from $q_{\mathcal{AS}}^{(0)}$ to $q_{\mathcal{AS}}^{(goal)}$. Report failure if no such path exists.

Problem 2 – Exploration problem: Given a robot+sensor, an initial known free region $\mathcal{P}_{free}^{(0)}$ and an initial configuration $q_{\mathcal{AS}}^{(0)}$ ($\mathcal{A}(q_{\mathcal{AS}}^{(0)}) \subset \mathcal{P}_{free}^{(0)}$), determine an s-feasible path, such that the volume $\mathcal{C}_{unk}$ at the end of the path is minimized.

There might be different measures for the exploration, suitable for different purposes. For example, if the robot task is to search an object, then the measure for exploration should be defined in physical space.

Problem 3 – Hybrid problem: Combination of problem 1 and 2. A task of this type has a goal and also more exploration is appreciated. The result of the exploration can be beneficial for the next task. The weights for exploration and reaching the goal can vary for different tasks. For example, if many subsequent tasks follow, the first task may weight the exploration more heavily.

Problem 1 is the extension of basic model-based motion planning problem – finding a collision free path between given start and goal configurations. Problem 2 is unique

to sensor-based motion planning problem since the entire environment, and hence the C-space, is assumed to be known for model-based motion planning. The purpose of exploration problem is to construct the C-space for the robot. Problems 1 and 3 are what we discuss in this thesis. Section 4.3 will discuss an implementation of the weight parameters to an "exploratory" component and a "goal" component for hybrid problems.

2.4 Completeness for Sensor-based Motion Planning

Completeness has different implications for the sensor-based problems mentioned in the previous section. In this thesis, we only study the completeness issue of start–goal problems for manipulator robots with eye sensors. The completeness of exploration problem is beyond the scope of this thesis.

Completeness for the model-based motion planning problem is defined as whether an algorithm can find a collision-free path to an arbitrary goal if there exists one (goal is reachable) and report a failure if no path exists. Because motion planning problems are very complex in real applications, many incomplete algorithms have been proposed. These algorithms can solve many motion planning problems in practical situations, but are not complete. For example, grid-based motion planning algorithms [40] can not find an existing collision-free path when this path goes through a channel which is narrower than the resolution of the grids. Completeness is proposed to characterize this property of the algorithms, see reference [18]. In motion planning, an algorithm is complete for a problem if it is guaranteed, for all instances of the problem, to find a solution when one exists or to return failure otherwise[2].

For sensor-based motion planning, paths exist for s-reachable goal only. The only meaningful completeness for sensor-based motion planning is s-completeness.

[2]The term complete has been used in many areas with very different meanings. For example, in algorithm complexity analysis (a problem to which all the problems in a class can be reduced is complete for this class – see [13]), signal processing (a set of functions which can represent any signal by their linear combination).

s-completeness: An s-complete algorithm for the start–goal problem should return an s-feasible path to the goal if such a path exists, for any given environment and any initial free region $\mathcal{P}_{free}^{(0)}$. Otherwise it should report that the goal is not s-reachable.

Before more discussion of the completeness issue, a special subset of physical space should be mentioned as follows. Let $\mathcal{E} = (\bigcup_{q \in cl(\mathcal{C}_{free}^{(i)}(q_0))} \mathcal{A}(q)) \bigcap \mathcal{P}_{unk}$, where $cl()$ stands for the closure of an open set, $\mathcal{C}_{free}(q_0)$ is the connected component of $\mathcal{C}_{free}^{(i)}$ containing q_0. $\mathcal{E}$ is essentially a part of the boundary between unknown physical space and free physical space. Let $\mathcal{E}_\mathcal{V} = \mathcal{E} \bigcap (\bigcup_{q \in \mathcal{C}_{free}^{(i)}(q_0)} \mathcal{V}_\mathcal{S}(q) \bigcup \partial \mathcal{V}_{\mathcal{S}obs}(q))$. Note $\mathcal{V}_\mathcal{S}(q) \bigcup \partial \mathcal{V}_{\mathcal{S}obs}(q)$ is the visible set of $\mathcal{P}$ from configuration q. $\mathcal{E}_\mathcal{V}$ is essentially the part of boundary between $\mathcal{P}_{unk}^{(i)}$ and $\mathcal{P}_{free}^{(i)}(q_0)$ that is touchable by a part of the robot if the robot were to move within the entire $cl(\mathcal{C}_{free}^{(i)}(q_0))$, and is visible from $\mathcal{C}_{free}^{(i)}(q_0)$.

Recall that the two key components of sensor-based motion planner are C-space expansion and view planning. The algorithm in Figure 1.4 is s-complete if (1) the underlying (model-based) planner is complete, and (2) the view planner is complete. Here, a complete view planner is defined: if there exists a point $x \in \mathcal{E}_\mathcal{V}$, then a complete view planner can find a configuration q, such that point x can be sensed from configuration q, i.e., $x \in \mathcal{V}_\mathcal{S}(q) \bigcup \partial \mathcal{V}_{\mathcal{S}obs}(q)$. Otherwise, report $\mathcal{E}_\mathcal{V} = \emptyset$.

The algorithm in Figure 1.4 will eventually stop when one of the following stop conditions is satisfied: (1) when goal is reached, or (2) when $\mathcal{E}_\mathcal{V} = \emptyset$ and goal is not reached.

The first condition of s-completeness (the underlying model-based planner is complete) is obvious. We only consider the second condition (the view planner is complete). The proof is by contradiction. We first assume that the algorithm stopped without finding an existing s-feasible path. Under this assumption, $\mathcal{E}_\mathcal{V}$ can not be an empty set, which is contradict with the stop condition. Therefore, the algorithm can not stop without finding the existing s-feasible path.

Assume that there exists an s-feasible path π starting from q_0 to q_{goal}, but the algorithm can not find an s-feasible path. Then, the second stop condition is satisfied, (see Figure 2.6). When the algorithm stops, known free C-space is $\mathcal{C}_{free}^{(i)}$, and known physical space is $\mathcal{P}_{free}^{(i)}$, and etc. Both $\mathcal{C}_{free}^{(i)}$ and $\mathcal{P}_{free}^{(i)}$ are open sets. Since π

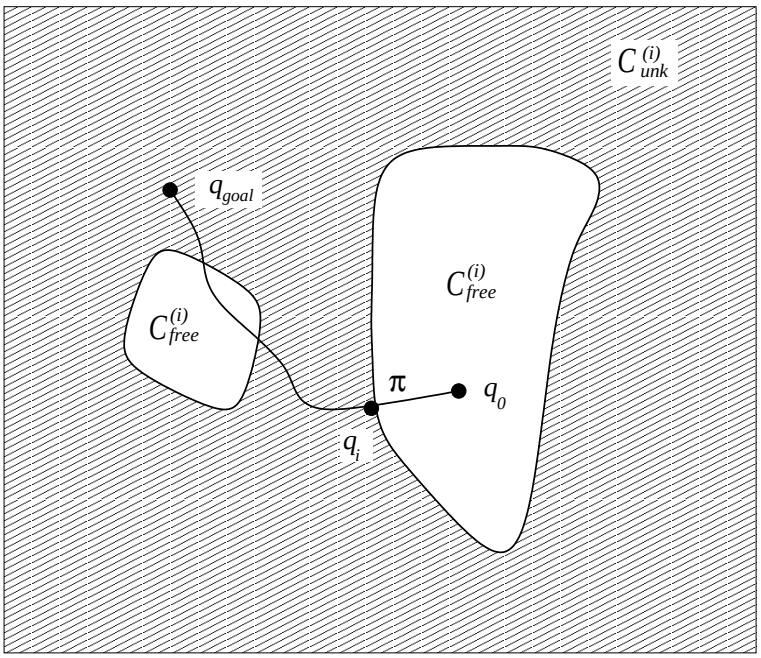


Figure 2.6: The status of the C-space when the algorithm stops without finding any s-feasible path, and there exists is an s-feasible path π. π intersects the boundary of $\mathcal{C}^{(i)}_{unk}$ and $\mathcal{C}^{(i)}_{free}(q_0)$ at q_i.

is an s-feasible path, π can not intersect $\mathcal{C}_{obs}$. π can not lie in $\mathcal{C}_{free}^{(i)}$ completely either (otherwise, underlying model-based planner would find this path). Therefore, π must intersect $\mathcal{C}_{unk}^{(i)}$ the unknown part of C-space at iteration i. Let q_i be the first intersection of π and the boundary between $\mathcal{C}_{unk}^{(i)}$ and $\mathcal{C}_{free}^{(i)}(q_0)$. Since $q_i \in \mathcal{C}_{unk}^{(i)}$, part of $\mathcal{A}(q_i)$ should intersect $\mathcal{P}_{unk}^{(i)}$, i.e, $\mathcal{A}_{unk}(q_i) \neq \emptyset$ ($\mathcal{A}_{unk}(q_i)$ is the part of $\mathcal{A}(q_i)$ in unknown physical space). Since q_i is on the boundary of $\mathcal{C}_{unk}^{(i)}$ and $\mathcal{C}_{free}^{(i)}(q_0)$, $q_i \in cl(\mathcal{C}_{free}(q_0))$, and then $\mathcal{A}_{unk}(q_i) \subset \mathcal{E}$. On the other hand, on this s-feasible path π, by definition, $\bigcup_{q \in \pi_{[q_0,q_i]}} \mathcal{A}(q) \subset \bigcup_{q \in \pi_{[q_0,q_i)}} \mathcal{V}_{\mathcal{S}}(q)$. This implies that the sensor can sense entire $\mathcal{A}_{unk}(q_i)$ on the path segment of π from q_0 to q_i (before reaching q_i). Specifically, $\mathcal{A}_{unk}(q_i) \subset \bigcup_{q \in \pi_{[q_0,q_i)}} \mathcal{V}_{\mathcal{S}}(q)$. Because the way q_i is chosen, (it is the first intersection point on π with $\mathcal{C}_{unk}$,) $\pi_{[q_0,q_i)} \subset \mathcal{C}_{free}^{(i)}(q_0)$. Hence, $\mathcal{A}_{unk}(q_i)$ is visible from $\mathcal{C}_{free}^{(i)}(q_0)$. Therefore, $\mathcal{A}_{unk}(q_i) \subset \mathcal{E}_{\mathcal{V}}$. Therefore, there is still a subset of $\mathcal{E}_{\mathcal{V}}$ which is visible when the algorithm stops. This contradicts the stop condition of the algorithm. Therefore, the algorithm can not stop without finding path π.

Hence, if the underlying model-based planner and the view planner are both complete, the corresponding senor-based planner with the algorithm in Figure 1.4 is s-complete.

Chapter 3

C-space Entropy and Its Application to View Planning

Sensor-based motion planning starts with a robot knowing nothing or little about the physical environment and hence the configuration space. In the process of planning, the knowledge about the environment increases as each scan is taken. We take an information theoretical view of sensing and consider it as increasing "knowledge/information" about the environment, or equivalently decreasing the "lack of information" or "ignorance" about the environment. The problem then becomes planning a scan which efficiently reduces the ignorance. However, since the natural planning space is the C-space, we need to pose this "information" concept in C-space.

In this chapter, we assume the sensor is not subject to occlusion constraint, i.e., this sensor returns the status of every point in a subset to be sensed in the physical space. Note that for real sensor, if there is an occluding obstacle, the region behind the obstacle may remain unknown after a scan. We also assume that the sensing region of the sensor is small. Therefore, the information gain in each iteration can be approximated by a linear function.

3.1 C-space Entropy $H(\mathcal{C})$: The Basic Notion

In this section, we first introduce some related background of the entropy theory, and then illustrate the concept of C-space entropy and its calculation with a simple robot example.

From a motion planning point of view, the status of a given configuration ($\in \mathcal{C}_{obs}$ or $\in \mathcal{C}_{free}$) is all the robot needs to know. Therefore, we need a function to measure the knowledge (or ignorance) of obstacles and free space in C-space.

The standard mathematical measure of "information" is the notion of entropy, which describes the lack of information about a system [61, 62]. Naturally, we use the entropy $H(\mathcal{C})$ as a measure of ignorance of the C-space, and call it C-space entropy. The first approach to calculate entropy is via its definition. Recall that the entropy of a discrete state random variable X (X takes values in a discrete set $\{a_1, a_2, ..., a_n\}$) is defined as $H(X) = -\sum_i p_i \log p_i$, where $p_i = p(X = a_i)$ [30]. Similarly, for two random variables X and Y (X takes values in a set $\{a_1, a_2, ..., a_n\}$; Y takes values in a set $\{b_1, b_2, ..., b_n\}$), $H(X,Y) = -\sum_i \sum_j p(X = a_i, Y = b_j) \log p(X = a_i, Y = b_j)$. With joint probabilities ($p(X = a_i, Y = b_j)$), the entropy can be calculated. The second approach to calculate the entropy is via mutual entropy. Entropy of two random variables can also be written as $H(X,Y) = H(X) + H(Y) - I(X,Y)$, where $I(X,Y)$ is the mutual information, with $I(X,Y) = \sum_i \sum_j p(X = a_i, Y = b_j) \log \frac{p(X = a_i, Y = b_j)}{p(X = a_i)p(Y = b_j)}$. Mutual information is the amount of common information contained in X and Y. When X and Y are independent, $I(X,Y) = 0$. In the above expression, $H(X)$ and $H(Y)$ can be calculated as in the single random variable case. Marginal probabilities $p(X = a_i) = \sum_j p(X = a_i, Y = b_j), p(Y = b_j) = \sum_i p(X = a_i, Y = b_j)$ are used in the calculation of $H(X)$ and $H(Y)$ respectively.

Both of these two approaches (direct calculation or via mutual information) can be used to calculate C-space entropy. We will illustrate the C-space entropy calculation with the first approach. The second approach is useful in deriving an approximate expression for practical cases when the C-space is complex. This is discussed in Section 3.2.

First we give a simple example to illustrate the concept of C-space entropy. Consider a single link robot with only two points a and b. Point b rotates around point a, which is fixed, as shown in Figure 3.1. Assume that the link can achieve two configurations, i.e. $\mathcal{C} = \{q_1 = 0^\circ \ , \ q_2 = -90^\circ\}$. The robot is equipped with an abstract sensor that senses a single physical point and determines its status (obstacle/free) in each scan. (We do not consider the problem "from where to scan this point" in this chapter.)

The probabilistic distribution of physical space determines the probabilistic distribution of C-space. In order to calculate the entropy of C-space for the two-point robot, we need a probabilistic model for the physical space. Because of the simple kinematics and the simple two-point C-space of the robot, the relevant physical space (workspace of the robot), $\mathcal{P} = \{A, B, C\}$. Let an obstacle point be represented as value 1 and a free point be represented as value 0. Assume that each point (unknown) in the physical space has a probability 0.5 for being free, i.e., $p(A = 0) = p(B = 0) = p(C = 0) = 0.5$, where $p(A = 0)$ denotes the probability that point A is free. We further assume that the status (free or obstacle) of points A, B and C are *independent* binary random variables, i.e. $p(A, B, C) = p(A)p(B)p(C)$.

Now, we can calculate the joint probability function of the C-space. Let random variable $C(q)$ represent the status (free or obstacle) of C-space at configuration q. $C(q_1)$ and $C(q_2)$ are then functions of physical space. Similar to the physical space, $C(q) = 0$ represents that configuration q is free, which implies entire robot is in free physical space. $C(q) = 1$ represents that the robot intersects obstacle points. For example, when $A = 0, B = 0, C = 1$, then $C(q_1) = 0, C(q_2) = 1$. $C(q_1)$ and $C(q_2)$ are binary random variables. From the probabilities of the points A, B and C, we can calculate the joint probabilities of $C(q_1)$ and $C(q_2)$. The probabilities in Table 3.1 is calculated by discrete random variable transformation (see page 141 in [53] for examples of discrete random random variable transformation). The basic idea of discrete random variable transformation is to find equivalent situations in the C-space and the physical space. For example, $(C(q_1) = 0, C(q_2) = 1)$ is equivalent to (A=0, B=0, C=1). Therefore, in the second row, $p(C(q_1) = 0, C(q_2) = 1) = p(A = 0, B = 0, C = 1) = p(A = 0)p(B = 0)p(C = 1) = 0.125$. The probabilities of all the

permutations are listed in this table.

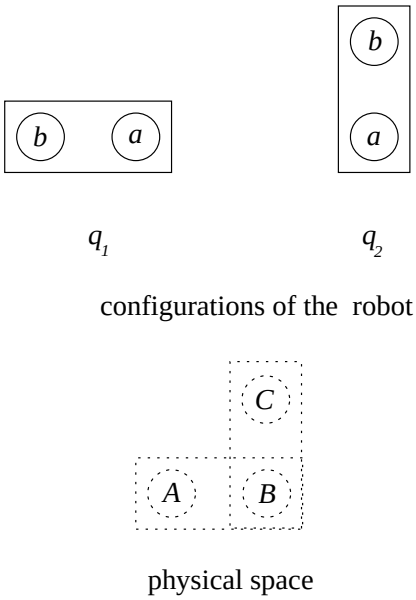


Figure 3.1: A simple single-link two-point robot and its physical space to illustrate the concept of C-space entropy and view planning via the maximal entropy reduction criterion. The physical space $\mathcal{P} = \{A, B, C\}$. C-space of the robot $\mathcal{C} = \{q_1 = 0^\circ\ , q_2 = -90^\circ\}$.

Using the definition of entropy of two random variables as mentioned earlier, the entropy of the C-space is $H(\mathcal{C}) = -\sum_{\substack{C(q_1)\in\{0,1\} \\ C(q_2)\in\{0,1\}}} p(C(q_1), C(q_2)) \log p(C(q_1), C(q_2)) = 1.5488$, where $p()$ is the joint probability listed in the right column in Table 3.1. This reflects the robot's ignorance of the C-space.

3.2 View Planning with C-space Entropy $H(\mathcal{C})$

As mentioned earlier, our view of sensing is that it reduces "ignorance" of the C-space. Now that $H(\mathcal{C})$ provides a measure, naturally, the next sensing must be to reduce the entropy as much as possible.

Ideally, it would be preferable to find a sequence of scanning, by which the entropy $H(\mathcal{C})$ is maximally reduced in the least number of scans. However, such a procedure is

C-space	physical space	probability $p(C(q_1), C(q_2))$
$C(q_1) = 0, C(q_2) = 0$	$A = 0, B = 0, C = 0$	0.125
$C(q_1) = 0, C(q_2) = 1$	$A = 0, B = 0, C = 1$	0.125
$C(q_1) = 1, C(q_2) = 0$	$A = 1, B = 0, C = 0$	0.125
$C(q_1) = 1, C(q_2) = 1$	*the rest*	0.625

Table 3.1: The probabilities of the status of different configurations before a scan. $C(q)$ is the status of C-space at q. In both the physical space and the C-space, the obstacle status is represented as 1, and the free status is represented as 0.

not possible because the environment is unknown. The only approach is to maximize the *expected value* of the entropy reduction, denoted by $E(-\Delta H())$, after a scan. We call this approach *maximal entropy reduction (MER) criterion.*

Formally, view planning is to choose a scan which

$$\max_{all\ possible\ scans} E(-\Delta H(\mathcal{C})) = \max_{all\ possible\ scans} (-E(H(\mathcal{C}_{after\ scan})) + H(\mathcal{C}_{before\ scan})), \tag{3.1}$$

where $\mathcal{C}_{after\ scan}$ is the C-space after a scan is made and $\mathcal{C}_{before\ scan}$ is the C-space before the scan.

With the same example as in Section 3.1, we illustrate the view planning. We calculate the expected entropy of the C-space after a scan is taken, i.e., one of the points in the physical space is sensed with the abstract sensor. In the following sections, we use $H(\mathcal{C}_x)$ to denote C-space entropy after point $x \in \mathcal{P}$ is scanned and $H(\mathcal{C}_{(x=\xi)})$ to represent the entropy after point $x \in \{A, B, C\}$ is sensed and the status of the point changes to $\xi \in \{0(free), 1(obstacle)\}$. For example, $H(\mathcal{C}_{(A=1)})$ represents the entropy after point A is sensed and changed to an obstacle point.

Sense Point A

When point A is sensed, we expect one of two possible outcomes:

- (i) point A becomes free. The probabilities of the configurations are listed in the left side of Table 3.2. $H(\mathcal{C}_{(A=0)}) = -\sum p(C(q_1), C(q_2) \log p(C(q_1), C(q_2)$. Plug in the values in Table 3.2, $H(\mathcal{C}_{(A=0)}) = -(0.25 \log_2 0.25 + 0.25 \log_2 0.25 + 0.0 + 0.5 \log_2 0.5) = 1.50$. Recall we calculate the entropy before any point is sensed, $H(\mathcal{C}) = 1.5488$ in the last section. We use $\underset{A \Rightarrow 0}{\Delta} H(\mathcal{C})$ to denote the entropy difference before point A is sensed (A is unknown) and after point A becomes free (and similarly $\underset{A \Rightarrow 1}{\Delta} H(\mathcal{C})$ represents the entropy difference before point A is sensed and after point A becomes obstacle), then

C-space	physical space	$p(C(q_1),C(q_2))$
$C(q_1)$=0,$C(q_2)$=0	B=0,C=0	0.25
$C(q_1)$=0,$C(q_2)$=1	B=0,C=1	0.25
$C(q_1)$=1,$C(q_2)$=0	impossible	0.0
$C(q_1)$=1,$C(q_2)$=1	B=1	0.5

C-space	physical space	$p(C(q_1),C(q_2))$
$C(q_1)$=0,$C(q_2)$=0	impossible	0.0
$C(q_1)$=0,$C(q_2)$=1	impossible	0.0
$C(q_1)$=1,$C(q_2)$=0	B=0,C=0	0.25
$C(q_1)$=1,$C(q_2)$=1	C=1 *or* (C=0,B=1)	0.75

Table 3.2: Left table: the probabilities of the status of different configurations after scanning point A, assuming A becomes free. The third row is a situation which can never happen when point A is free. Right table: The probabilities of the status of different configurations after scanning point A, assuming point A becomes obstacle.

$$\underset{A \Rightarrow 0}{\Delta} H(\mathcal{C}) = H(\mathcal{C}_{(A=0)}) - H(\mathcal{C}) = 1.50 - 1.5488 = -0.0488.$$

- (ii) point A becomes obstacle. The probabilities of the configurations are listed in the right side of Table 3.2. The entropy is $H(\mathcal{C}_{(A=1)}) = -\sum p \log p = 0.8113$, and then

$$\underset{A \Rightarrow 1}{\Delta} H(\mathcal{C}) = H(\mathcal{C}_{(A=1)}) - H(\mathcal{C}) = -0.7375.$$

The expected reduction in the entropy of the new C-space after point A is scanned is $E(\Delta H(\mathcal{C}_A)) = p(A=0) \underset{A \Rightarrow 0}{\Delta} H(\mathcal{C}) \quad + \quad p(A=1) \underset{A \Rightarrow 1}{\Delta} H(\mathcal{C}) = -0.3932.$

Sense Point B

Similarly, when point B is scanned, again we expect one of two possible outcomes:

- (i) point B becomes free. The probabilities of the configurations are listed in Table 3.3 . The entropy is $H(\mathcal{C}_{(B=0)}) = -\sum p \log p = 2.00$, and then $\underset{B \Rightarrow 0}{\Delta} H(\mathcal{C}) = H(\mathcal{C}_{(B=0)}) - H(\mathcal{C}) = 0.4512$.

C-space	physical space	$p(C(q_1),C(q_2))$
$C(q_1)$=0,$C(q_2)$=0	A=0,C=0	0.25
$C(q_1)$=0,$C(q_2)$=1	A=0,C=1	0.25
$C(q_1)$=1,$C(q_2)$=0	A=1,C=0	0.25
$C(q_1)$=1,$C(q_2)$=1	A=1,C=1	0.25

C-space	physical space	$p(C(q_1),C(q_2))$
$C(q_1)$=0,$C(q_2)$=0	*impossible*	0.0
$C(q_1)$=0,$C(q_2)$=1	*impossible*	0.0
$C(q_1)$=1,$C(q_2)$=0	*impossible*	0.0
$C(q_1)$=1,$C(q_2)$=1	A,C *can be either* 0 *or* 1	1.0

Table 3.3: Left table: the probabilities of the status of different configurations after scanning point B, assuming point B becomes free. Right table: The probabilities of the status of different configurations after scanning point B, assuming point B becomes obstacle.

- (ii) point B becomes obstacle. The probabilities of the configurations are listed in Table 3.3 . The entropy is $H(\mathcal{C}_{(B=1)}) = -\sum p \log p = 0.00$, and then $\underset{B \Rightarrow 1}{\Delta} H(\mathcal{C}) = H(\mathcal{C}_{(B=1)}) - H(\mathcal{C}) = -1.5488$.

The expected entropy reduction of the new C-space, after point B is scanned, is $E(\Delta H(\mathcal{C}_B)) = p(B=0) \underset{B \Rightarrow 0}{\Delta} H(\mathcal{C}) + p(B=1) \underset{B \Rightarrow 1}{\Delta} H(\mathcal{C}) = -0.5488$.

We can similarly calculate the expected reduction in entropy of the C-space after point C is sensed. $E(\Delta H(\mathcal{C}_C)) = E(H(\mathcal{C}_C)) - H(\mathcal{C}) = -0.3932$.

$x_{max} = \{x : \max_{x \in \mathcal{P}}\{-E(\Delta H(\mathcal{C}_x))\}\} = B$. Scanning B will result in the maximum entropy reduction. Indeed, as intuition directed us, point B is the point that results in maximum entropy reduction, and hence is the place to be sensed in the next step.

3.3 C-space Entropy for Real Robots

In practice, the C-space generally is a multi-dimensional continuous space. For example, the robot in our Eye-in-Hand system has six degrees of freedom. Hence, the robot C-space is six dimensional. Because the $\mathcal{C}_{obs}$ (and $\mathcal{C}_{free}$) distribution is unknown, the status of each configuration in the C-space is a random variable. We can view the C-space as a 6-dimensional stochastic process. Then, the definition of entropy of stochastic process can be used in our problem.

From the signal transmission problems treated in the classic papers by Shannon[61, 62], we know that the exact description of a continuous time stochastic process requires an infinite number of bits for representation. To overcome this problem, Shannon studied the minimum information rate that is needed to achieve the description and transmission of a continuous information source within pre-defined particular distortions. One of the distortion types allows the stochastic process to be discretized. Let us consider a discrete time stochastic process $X = \{X_n\}$, where each X_n is a discrete state random variable. Then the information rate of the discrete stochastic process can be expressed as [30] $\bar{H}(X) = \lim_{N\to\infty} \frac{1}{N} H(X_1, X_2, \cdots, X_N)$, where $H()$ is the entropy function of N random variables.

In our case, we discretize the C-space to a certain spatial resolution (say r discretized levels in each dimension). We view the C-space as a 6-dimensional discrete stochastic signal, in which each discretized configuration takes a value in a binary set {0(free), 1(obstacle)}. Spatial discretization of the C-space is reasonable due to many facts such as: (1) we are interested in gross motion planning, and (2) robots have finite accuracy, etc. Since the C-space of the articulated robots is compact in most cases (either intrinsically or in practice via joint limits) [40], the C-space is then viewed as a random vector $(C(q_0), C(q_1), \cdots, C(q_{N-1}))$. $N = r^{n_{\mathcal{A}}}$ is the total number of discretized configurations, where r is the number of discretized values for each degree of freedom, and $n_{\mathcal{A}}$ is dimension of C-space.

The entropy of C-space is therefore the entropy of the random vector, i.e. $H(\mathcal{C}) = H(C(q_0), C(q_1), \cdots, C(q_{N-1}))$. The joint probability of $C(q_0), C(q_1), \cdots, C(q_{N-1}), p(C(q_0), C(q_1), \cdots, C(q_{N-1}))$ should be used to compute $H(\mathcal{C})$.

Calculation of the joint distribution $p(C(q_0), C(q_1), \cdots, C(q_{N-1}))$ would be extremely intensive for any non-trivial robot. We now examine the second approach (via mutual information) to C-space entropy calculation as mentioned in Section 3.1. Similar to the two variable case,

$$\begin{aligned} H(\mathcal{C}) &= - \sum_{C(q_i)\in\{0,1\}} p(C(q_0), C(q_1), \cdots, C(q_{N-1})) \log p(C(q_0), C(q_1), \cdots, C(q_{N-1})) \\ &= \sum_{i=0}^{N-1} H(C(q_i)) - \sum_{i\neq j} I(C(q_i), C(q_j)) + \\ &\quad \sum_{i,j,k \ \ different} I(C(q_i), C(q_j), C(q_k)) - \cdots, \end{aligned} \tag{3.2}$$

where $I(\cdot)$ is the mutual information of multiple random variables. Mutual information of random variables is the information about one of the random variables contained in the other [53]. The mutual information is caused by the independence between the two random variable. In our case, the random variables are the status of the configurations in C-space. The following two sources cause the independence of the status of two configurations: (1) The overlap between the physical space the robot occupied at different configurations, e.g. $\mathcal{A}(q_i)$ and $\mathcal{A}(q_j)$. The overlap part of $\mathcal{A}(q_i)$ and $\mathcal{A}(q_j)$ affects the status of both $C(q_1)$ and $C(q_2)$, and therefore causes the dependency of $C(q_1)$ and $C(q_2)$; (2) the correlation between the status (free/obstacle) of different positions in physical space $C(q_1)$ and $C(q_2)$. When certain parts of $\mathcal{A}(q_i)$ and $\mathcal{A}(q_j)$ (in physical space) are related, $C(q_1)$ and $C(q_2)$ are also related.

In order to calculate the mutual information, it is necessary to know the intersections of all pair $\mathcal{A}(q_i)$'s, all triplets and etc. Although it is possible to compute the intersections, such computation would be expensive. Even for the terms of the mutual information of two random variables in Equation 3.3, the computational complexity is an order of $O(N^2)$. Calculating these terms could require hours of computation on our current computers[1]. Therefore, we neglect all the mutual entropies in the previous formula. The approximated calculation of C-space entropy is based on the marginal

[1]Such mutual information indeed could be useful; however we are ignoring it for computational ease. Its relative importance needs further investigation.

probabilities only, which are calculated much faster. We define

$$\widetilde{H}(\mathcal{C}) = \sum_{i=0}^{N-1} H(C(q_i)) \tag{3.3}$$

as an approximation of the real $H(\mathcal{C})$, where $H(C(q_i))$ is the entropy of $C(q_i)$. $H(C(q))$ can be calculated using the following equation

$$H(C(q)) = -(p(q)\log p(q) + (1-p(q))\log(1-p(q))), \tag{3.4}$$

where $p(q)$ is an abbreviation for $p(C(q)=0)$, i.e. the probability of $q \in \mathcal{C}_{free}$.

	$H(\mathcal{C})$	$\widetilde{H}(\mathcal{C})$
Before scanning	1.5488	1.6226
Expected value after scanning A or C	1.1556	1.3113
Expected value after scanning B	1.0	1.0

Table 3.4: Comparison of C-space entropy $H(\mathcal{C})$ and its approximation $\widetilde{H}(\mathcal{C})$ for the two-point single joint robot example in previous sections.

This approximation essentially neglects the dependency between different $C(q_i)$'s, which is reflected in the mutual information. Therefore, it over-estimates the importance of those regions in physical space, which are occupied frequently by the robot. In the two-point robot example, the importance of scanning point B will be affected to a certain extent (although point B *is* important as shown in Section 3.2 with the exact calculation).

Table 3.4 lists the real entropies $H(\mathcal{C})$ and the approximated entropies $\widetilde{H}(\mathcal{C})$. It is interesting to point out that $H(\mathcal{C})$ and $\widetilde{H}(\mathcal{C})$ are equal after point B is scanned. This is because there is no common point jointly affecting the status of both q_1 and q_2 after B is scanned, and hence $C(q_1)$ and $C(q_2)$ become independent (assuming A, B, and C are independent).

3.4 Information Gain Density

$H(\mathcal{C})$ is defined as a function of C-space. But it is ultimately determined by the status of physical space. From this point of view, when a region in the physical space is scanned, the entropy would be changed. We could compute this entropy change for every region. The best region to be scanned is the one that results in a maximum reduction of $H(\mathcal{C})$. When the size of the region is small, the entropy reduction can be approximated as $(\lim_{vol(\mathcal{B}(x))\to 0} \frac{-E\Delta H(\mathcal{C})}{vol(\mathcal{B}(x))}) vol(\mathcal{B}(x))$, where $\mathcal{B}(x)$ is an open region in physical space containing x. Then, searching for the region to scan in order to achieve the maximum entropy reduction is equivalent to searching for the position which maximizes $\lim_{vol(\mathcal{B}(x))\to 0} \frac{-E\Delta H(\mathcal{C})}{vol(\mathcal{B}(x))}$ because the sensing region size is fixed. The concept that captures this process is the *information gain density function (IGDF).*

IGDF is defined as $G_e(x) = \lim_{vol(\mathcal{B}(x))\to 0} \frac{-E\Delta H(\mathcal{C})}{vol(\mathcal{B}(x))}$, where $\mathcal{B}(x)$ is an open region in physical space containing x, $vol(\mathcal{B}(x))$ denotes the volume of region $\mathcal{B}(x)$, and $E\Delta H(\mathcal{C})$ is entropy reduction after $\mathcal{B}(x)$ is sensed and the status (obstacle or free) of entire $\mathcal{B}(x)$ is determined[2]. $G_e(x) \cdot vol(\mathcal{B}(x))$ approximates the expected information gain in the C-space when a small region, $\mathcal{B}(x)$, around x is sensed. Here, we assume that the status (free or obstacle) of $\mathcal{B}(x)$ can be determined after the scan, i.e. no obstacle occludes the view.

As in Section 3.3, for computational reasons, we use the approximated entropy, $\widetilde{H}(\mathcal{C})$, instead of $H(\mathcal{C})$, to obtain $G_e(x)$.

$$G_e(x) = \lim_{vol(\mathcal{B}(x))\to 0} \frac{-E\Delta \widetilde{H}(\mathcal{C})}{vol(\mathcal{B}(x))} = \lim_{vol(\mathcal{B}(x))\to 0} \frac{-\sum_q E\Delta H(C(q))}{vol(\mathcal{B}(x))}.$$

Interchanging the limit and summation in the above equation,

$$G_e(x) = \sum_q \lim_{vol(\mathcal{B}(x))\to 0} \frac{-E\Delta H(C(q))}{vol(\mathcal{B}(x))}. \tag{3.5}$$

[2]This limit may not always exist for arbitrary obstacle distributions. For example, when $\lim_{vol(\mathcal{B}(x))\to 0} E\Delta H(\mathcal{C}) \neq 0, G_e(x)$ does not exist. In this situation, other approaches of approximation has to be used. For now, we assume that this limit exists.

Let $g_q(x) = \lim_{vol(\mathcal{B}(x))\to 0} \frac{-E\Delta H(C(q))}{vol(\mathcal{B}(x))}$, then

$$G_e(x) = \sum_q g_q(x). \tag{3.6}$$

An interpretation of the above expression is as follows. When a point $x \in \mathcal{P}$ is scanned, it affects the C-space entropy via each configuration q. $g_q(x)$ is the expected contribution of q to the C-space entropy reduction when point x is scanned. $G_e(x)$ is the summation of $g_q(x)$ over all the configurations in the entire C-space. However, note that certain configurations do not contribute to $G_e(x)$. $g_q(x)$ equals to 0 when $\mathcal{B}(x) \not\subset \mathcal{A}(q)$, because changing the status of $\mathcal{B}(x)$ will not change the C-space entropy from configuration q. We call the region which has non-zero contribution *C-zone* of x. C-zone of x is denoted by $\mathcal{X}(x)$. Formally, C-zone of x, $\mathcal{X}(x) = \{q : x \in \mathcal{A}(q)\}$. $\mathcal{X}(x)$ is the set of configurations such that $\mathcal{A}(q)$ contains point x. Therefore, we explicitly write

$$G_e(x) = \sum_{q\in\mathcal{X}(x)} g_q(x). \tag{3.7}$$

An alternative computational view is to write

$$G_e(x) = \sum g_q(x)\delta(\mathcal{A}_{unk}(q)),$$

where $\mathcal{A}_{unk}(q)$ stands for the part of $\mathcal{A}(q)$ in $\mathcal{P}^{(i)}_{unk}$, and

$$\delta(\mathcal{A}_{unk}(q)) = \begin{cases} 1, \ x \in \mathcal{A}_{unk}(q), \\ \\ 0, \ otherwise. \end{cases}$$

Now, we discuss how to determine $g_q(x)$ or $E(\Delta H(C(q))$. Using a notation similar to the simple example we discussed, and assuming $\mathcal{B}(x) \subset \mathcal{A}(q)$, let

$$\underset{\mathcal{B}(x)\Rightarrow 0}{\Delta} H(C(q)) = H(C_{\mathcal{B}(x)=0}(q)) - H(C(q)), \tag{3.8}$$

$$\underset{\mathcal{B}(x)\Rightarrow 1}{\Delta} H(C(q)) = H(C_{\mathcal{B}(x)=1}(q)) - H(C(q)). \tag{3.9}$$

$H(C_{\mathcal{B}(x)=0}(q))$ is the entropy of $C(q)$ if $\mathcal{B}(x)$ is free. $H(C_{\mathcal{B}(x)=1}(q))$ is the entropy of $C(q)$ if any part of $\mathcal{B}(x)$ is obstacle. $H(C(q))$ is the entropy of $C(q)$ in (an unknown part of) C-space before the status of $\mathcal{B}(x)$ is determined.

$\underset{\mathcal{B}(x)\Rightarrow 0}{\Delta} H(C(q))$ is the difference of the entropy at configuration q when $\mathcal{B}(x)$ becomes free. The entropy difference is due to the fact that the probability that $q \in \mathcal{C}_{free}$ will increase because $\mathcal{A}_{unk}(q)$ will be reduced by $\mathcal{B}(x)$. Similarly, $\underset{\mathcal{B}(x)\Rightarrow 1}{\Delta} H(C(q))$ is the entropy difference when any part of $\mathcal{B}(x)$ becomes obstacle. The entropy difference is due to the fact that $q \in \mathcal{C}_{obs}$ if any part of $\mathcal{B}(x)$ becomes obstacle.

Here we assume that the situations when $\mathcal{B}(x)$ intersects the boundary of the robot $\mathcal{A}(q)$, can be ignored. We only consider the situations when $\mathcal{B}$ is completely inside or outside of $\mathcal{A}(q)$. This is a reasonable assumption since the size of $\mathcal{B}(x)$ is far smaller than the size of the robot (note in the limit $vol(\mathcal{B}(x)) \rightarrow 0$). The situations when $\mathcal{B}(x)$ are neither completely inside nor completely outside the robot are negligible as $\mathcal{B}(x) \rightarrow 0$, and their contributions to the total entropy reduction are ignored.

Recall that when $\mathcal{B}(x) \not\subset \mathcal{A}(q)$, $g_q(x)$ equals 0. We only need to consider the situations when $\mathcal{B}(x) \subset \mathcal{A}(q)$. The expected value of entropy reduction at configuration q, when $\mathcal{B}(x) \subset \mathcal{A}(q)$, is sensed is now given by

$$E\Delta H(C(q)) = p(\mathcal{B}(x)) \underset{\mathcal{B}(x)\Rightarrow 0}{\Delta} H(C(q)) + (1 - p(\mathcal{B}(x))) \underset{\mathcal{B}(x)\Rightarrow 1}{\Delta} H(C(q))), \qquad (3.10)$$

where $p(\mathcal{B}(x)) = p(\mathcal{B}(x) \subset \mathcal{P}_{free})$, the probability that $\mathcal{B}(x)$ is in $\mathcal{P}_{free}$, given all the known information (i.e. $\mathcal{P}^{(i)}_{free}$ and $\mathcal{P}^{(i)}_{obs}$) at iteration i. Strictly speaking, we are dealing with $p(\mathcal{B}(x)|\mathcal{P}^{(i)}_{free}, \mathcal{P}^{(i)}_{obs})$, where $p(\mathcal{B}(x)|\mathcal{P}^{(i)}_{free}, \mathcal{P}^{(i)}_{obs})$ is the conditional probability that $\mathcal{B}(x)$ is in $\mathcal{P}_{free}$ given $\mathcal{P}^{(i)}_{free}$ and $\mathcal{P}^{(i)}_{obs}$). However, we omit the conditions in the notation for simplicity. $1 - p(\mathcal{B}(x))$ is the probability that (at least) part of $\mathcal{B}(x)$ is obstacle. Since $\mathcal{B}(x) \subset \mathcal{A}(q)$, we have only two cases: (i) $\mathcal{B}(x)$ is free, or (ii) $\mathcal{B}(x)$ contains some obstacle region. Therefore, $\Delta H(C(q))$ will have a probability of $p(\mathcal{B}(x))$ to decrease by $\underset{\mathcal{B}(x)\Rightarrow 0}{\Delta} H(C(q))$ and a probability of $1 - p(\mathcal{B}(x))$ to decrease by $\underset{\mathcal{B}(x)\Rightarrow 1}{\Delta} H(C(q))$, resulting in Equation 3.10 for $E(\Delta H(C(q)))$.

What remains to be calculated now is $H(C(q))$. Note that $C(q)$ is a binary random variable (free/obstacle). Recall that

$$H(C(q)) = -(p(q)\log p(q) + (1 - p(q))\log(1 - p(q))), \qquad (3.11)$$

where $p(q)$ is the probability that $q \in \mathcal{C}_{free}$, given all the known information ($\mathcal{P}^{(i)}_{free}$,$\mathcal{P}^{(i)}_{obs}$). Note that, $H(C(q))$ is non-zero only at those configurations where the collision status

of the robot is unknown, since $p(q) = 0$ for an obstacle configuration, i.e. if $\mathcal{A}(q)$ intersects a known obstacle; and $p(q) = 1$ for a free configuration.

We have developed the mathematical machinery to compute $G_e(x)$, assuming that $p(\mathcal{B}(x))$ and $p(q)$ are known. These quantities, $p(\mathcal{B}(x))$ and $p(q)$ can be further derived from the distribution of obstacles, $\mathcal{P}_{obs}$, in physical space, as shown in Section 3.4.1. Using these expressions and the results from this section, we now show that $G_e(x)$ exists and can be easily calculated for the physical spaces with Poisson distribution obstacles.

3.4.1 Probabilistic Model of Physical Environment

We now discuss a simple probabilistic model of physical space – the Poisson point process, and give analytical expressions for $p(q)$ and $p(\mathcal{B}(x))$. While it is somewhat simplistic to use the Poisson model to represent the obstacles in the physical space, it gives us the insight into the C-space entropy. It also gives the ability to approximate the physical space to a certain degree. For example, the density of the Poisson point process can be used to represent the density of obstacles around the robot in the environment.

A stationary Poisson point process Φ is characterized by uniformly distributed points in space. These points can be viewed as obstacles in the physical space. The number of points in a set $\mathcal{B}$ is usually denoted by $\Phi(\mathcal{B})$. A Poisson point process has two fundamental properties [66]:

(1) Poisson distribution of point-counts. $\Phi(\mathcal{B})$ has a Poisson distribution mean, $\lambda\, vol(\mathcal{B})$ for some finite constant $\lambda > 0$. λ is usually called the intensity or density of the Poisson point process Φ.

(2) Independent scattering. The numbers of points of Φ in k disjoint sets form k independent variables, for any arbitrary k. For example, let $\mathcal{A}$ and $\mathcal{B}$ are two sets, when $k = 2$, if $\mathcal{A} \bigcap \mathcal{B} = \emptyset$, then $\Phi(\mathcal{A})$ and $\Phi(\mathcal{B})$ are independent.

From the motion planning point of view, these points are obstacles in the physical space of the robot. The density parameter of this model, λ, gives us the ability to set

the degree of "clustering" of the physical space[3]. For a more clustered environment, λ is higher.

Given any arbitrary region, a set $\mathcal{B}$, the probability that there is no point (obstacle) in set $\mathcal{B}$, i.e. $p(\mathcal{B}) = p(\mathcal{B} \in \mathcal{P}_{free}) = p(\Phi(\mathcal{B}) = 0) = e^{(-\lambda\, vol(\mathcal{B}))}$ [66]. The independent scattering property of the Poisson point process implies $p(\mathcal{B}_1|\mathcal{B}_2) = p(\mathcal{B}_1)$ if $(\mathcal{B}_1 \cap \mathcal{B}_2) = \emptyset$. Therefore, let $\mathcal{B} \subset \mathcal{P}_{unk}^{(i)}$, then $\mathcal{B} \bigcap \mathcal{P}_{free}^{(i)} = \emptyset$, $\mathcal{B} \bigcap \mathcal{P}_{obs}^{(i)} = \emptyset$,

$$p(\mathcal{B}) = p(\mathcal{B} \subset \mathcal{P}_{free}|(\mathcal{P}_{free}^{(i)}, \mathcal{P}_{obs}^{(i)})) = p(\mathcal{B} \subset \mathcal{P}_{free}) = e^{(-\lambda\, vol(\mathcal{B}))}. \tag{3.12}$$

Recall that $p(q)$ is the probability that the robot (at configuration q) lies completely inside $\mathcal{P}_{free}$ at iteration i. Obviously, $\mathcal{A}_{unk}(q) \bigcap \mathcal{P}_{free}^{(i)} = \emptyset$, $\mathcal{A}_{unk}(q) \bigcap \mathcal{P}_{obs}^{(i)} = \emptyset$, and therefore

$$p(q) = p(\mathcal{A}_{unk}(q) \subset \mathcal{P}_{free}|(\mathcal{P}_{free}^{(i)}, \mathcal{P}_{obs}^{(i)})) = p(\mathcal{A}_{unk}(q) \subset \mathcal{P}_{free}) = e^{(-\lambda\, vol(\mathcal{A}_{unk}(q)))}. \tag{3.13}$$

3.4.2 Information Gain Density for Point Poisson Process Model

Let us discuss $g_q(x) = \lim\limits_{vol(\mathcal{B}(x))\to 0} \dfrac{-E\Delta H(C(q))}{vol(\mathcal{B}(x))}$ under the assumption that the obstacles have a Poisson point distribution.

Using Equation 3.10, we get

$$g_q(x) = -\lim_{vol(\mathcal{B}(x))\to 0} \frac{p(\mathcal{B}(x)) \underset{\mathcal{B}(x)\Rightarrow 0}{\Delta} H(C(q)) + (1 - p(\mathcal{B}(x)) \underset{\mathcal{B}(x)\Rightarrow 1}{\Delta} H(C(q))}{vol(\mathcal{B}(x))}.$$

Using Equation 3.12, we get

$$\lim_{vol(\mathcal{B}(x))\to 0} \frac{1 - p(\mathcal{B}(x))}{vol(\mathcal{B}(x))} = \lambda \quad \text{and} \quad \lim_{vol(\mathcal{B}(x))\to 0} p(\mathcal{B}(x)) = e^0 = 1.$$

[3]We also are investigating a more realistic model called Boolean Model [66]. However, computing the probabilities under this model is rather complicated and much more computationally intensive. In addition, only approximate analytical expressions seem obtainable.

Applying the above two expressions to $g_q(x)$, we get

$$g_q(x) = -\lim_{vol(\mathcal{B}(x))\to 0}(\frac{\underset{\mathcal{B}(x)\Rightarrow 0}{\Delta} H(C(q))}{vol(\mathcal{B}(x))} + \lambda \underset{\mathcal{B}(x)\Rightarrow 1}{\Delta} H(C(q))).$$

Given $\mathcal{B}(x) \subset \mathcal{A}(q)$ and $\mathcal{B}(x) \subset \mathcal{C}_{free}$, after $\mathcal{B}(x)$ is being sensed, $p(q)$ is increased by a certain value because $\mathcal{A}(q)$ has an additional portion $\mathcal{B}(x)$ changed to free space. New $p(q)$ (after scanning) is a function of $vol(\mathcal{B}(x))$. Let $\underset{\mathcal{B}(x)\Rightarrow 0}{\Delta}(p(q))$ be the change in $p(q)$ due to $\mathcal{B}(x)$ changing from unknown to free. $g_q(x)$ can be rewritten as

$$\begin{aligned} g_q(x) &= -\lim_{vol(\mathcal{B}(x))\to 0}(\frac{\underset{\mathcal{B}(x)\Rightarrow 0}{\Delta} H(C(q))}{\underset{\mathcal{B}(x)\Rightarrow 0}{\Delta}(p(q))}\frac{\underset{\mathcal{B}(x)\Rightarrow 0}{\Delta}(p(q))}{vol(\mathcal{B}(x))} + \lambda \underset{\mathcal{B}(x)\Rightarrow 1}{\Delta} H(C(q))) \\ &= -\lim_{vol(\mathcal{B}(x))\to 0}\frac{\underset{\mathcal{B}(x)\Rightarrow 0}{\Delta} H(C(q))}{\underset{\mathcal{B}(x)\Rightarrow 0}{\Delta}(p(q))}\lim_{vol(\mathcal{B}(x))\to 0}\frac{\underset{\mathcal{B}(x)\Rightarrow 0}{\Delta}(p(q))}{vol(\mathcal{B}(x))} - \lambda \underset{\mathcal{B}(x)\Rightarrow 1}{\Delta} H(C(q)). \end{aligned}$$

When $\mathcal{B}(x)$ is free, we get the following equation from Equation 3.13,

$$\lim_{vol(\mathcal{B}(x))\to 0}(\frac{\underset{\mathcal{B}(x)\Rightarrow 0}{\Delta}(p(q))}{vol(\mathcal{B}(x))}) = \lambda.$$

This is essentially the derivative of $p(q)$ with respect to the volume of $\mathcal{B}(x)$.

From Equation 3.8, we get

$$\lim_{vol(\mathcal{B}(x))\to 0}(\frac{\underset{\mathcal{B}(x)\Rightarrow 0}{\Delta} H(C(q))}{\underset{\mathcal{B}(x)\Rightarrow 0}{\Delta}(p(q))}) = -\log(\frac{p(q)}{1-p(q)}).$$

When $\mathcal{B}(x)$ contains an obstacle, q will be a part of C-obstacle. $p(q)$, the probability of q in free C-space, becomes 0. Therefore, $H(C(q))$ becomes 0. We get the following equation from Equation 3.9

$$\underset{\mathcal{B}(x)\Rightarrow 1}{\Delta} H(C(q)) = -H(C(q)).$$

Again, we plug the above three equations into $g_q(x)$. Then,

$$g_q(x) = \lambda(\log(\frac{p(q)}{1-p(q)}) + H(C(q))).$$

We can consider that the above equation consists of two parts. The first part, $\lambda \log(\frac{p(q)}{1-p(q)})$, is the contribution of the entropy reduction when $\mathcal{B}(x)$ changes to free, and $\lambda H(C(q))$ is the part of the contribution when $\mathcal{B}(x)$ changes to obstacle.

Substituting for $H(C(q))$ from Equation 3.4, we get

$$g_q(x) = \lambda(\log \frac{p(q)}{1-p(q)} - p(q)\log p(q) - (1-p(q))\log(1-p(q))). \tag{3.14}$$

Recall that $G_e(x) = \sum_{q \in \mathcal{X}(x)} g_q(x)$. Then,

$$G_e(x) = \sum_{q \in \mathcal{X}(x)} \lambda(\log \frac{p(q)}{1-p(q)} - p(q)\log p(q) - (1-p(q))\log(1-p(q))). \tag{3.15}$$

This expression, combined with the $p(q)$ expression in Equation 3.13 completely determines the IGDF for a physical space with a Poisson point distribution. The point to be scanned next is the one that maximizes $G_e(x)$.

3.4.3 MER Criterion: A Deterministic Geometric Interpretation

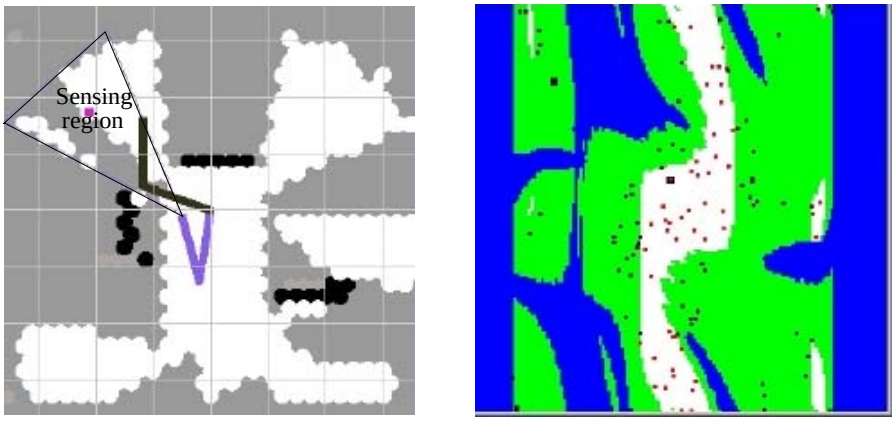
The MER criterion developed so far assumes a stochastic model of the environment. However, it also has an elegant deterministic interpretation. Note that the expression for IGDF $= \sum_{q \in \mathcal{X}(x)} g_q(x)$ is a weighted summation over C-zone of x, where the weight ($g_q(x)$) is essentially the contribution of configuration q to the total entropy reduction. Assuming we ignore the probabilistic model, i.e. assume that each configuration contributes the same constant value D_e. In other word, assume $g_q(x) = D_e$. Then, the IGDF is directly proportional to the sizes (or volumes) of the C-zones of each point in the physical space. *Therefore, the next view will be at a place in unknown physical space that has a maximal volume of C-zone.*

Recall that in Kruse's algorithm (see Section 1.5.2), criterion 3 in choosing a view is to maximize the volume of unknown physical space. This criterion treats every point in physical space having the same priority to be scanned. This criterion is obviously incorrect when the effect of the C-zone volume is taken into account. Those

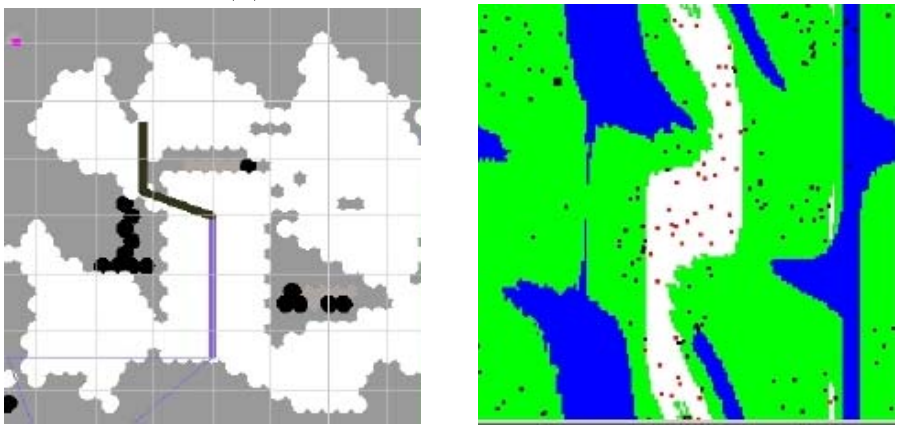
places, which affect large set of C-space, i.e. having large C-zone, should have a higher priority to be sensed. On the other hand, the places which are rarely occupied by the robot should be given less priority to be sensed. In an extreme case, some regions are not occupiable by the robot (see a later sub-section in Section 3.4.4). The unoccupiable regions need not to be sensed at least from pure motion planning point of view[4]. The volume of C-zones should be an important factor when the C-space is explored even without assuming any probabilistic model of physical space.

Figure 3.2 shows a comparison of the results of physical space and C-space after 14 scans with three different view planning strategies: (1) the next view is randomly chosen, i.e. the position and direction in each view are randomly selected; (2) the next view is chosen such that it maximizes the unknown physical volume in the scan (criterion 3 in Kruse's algorithm); (3) the next view is chosen based on MER criterion. The robot in this simulation is a two-link planar robot. The sensor field of view is the triangular region shown in Figure 3.2a. In both physical space and C-space shown in the figure, the dark regions are obstacles; the gray regions are unknown; the white regions are free. Figure 3.2a is the result with random views. Figure 3.2b is the result using maximizing unknown physical space volume criterion. Figure 3.2c is the result of MER criterion. After 14 scans, the percentages of known C-spaces have increased to 47% (random), 33% (maximizing unknown physical space volume) and 70% (MER) respectively from an initial known C-space which is 14% of the entire C-space. It is interesting to point out that even the random view strategy is better than maximizing unknown physical space volume criterion. The latter tries to scan the unknown physical space as much as possible. Therefore, the overlaps among the viewing regions are minimized. Known physical space tends to be scattered around the workspace after a number of scans. On the other hand, only lumped known regions can form larger sets of known C-space. Therefore, maximizing unknown physical space volume criterion will in general generate less efficient view planning strategy from C-space exploration point of view.

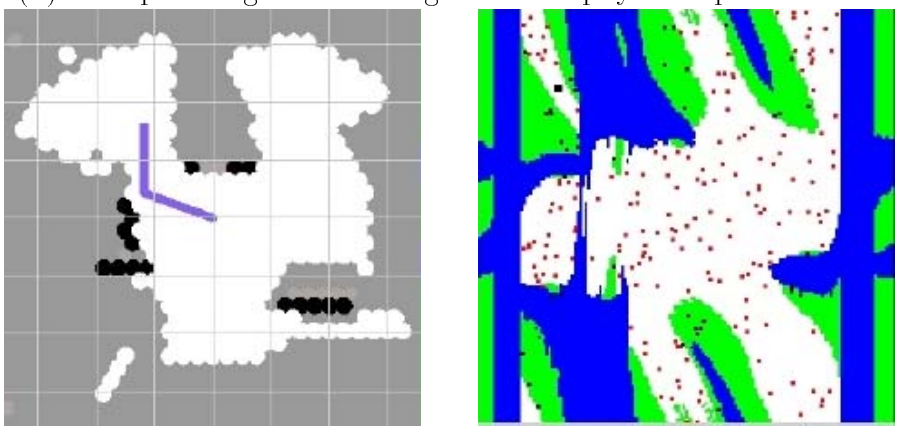
[4]Such points may be of interest for other tasks, such as inspection.



(a) view planning: random views



(b) view planning: maximizing unknown physical space volume



(c) view planning: MER criterion

Figure 3.2: A comparison of explored C-space and physical space for different view planning strategies. The left side shows the physical space and the right side shows the C-space.

3.4.4 Discussion on Properties of IGDF

Now let's examine some properties of the information gain density function, $G_e(x)$ assuming that the physical space is a Poisson point process. We discuss the properties of $G_e(x)$ qualitatively in this section. $G_e(x)$ is affected by configuration q in three ways as described below.

(1) Dependency on C-zone $\mathcal{X}(x)$ volume

$G_e(x)$ is larger when the volume of the C-zone of x is larger.

As mentioned in Section 3.4.3, information gain density $G_e(x)$ is the summation of $g_q(x)$ over its C-zone for any x in unknown physical space, $\mathcal{X}(x)$. Those regions in the physical space that have a larger C-zone would have a higher value of $G_e(x)$.

This property can also be seen in the example in Section 3.2, where point B intersects both $\mathcal{A}(q_1)$ and $\mathcal{A}(q_2)$, and therefore point B causes C-space entropy to be reduced faster than the other points.

(2) Dependency on the part of the robot in unknown physical space – $\mathcal{A}_{unk}(q)$

If for a configuration q, $\mathcal{A}(q)$ has a larger portion in free space, $G_e(x)$ is higher for $x \in \mathcal{A}_{unk}(q)$.

$g_q(x)$ increases monotonically with the probability $p(q)$ (see Equation 3.14). Figure 3.3 shows the relationship between $g_q(x)$ and $p(q)$. Note that with the Poisson model, the probability $p(q)$ is a monotonically increasing function of the volume of $\mathcal{A}(q)$ in known free space (see Equation 3.13). Therefore, $g_q(x)$ is a monotonic function of the fraction of $\mathcal{A}(q)$ in known free space.

The above feature implies that if $\mathcal{A}(q)$ has a large part of the total volume is in free physical space, then $\mathcal{A}_{unk}(q)$ is more likely to be scanned in the next step. For this type of configurations, the sensor needs less effort to determine the status of them. For example, in Figure 3.4, configuration q_2 may need more than two scans to fully determine the status (obstacle or free) and configuration q_1 needs only one. $\mathcal{A}(q_1)$ has

a higher scanning priority than $\mathcal{A}(q_2)$ does. The benefit of scanning $\mathcal{A}_{unk}(q_1)$ is that the known part of C-space tends to expand faster after this scan.

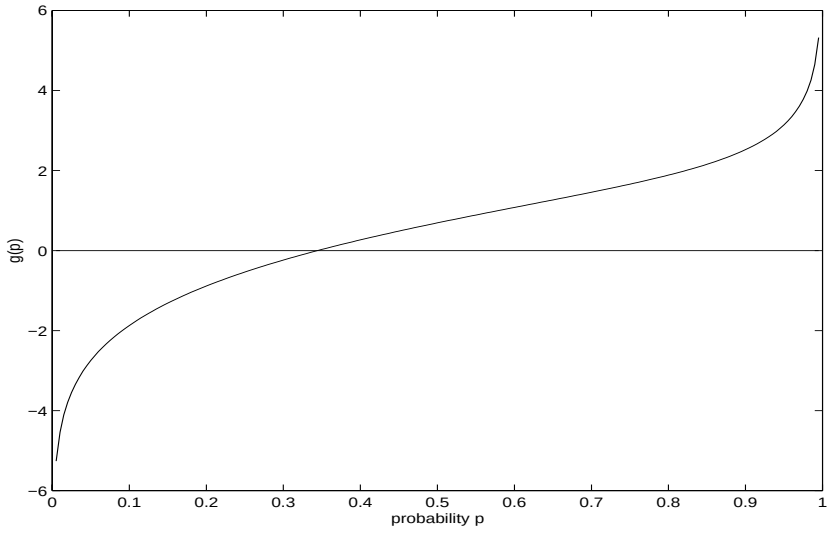


Figure 3.3: Illustration of the contribution of a configuration towards the entire entropy reduction, $g_q(x)$, for the Poisson model. $g_q(x)$ increases as $p(q)$, the probability that configuration q is in $\mathcal{C}_{free}$, increases.

(3) Unoccupiable Region

Some regions in the physical space will never intersect any part of the robot. It does not matter if these regions are free or have obstacles from a MP point of view. We call these regions *unoccupiable regions*. Unoccupiable regions are not of concern from a pure robot motion planning point of view. They do not contribute to the reduction of C-space entropy, and are therefore excluded from being scanned automatically with MER.

There are two types of unoccupiable regions: Type 1, the regions that can not be intersected by the robot because of the robot's kinematic limits. These regions are represented by $\mathcal{U}_1$; Type 2, the regions that can not be intersected by the robot due to obstacles in physical space. These regions are denoted by $\mathcal{U}_2$. Formally, $\mathcal{U}_2$ is characterized by the following implicit constraint: $\forall q$, if $\mathcal{A}(q) \bigcap \mathcal{U}_2 \neq \emptyset$, then, $q \in \mathcal{C}_{obs}$ (See Figure 3.5).

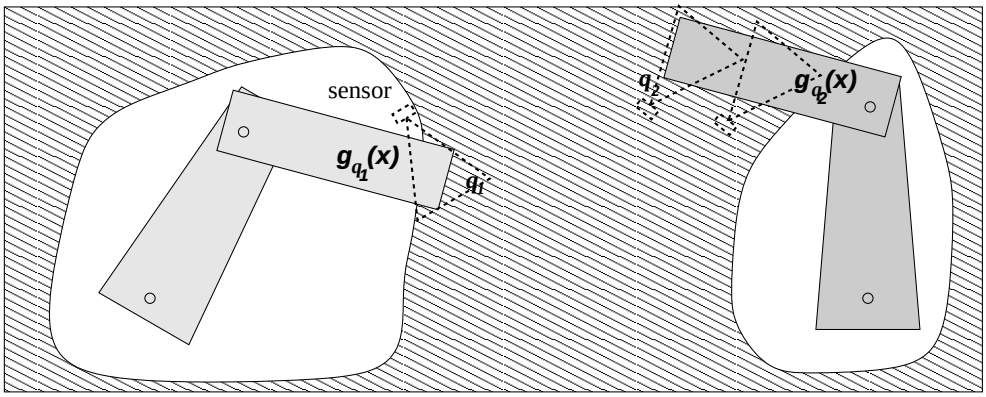


Figure 3.4: Scanning priorities for two configurations. $\mathcal{A}(q_1)$ (the left sub-figure) is more likely to be scanned than $\mathcal{A}(q_2)$ (the right sub-figure) because scanning as in the left sub-figure will determine the status of configuration q_1 immediately after this scan.

Clearly, for the first type unoccupiable regions, there is not any robot configuration that will contribute to information gain if those regions are scanned, i.e. $G_e(x) = 0, x \in \mathcal{U}_1$. For the second type, $g_q(x) = 0$, $x \in \mathcal{U}_2$. This is because, by definition, all the robot configurations intersecting x have $p(q) = 0$, since the robot intersects some obstacles. Hence, $G_e(x) = 0$.

As more obstacles are found, $\mathcal{U}_2$ can also increase. $\mathcal{U}_1$ is determined by robot kinematics only, and therefore is static during the process of planning.

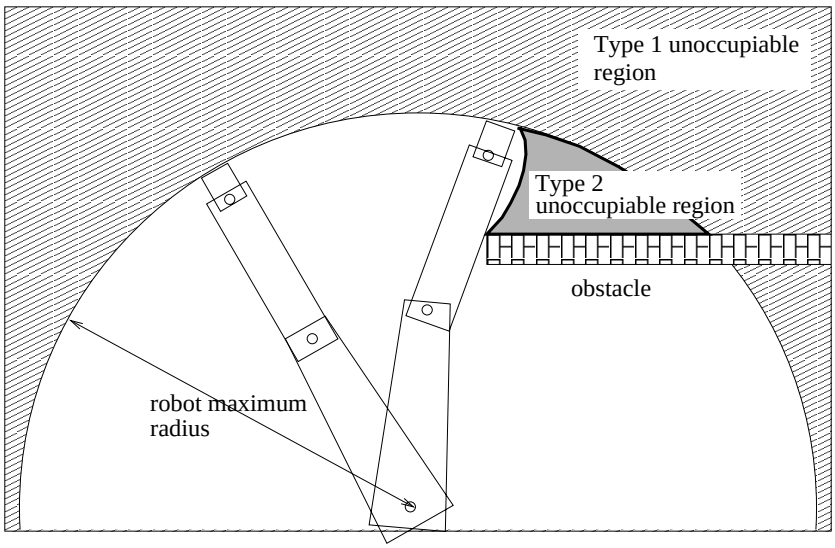


Figure 3.5: Two types of unoccupiable regions. These regions will not contribute to entropy reduction. Type 1 unoccupiable regions are beyond the scope of the robot workspace. Type 2 unoccupiable regions are formed due to obstacles.

3.4.5 Illustration of IGDF Properties

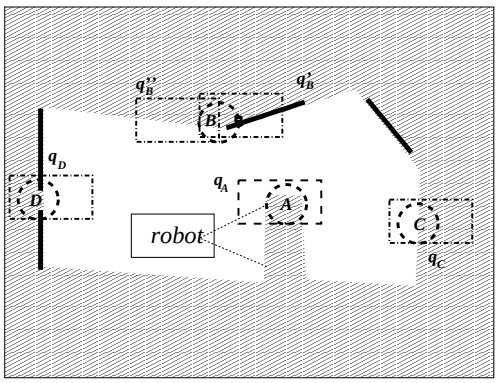


Figure 3.6: Illustration of $G_e(x)$ with a single link, translation robot. The black regions are obstacles found by the robot. $G_e(x)$ at Region A has the maximum value and zero value at Region D.

A simpler example of view planning for a single link, translational robot is given in Figure 3.6. This figure illustrates the properties of the view planning strategy based on MER. Assume that the robot is a rectangle with translational movement only. There is a range sensor mounted on the robot. Figure 3.6 shows a certain stage of the planning.

We analyze four possible scanning regions which are represented as dash line circles marked A, B, C and D. Dotted rectangles represent the robot at the different configurations. The white region has already been found to be free by the sensor. The black regions are known obstacles for the robot. The shaded region is still unknown at this stage.

Let us now consider the four candidates for sensing. (1) In region B, $G_e(x)$ value is less, because x is less frequently occupied by the robot without colliding with known obstacles. Many of the configurations, for example q'_B, will result in robot collisions with obstacles. q'_B will not contribute to entropy reduction. Only those configurations with non-zero probabilities in $\mathcal{C}_{free}$ contribute to $G_e(x)$ at region B, such as q''_B. (2) When the robot occupies region C, $\mathcal{A}_{unk}(q_c)$ is larger, and hence $p(q)$ is less. $G_e(x)$ value in region C tends to be smaller. (3) Region D is an unoccupiable region, here $G_e(x) = 0$. C-space entropy is not affected by the unknown region in D. Intuitively,

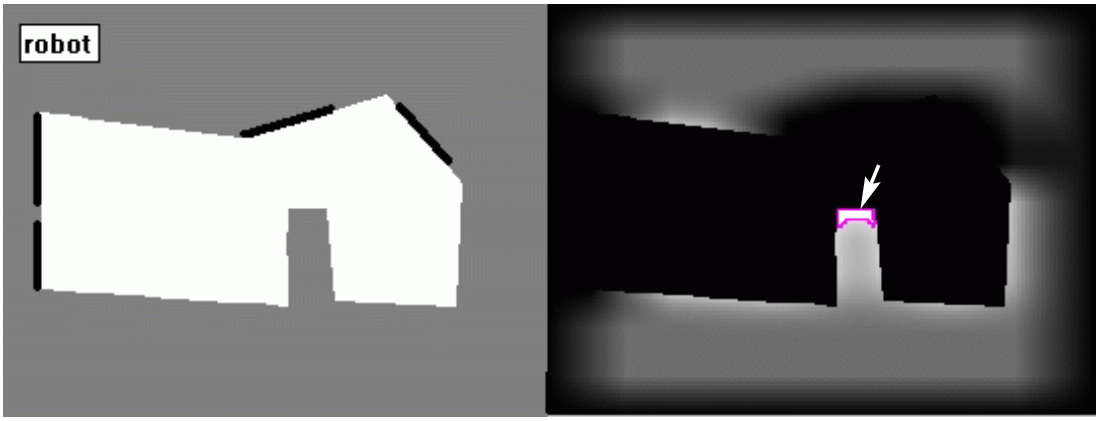


Figure 3.7: This figure (the right part) shows the IGDF calculated from the previous example. The physical space is also shown on the right of the figure for comparison. The lighter region has higher information gain density. The IGDF in the oblong white region (same position as region A in Figure 3.6) is greater than 90% of $\max G_e(x)$.

region D becoming free or presenting obstacles will not change robot's ability to move around. (4) Region A will have a C-zone volume comparable to that of region C. However, when the robot occupies region A, a larger portion of the robot is in $\mathcal{P}_{free}$. Hence, $p(q)$ is high, which implies that $g_q(x)$ is higher. $G_e(x)$ at region A, therefore, would have the maximum value among all the regions.

A simulation result of view planning for this one link free-flying robot is shown in Figure 3.7. The left side of the figure shows the physical space. The right side of the figure shows $G_e(x)$. The lighter region has higher $G_e(x)$. Indeed region A has the highest IGDF.

3.5 Computational and Implementation Issues

Equation 3.15 tells us that the IGDF, $G_e(x) = \sum_{q \in \mathcal{X}(x)} g_q(x)$ can be calculated as the summation of $g_q(x)$ when $q \in \mathcal{X}(x)$. The computational cost of finding the C-zone $\mathcal{X}(x)$ is high. One way out would be to approximate the summation of $g_q(x)$ over randomly selected configurations $q \in \mathcal{C}_{unk}$ when $x \in \mathcal{A}(q)$, i.e. $G_e(x) = \sum_{\substack{randomly \\ selected\ q \in \mathcal{C}_{unk}}} g_q(x)\delta(\mathcal{A}_{unk}(q))$. In this case, $G_e(x)$ is normalized (divided by the density of the

randomly selected configurations). This normalization is necessary otherwise $G_e(x)$ would be sensitive to the number of random selected configurations.

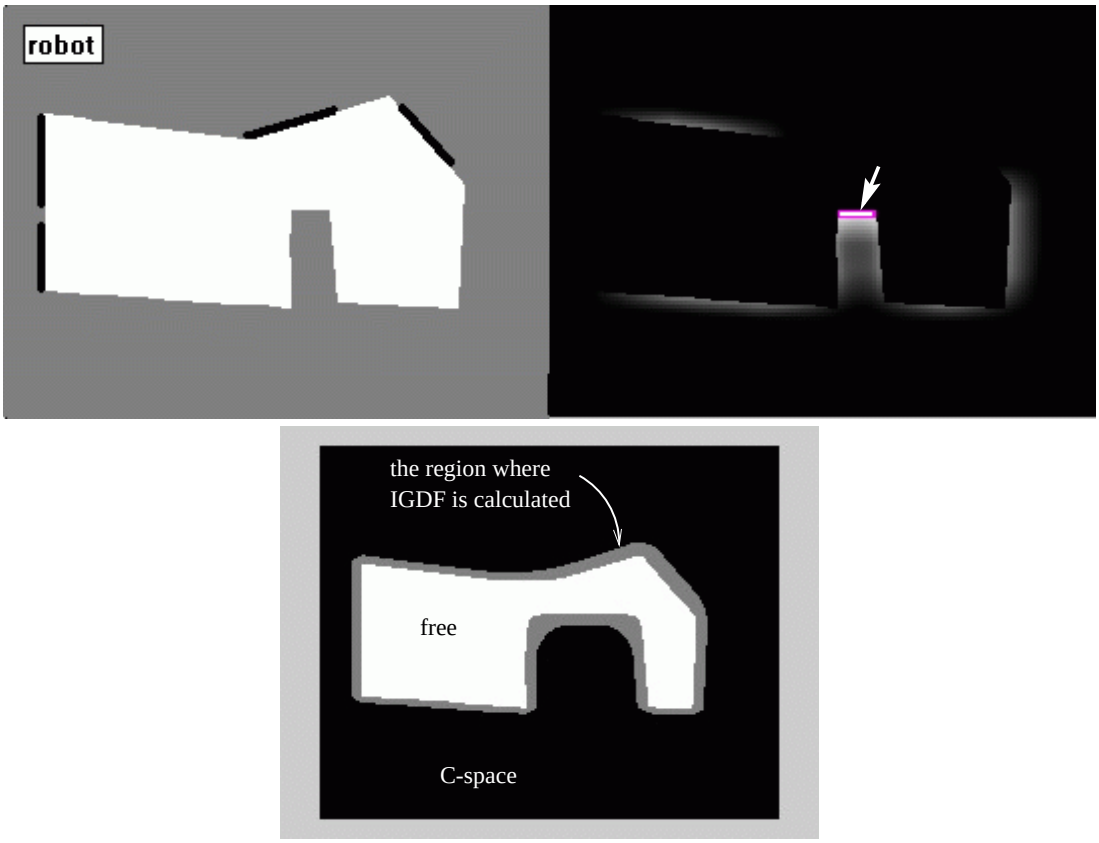


Figure 3.8: The IGDF calculated from random selected configurations within a much thin layer of region (portion in $\mathcal{P}_{free}$ $m > 0.85$), as shown in the bottom figure. Similar to Figure 3.7, $G_e(x)$ in the oblong white region is greater than 90% of $\max G_e(x)$. This region is very similar to the one shown in Figure 3.7, which is calculated in the entire $\mathcal{C}_{unk}$.

An important point for computational efficiency is that most of the contribution to IGDF comes from the configurations in a layer that lies close to the boundary interface between $\mathcal{C}^{(i)}_{free}$ and $\mathcal{C}^{(i)}_{unk}$. In fact, x_{max}, the position where $G_e(x)$ reaches maximum value, is not susceptible, to a certain extent, to the thickness (in a certain measure) of this layer. This is because a configuration, which is distant from $\mathcal{C}_{free}$, would have less fraction of $\mathcal{A}(q)$ in $\mathcal{P}_{free}$. Hence, $g_q(x)$ is less, as was discussed in

Section 3.4.4. IGDF and x_{max} (position in physical space where IDGF reaches its maximal value) are less affected by these configurations. Therefore, we need only a relatively thin layer of unknown configurations to determine x_{max}. The thickness of this layer can be represented by the fraction of $\mathcal{A}(q)$ is in $\mathcal{P}_{free}$ (see Section 4.1.1 for a precise definition). Figure 3.8 shows $G_e(x)$ computed from random selected configurations within a much thinner layer of randomly selected configurations in $\mathcal{C}_{unk}^{(i)}$ (portion in $\mathcal{P}_{free}$ > 0.85). Compared with Figure 3.7, the positions of the regions where $G_e(x) > 0.90 * \max G_e(x)$ in the two figures are very close (an oblong region at the center of the right side figure). These randomly selected configurations in $\mathcal{C}_{unk}^{(i)}$ correspond to gray landmarks in our algorithm. Computing IGDF is a key motivation of putting gray landmarks on the boundary of $\mathcal{C}_{free}$, to be introduced in Chapter 4.

When the real size of the sensing region, $\mathcal{V}_{\mathcal{S}}$, can not be viewed as a very small region, IGDF has a large error in approximating the entropy reduction in each scan. This fundamental problem needs further investigation. Here we propose a simple solution which is used in our implementation. Since the calculation of $G_e(x)$ assumes a very small sensing region $\mathcal{B}(x)$, x_{max} is always on the boundary of $\mathcal{P}_{unk}$ and $\mathcal{P}_{free}$ (see Figures 3.7 and 3.8). A practical approach to solve this problem is to move x_{max} away from $\mathcal{P}_{free}$ by a certain distance, for example, by 1/2 of the radius of $\mathcal{V}_{\mathcal{S}}$.

3.6 Maximization of $E\Delta H()$ Over Multiple Scans

In order to scan point x_{max} (the point with the highest information density), the robot has to move and scan from a certain configuration. However, occasionally it is not possible to move to a desired place from where the sensor scans, either because of (known) obstacles or unknown regions in the physical space.

If this movement is blocked by obstacles, then scanning the second best place is the only choice. However, when the movement is prevented by unknown regions (see Figure 3.9 a), there are two schemes. The first scheme is to let the sensor scan the second best point according to MER (assume this point is scannable, otherwise choose the next best scannable region). Let us call this second best point $x'_{max} = \{x :$

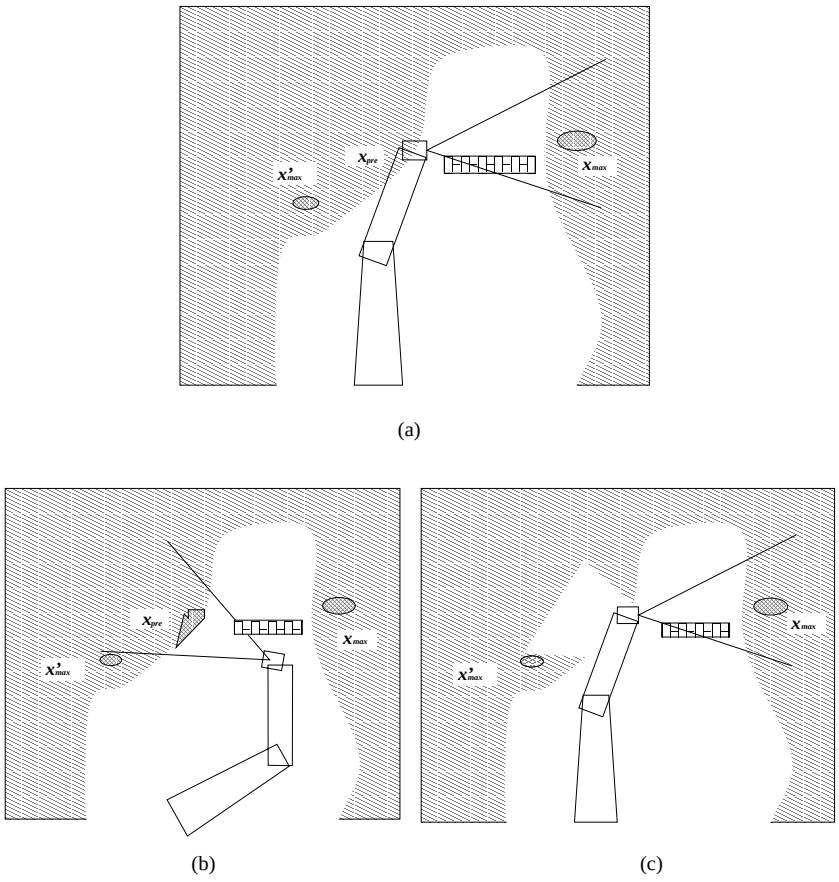


Figure 3.9: (a) shows an example of a situation when the place to be scanned, x_{max}, is not scannable because of an unknown region x_{pre} intersecting the robot in the configuration from where it takes the scan. (b) and (c) show a possible scheme – scan the unknown region x_{pre} first, and then scan x_{max} if x_{pre} is free.

second $\max_{x \in \mathcal{P}}(G_e(x))\}$. Or, the second scheme is to first scan x_{pre}, and then scan x_{max}. Which scheme to choose can be elegantly resolved based on MER over two or multiple iterations.

Let $G_m = G_e(x_{max})$ and $G'_m = G_e(x'_{max})$. Let $\mathcal{B}(x_{pre})$ be the unknown region which prevents the sensor from scanning x_{max}, and let x_{pre} represent the "center" of this region. If scheme (2) is chosen, the expected entropy reduction over two iterations can be written as follows: $E(\Delta^2 H) = G_e(x_{pre}) + p(\mathcal{B}(x_{pre})) \cdot G_m + (1 - p(\mathcal{B}(x_{pre})) \cdot G'_m$. We assume that $\mathcal{B}(x_{pre})$ can be covered by one scan. The first term in the above equation is the expected entropy reduction from the first scan. The outcome of the first scan can be either free or obstacle for $\mathcal{B}(x_{pre})$. The second term is the entropy reduction when $\mathcal{B}(x_{pre})$ is free and then x_{max} is scanned. The third term is the entropy reduction when $\mathcal{B}(x_{pre})$ is found to be an obstacle and the second best position x'_{max} is scanned. Because the entropy reduction is obtained over two iterations, the average entropy reduction is $(E(\Delta^2 H))/2$. Figure 3.9 (b) and (c) show the two steps, first scanning x_{pre} and then scanning x_{max}. $E(\Delta^2 H)/2$ is then compared with G'_m. If the former is larger, we follow the second scheme because more entropy will be reduced on average over the two scans than over one scan at x'_{max} (assume that every iteration has the same cost and the cost of determining above alternatives is negligible). Otherwise, the first scheme will be adopted. Although here we only give the schemes of MER over two iterations, this procedure can be generalized to n scans.

This, then completes the algorithm description for computing the region for the next scan based on MER (see Appendix B.10 for psudo-code of the process). In this chapter, we introduced the following key ideas: we view the sensing action as reducing ignorance of the C-space. The C-space entropy is introduced to measure the effectiveness of a scan. Maximal entropy reduction (MER) is then achieved by sensing the position maximizing IGDF.

Chapter 4

Sensor-Based Incremental Construction of Probabilistic Roadmap (PRM)

Having solved the view planning problem, i.e. where to take the next scan, we now integrate it into our algorithm SBIC-PRM. In this chapter, we first present the notation of SBIC-PRM in Section 4.1, and then explain the algorithm in Section 4.2. Section 4.3 and 4.4 give details of the two major components of SBIC-PRM, view planning and roadmap expansion.

4.1 Definitions and Notation for SBIC-PRM

4.1.1 Landmarks and Roadmap

Let l denote a landmark – some of these landmarks are nodes in our roadmap (as will become clear later); q_l denotes the configuration corresponding to a landmark l. $\mathcal{L}$ denotes the set of all the landmarks. We distinguish among three types of landmarks. See Figure 4.1 for the robot poses at configurations corresponding to different types of landmarks.

White landmarks, similar to the nodes in [36] belong to $\mathcal{C}_{free}^{(i)}$, i.e. the robot in this configuration lies completely in $\mathcal{P}_{free}^{(i)}$. *Black* landmarks correspond to the robot colliding with a *known* obstacle, $\mathcal{P}_{obs}^{(i)}$, at the corresponding configuration. *Gray* landmarks correspond to part of the robot in $\mathcal{P}_{unk}^{(i)}$, i.e. at the configuration corresponding to a gray landmark, the robot does not intersect $\mathcal{P}_{obs}^{(i)}$, and does not lie completely in $\mathcal{P}_{free}^{(i)}$ either (see Figure 4.2). $\mathcal{L}_w$ denotes the set of white landmarks, $\mathcal{L}_b$, the set of black landmarks, and $\mathcal{L}_g$, the set of gray landmarks. $\mathcal{L} = \mathcal{L}_w \bigcup \mathcal{L}_b \bigcup \mathcal{L}_g$.

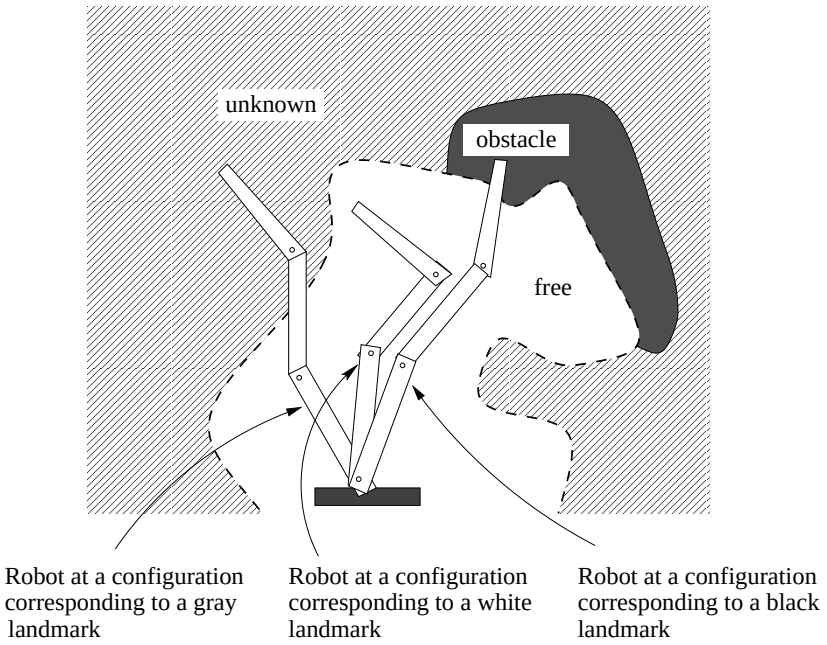


Figure 4.1: A schematic example of gray, white and black landmarks in physical space.

Recall that the C-space entropy is calculated over randomly selected configurations in unknown C-space, and that most of the contribution comes from the configurations lying in $\mathcal{C}_{unk}$ close to the boundary between $\mathcal{C}_{unk}$ and $\mathcal{C}_{free}$. This corresponds to configurations at which a large fraction of the robot is in free physical space (see Section 3.5). Intuitively, we can imagine that this region forms the adjacent region around $\mathcal{C}_{free}$. Let $\mathcal{C}_{free+}^{(i)}(m)$ denote this region in C-space. Formally $\mathcal{C}_{free+}^{(i)}(m) = \{q \in \mathcal{C} : \text{the fraction of } \mathcal{A}(q) \text{ in } \mathcal{P}_{free} > m\}$. When the robot takes a configuration in $\mathcal{C}_{free+}^{(i)}(m)$, the robot has a fraction, greater than m, of the volume of $\mathcal{A}(q)$ that lies within free physical space, $\mathcal{P}_{free}^{(i)}$. $\mathcal{C}_{free+}^{(i)}(m)$ is the region where (all colors of)

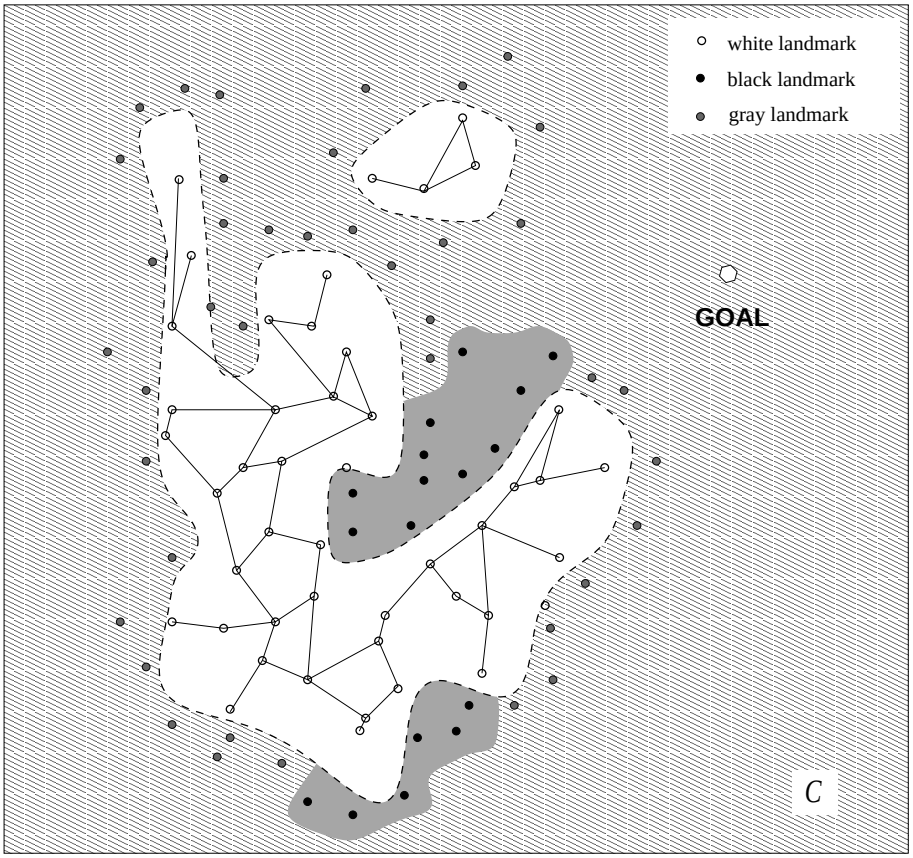


Figure 4.2: Three types of landmarks (all denoted by ○ with different colors) are schematically shown above in the configuration space. The white region is known $\mathcal{C}_{free}$ and the landmarks in this region are white landmarks ($\mathcal{L}_w$). The *Roadmap* is the graph shown in this region. The dark region indicates known $\mathcal{C}_{obs}$ and the landmarks in this region are black landmarks ($\mathcal{L}_b$). The gray region is the unknown configuration space and the landmarks in this region are gray landmarks ($\mathcal{L}_g$). The gray and black landmarks form a "crust" enclosing $\mathcal{C}_{free}$.

landmarks are placed at a given iteration. Gray landmarks, $\mathcal{L}_g \subset \mathcal{C}^{(i)}_{free+}(m) - \mathcal{C}^{(i)}_{free}$.

$\mathcal{C}^{(i)}_{free+}(m)$ is $\mathcal{C}^{(i)}_{free}$ enlarged by a layer. Parameter m controls the thickness of the enlarged layer of $\mathcal{C}^{(i)}_{free}$. The smaller it is, the "thicker" is the envelope of gray landmarks around the current boundary of $\mathcal{C}^{(i)}_{free}$ [1]. See Figure 4.3 for $\mathcal{C}^{(i)}_{free+}(m)$ with different m's.

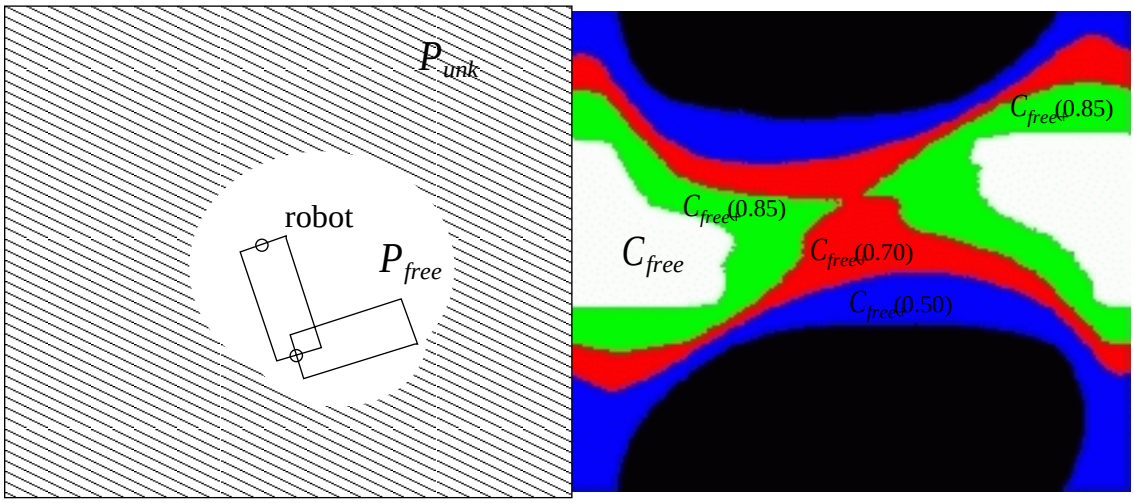


Figure 4.3: $\mathcal{C}_{free+}(m)$ with $m = 0.85, 0.70$ and 0.50 for a two-link robot. The left figure shows the robot and its physical space. The right figure shows the C-space.

$\mathcal{C}^{(i)}_{free}$ is characterized by a *roadmap* $\mathcal{R}$ (although $\mathcal{R}$ changes in each iteration, we omit the superscript (i) on $\mathcal{R}$ because we are only concerned with the roadmap at current iteration). The *roadmap* $\mathcal{R}$ is defined as an undirected graph with its nodes as the white landmarks i.e. $\mathcal{R} = \{\mathcal{L}_w, \mathcal{E}\}$ where $\mathcal{L}_w$ is the set of nodes of the graph, i.e. the landmarks, , $\mathcal{E}$ is the set of the edges of the graph. Two landmarks, l_1 and l_2, in the roadmap are connected by an edge in $\mathcal{E}$ if a *local_planner*(q_{l_1}, q_{l_2}) returns a simple collision-free path from q_{l_1} to q_{l_2}. The *local_planner*() simply tries to execute a discretized straight line in configuration space. It checks for collision detection at

[1] An alternative definition for $\mathcal{C}_{free+}$ would be to first enlarge $\mathcal{P}_{free}$ isotropically by a layer enclosing it. Call this enlarged space $\mathcal{P}_{free+}$. If, for a configuration q, $\mathcal{A}(q) \subset \mathcal{P}_{free+}$, then $q \in \mathcal{C}_{free+}$. However, since we also have to determine the probability of a gray landmark being in $\mathcal{C}_{free}$, and this computation needs to know the volume of the robot in $\mathcal{P}_{free}$, the first definition of $\mathcal{C}_{free+}$ is computationally more attractive.

each discretized point. If any of the points along the straight line results in collision, the *local_planner*() returns failure.

Note that roadmap $\mathcal{R}$ may not be a connected graph because $\mathcal{C}_{free}^{(i)}$ may not be connected. We use $\mathcal{R}(l_0)$ to represent the connected component of $\mathcal{R}$ that contains l_0, the landmark corresponding to the start configuration q_0. Clearly, robot motion can be carried out only in $\mathcal{R}(l_0)$ at a given iteration.

4.1.2 View Node

The view planning approach presented in Chapter 3 finds $x_{max} \in \mathcal{P}$, the center of the next sensing region. However, the planner also has to find a sensor configuration $q_{\mathcal{S}}$ (equivalent to q in our system), from which x_{max} can be scanned.

In SBIC-PRM, nodes, or robot configurations, from which the next sensing action takes place, are called view nodes. A view node is denoted by v with the corresponding robot configuration, denoted by q_v in $\mathcal{C}_{free}$. Obviously, v must be connected to $\mathcal{R}(l_0)$, since the robot needs to reach it in order to scan.

As explained in Section 2.1.2, a natural way to find a view node is to search in the sensor configuration space $\mathcal{C}_{\mathcal{S}}$ (equivalent to $\mathcal{C}$ in our system). However, $\mathcal{C}_{\mathcal{S}}$ is huge (6-d in our system). In SBIC-PRM, (for a detailed explanation see Section 4.2.2), we search the landmarks of the current roadmap (a set of random selected configurations in C-space) to find the next view node for computational efficiency (in stead of searching the entire sensor configuration space). From each white landmark, we let the external sensing *dofs* (the base and elbow) to be fixed and search the space spanned by the wrist *dofs* to find the best joint angle values to view a given position in the physical space. We limit the search to $q_l \times \mathcal{C}_{\tilde{\mathcal{S}}_{int}} \times \mathcal{C}_{\mathcal{S}_{int}}$, i.e. the search space is spanned by the internal and quasi internal sensing degrees (the wrist *dofs* in our system) at each landmark. In other word, $n_{\mathcal{S}_{ext}}$ - $n_{\tilde{\mathcal{S}}_{int}}$ dofs (the two base and one elbow degrees of freedom in our system) remain the value of the white landmarks, while the internal (and quasi-internal) *dofs*, $n_{\mathcal{S}_{int}}$ and $n_{\tilde{\mathcal{S}}_{int}}$ (the three wrist degrees of freedom), are allowed to be changed to find the "best" viewing position. The corresponding view node is denoted by v_e.

We use $v_e(x)$ to represent a view node with the sensor directed toward a point $x \in \mathcal{P}$. Note that $q_{v_e}(x)$ may not exist for some pair of e and x because of kinematic and geometric limits of the robot and sensor.

4.2 Algorithm

We now state the pseudo-code for SBIC-PRM. The pseudo-code (as is our implementation) is stated in an object-oriented manner. There are four major objects in the algorithm: (i) the *PhysicalSpace* object contains all the information about $\mathcal{P}$; (ii) the *Roadmap* object keeps the information about landmarks and their connectivity; (iii) *ViewPlanner* object plans the next view, and (iv) the *Robot* object keeps information about the robot and the sensor. In addition to moving the robot physically, the *Robot* object also provides functions to move the robot "virtually" to a configuration $q \in \mathcal{C}$, and interacts with the *PhysicalSpace* object to detect collision of the robot at that configuration.

The main loop in SBIC-PRM(q_{goal}) consists of scanning, updating physical space, expanding the roadmap, view planning and moving to a view node (see Program 4.2). Recall that q_{goal} is a given goal configuration for "start-goal" tasks or hybrid tasks. For exploration tasks, $q_{goal} = NULL$. First we state the termination conditions.

4.2.1 Termination Conditions:

When the planner is given a goal, i.e. start-goal tasks or hybrid tasks, it terminates when one of the following conditions is met:

(1) q_{goal} is reached. The planner terminates with a success.

(2) q_{goal} is in collision with known obstacle. The planner terminates with a failure – goal not reachable

(3) there exists no scannable region which can change the C-space entropy. In this situation, two sub-cases exist:

(3a) There are no gray landmarks left. The planner reports that q_{goal} is not m-reachable (see discussion in Section 2.2). This is because $\mathcal{C}_{free}$ is

```
SBIC-PRM(q_goal)
{
PhysicalSpace.initialization()
roadmap.initialization()
i ← 0
repeat
   if (roadmap.reachGoal(q_goal) = TRUE)
      robot.move(q_goal)
      report SUCCESS and exit
   else if (GoalInCollision)
      report FAILURE-GoalInCollision
      exit
   else
      i ← i + 1
      v_e(x_max)=ViewPlanner.planViewNode()
            /* e is a landmark, x_max is where to scan */
      if v_e(x_max) =NULL
            /*failure from choosing next view node*/
         if no gray landmarks
            report FAILURE-GoalNotMReachable and exit
         else report FAILURE-GoalNotSReachable and exit
      else robot.move(v_e(x_max))
      PhysicalSpace.scan(i)
      PhysicalSpace.update(i)
      roadmap.expand()
until i > maxIterations
report FAILURE-MaxIterations and exit
}
```

Program 4.1: Pseudo code for SBIC-PRM.

completely enclosed by obstacles (in a probabilistic sense). The planner has identified the goal is not m-reachable in this case.
(3b) There are still gray landmarks, the planner reports that q_{goal} is not s-reachable. This is because there are no scannable regions that can reduce C-space entropy, and there are still unknown region in physical space. All the regions which can reduce the ambiguity of C-space are blocked by obstacles.

(4) A maximum number of iterations has been carried out. The planner terminates with a failure.

For a pure exploration task , conditions (3) and (4) make the planner stop.

4.2.2 Description of Each Function

We now describe each function. For pseudo-code of these functions, see Appendix B.

Roadmap Initialization:

Recall that a small region of physical space, $\mathcal{P}^{(0)}_{free}$, is free. This allows the robot and the sensor to make initial movements (see Section 2.3).

Roadmap initialization function first randomly chooses landmarks within $\mathcal{C}^{(0)}_{free+}(m)$. The probabilistic roadmap algorithm [36] is then used to connect the white landmarks in the roadmap (a graph structure). The initial robot configuration is treated as a white landmark. Gray and black landmarks are stored in two separate lists.

Plan View Node:

$planViewNode()$ returns the "best" place to be scanned according to the MER criterion and a view node (a configuration from which to scan). See Section 4.3 for details. The robot will move to the view node and take a new scan in the next iteration.

Expand Roadmap:

roadmap.expand() places new landmarks and carries out the roadmap expansion in a wavefront fashion. It also connects the new white landmarks in the $\Delta\mathcal{C}_{free}^{(i)}$ just acquired in the recent scan. Details of roadmap expansion are explained in Section 4.4.

Scan and Update Physical Space:

scan() activates the sensor at the robot's current configuration, and as mentioned earlier, returns a two dimensional array of points $\in \mathcal{P}$ on the obstacles' surfaces, $\partial\mathcal{V}_{\mathcal{S}}(q)$. For representing physical space, we maintain two octrees, one for the free physical space, called $\mathcal{P}_{free}$-octree, and the other for the obstacles, called $\mathcal{P}_{obs}$-octree. Note that $\mathcal{P}_{obs}^{(i)}$ is the "surface" of obstacles seen by the sensor. We use a collection of finest resolution octree nodes on the obstacle surfaces to represent obstacles. These octrees are directly constructed in an on-line manner from sensed range data provided by the sensor using a novel algorithm, denoted here by *constructOctree*(). The algorithm is explained in Chapter 5.

Move Robot:

This involves searching the roadmap as a graph and moving the robot to a given configuration, either to a view node when the robot needs to take a new scan, or to the goal configuration q_{goal}. Dijkstra's single source shortest path algorithm [13] is applied to $\mathcal{R}^{(i)}(q_0)$ to find a path from the current landmark to the given configuration. This algorithm also computes the connectivity of $\mathcal{R}^{(i)}$. The result of graph search is a sequence of robot movements to the desired configuration. Each movement command is sent to the robot server (see Chapter 6) in our system.

Reach Goal:

The function first calculates the status of q_{goal}.

- If $q_{goal} \in \mathcal{C}_{obs}$, then return a FALSE with flag $GoalInCollision = TRUE$ indicating the goal is not reachable.
- If $q_{goal} \in \mathcal{C}_{unk}$, return a $FALSE$.
- Otherwise (i.e. $q_{goal} \in \mathcal{C}_{free}$), when a goal is given, the planner tries to reach the goal with the $local\_planner(q_l, q_{goal})$ from each of the white landmarks $l \in \mathcal{R}(l_0)$. If any of them returns $TRUE$, the planner has effectively found a collision-free path within currently known free physical space $\mathcal{P}^{(i)}_{free}$.

For an exploration task, $q_{goal} = NULL$, and *reachGoal*() simply returns $FALSE$.

4.3 Planning a View Node

The view node is determined in a two-step process as follows. The first step is to *determine where to scan*, i.e. the physical region (a direction or a point $x \in \mathcal{P}$) that should be scanned. Routine *getWhereToScan*() determines this and returns a point $x \in \mathcal{P}$ where $\mathcal{V}_{\mathcal{S}}$ should be centered. The second step is to determine from where to scan, i.e. to *find a view node* (a robot configuration), v_l, from which x is scanned. Given x, the connected component $\mathcal{R}^{(i)}(l_0)$ of the roadmap is searched to obtain a (reachable) white landmark that places the center of $\mathcal{V}_{\mathcal{S}}$ closest to x.

If the *getWhereToScan*() has exhausted all the observable physical space, it will stop and report a failure. If this happens, the sensor-based motion planner will not be able find a path because the given goal is not reachable (s-reachable or not m-reachable).

The two steps are detailed as follows:

Determine Where To Scan: There are two main objectives in choosing such a region to scan, (i) to maximize entropy reduction (exploratory component), and (ii) to preferentially "expand" toward the goal (goal component).

The first objective is achieved by maximizing IGDF, $G_e(x)$, already discussed in Section 3.4. IGDF approximates the expected information gain in the C-space when a small region, $\mathcal{B}(x) \in \mathcal{P}$, around x is sensed.

To satisfy the second objective, that of preferentially directing the expansion of free space toward the goal, the (unknown) boundary point of $\mathcal{C}_{free}$ closest to the goal should be sensed. This boundary point should have the following two properties: (1) it should be close to the goal; (2) it should not be too far from $\mathcal{C}_{free}^{(i)}$. Otherwise, this scan is not very effective.

Combining the two factors mentioned above, a reasonable choice is to find the gray landmark closest to the goal (among all the gray landmarks) subject to the condition that $\mathcal{A}(q_{l_g})$ has at least a certain fraction m_g in $\mathcal{P}_{free}$. We use l_G to represent this gray landmark. m_g is determined by the dimension of the robot and the sensing region(for dimensions, see Chapter 5). m_g is set to 0.85 in our implementation[2]. $\mathcal{A}_{unk}(q_{l_G})$ is the region that should be sensed. This is achieved by defining an objective function in physical space. This function has a value of 1 in the unknown region that the robot occupies in configuration q_{l_G}, and has a value of 0 in the rest of the physical space. Formally,

$$G_g(x) = \delta(\mathcal{A}_{unk}(q_{l_G})) = \begin{cases} 1, \ x \in \mathcal{A}_{unk}(q_{l_G}), \\ 0, \ otherwise \end{cases} \tag{4.1}$$

where $\delta(\mathcal{A}_{unk}(q_{l_G}))$ is a unit function in physical space.

The combined objective function is then a weighted sum of G_e and G_g, i.e. $G(x) = w_e G_e(x) + w_g G_g(x)$. $getWhereToScan()$ returns a point (in physical space) that maximizes $G(x)$, i.e. it returns $x_{max} = \{x : \underset{x \in \mathcal{P}_{unk}}{max} \{G(x)\}\}$.

The form of $G_g(x) = \delta(\mathcal{A}_{unk}(q_{l_G}))$ is very similar to $G_e(x) = \sum_q g_q(x)\delta(\mathcal{A}_{unk}(q))$. In fact, both components are determined by discovering the status of unknown C-space. $G(x)$ can be considered as a generalized information gain density function with a larger weight on configuration q_{l_G}.

w_e and w_g are the weights for exploratory and goal components, as mentioned in Section 2.3. When w_g is greater, the algorithm emphasizes the goal component, and

[2]Roughly speaking, the unknown part of the robot, $\mathcal{A}_{unk}(q_{l_G})$, should be covered with one scan. When we calculate m_g, we simplify the sensing region as a sphere with a diameter of 40cm. If 85% of the robot is in free physical space, this sphere sensing region covers the rest of the robot

when w_e is greater, the exploratory component is emphasized. The selection of w_g and w_e depends on the purpose of this motion planning task as discussed in Section 2.3.2.

When w_e is 0, $G(x) = \delta(\mathcal{A}_{unk}(q_{l_G}))$. This function has a constant value over region $\mathcal{A}_{unk}(q_{l_G})$. x_{max} does not have a well defined value. In this situation, the center of $\mathcal{A}_{unk}(q_{l_G})$ should be chosen as the next point to be scanned.

Given a point x_{max} to be scanned, the next step is to determine a reachable configuration from which to scan the chosen point x_{max}, i.e. to determine a view node. This is described in the following paragraphs.

Find a View Node: The feasible set of landmarks from which a view node can be derived must satisfy the following constraints: (i) *field of view* – view node $v_l(x)$ exists such that x lies within the scannable region $\mathcal{V}_{\mathcal{S}}(q_{v_l})$, which implies that x lies within the finite sensing volume; (ii) *visibility* – x is visible from the sensor, i.e. no known obstacle obstructs the view to x. (iii) *landmark reachability* – the view node is reachable, i.e. $l \in \mathcal{R}^{(i)}(l_0)$, so that the robot can move to it, and (iv) *view node reachability* – view node $v_l(x)$ is reachable from l, i.e. a simple sub-path from l to the view node within $\mathcal{C}_{\tilde{\mathcal{S}}_{int}} \times \mathcal{C}_{\mathcal{S}_{int}}$ exists (In fact, this constraint is satisfied automatically if the sub-path is in $\mathcal{C}_{\mathcal{S}_{int}}$ only.). It involves much less computation because moving over $\mathcal{C}_{\tilde{\mathcal{S}}_{int}} \times \mathcal{C}_{\mathcal{S}_{int}}$ needs little or no collision detection.

The first two constraints, as well as others not applicable to this situation (for example, illumination constraints), are studied in the computer vision area [14, 68]. These constraints are called sensor constraints [68]. The last two are special to our sensor-based motion planning problem because we can not assume all robot (sensor) movements are free from collision.

From all the landmarks which satisfy the above constraints, we choose the one whose view node configuration results in the center of the sensing region $\mathcal{V}_{\mathcal{S}}(q_{v_l})$ being closest to x. This is the landmark from which the view node is derived and is returned by $planViewNode()$ along with the corresponding view node $v_l(x)$.

The precise mechanism to determine the view node depends on the particulars of the system. The search for the view node in this system is a practical one, and is

done as follows. From each white landmark l in $\mathcal{R}^{(i)}(l_0)$, we let the first three *dofs* (because they are not quasi-internal/internal sensing degrees) be fixed, while varying the last three *dofs* (quasi-internal sensing degrees). Using inverse kinematics of the last three joints, we calculate the values of the last three *dofs* needed to see point x_{max} with the first three *dofs* fixed. Call this view node configuration q_{v_l} (different values for the last 3 joints). Ideally, q_{v_l} should be treated like a query point for the roadmap. Planner should try to reach q_{v_l} from all the white landmarks in $\mathcal{R}^{(i)}(l_0)$. However, since q_{v_l} is derived from landmark l, only *local_planner*(q_l, q_{v_l}) is invoked. (If *local_planner*(q_l, q_{v_l}) fails, the chance of connecting q_{v_l} to other landmarks is very small since q_{v_l} is much closer to q_l than to the other landmarks. We ignore the other landmarks for efficiency reason.) If *local_planner*(q_l, q_{v_l}) returns success, then q_{v_l} is a reachable view node. Note that this *local_planner*(q_l, q_{v_l}) does not compute the collision of the entire robot. It checks the collision only for the (physical part) sensor which is affected by quasi-internal *dofs*.

It may so happen that there are no landmarks to derive a view node for a given x, i.e. the above constraints may not be satisfied by any of the landmarks for a given x. In this event, $getWhereToScan()$ is called again. It then locates the next best point x'_{max} from $G(x)$. This is achieved simply by tagging the best points and remembering them. As introduced in Section 3.6, if the robot's movement to q_{v_l} is prevented by unknown region x_{pre}, the view planner can either scan x'_{max}, or first scan x_{pre} then scan x_{max}. The scheme will be determined by MER over two (or multiple) scans. Because of computational cost, we do not use more than two steps in our implementation.

4.4 Expand Roadmap

4.4.1 Adding New Landmarks

At each iteration (say i), the robot takes a snapshot range image, and $\Delta\mathcal{P}^{(i)}_{free}$ is the corresponding additional physical free space (see Chapter 4 for adding $\Delta\mathcal{P}^{(i)}_{free}$). Our

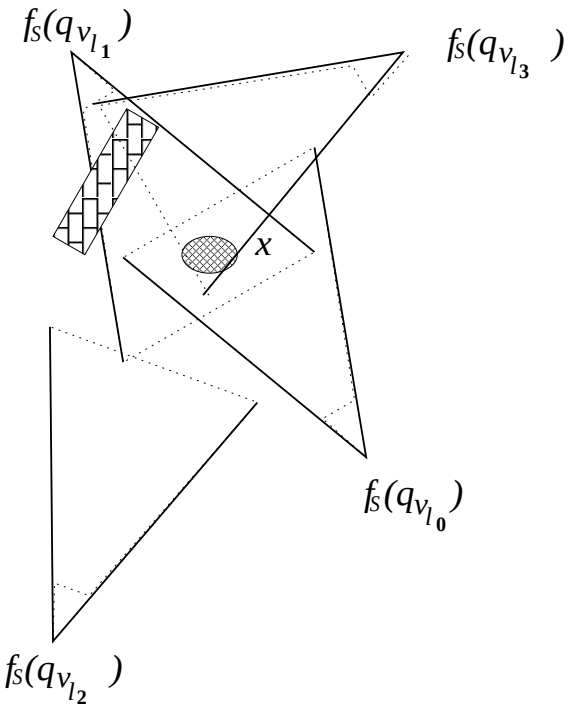


Figure 4.4: Schematic showing various constraints for determining a landmark to scan a point $x \in \mathcal{P}$. The shaded oval represents the region to be scanned and the triangular area is the field of view of the sensor. Landmark l_1 does not satisfy the visibility constraint, since an obstacle occludes point x. Landmark l_2 does not satisfy the containment constraint since x is out of the sensing volume $\mathcal{V}_{\mathcal{S}}(q_{l_2})$. We choose landmark l_0 because x is closer to the center of $\mathcal{V}_{\mathcal{S}}(q_{l_0})$ than to the center of $\mathcal{V}_{\mathcal{S}}(q_{l_3})$.

underlying model-based planner (PRM) is probabilistically complete[3]. This implies that $\mathcal{R}^{(i-1)}$ is an adequate representation of $\mathcal{C}_{free}^{(i-1)}$ in a probabilistic sense. Therefore, the roadmap expansion at each iteration i should be limited to $\Delta\mathcal{C}_{free}^{(i)}$, which is the additional free configuration space that arises because of the new $\Delta\mathcal{P}_{free}^{(i)}$. This ensures that the algorithm maintains a uniform distribution of the landmarks in C-space. Otherwise, as in [38], the distribution may degenerate during the planning process as more and more landmarks are placed in the same areas of $\mathcal{C}_{free}$, leading to a very large number of landmarks. Although these redundant landmarks represent a point in free/obstacle/unknown regions, large numbers of landmarks cause inefficiency.

To avoid degeneration of landmark distribution as mentioned in [38] (the random distribution landmark becomes non-uniform and clustered), two approaches can be used to place landmarks (Both of the two approaches can avoid landmark distribution degeneration. We adopted the second approach in our implementation for conceptual simplicity.). Let ρ (chosen *a priori*) be the density of landmarks in C-space[4]. $k(= \rho \times vol(\mathcal{C}))$ landmarks would be generated in the entire C-space.

Approach 1:

k configurations are generated randomly in $\mathcal{C}$ and stored in a candidate landmark list, $\mathcal{L}_c$, during the initialization.

Recall that $\mathcal{C}_{free+}(m)$ in Section 4.1.1 is defined as $\{q \in \mathcal{C} : (1 - \frac{\mathcal{A}_{unk}(q)}{\mathcal{A}(q)}) > m\}$. Therefore, at each iteration (including iteration 0, the initialization), $\forall q \in \mathcal{L}_c$, if $1 - \frac{\mathcal{A}_{unk}(q)}{\mathcal{A}(q)}) > m$, q is added to $\Delta\mathcal{L}$ (becomes a new landmark), and is *removed* from $\mathcal{L}_c$. The configurations remaining in $\mathcal{L}_c$ are used in subsequent iterations. Since the landmark candidates are removed from $\mathcal{L}_c$ when they become landmarks, the landmarks are always derived from the original $\mathcal{L}_c$. Therefore, their distribution obviously is uniform.

[3]A planner is called probabilistically complete if, given a solvable problem, the probability of solving it converges to 1 as the running time goes to infinity. Such a planner is guaranteed to solve any solvable problem within finite time (in a probabilistic sense) [67].

[4]For choosing the density of landmark ρ, please see the experimental results in [36, 49].

Approach 2:

k new configurations are generated randomly in $\mathcal{C}$ in each iteration. A configuration $q \in \Delta\mathcal{L}$ (becomes a new landmark) if and only if this configuration lies in $\mathcal{C}^{(i)}_{free+}(m)$, but not in $\mathcal{C}^{(i-1)}_{free+}(m)$. In other words, a new configuration q becomes a landmark if : (i) the volume of $\mathcal{A}(q)$ that lies within $\mathcal{P}^{(i)}_{free}$ is greater than $m \cdot \mathcal{A}(q)$; and (ii) the volume of $\mathcal{A}(q)$ that lies within $\mathcal{P}^{(i-1)}_{free}$ is less than $m \cdot \mathcal{A}(q)$.

After carrying out either one of the processes mentioned above, we then check the color(s) of the landmarks in $\mathcal{L}_g$ and $\Delta\mathcal{L}$. $\mathcal{L}_g$ contains the gray landmarks whose color could be changed due to a new scan. $\Delta\mathcal{L}$ contains the new landmarks whose colors have not been determined. If a gray landmark remains gray in the updated physical space (represented by $\mathcal{P}^{(i)}_{free}$ and $\mathcal{P}^{(i)}_{obs}$), it is retained in $\mathcal{L}_g$; if it becomes black, it is removed from $\mathcal{L}_g$ and inserted in $\mathcal{L}_b$; and if it becomes white, it is removed from $\mathcal{L}_g$ and added (as a node) in the roadmap.

4.4.2 Updating Roadmap

After the new white landmarks are added to the roadmap, the *local_planner*() is repeatedly called to try to connect them with pre-existing (white) landmarks in the roadmap.

When the local planner tries to connect two landmarks, it starts from both landmarks towards each other, along the straight line segment connecting the two. It will encounter three situations: (1) *local_planner*() finds a free path between two landmarks, and an edge is added to roadmap $\mathcal{R}$; (2) *local_planner*() finds an obstacle between two landmarks; (3) *local_planner*() finds an unknown region (edge segment) along the path between the two landmarks. This third case is handled as follows primarily for efficiency. We add two temporary gray landmarks, one on each side of the unknown edge segment. After the sensor scans the environment and the physical space has been updated in the next iteration, an attempt will be made to connect

the same edge between the two temporary landmarks, instead of the two white landmarks, thus avoiding duplicating collision detection computations from one iteration to the next. These temporary landmarks are removed when an edge is connected successfully (the two temporary landmarks meet at a certain place on the edge), or an obstacle is found between the two temporary landmarks.

The core computation in this process is that of collision detection between the robot model and the environment representation. The geometry of the robot and the sensor is modeled by a collection of polyhedra according to their real shapes, sizes and kinematic relations (see [15]). Recall that the environment is represented as octrees. Each node in the octree is represented as a "cube" with its size determined by the level of the node. An existing collision detection algorithm [55] for polyhedral primitives is then used for collision detection. Routine *colorOf()* uses this collision detection procedure to determine the color of a landmark. The volume computation, i.e. computing whether m fraction of the robot total volume lies in $\mathcal{P}_{free}$, is in practice replaced by calculating the fraction of surface area for algorithmic reasons. Because the collision detection algorithm used here checks the intersections of the surface of the robot model with the obstacles. We compute the fraction of triangles on the surface of the polyhedral robot model that lies within $\mathcal{P}_{free}^{(i)}$, again using the same collision detection procedure as described above.

4.5 Query

When the planner finishes the task, it has a roadmap representing (part of) the free C-space. Similar to model based planning [37], the roadmap can be used to query the planner if a collision free path to a given goal configuration exists within the existing roadmap. The planner tries to connect the goal configuration to each node in the roadmap by invoking the *local_planner*(). If any of the invocation succeeds, the goal is reachable. Otherwise, it is not.

A unique feature of the queries in our sensor based motion planning problem is

that the exploration still may not be complete after a given task. This is especially true when the weight on exploratory component is not large. Therefore, similar to updating the roadmap, there might be three different cases for the "off ramp", the connection to *new goal* from the roadmap: (a) the connection intersects obstacles, (b) the entire connection is free, and (c) the connection partially intersects unknown space, but does not intersect obstacles. Obviously, if a (b) type of connection exists, a free path is found, and success is returned. If all the candidate connections are (a) type, a failure is reported. Otherwise, further exploration is needed. SBIC-PRM is then re-started with the given new goal.

4.6 Discussion on Completeness

The model-based algorithm we adopt is a probabilistically complete algorithm. In order to determine the completeness of SBIC-PRM, we need to examine the completeness of the view planner only.

Recall that a complete view planner is defined as: if there exists a point $x \in \mathcal{E}_{\mathcal{V}}$, then a complete view planner can find a configuration q, such that point x can be sensed from configuration q. Otherwise, report $\mathcal{E}_{\mathcal{V}} = \emptyset$. $\mathcal{E}_{\mathcal{V}}$ is a part of the boundary of $\mathcal{P}_{unk}/\mathcal{P}_{free}$, which is visible from $\mathcal{C}^{(i)}_{free}(q_0)$ and touchable from the closure of $\mathcal{C}^{(i)}_{free}(q_0)$. The completeness of SBIC-PRM essentially depends on whether $\bigcup_{q \in \mathcal{L}_w}(\mathcal{V}_{\mathcal{S}}(q) \bigcup \partial\mathcal{V}_{\mathcal{S}\lfloor f}(q))$ will cover the entire $\mathcal{E}_{\mathcal{V}}$ (again, in a probabilistic sense) when the number of white landmarks is large enough. We are not able to answer this question for now. Our conjecture is that our view planner is complete, since for the majority of the white landmarks (except for degenerate cases) each scan from a landmark covers a certain region in the observable physical space . Therefore, our conjecture is that SBIC-PRM is a (probabilistically) s-complete algorithm.

Chapter 5

Physical Space Representation – Octree

As previously stated, the planner has a graph representation for the free C-space, gray landmarks for the unknown and black landmarks for the obstacle regions in C-space. A main computation in the planning process is that of updating the roadmap due to the newly acquired scan. A key part of this update process is to incrementally add a new set of landmarks ($\Delta\mathcal{L}^{(i)}$). However, when a new scan $\mathcal{V}_\mathcal{S}$ is made, it is not possible to calculate $\Delta\mathcal{L}^{(i)}$ from $\Delta\mathcal{P}_{free}$ only. This is because the $\Delta\mathcal{C}_{free}^{(i)}$ (and $\Delta\mathcal{L}^{(i)}$) depends not only on $\Delta\mathcal{P}_{free}$ but also on $\mathcal{P}_{free}^{(i-1)}$. Therefore, the planner needs to maintain and store $\mathcal{P}_{free}$ and $\mathcal{P}_{obs}$ throughout the entire planning process. The question then arises as to what type of representation is best suited for representing the environment.

As we mentioned in the introduction, the octree is the best choice for our sensor-based motion planning problem. However, constructing octrees is usually a time consuming process especially when octree resolution is high. For example, given the voxels on the visible surface (see Figure 5.1), we could first convert the visible surface into a voxel map and then construct an octree from the voxel map. This brute force method is very inefficient in space as well as in time. In this chapter, we present a time and (memory) space efficient algorithm which constructs octrees directly from range images. Our algorithm makes use of a key *visibility* property of most range sensors – all the voxels on the object surfaces in each range image can be "seen" from

the view point, i.e. there is no object between the view point and any visible surfaces. With this key property, it is possible to define a projection map which stores certain distance information extracted from range data, and which makes octree construction fast.

In particular, we explicitly represent the free space and obstacles as octrees, which are denoted by $\mathcal{P}_{free}$-octree and $\mathcal{P}_{obs}$-octree respectively. We present algorithms for constructing octrees from range images $\mathcal{V}_{\mathcal{S}}$ in Section 5.2. In Section 5.3, we presents algorithms for manipulating the octrees such as union, intersection and difference logical operators. In section 5.4, we discuss our approach to derive $\mathcal{P}_{obs}$-octree. Throughout the entire algorithm, we follow the convention that white nodes represent the effective components with respect to their names in physical space. For example, white nodes in $\mathcal{P}_{obs}$-octree represent obstacles; white nodes in $\mathcal{P}_{free}$-octree represent free space.

5.1 Model of Range Sensor

The free space obtained from one scan is modeled as a cone $\mathcal{V}_{\mathcal{S}}$, whose apex is at the center of the sensor. The base of $\mathcal{V}_{\mathcal{S}}$, i.e. $\partial\mathcal{V}_{\mathcal{S}}$, is composed of visible surface segments on objects. The range image is a discretized array with each element representing a voxel in physical space.

The visibility property of the range sensor implies that there is a single-valued distance function (in spherical coordinates) with an origin at the center of the sensor. A ray in any direction within $\mathcal{V}_{\mathcal{S}}$, starting at the cone apex, will intersect $\partial\mathcal{V}_{\mathcal{S}}$ exactly once (see Figure 5.1).

5.2 Constructing $\mathcal{P}_{free}$-octree from Range images

We now present our octree construction algorithm *constructOctree*(). We use a pointer based octree. Octrees have three types of nodes: black, white and gray. The black node represents a region which is not free (obstacle/unknown). The white node represents a free region and the gray node represents a mixture of both non-free and

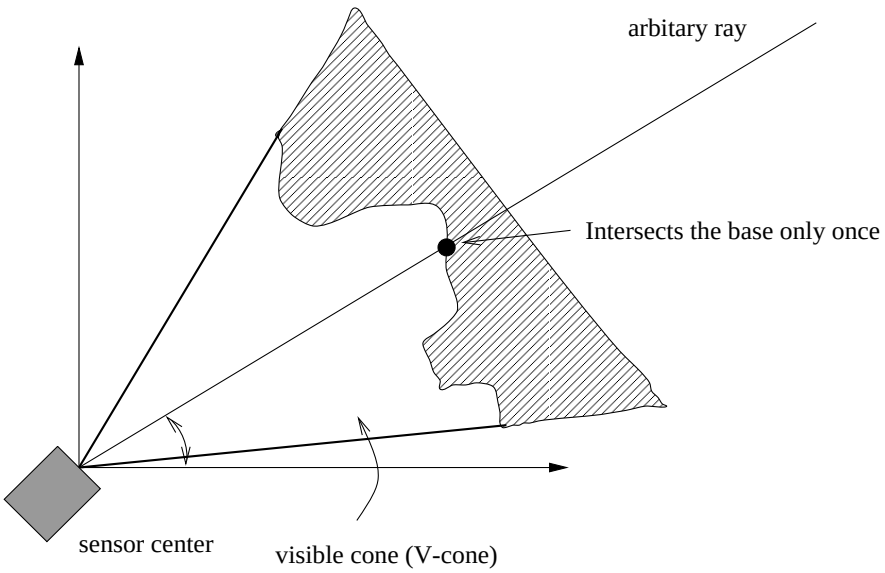


Figure 5.1: The sensor model.

free region. All non-leaf nodes have 8 pointers pointing to their respective children. The basic structure of our octree construction algorithm is as follows: it first checks whether a node lies completely inside or outside the $\mathcal{V}_{\mathcal{S}}$. If the node is completely inside $\mathcal{V}_{\mathcal{S}}$, it is a white node. If the node is completely outside $\mathcal{V}_{\mathcal{S}}$, it is a black node. Otherwise, it is labeled gray and the algorithm then performs decomposition of this node into 8 sub-nodes. The process is repeated until the node is decomposed into small regions which can be represented by a white or a black node.

Function $constructOctree()$ inputs a range image $\partial\mathcal{V}_s$, and recursively decomposes the workspace into eight smaller subspaces and calls $compute\_node\_color()$, until the color of the nodes is determined or until maximum resolution has been reached.

In order to efficiently determine the color of the nodes, a projection map is pre-computed. In this section, we first introduce the definition of a projection map and then discuss the procedure of computing the color of the octree nodes.

5.2.1 Projection Map

Based on the visibility property of the sensor model (see Section 5.1), we define a projection map. Imagine a bounding box enclosing the workspace. The box's surface

is discretized at the finest resolution of the robot workspace (see Figure 5.2). The six sides in the projection map are:

$$x_1 = \pm l_{max}, -l_{max} \leq x_2, x_3 < l_{max};$$
$$x_2 = \pm l_{max}, -l_{max} \leq x_1, x_3 < l_{max};$$
$$x_3 = \pm l_{max}, -l_{max} \leq x_1, x_2 < l_{max}.$$

where l_{max} is the dimension of the work space. In our experiments, $l_{max} = 128cm$.

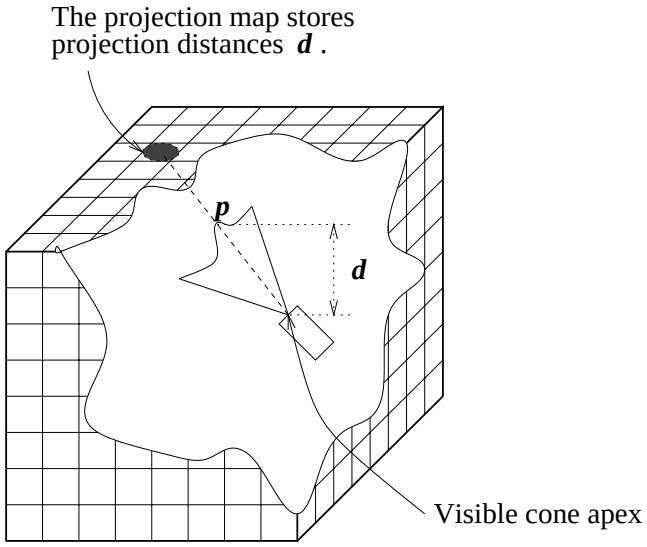


Figure 5.2: The projection map.

Let $\mathbf{x_p} = (x_{p1}, x_{p2}, x_{p3})$ be the coordinate of a voxel $\mathbf{p}$[1], and $\mathbf{x_a} = (x_{a1}, x_{a2}, x_{a3})$ be the coordinate of the apex of $\mathcal{V}_\mathcal{S}$. $\mathbf{x_a}$ can be obtained from the robot forward kinematics $f_s(q_v)$ when the robot is at a view node with configuration q_v. Let $\mathbf{x} = t \cdot (\mathbf{x_p} - \mathbf{x_a}) + \mathbf{x_a},\ t > 0$, be a ray starting from $\mathbf{x_a}$ in the direction of $\mathbf{x_p}$. This ray will intersect one of the six faces of the bounding cube. We use $P(\mathbf{p})$ to denote the 2-D coordinates (on the particular cube face) of this intersection point. The point projection distance of $\mathbf{p}$ is $d(\mathbf{x_p}) = |x_{pj} - x_{aj}|$, where j corresponds to the coordinate direction that is perpendicular to the intersected cube face. $d(\mathbf{x_p})$ is used to efficiently

[1]We don't make a strict distinction between a voxel and a point.

determine if a point $\mathbf{x_p} \in \mathcal{V_S}$.

The projection map, denoted by M, essentially stores the point projection distances of the voxels on $\partial\mathcal{V_S}$ (these are provided by the range sensor) and is defined as $M(P(\mathbf{p})) = d(\mathbf{p})$, where $\mathbf{p} \in \partial\mathcal{V_S}$. However, since voxels in the range data may project rather sparsely on the cube faces, the algorithm uses a simple linear interpolation to fill in distance values in between the projected values of $\partial\mathcal{V_S}$.

5.2.2 Determining Node Color

Given a node of the octree, *compute_node_color*() returns its color (black, white or gray) according to the relative spatial relationship of $\mathcal{V_S}$ and the node (a cube). Since $\mathcal{V_S}$ is a simply connected region (no holes), only the surface of the node has to be checked in order to determine its color. Obviously, if all the faces on the node surface are in $\mathcal{V_S}$, then this node is a white node; if the faces are partially inside $\mathcal{V_S}$, then the node is a gray one; if all the faces are outside $\mathcal{V_S}$, then this node can be either black when the entire node is outside $\mathcal{V_S}$, or gray when $\mathcal{V_S}$ is completely inside this node. The two cases can be distinguished by checking whether the apex of $\mathcal{V_S}$, $\mathbf{x_a}$ is inside the node. The pseudo-code of the function *compute_node_color*() is listed in Appendix B.11.

We now examine a characteristic of node faces, which leads to time efficiency in the octree construction algorithm. We found the faces of the nodes to be checked can be classified into two classes, *near-faces* and *far-faces*. Imagine a ray starting from the apex towards the node. The ray intersects the node surface twice, except for some degenerate cases when the two intersecting points merge into one point on the edge of the node. We call a face *near-face* if all intersection points on that face are the intersection points closer to the apex. We call a face *far-face* if all intersection points are the intersection points further from the apex. We can always have this classification because nodes are convex cubes. A third possibility is that a face is coplanar with $\mathcal{V_S}$ apex. We ignore this type of face in the color determination function because it would not affect the color of the node after other faces have been checked. Figure 5.3 shows near-faces, far-faces and faces to be ignored.

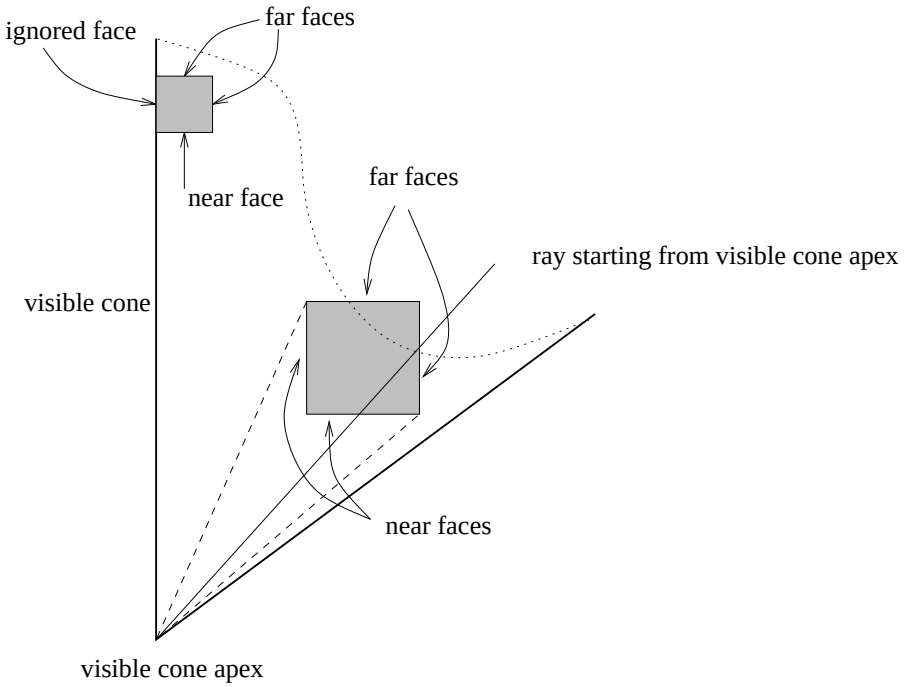


Figure 5.3: Near-faces and far-faces.

Let $\mathbf{x_p}$ be an arbitrary point on a node face. $\mathbf{x_a}$, as before, is $\mathcal{V}_\mathcal{S}$ apex, and $\mathbf{n}$ denotes the outward pointing normal of the node face. Then

if $(\mathbf{x_p} - \mathbf{x_a}) \cdot \mathbf{n} > 0$, the face is a near-face
if $(\mathbf{x_p} - \mathbf{x_a}) \cdot \mathbf{n} < 0$, the face is a far-face.
if $(\mathbf{x_p} - \mathbf{x_a}) \cdot \mathbf{n} = 0$, neglect this face.

On average, *compute_node_color*() checks less than half of the faces. The reason that this approach is correct results from the visibility property which guarantees that all the voxels on a line segment, from the apex to any voxel on $\partial\mathcal{V}_\mathcal{S}$, are free. If all the near-faces are outside of $\mathcal{V}_\mathcal{S}$, the node is completely outside $\mathcal{V}_\mathcal{S}$, since all the voxels in the node are further (from the apex) than some voxel on the near-faces. Analogously, the function can determine if a node is completely inside $\mathcal{V}_\mathcal{S}$ by checking far-faces.

5.2.3 Checking Faces

To check whether a face of a node is inside $\mathcal{V}_\mathcal{S}$, the face is discretized at the finest resolution of the octree. Each element is then checked if it is inside $\mathcal{V}_\mathcal{S}$ or not. This can be done by projecting the elements onto the projection cube and by comparing the distances stored in the corresponding position on the map. Formally,
$(\forall \mathbf{p} \in \{face\ elements\}),\ M(P(\mathbf{p})) \geq d(\mathbf{p}) \Rightarrow the\ face\ is\ inside;$
$(\forall \mathbf{p} \in \{face\ elements\}),\ M(P(\mathbf{p})) < d(\mathbf{p}) \Rightarrow the\ face\ is\ outside.$

Thus, we now have an algorithm *constructOctree*() that computes octrees from the range image $\mathcal{V}_\mathcal{S}$.

5.3 Logical Operators

In SBIC-PRM, we need to calculate the union and difference of octrees in each iteration, for example, $\mathcal{P}_{free}^{(i)} = \mathcal{P}_{free}^{(i-1)} \bigcup \Delta\mathcal{P}_{free}^{(i)}$.

Standard logical operator algorithms can be used in SBIC-PRM. However, we observe the following facts which lead to efficient algorithms: (i) At iteration i, $\mathcal{V}_\mathcal{S}$ always overlaps partially with $\mathcal{P}_{free}^{(i-1)}$ because the sensor itself has to be always in free space. This implies that the volume of net increment, $\Delta\mathcal{P}_{free}^{(i)}$, is smaller than the volume of $\mathcal{V}_\mathcal{S}$. (ii) We also know that the computational time of constructing an octree is related to the number of nodes in the octree. The relationship between the number of nodes and the computational time leads us to explore the approaches that directly obtain union, intersection and difference of octrees, rather than calling *constructOctree*() and then using standard logical operators on octrees.

Our logical operation algorithms construct the union, difference or intersection of two octrees. We use the union as an example to illustrate the procedure. The other two procedures, difference and intersection, are quite similar to the construction of the union.

The algorithm has two input data structures, $\mathcal{P}_{free}^{(i-1)}$-octree and $\partial\mathcal{V}_\mathcal{S}$. The output

of the algorithm is the octree representation of $\mathcal{P}_{free}^{(i-1)} \bigcup \mathcal{V}_{\mathcal{S}}$, which is $\mathcal{P}_{free}^{(i)}$-octree.

The basic idea of the octree union algorithm is: in order to save unnecessary computation, we construct only part of the octree for $\mathcal{V}_{\mathcal{S}}$, which is *not* inside the existing free space, $\mathcal{P}_{free}^{(i-1)}$. From the octree point of view, we need to consider only the black nodes of $\mathcal{P}_{free}^{(i-1)}$-octree, which are not free (unknown/obstacle) according to our convention. The white nodes are already in $\mathcal{P}_{free}^{(i-1)}$. Given $\partial\mathcal{V}_{\mathcal{S}}$, the algorithm first constructs the projection map as before. Then, the algorithm traverses $\mathcal{P}_{free}^{(i-1)}$-octree. The algorithm will encounter three cases during the traversal.

1) White nodes: do nothing and return.

2) Gray nodes: keep the current structure, and traverse its child nodes.

3) Black nodes: construct the octree within the space specified by this black node to the finest spatial resolution. This sub-octree is built from the projection map of $\partial\mathcal{V}_{\mathcal{S}}$. Replace the black nodes with this sub-octree.

The result of the traversal is the $\mathcal{P}_{free}^{(i)}$-octree. However, this new octree generated by this approach is likely to be a degenerate one, i.e., some of the gray nodes may have all the leaf child nodes with the same color (either black or white). In this case, these gray octree nodes should be replaced with black or white nodes. A following clean-up process has to be applied to make the octree become a regular one.

The efficiency of the logical operator algorithms depends on the degree of overlap between $\mathcal{P}_{free}^{(i-1)}$-octree and $\mathcal{V}_{\mathcal{S}}$. The overlap is the part of free physical space which has been determined before. When there is more overlap, the union algorithm will be more efficient than the standard algorithms (constructing an octree for $\partial\mathcal{V}_{\mathcal{S}}$, and then applying standard logical operations). If there is no overlap, this algorithm is slightly less efficient.

5.4 Deriving $\mathcal{P}_{obs}^{(i)}$-octree

$\mathcal{P}_{obs}$ is different from the boundary of $\mathcal{P}_{free}$. The latter differs from the former in the following three ways: (i) it may include the boundary of shadows caused by occlusion of obstacles (Figure 5.4), (ii) it may include the boundary caused by the limit of the sensing distance (Figure 5.4), and (iii) it may include the boundary caused by the

robot itself, since the sensor may scan part of the robot (Figure 5.5). We determine $\Delta\mathcal{P}_{obs}^{(i)}$ from the boundary of $\Delta\mathcal{P}_{free}^{(i)}$ by excluding the spurious boundary caused by the above three cases as follows.

We first construct an octree O_1 that represents $\Delta\mathcal{P}_{free}^{(i)}$ from the range image $\partial\mathcal{V}_s$ and $\mathcal{P}_{free}^{(i-1)}$-octree, i.e., $O_1 = \Delta\mathcal{P}_{free}^{(i)} = constructOctree(\partial\mathcal{V}_s) - \mathcal{P}_{free}^{(i-1)}$-octree. Next, we construct a new range image, $\partial\mathcal{V}_{Sobs}$, by "contracting", along the sensing direction, all the data elements of the range image other than the ones that are at the maximum range value (which is easily checked), i.e., the elements at maximum range are left unchanged. We construct another octree, O_2, from this modified range image, i.e. $O_2 = constructOctree(\partial\mathcal{V}_{Sobs})$. $\mathcal{P}'_{obs}$-octree $= O_1 - O_2$ then represents an octree of the boundary of $\mathcal{P}_{free}$ with type 1 and type 2 spurious boundaries excluded.

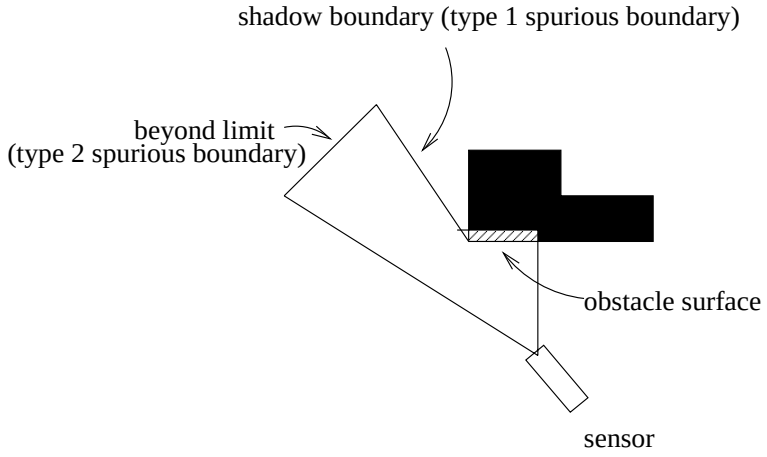


Figure 5.4: An example of type 1 and type 2 spurious boundaries that must be removed to get the obstacle octree.

For excluding the type 3 spurious boundary, we exploit the fact that the robot is initially in free physical space and always stays in $\mathcal{P}_{free}$. Any obstacles in $\mathcal{P}'_{obs}$-octree and in $\mathcal{P}_{free}^{(i)}$ must correspond to Type 3 spurious boundaries. Therefore, we can say that $\Delta\mathcal{P}_{obs}^{(i)} = \mathcal{P}'_{obs} - \mathcal{P}_{free}$, where $\Delta\mathcal{P}_{obs}^{(i)}$ denotes the part of the obstacles that becomes known in the new scan.

After all spurious boundaries are eliminated, the net obstacle is added to $\mathcal{P}_{obs}^{(i-1)}$, i.e., $\mathcal{P}_{obs}^{(i)} = \Delta\mathcal{P}_{obs}^{(i)} \bigcup \mathcal{P}_{obs}^{(i-1)}$.

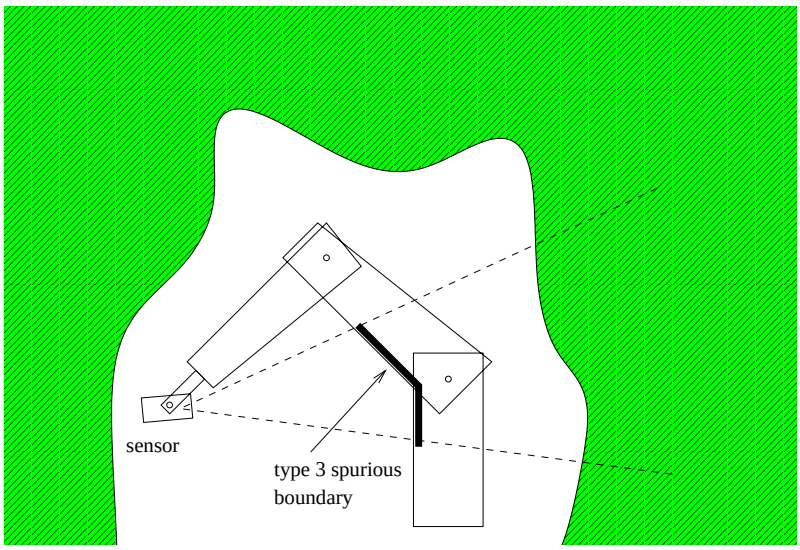


Figure 5.5: An example of (type 3) spurious boundaries caused by the sensor scanning the robot itself. The thick line corresponds to the part of the robot surface scanned by the sensor.

5.5 Examples

We now give some examples from the results of the octree construction, obstacle calculation and logical operator algorithms.

Figure 5.6 (left) shows the free spaces from single shots of range images. For clarity, we only show leaf nodes corresponding to free space in the octree. These white leaf nodes are shown in gray cubes at their real positions and dimensions. In the left figure, free spaces are represented as a gray region. The pointy part on the left of the free region is the apex of $\mathcal{V}_{\mathcal{S}}$. The sensor is scanning to the right side in the figure. In the right part of the figure, the gray areas are obstacles. Again the obstacles are composed of cubes corresponding to the leaf nodes of the obstacle octree. The obstacles are obtained from the same scan and calculated with the previously mentioned algorithms.

Figure 5.7 shows the union of obstacles obtained from different scans. In order

to show the correspondence to the real objects, we also show a photo of the robot's physical space in Figure 5.5 with labels on the obstacles.

Constructing an octree (at $128 \times 128 \times 128$ resolution level) takes about 1–2.5 seconds on a Pentium II–450 computer running Windows NT. Union, intersection and negation take about $0.1 - 0.5$ seconds on the same computer.

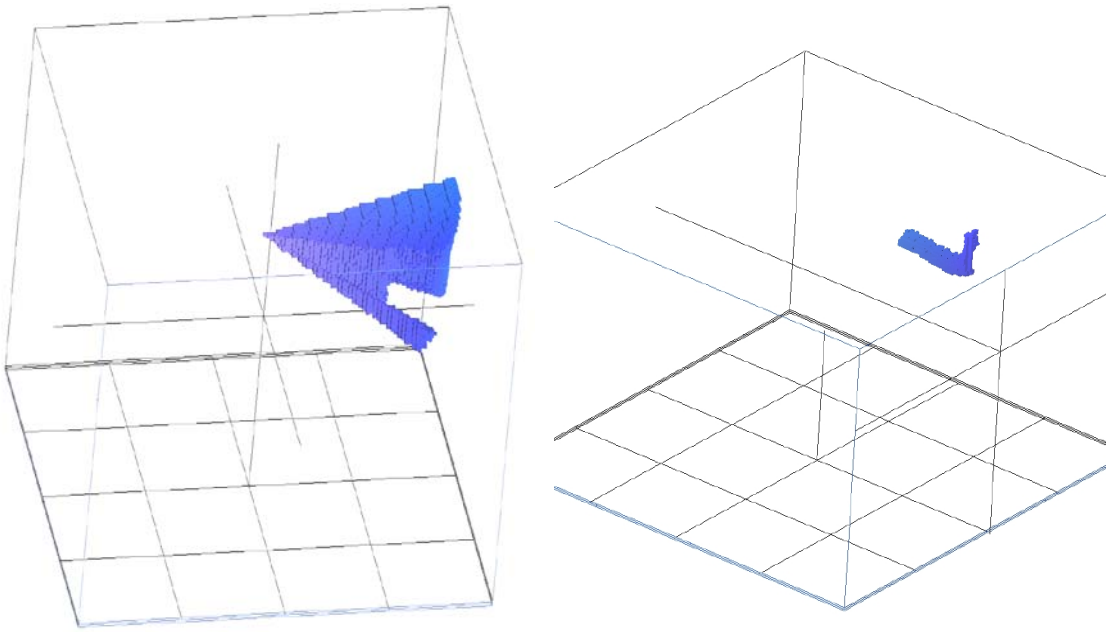


Figure 5.6: Examples of a free space octree and an obstacle octree. The pictures show the free space (left) and the obstacle surface (right) from one range image. In this scan, there is only one curved obstacle in the visible region.

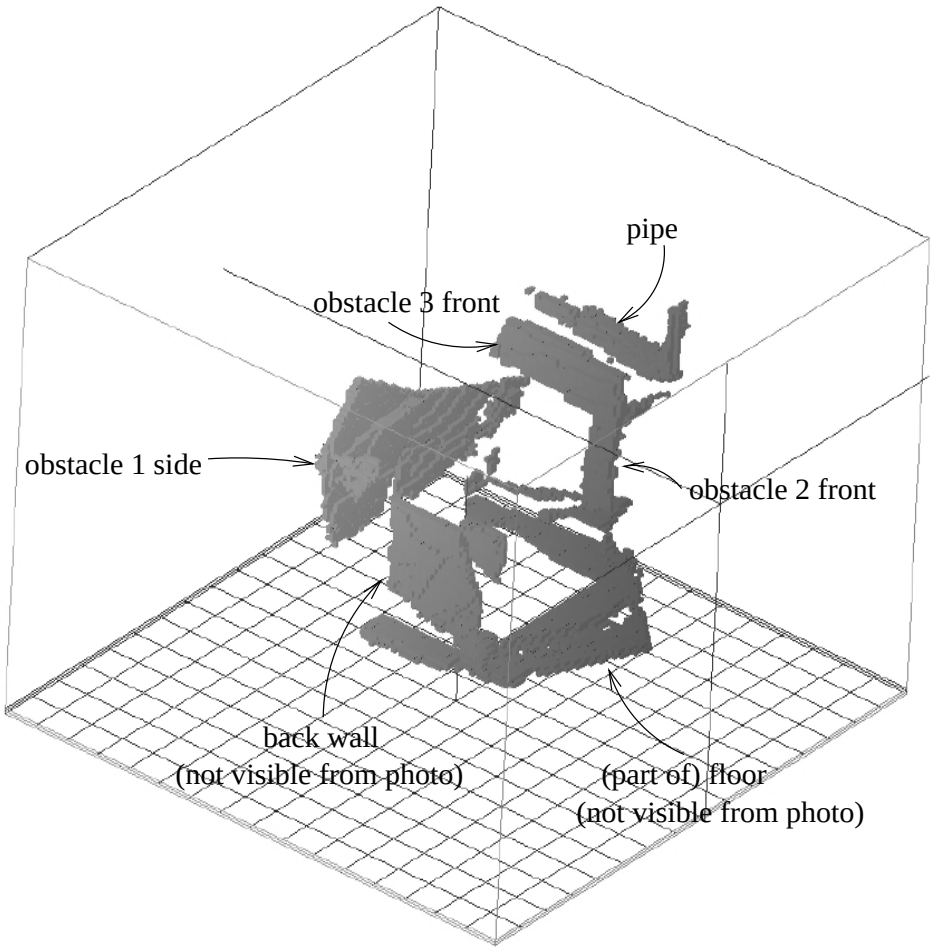


Figure 5.7: This figure shows the correspondence of $\mathcal{P}_{obs}$ with the real photo shown on the next page.

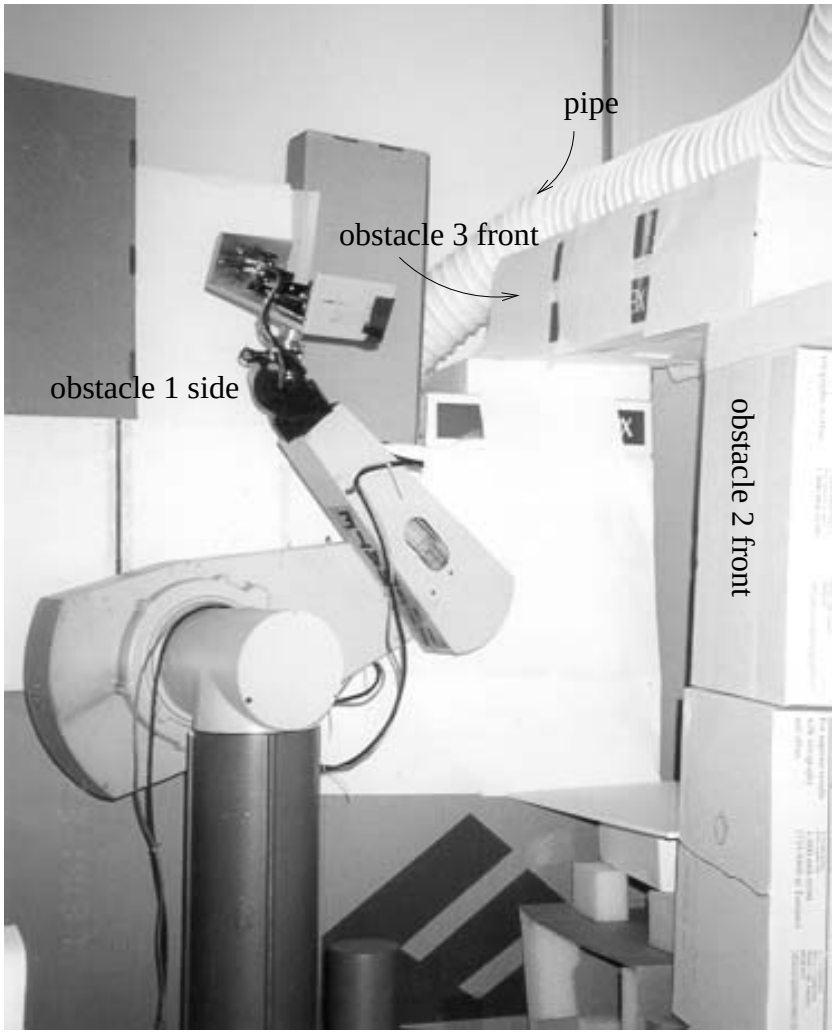


Figure 5.7: A photo used to show the correspondence of real objects, with their internal representation of the physical space shown in the previous page.

Chapter 6

System

6.1 Sensor-Based Planning Testbed

We describe the hardware of the system in Section 6.1.1 and the software architecture in Section 6.1.2.

6.1.1 The Physical System

The physical system test-bed, shown in Figure 1.3, consists of a six-dof PUMA 560 equipped with a wrist-mounted range sensor – an area-scan triangulation based laser range scanner. The decision to mount the sensor on the robot end-effector, thus providing a six-dof scanning ability, was motivated by the additional maneuverability for sensing. The sensor can scan from any point in the reachable workspace of the robot and in almost any direction (subject to joint limits).

The sensor hardware is composed of a solid state laser striper, a CCD camera, and a framer grabber residing in a PC. The camera and the laser striper are mounted on the PUMA end-effector with their (the camera and the striper) relative positions fixed.

Based on the geometrical relationship between the camera and laser striper, the spatial position of the laser reflection in the sensor's frame can be calculated. Without moving the sensor, only one static stripe of laser beam is projected into the scene.

The sensor can only obtain the distances along a "laser scan line". In our system, the last joint (joint 6) of the Puma robot is rotated to scan the laser light stripe across the scene to obtain range data over an "area" in the scene. This is then represented as a range image.

In order to integrate the distances of each "laser scan line", the value of joint 6 is needed at each time when the camera takes video image. When the sensor scans, joint 6 of the robot is moved at an even rotation speed except for the accelerating and decelerating periods. Therefore, we use the time elapsed from the starting time of the scanner to the time when the camera takes video image to compute the instantaneous angle of joint 6. This angle, together with each stripe of range image, is used to calculate the entire range image. The sensor in our system is modeled after the one reported in [65] [1].

The sensor specifications are as follows. The sensor is configured to sense in the range of 30 cm - 105 cm from the camera. Within this measurable range, the sensor is accurate to about 0.5 cm (in all three coordinates). This range and resolution is just adequate for our Puma robot whose positioning accuracy is now roughly 1cm - 3cm, and whose workspace radius is about 1m. The field of view of the sensor in the direction perpendicular to the laser stripe (joint 6 rotating direction) is not limited. However, in the direction of the laser stripe, it is limited to 35 degrees. The scanning time depends on the scanning resolution and the field of view. In our typical application, it takes about 30 seconds to get a 256 by 256 resolution range image.

6.1.2 System Software Architecture

The main component blocks of the testbed software (written in C/C++) are: (i) the robot server, (ii) the sensor server, and (iii) the planner. The two server processes, the planner, the sensor and the robot communicate with each other through sockets using TCP/IP and UDP/IP protocols. Such an implementation keeps the system modular and hence easier to maintain and add to. Furthermore, to achieve a fast response and to facilitate intensive data transmission between the processes, we use a switched

[1] Sensor was built by Gilbert Soucy.

hub for the system to screen the outside network traffic. See Figure 6.1 for the basic system architecture and data flow. We now describe the system details for each of the processes.

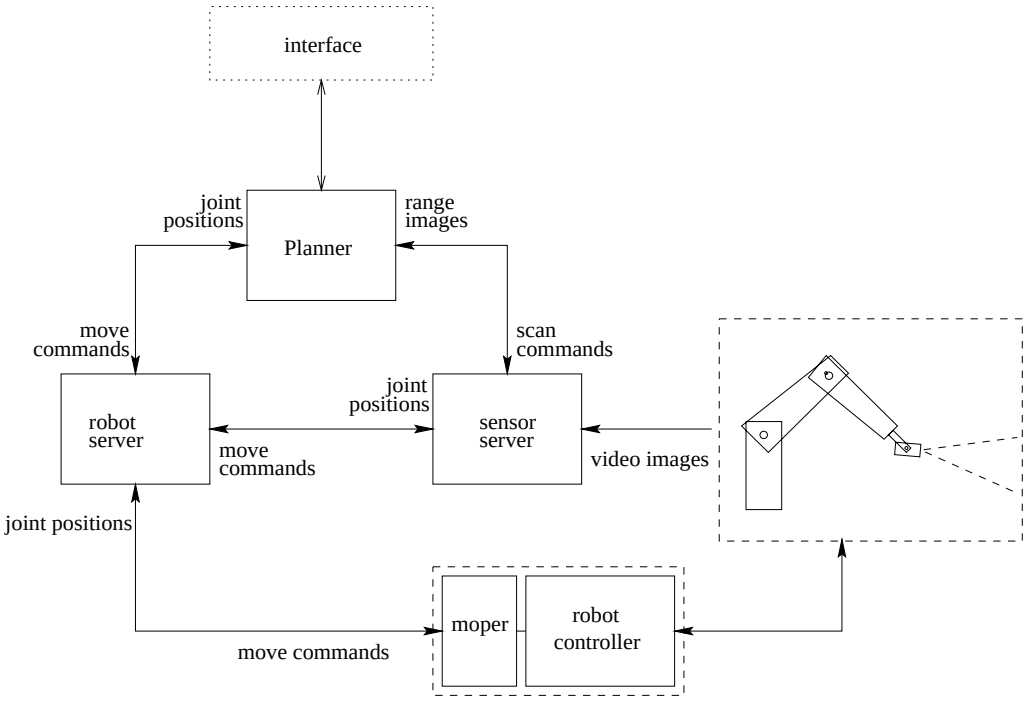


Figure 6.1: The sensor-based testbed system configuration and data flow.

The Robot Server

The robot server is essentially a multiple thread robot control program that runs on a host computer, an SGI INDIGO workstation. Its application control interface is RCCL (Robot Control C Library) [24]. It communicates via a parallel port with a program (MOPER) that runs on the Unimation controller (an LSI 11). MOPER dispatches the joint setpoint commands to individual (original Unimation) joint servo controllers. The robot server accepts robot movement commands through a UDP/IP socket and a TCP/IP socket.

The Sensor Server

The sensor server acquires and transmits images when requested. A range image process (also implemented as a COM object) running on the PC analyzes (using triangulation) the laser stripes in the images. Combining the joint 6 value, it outputs a range image, which is an array of points in the physical 3-dimensional workspace. The server communicates with other processes on the net with TCP/IP protocols through two sockets. The first socket accepts commands from the client process, and transmits the range images to the client. These commands include the specification of field of view, interpolation (between range data points) on/off, etc. The second socket communicates with the robot server. The scanning mechanism is started and synchronized through this socket.

Planner

The planner is the client of both the robot server and the sensor server. It could be launched at any computer with internet access. For now, the planner runs on the PC (a Pentium II 450), the same machine that the sensor server runs on.

Data flow

There are several major data flows in the system: (i) the planner sends robot movement commands to the robot server, and receives the current robot configuration (joint position) data from the robot server; (ii) the planner sends scan commands (along with the scanning configuration of the robot) to the sensor server which, in turn, generates and sends movement commands to the robot server; and (iii) the sensor server returns range images along with joint positions (received from the robot server) back to the planner.

6.2 Determining Server Type

Our key objective is to provide a reliable test-bed for sensor based robot motion planning research. A client-server type of structure is a convenient way to provide a friendly interface to the planning programs. Now we study the features of the servers mentioned in the last section and determine the type (connection oriented/connectionless) for each server in this section.

Response time is an important factor. An ideal system for senor-based MP would include a real time network. The responses of some system components should be as prompt as possible and must occur within a hard time constraint. For example, sensor software requires robot controller to respond within several milliseconds and the time delay should be predictable. However, given current available technologies and other limitations, a dedicated switched hub is a good compromise among all the factors. This switched hub screens out the data traffic, which is irrelevant to this system. The internal communication between the computers that those servers run on is 10 Mb/s.

6.2.1 Connection-oriented vs Connectionless Server:

The two client-server interaction styles correspond directly to the two major transport protocols that TCP/IP and UDP/IP protocol suites supply. If the client and server communication uses TCP protocol, the interaction is connection oriented. Connection oriented communication establishes a link before real communication starts. This link will last until the entire communication finishes. If the communication uses UDP protocol, the interaction is connectionless. Connectionless servers provide faster responses compared with connection-oriented servers. They eliminate the computational overhead and delay for the establishment a TCP connection.

An experiment was conducted to test the suitability of different types of servers. We tested the round trip time delays of a set of commands, i.e. the lapse between the time a command is sent and the time it is acknowledged. Table 6.1 shows an average of 10000 time delays. The numbers in the brackets show the standard deviation. The UDP server responds 3 - 8 times faster than the TCP one. When a UDP server tries to

communicate over the network, about 1% of trials result in failure (either commands or acknowledgments are missed). From the experimental result, we conclude that UDP transmission within a hub provides the least (and almost fixed) time delay for the communication between two components. Note that the control cycle of PUMA robot is 20ms. This delay (1.4ms) is tolerable for the robot control in our system. Based on this experiment, the server types are determined as follows:

	TCP	UDP
same hub	5.2(4.7)ms	1.4(0.03)ms
over net	65.3(344.2)ms	7.0(1.7) ms (1% failure)

Table 6.1: Round trip time used for robot server in different situations.

Robot Server: As we said, the response time is crucial for our system. A connectionless server is a naturally good choice for the robot server. In fact, this type of time-variant (e.g. current robot joint positions when it is moving) information, which is only valid at the time when the data is generated, is particularly suitable for UDP transmission (a connectionless server).

Although connectionless servers have some disadvantages (e.g. lack of a higher level error control mechanism) over connection-oriented servers, most of the disadvantages can be compensated in our system. (1) The computers in this system are connected to the same switched hub. Packets are very seldom or never dropped, nor delivered out of order, nor duplicated. Those higher level error control mechanisms in TCP (connection-oriented) are not necessary. (2) The robot server in this system is unlikely to be distributed to remote computers.

Sensor Server: The sensor we have in this system is a range sensor. The data being sent out from sensor server has the following properties: (1) It is a massive amount of detailed information of the environment. One range image with 256×256 resolution

requires at least 190KB. (2) Range images are taken every 1-2 minutes and the time used for taking a range image is long (about 30 seconds). The transmission time is not as critical as in the case of the robot server (the latter responds within milliseconds). (3) Every part of the data is crucial for the entire range image. Therefore, it is reasonable to choose the TCP protocol for range image transmission, and hence a connection oriented server.

6.3 Multiple Thread Implementation of Planner

Sensor based motion planning is, by its nature, an on-line process. Although client-server programming implies parallelization of clients and servers, the planner – the most CPU-intensive part– is sequentially executed. This section discusses the parallelization of the planner in order to utilize the CPU resource effectively in one computer.

In each iteration, there are two different phases, namely planning the phase and the executing phase (physical motion of the robot). In the planning phase, the robot plans for the next step (a sequence of movement or scan). The planner needs intensive computation in this phase, including for updating physical space, roadmap expansion and etc. On the other hand, in the executing phase, the robot basically follows a given path and does not require intensive computing. See Figure 6.2 for the flow of this process, and Table 7.1 for the average time used for each part.

The executing phase is ideal for background data processing in each plan-scan-executing cycle. Therefore, the planner should be clearly designed with two parts. The first part only processes the necessary information for the next step movement. In our system, the planner expands the roadmap in this planning phase. The planner connects only the new white landmarks to the existing roadmap, and does not try to reach the goal by $reachGoal()$ function. The executing phase should include the second part, which includes all the rest of the non-crucial tasks. This part of the task runs in the background when the robot moves along the path determined by the crucial information process phase. The executing phase includes connecting all pairs of the white landmarks which are not connected, and connecting the goal to $\mathcal{R}(l_0)$.

When $reachGoal()$ returns successfully, the background thread notifies the foreground thread. The foreground thread searches the roadmap again and re-plans a new path to the goal.

The multi-thread implementation does not change the flow of the program. It saves time for the work processed in the background thread by parallelizing the robot movement and computation. One consequence of this parallelization is that when function $reachGoal()$ finds a path to the goal, the robot may have already moved to a new position, and therefore needs to move back.

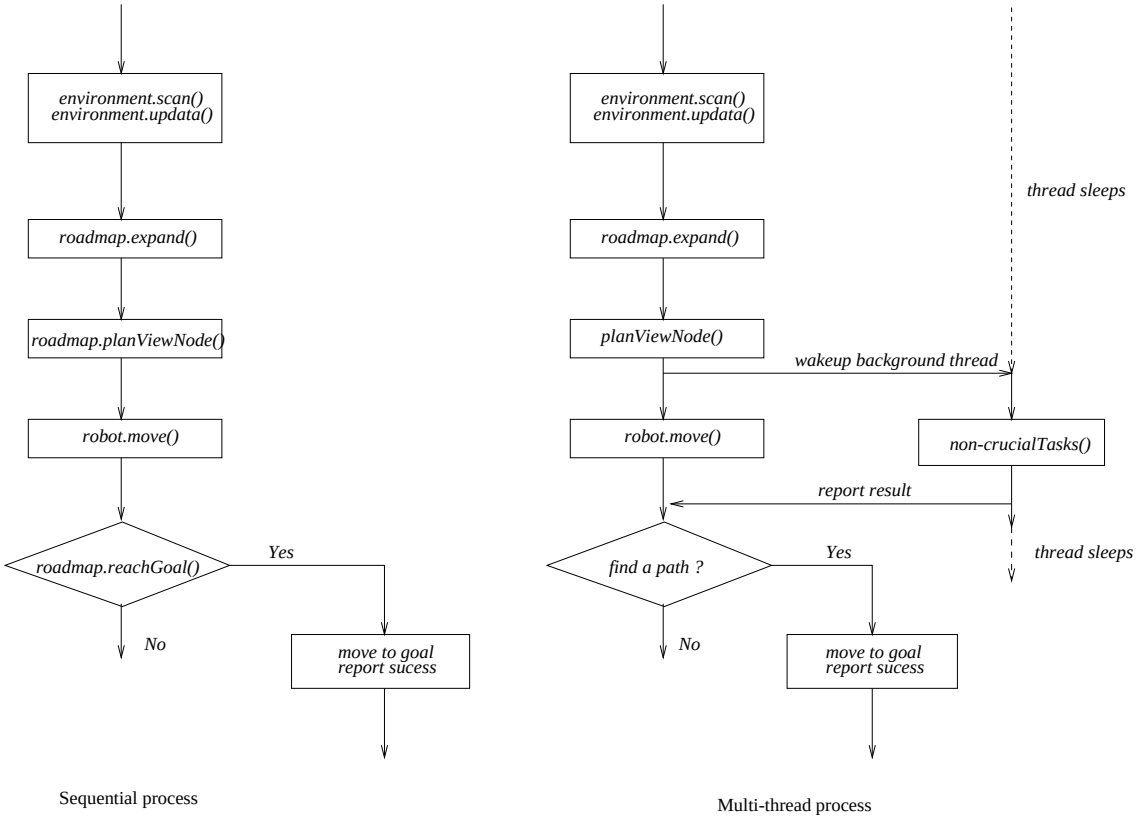


Figure 6.2: Flow chart of the single thread process and the multiple thread process implementation of the overall planner.

Chapter 7

Experimental Results

This chapter presents the experimental results for SBIC-PRM running on the real testbed as described in the last chapter. We first present the average run time breakdown for the major components of SBIC-PRM in Section 7.1. In Section 7.2, we give some examples of the planning process. In the examples, we show the obstacles sensed by the sensor, snapshots of the robot when scanning, and the evolution of the roadmap at each iteration. In Section 7.3, we analyze the effects of the parameters on the behavior of the algorithm. We examine the influence of the weights of the exploratory and goal components, verify the fact that the landmarks are evenly distributed in C-space, and show the relationship between the number of gray landmarks and parameter m, which controls the thickness of $\mathcal{C}_{free+}(m)$, etc.

We assumed that a small cuboid region of $80cm \times 80cm \times 220cm$ surrounding the robot is free. This is $\mathcal{P}^{(0)}_{free}$. Note that with the PUMA in its most "compact" configuration, the smallest cube enclosing the projection of the PUMA on the horizontal plane is $67cm \times 67cm$. The minimum distance from $\mathcal{A}(q_0)$, the robot at the start configuration to the boundary of $\mathcal{P}^{(0)}_{free}$, is therefore only $6.5cm$ on each side. This implies that the assumed initial free region is indeed very tight. Compared with a similar setting presented by Renton [58], in which the volume of the initial free region in their experiments was 2.34 times as the one in our experiments.

7.1 Run Time Breakdown

The run time for one iteration varies from 1 minute to 2 minutes, depending on the number of landmarks generated and on the complexity of the physical space. The algorithm is executed on a Pentium II 450 computer. Table 7.1 shows the breakdown of the average time used in each part of the planner at a stage when there are about 1000 - 6000 landmarks. A typical robot movement in each iteration is composed of 3 - 7 primitive movements (straight line in C-space).

Components	Time
$PhysicalSpace.scan()$	30 - 60 sec
$PhysicalSpace.update()$	12 - 15 sec
$roadmap.expand()$ *(1 - 6k landmarks in total)*	20 - 33 sec
$ViewPlanner.planViewNode()$	5 sec
$robot.move()$ *(3 - 7 primitive movements)*	5 - 25 sec
$roadmap.reachGoal()$	3 - 5 sec
$Total$	69 - 115 sec

Table 7.1: Breakdown of run time for one iteration.

7.2 Examples of Different Tasks

7.2.1 Physical Space Evolution

Recall that the objective function $G(x)$, which is maximized for choosing the next scan, is the sum of an exploratory component and a goal component, with weights w_e and w_g respectively. Figure 7.1 shows a sequence of views of the sensor for a hybrid task with $w_e = 0.25$, and $w_g = 1.5$. In this task, the robot stands vertically upwards at the start configuration, and is inside a narrow opening towards right at the goal

configuration. Since the goal component is heavily weighted, most of the views are directed toward the region (to the right and downwards) that the robot would occupy in its goal configuration.

Figure 7.2 shows the gradual emerging of the internal representation (in particular, the obstacles) of the physical space built by the robot during the planning process for the task in Figure 7.1. The sensor mostly scans the physical region that the robot sweeps along a path to the goal. Therefore, the obstacles are in the same direction.

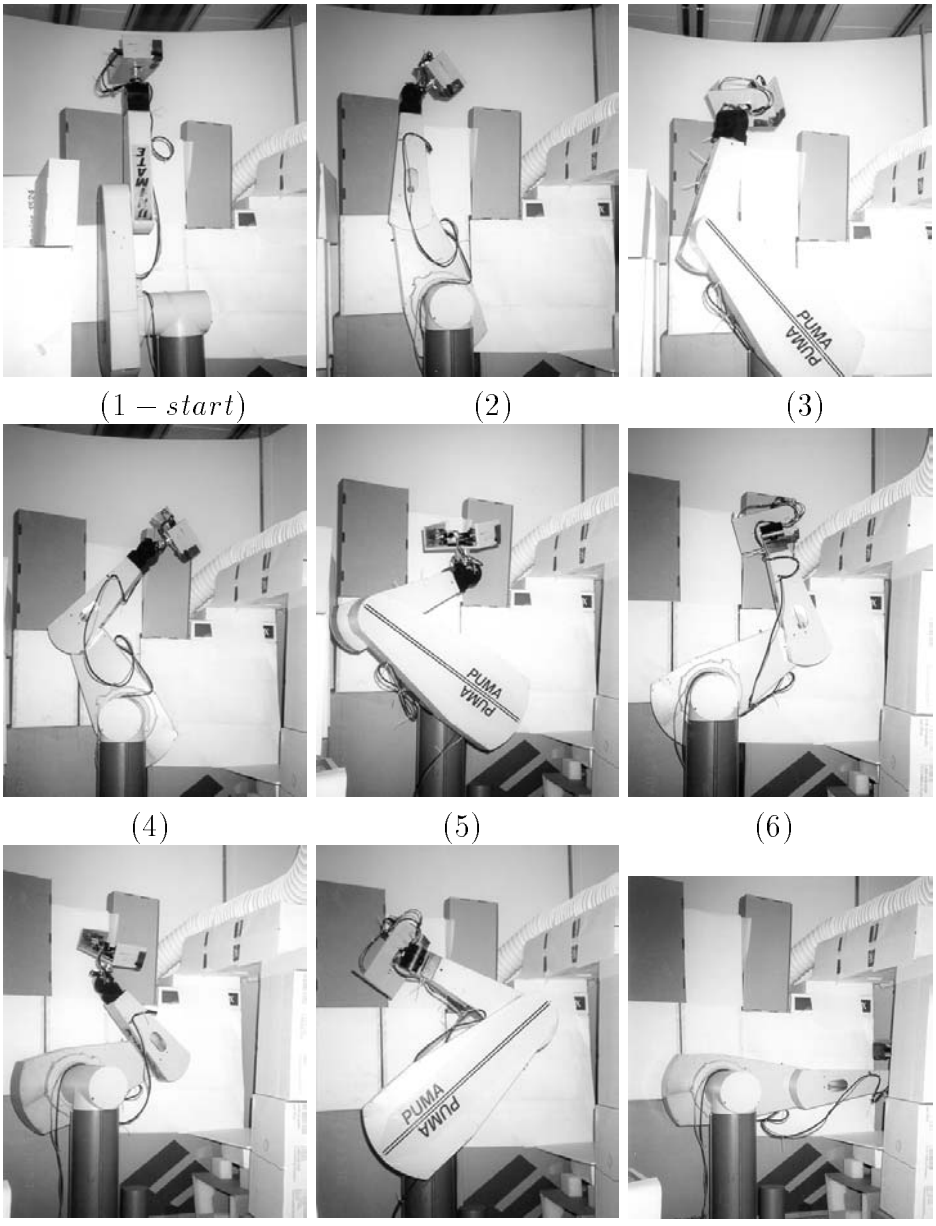
Figure 7.3 shows the views of the sensor in a pure exploration task, $w_g = 0, w_e = 1.0$, as a comparison. The views in this example are not concentrated towards the goal direction, as were the ones in the previous example. Figure 7.4 shows the gradual emerging of obstacles from the unknown physical space in the same example. In contrast to Figure 7.2, the obstacles are scattered around the scene.

7.2.2 Roadmap Evolution

Figures 7.5 and 7.6 show the evolution of two roadmaps (projected on the first three $dofs$) as the planning process proceeds. For clarity, only white landmarks are shown in the figures. An edge between two white landmarks is shown by a straight line between them.

Figure 7.5 shows a hybrid task with small exploratory component ($w_e = 0.25$, $w_g = 1.5$, as shown in Figure 7.2). The small cube below the roadmap is the goal configuration. Note that the robot has to pass through a very narrow (and quite complex) channel in C-space in order to reach the goal. The roadmap grows mostly around a path to the goal.

Figure 7.6 is the roadmap for an exploration task. In this task, w_g is set to zero. Compared with Figure 7.5, the roadmap grows at a higher speed, and occupies a larger part of $\mathcal{C}_{free}$.

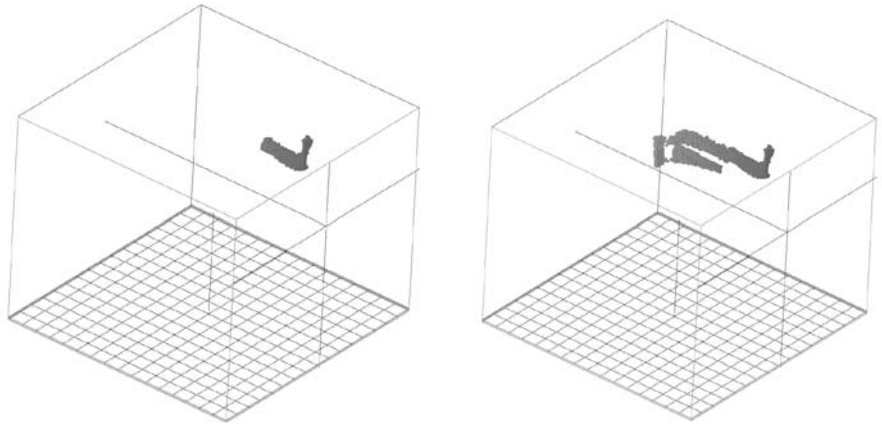


$(1-start)$ (2) (3)

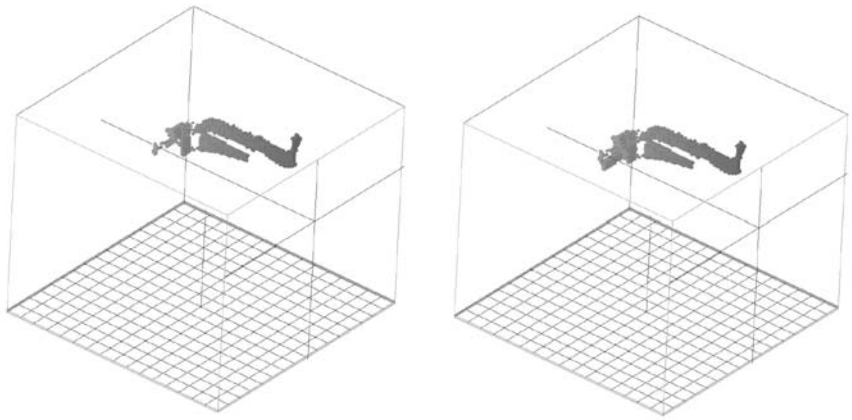
(4) (5) (6)

(7) (8) *goal*

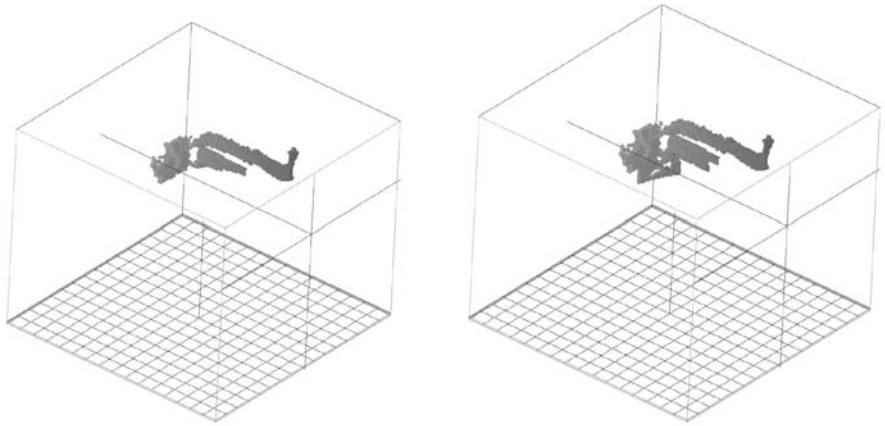
Figure 7.1: Some of the robot scans in a hybrid task with a smaller exploratory component weight $w_e = 0.25$, and larger goal component weight $w_g = 1.5$.



$\mathcal{P}_{obs}^{(1)}$ $\mathcal{P}_{obs}^{(2)}$



$\mathcal{P}_{obs}^{(3)}$ $\mathcal{P}_{obs}^{(4)}$



$\mathcal{P}_{obs}^{(5)}$ $\mathcal{P}_{obs}^{(6)}$

Figure 7.2: Obstacles found by the planner at different stages of a hybrid task with a smaller weight $w_e = 0.25$, and larger $w_g = 1.5$ (continued on next page).

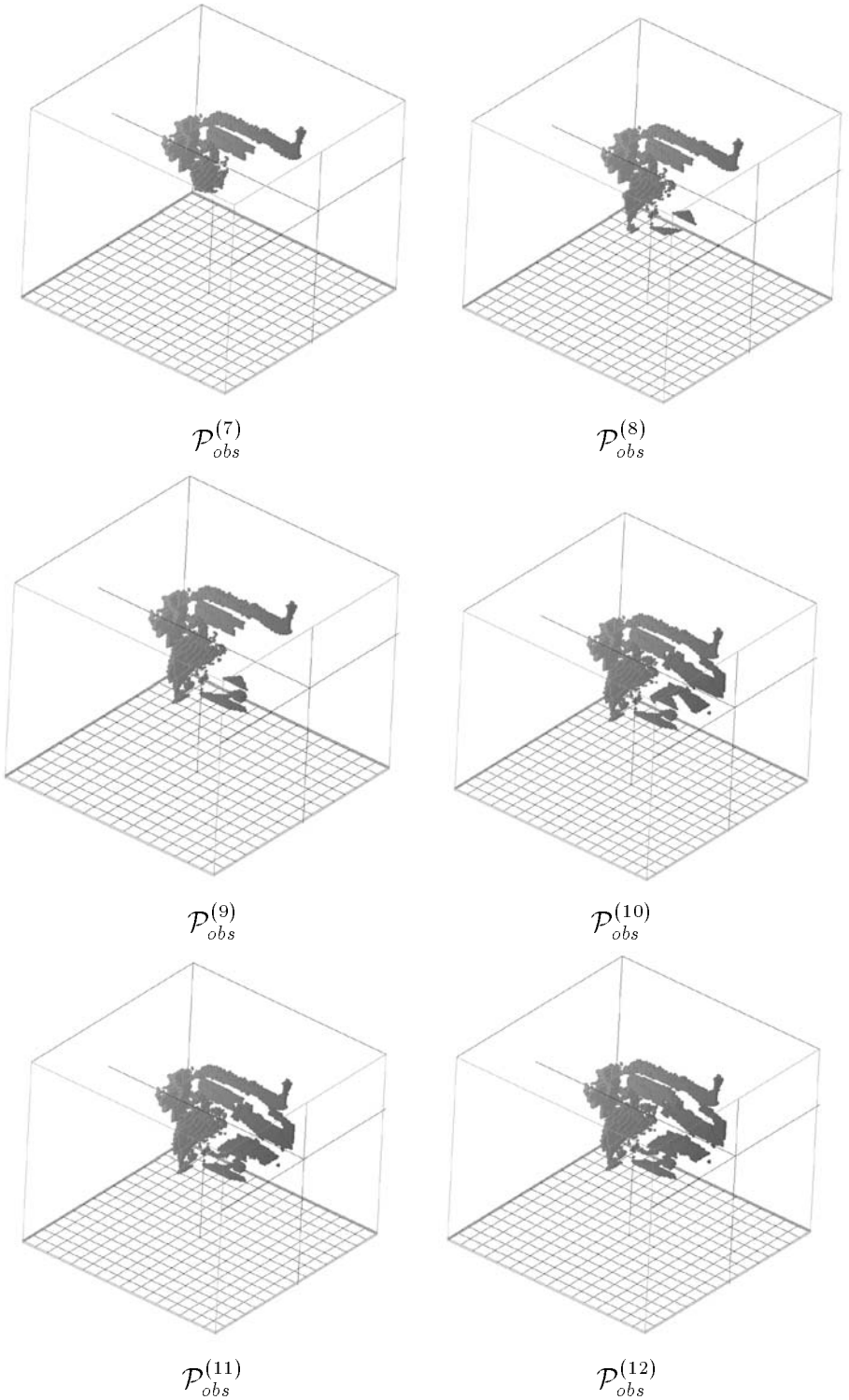
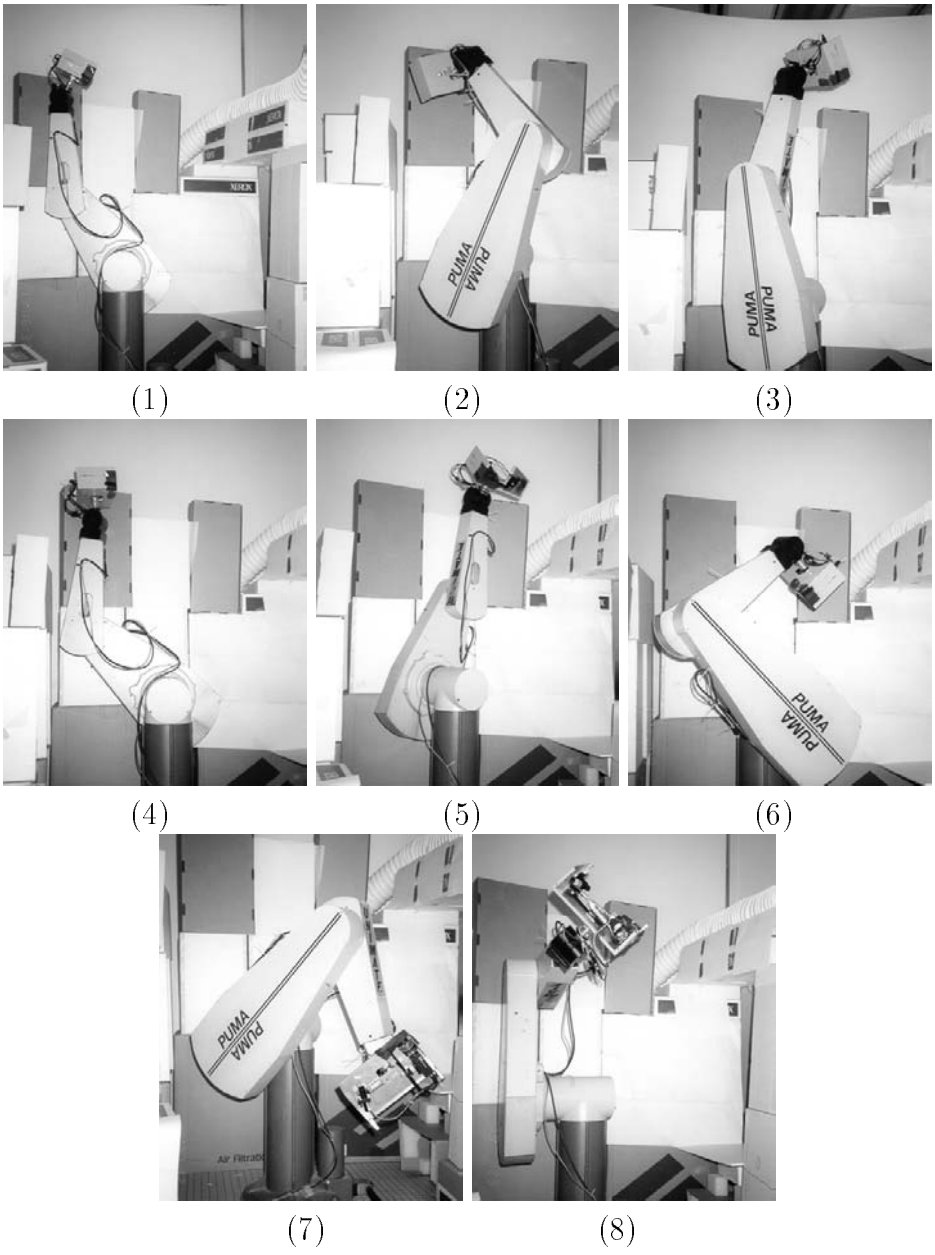


Figure 7.2: (continued from last page) Obstacles found by the planner at different stages of a hybrid task with a smaller weight $w_e = 0.25$, and larger $w_g = 1.5$.

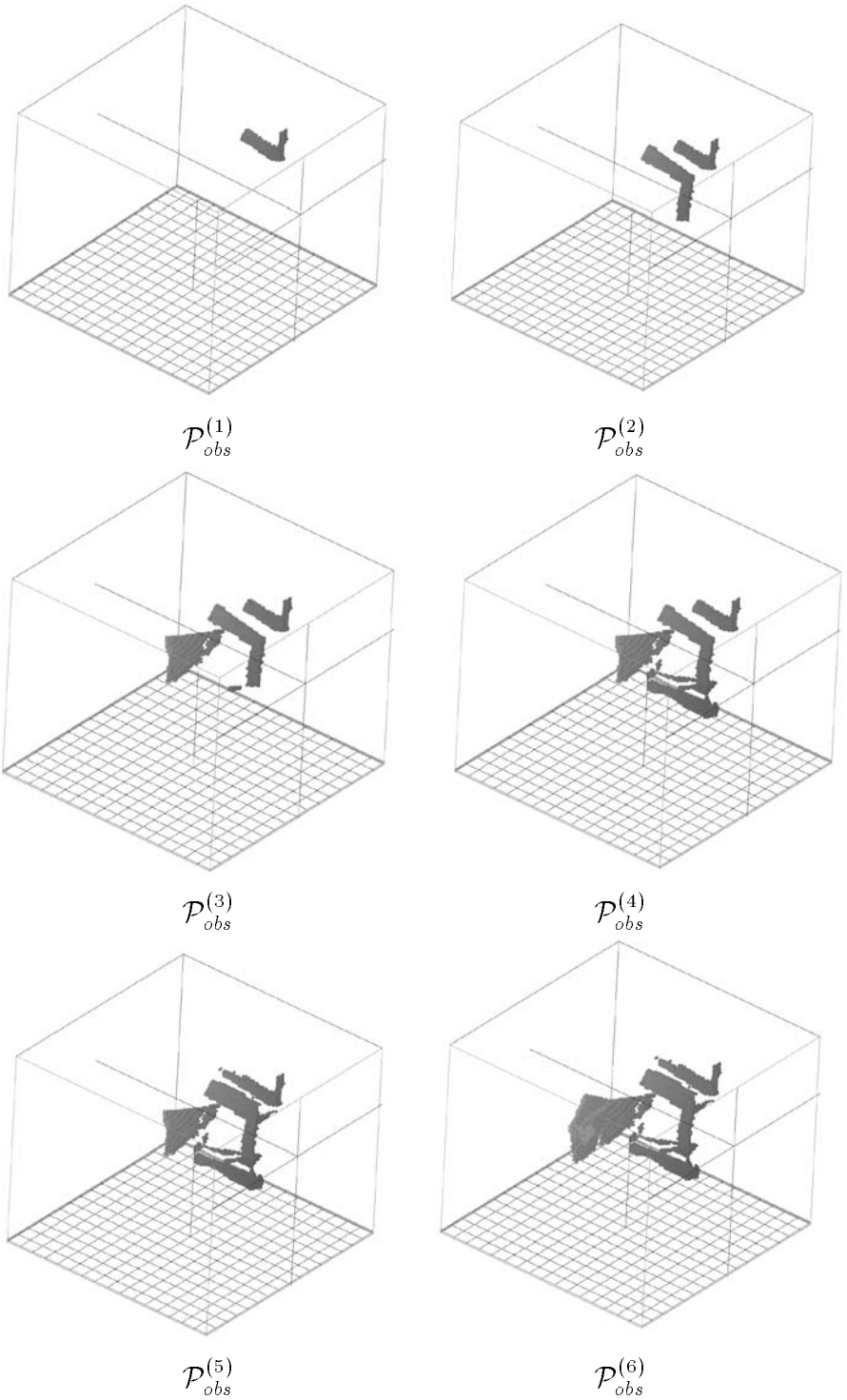


(1) (2) (3)

(4) (5) (6)

(7) (8)

Figure 7.3: Robot scans in an exploration task ($w_g = 0.0$).

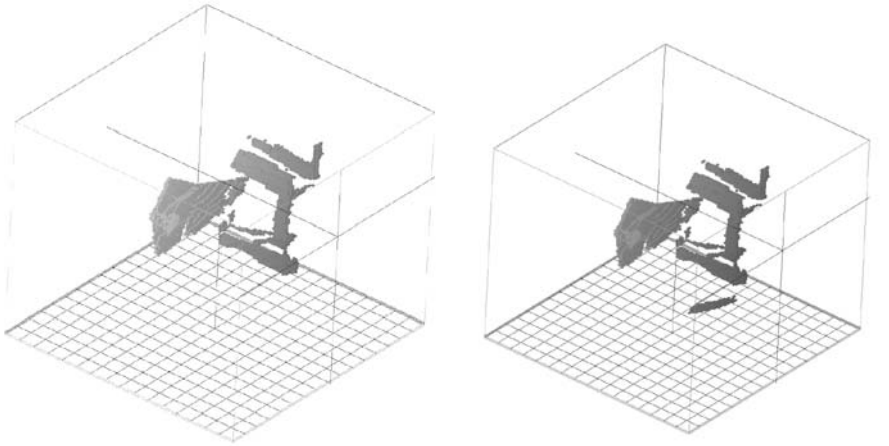


$\mathcal{P}_{obs}^{(1)}$ $\mathcal{P}_{obs}^{(2)}$

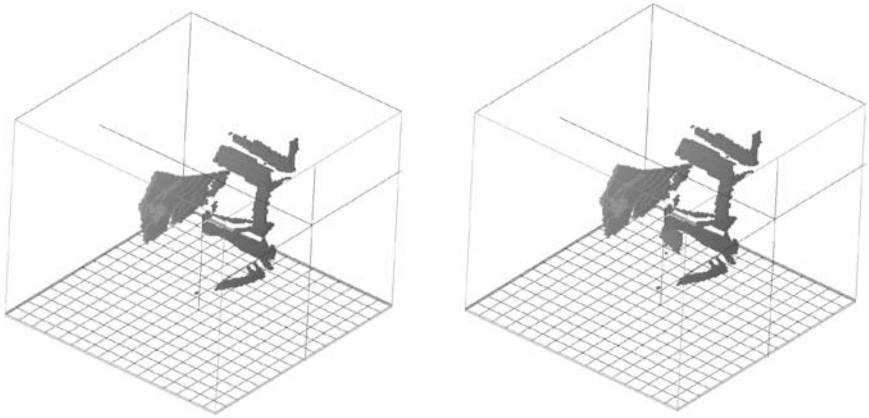
$\mathcal{P}_{obs}^{(3)}$ $\mathcal{P}_{obs}^{(4)}$

$\mathcal{P}_{obs}^{(5)}$ $\mathcal{P}_{obs}^{(6)}$

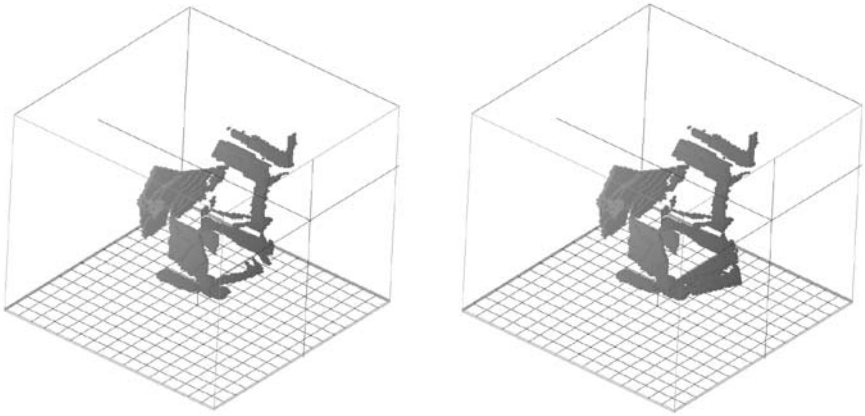
Figure 7.4: Obstacles found by the planner at different stages of an exploration task (continued on next page).



$\mathcal{P}_{obs}^{(7)}$ $\mathcal{P}_{obs}^{(8)}$



$\mathcal{P}_{obs}^{(9)}$ $\mathcal{P}_{obs}^{(10)}$



$\mathcal{P}_{obs}^{(11)}$ $\mathcal{P}_{obs}^{(12)}$

Figure 7.4: (continued from last page) Obstacles found by the planner at different stages of an exploration task.

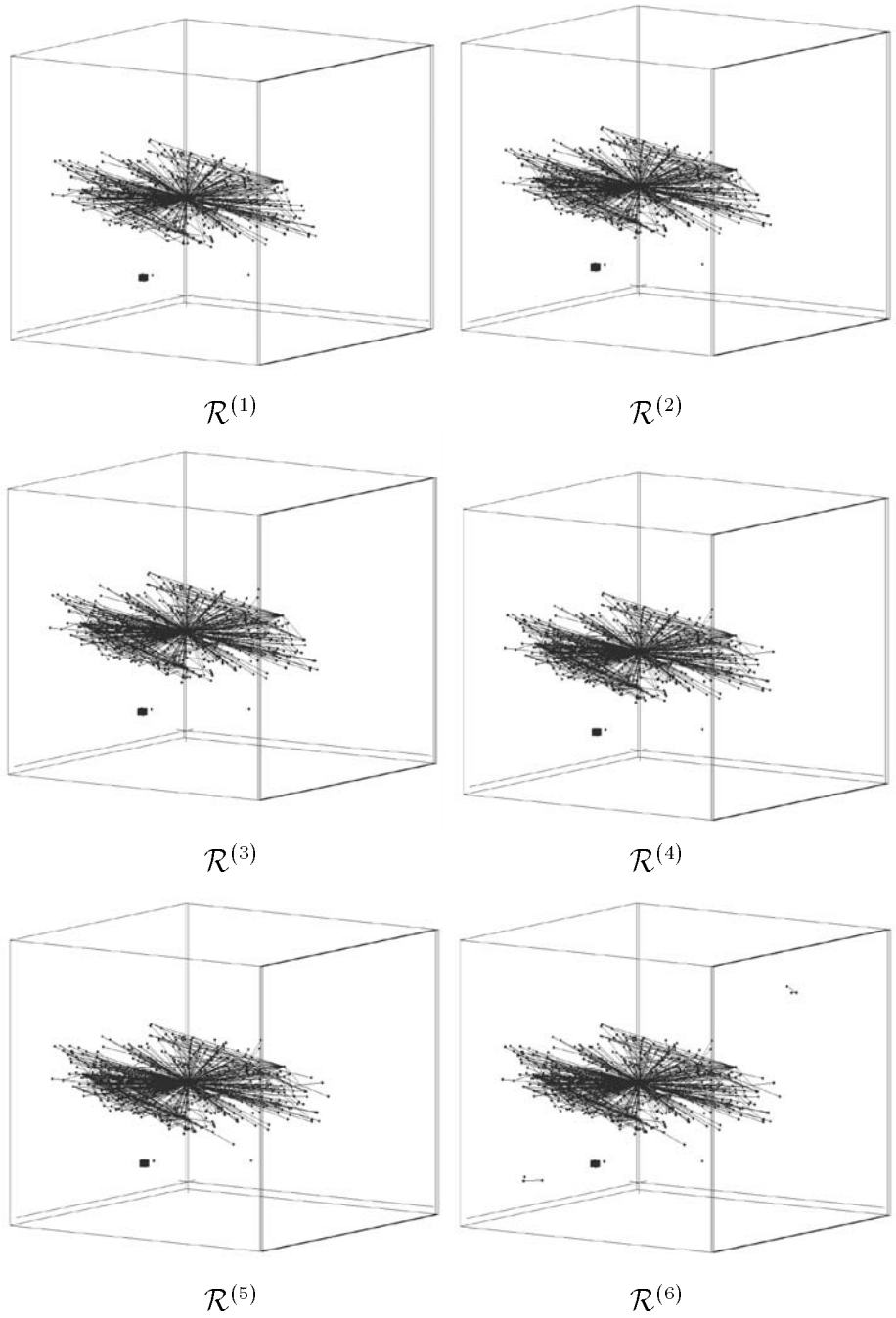


Figure 7.5: Roadmap evolution of the same hybrid task as in Figure 7.2 (continued on next page).

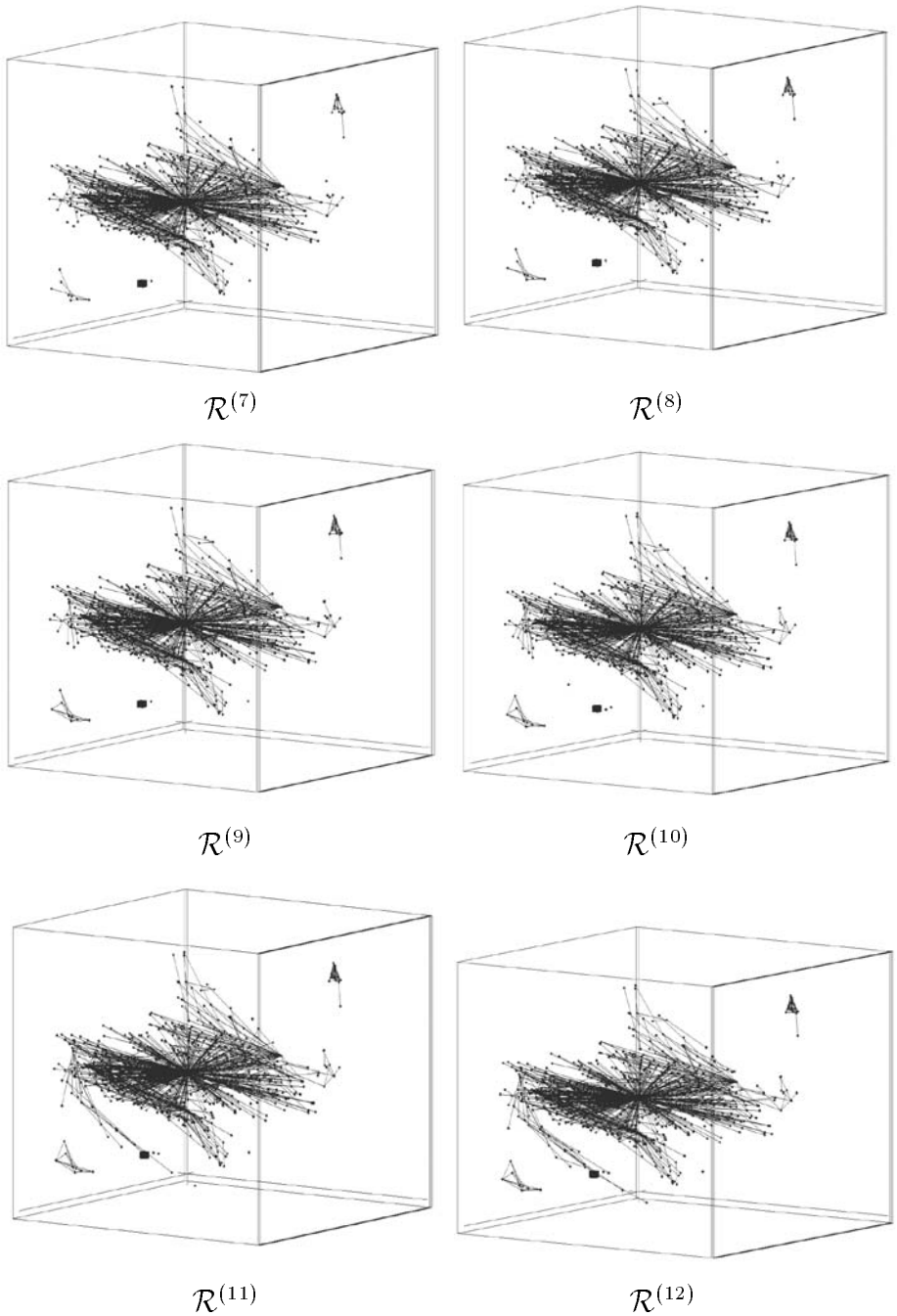


Figure 7.5: (continued from last page) Roadmap evolution of the same hybrid task as in Figure 7.2.

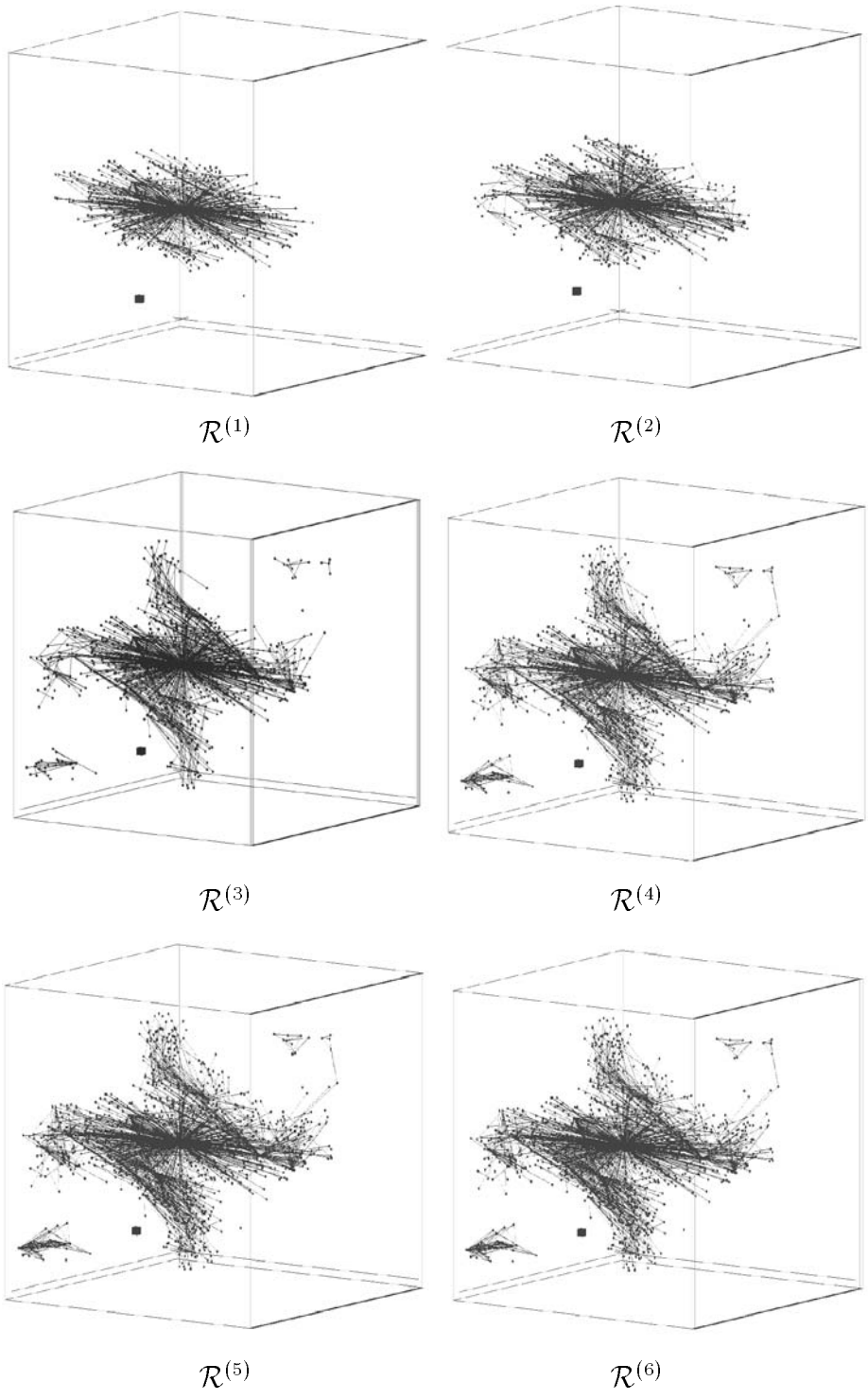


Figure 7.6: Roadmap evolution of the exploration task (continued on next page).

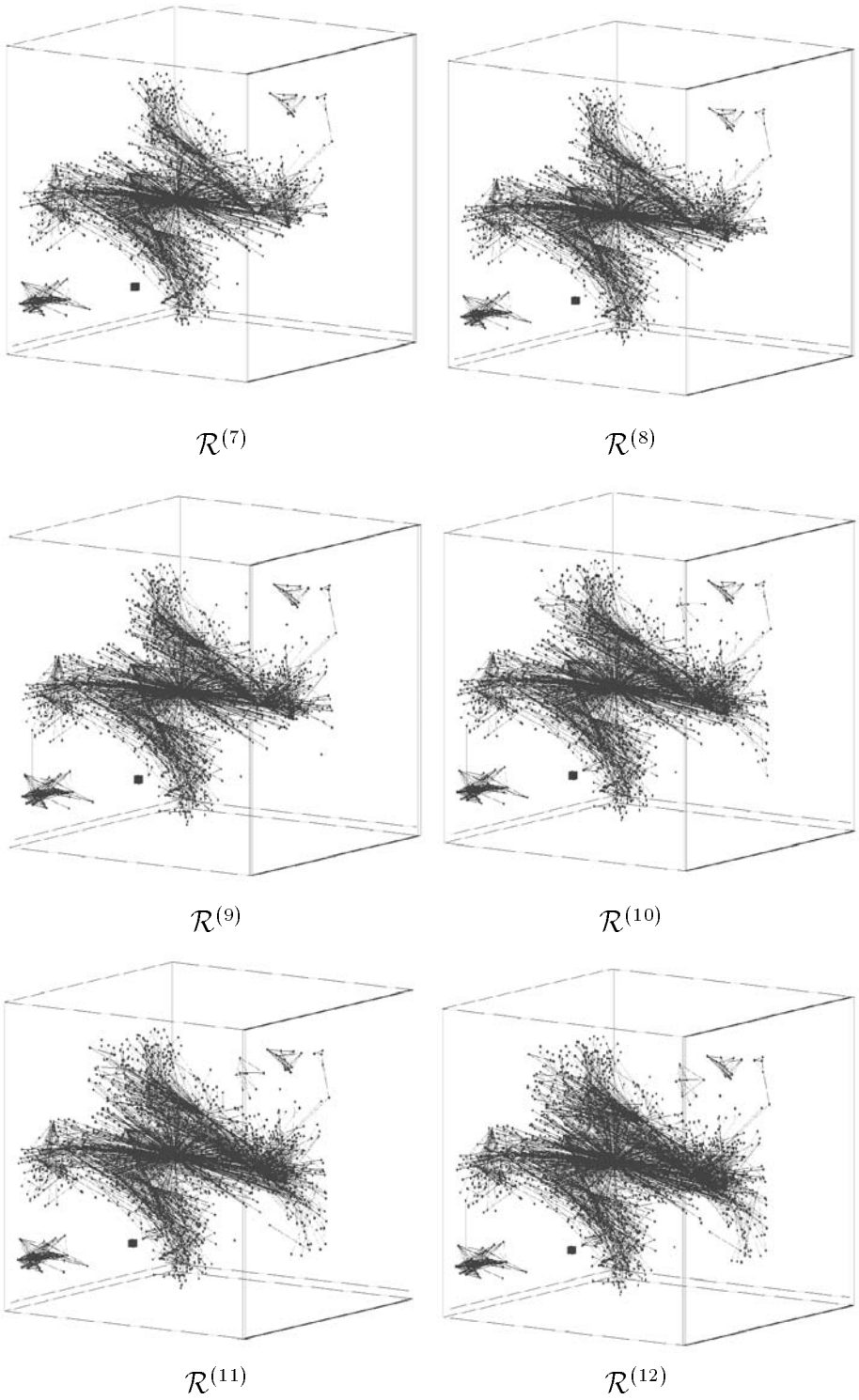


Figure 7.6: (continued from last page) Roadmap evolution of the exploration task.

7.3 Data Analysis

This section gives quantitative analysis of the experimental results with SBIC-PRM. Section 7.3.1 compares the speed of C-space entropy reductions for different experimental settings to verify the effectiveness of IGDF. Section 7.3.2 gives the number of landmarks as the planner iterates over the scan and plan cycles. We then use a statistical approach to verify the uniform distribution of the landmarks in C-space in Section 7.3.3. We observe the change in the number of landmarks vs. parameter m in Section 7.3.4. Finally, the number of scans before reaching the goal is discussed in Section 7.3.5.

Landmark density is an important parameter in all motion planning algorithms based on landmarks. The probability of connecting any two configurations for a given density of landmarks, has been addressed very well in [1, 35]. We will not discuss the landmark density in this thesis.

7.3.1 C-space Entropy

As we mentioned in Chapter 3, C-space entropy $H(\mathcal{C})$ represents the ambiguity of the entire C-space to the robot. Therefore, C-space entropy should be reduced as fast as possible. The rate of C-space entropy reduction represents a measure for the effectiveness of C-space exploration.

In the experiment in this sub-section, we show the rates of C-space entropy reduction. At each iteration, $\widetilde{H}(\mathcal{C})$ is calculated from Equation 3.3 at randomly selected unknown C-space configurations with the same density as the landmarks.

The curves in Figure 7.7 show that $\widetilde{H}(\mathcal{C})$ decreases as the sensor scans the physical space. The horizontal axis is the number of scans. The vertical axis is the C-space entropy. The curve with square markers shows $\widetilde{H}(\mathcal{C})$ as a function of the number of scans, when the exploratory component weight w_e is zero. In this case, all the scans are determined by the goal component, $G_g(x)$, of the objective function. The curve with triangular markers is $\widetilde{H}(\mathcal{C})$ when w_e is set to 0.25. The curve with cross markers represents $\widetilde{H}(\mathcal{C})$ when $w_e = 0.50$. As the exploration weight becomes larger, the slope of $\widetilde{H}(\mathcal{C})$ is steeper. The lowest curve (with asterisk) is the $\widetilde{H}(\mathcal{C})$ curve when w_g is set

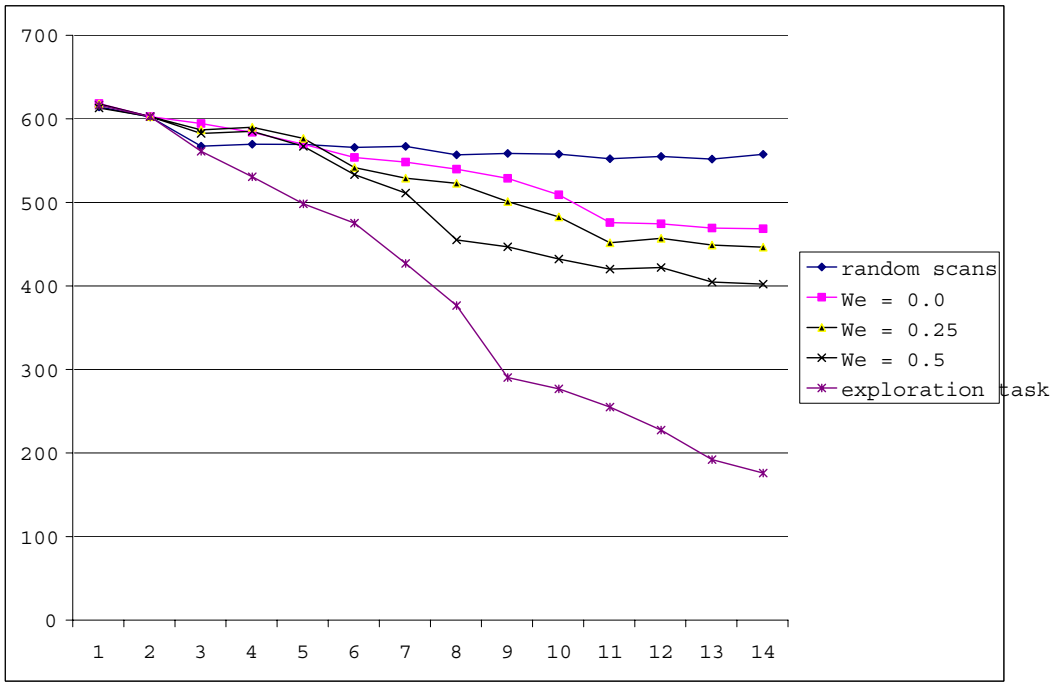


Figure 7.7: C-space entropy $\widetilde{H}(\mathcal{C})$ decreases as the sensor scans. The decreasing speed changes with the exploration weight w_e.

to zero. This means that all the scans are dedicated to exploration. The task is a pure exploration task.

In order to show the effectiveness of the sensing scheme, we also experimented when the scanner was directed to a random direction from a randomly chosen landmark. This curve is shown with diamond markers. From the figure, we clearly see that, in terms of C-space entropy reduction, the views determined by the objective function in SBIC-PRM are significantly better than the random scans.

From this experiment, we conclude that information gain density function defined in Chapter 3 provides an approach of determining efficient scanning regions. Scanning those regions reduces C-space entropy quickly. From Figure 7.7, we can clearly see that w_e, the exploration weight, affects the speed of C-space entropy reduction.

7.3.2 Effect of Exploration Weight on Number of Landmarks

We now analyze the effect of exploration weight parameter, w_e, on the number of landmarks. In this experiment, we let $w_e = 0.25, 0.50, 1.00$ respectively for three groups of experiments, with the same start and goal configurations. In each group, the algorithm, SBIC-PRM, is run 10 times to reduce the randomness of the experimental results and average results are reported.

As we analyzed, the exploration weight parameter w_e represents the effort that the algorithm devotes to exploring unknown C-space. Such effort aims at changing unknown C-space to known C-space, either obstacle or free space. Since our C-space is represented by landmarks, the effect of such effort in SBIC-PRM is reflected by changing gray landmarks to white/black landmarks. Because of the random approach in generating landmarks, the number of landmarks of different colors (white, black and gray) would be roughly proportional to the volume of C-space of the corresponding status (free, obstacle or unknown). Therefore, parameter w_e should affect the number of landmarks. The larger w_e becomes, the more black and white landmarks (in total) one would expect.

This trend is clearly shown in the experimental result (see Figure 7.8). The curve with triangular markers shows the total numbers of black/white landmarks vs. the

number of iterations, when $w_e = 0.25$. The curve with square markers is the number of these landmarks when w_e is set to 0.50. The curve with diamond shape markers is the number of these landmarks when w_e is set to 1.00. The number of landmarks with known status increases at a faster rate for greater values of w_e.

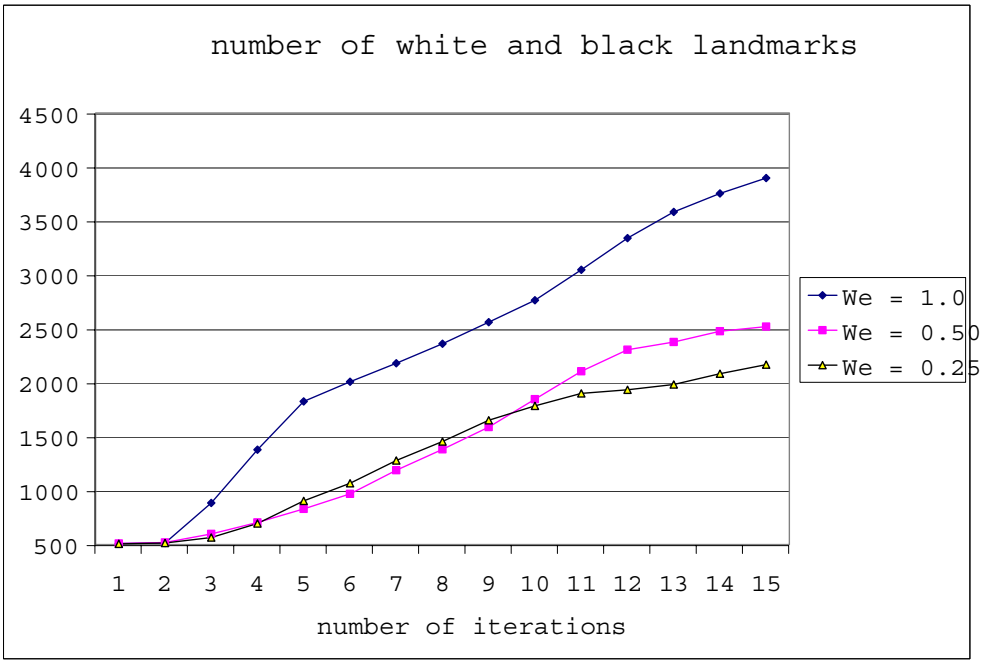


Figure 7.8: The total number of black and white landmarks. Three curves have different slopes because the weights w_e are different.

On the other hand, gray landmarks represent the wave front "boundary" of free C-space. The exploratory component strives to eliminate the gray landmarks. However, since the gray landmarks are on the boundary of $\mathcal{C}_{free}$ (therefore, their number is roughly proportional to the surface of $\mathcal{C}_{free}$ and $\mathcal{C}_{unk}$ boundary), they can not be eliminated completely until the entire C-space is explored. In fact, while some gray landmarks are changed to black/white landmarks, new gray landmarks are added to

the gray landmark set.

Figure 7.9 (b) shows this process. The number of gray landmarks stays at a relatively constant value as the algorithm iterates in contrast to the fact that the number of white and black landmarks increases quickly. Therefore, from this sub-section and from the last sub-section, we conclude that maximizing the information gain density function is an effective way of C-space exploration.

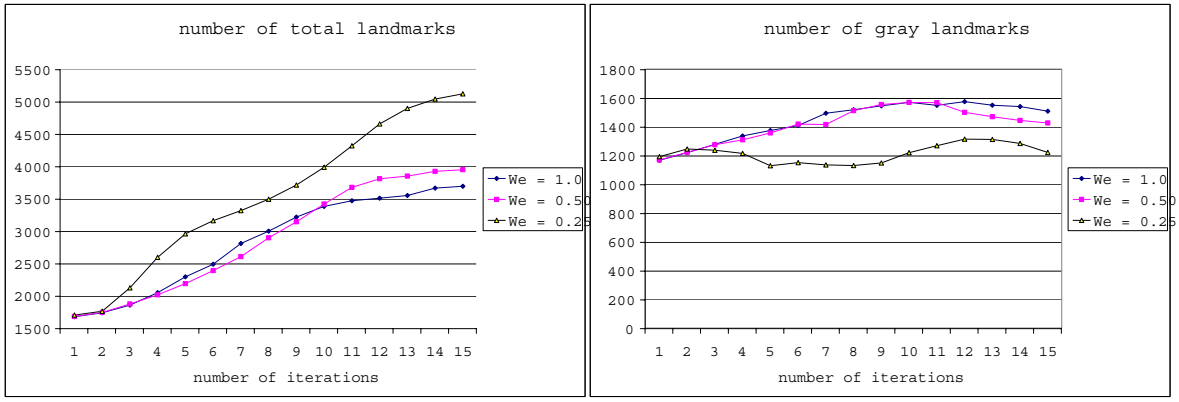
Figure 7.9 shows the total landmarks and breakdown of each category at different iterations with the same setting of w_e parameters. On the right hand side of each sub-plot, the legend shows w_e for each curve. At the 0th iteration, about 1100 gray landmarks are generated around the $\mathcal{C}_{free}$ which is represented by a graph of around 510 white landmarks. The reason why there are many gray landmarks is that the initial $\mathcal{C}_{free}^{(0)}$ is a flat region in the C-space. Therefore, the surface of $\mathcal{C}_{free}/\mathcal{C}_{unk}$ boundary is large compared with the volume of $\mathcal{C}_{free}^{(0)}$. Since the algorithm has no information about the obstacles, there are no black landmarks at the 0th iteration.

7.3.3 Uniform Distribution of Landmarks

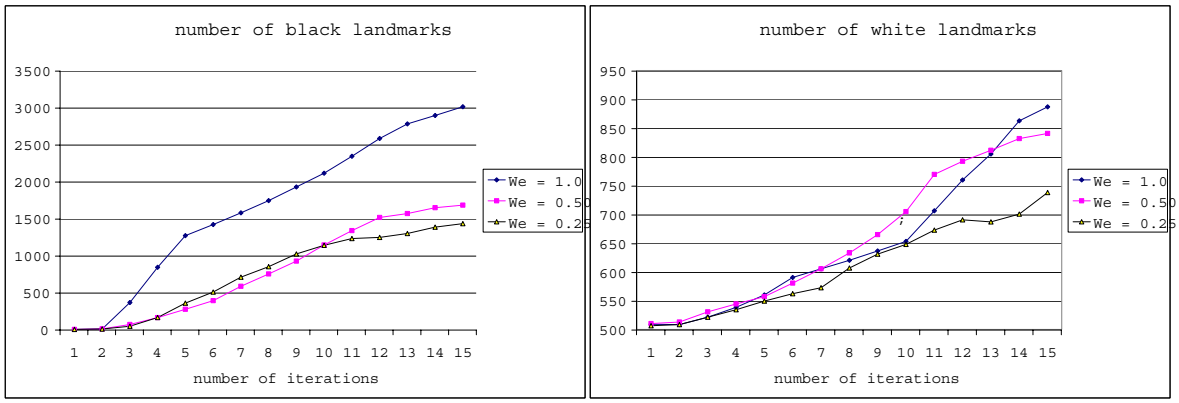
The algorithm needs to maintain a uniform distribution of the landmarks in C-space. As pointed in [38], the distribution could easily degenerate in the planning process. Our approach generates new landmarks only in the new free region ($\Delta\mathcal{C}_{free}$) and the (expanding) boundary (wavefront) of $\mathcal{C}_{free}$. This leads to a uniform distribution of landmarks during the entire planning process.

To show the uniform distribution, we need a measure of "uniformity" of distribution. We use a standard statistical measure [10] (often used in ecology to characterize distribution of population) to verify uniform distribution of the landmarks.

Each landmark has a minimum distance associated with it, the distance to the closest landmark, $d_{min}(l) = \min_{l' \neq l} \{\|q_l - q_{l'}\|\}$, where $\|\cdot\|$ is a distance from which uniformity is defined. Here, we use Euclidean distance. We use the average of these minimum distances, $\bar{d}_{min} = \frac{\sum_{l \in \mathcal{L}} d_{min}(l)}{total\ landmarks}$, as a measure of the deviation from uniform distribution [10]. The basic idea behind this is that clustered landmarks result in lower average minimum distances. If the landmarks are randomly distributed, then



(a) (b)



(c) (d)

Figure 7.9: Number of landmarks.

the uniform distribution should have the maximum value of $\bar{d}_{min}$. The value of this measure (most uniformly distributed) is given by $D(\rho, n) = \frac{|\Gamma(n/2+1)|^{1/n}\Gamma(1/n+1)}{\rho^{1/n}\pi^{1/2}}$, where ρ is the density of landmarks, n is the dimension of the C-space, and $\Gamma()$ is the standard gamma function. The approach to verify the uniformness of the distribution assumes there is no boundary effect. Landmarks on the boundary of the explored C-space have a greater chance of having a larger $d_{min}(l)$ because these landmarks have less "neighbor" landmarks than the interior landmarks of the explored C-space do. Therefore, $\bar{d}_{min}$ from real cases is larger than the theoretical value $D(\rho, n)$.

Figure 7.10 shows this measure for a series of tests, plotted as a function of number of iterations. The horizontal line represents the theoretical value for a uniform distribution. This value corresponds to $D(\rho, n) = 0.74$, for $\rho = 0.765$, and $n = 6$.

This suggests that our landmarks are uniformly distributed in C-space. The fact that our average minimum distance is somewhat greater than the theoretical value is due to boundary effects.

7.3.4 Thickness of $\mathcal{C}_{free+}(m)$ Wavefront Region

The wavefront is composed of gray landmarks. Recall that SBIC-PRM uses the gray landmarks to determine the next views. As we discussed in Chapter 4, the thickness of the gray landmark region (and the number of gray landmarks) varies inversely with the fraction m. As m increases, the "thickness" of $\mathcal{C}_{free+}(m)$ decreases and hence the number of gray landmarks decreases. This phenomena can be seen from Figure 7.11. This figure shows the average number of gray landmarks after 7 iterations.

7.3.5 Number of Scans with Respect to Exploration Weight

As a general measure of SBIC-PRM's ability to plan paths in cluttered environments, we tested the number of iterations needed to reach a goal. In this experiment, the robot forearm and upper arm are inside a hole in the wall at the goal configuration. According to [29], this type of task is usually classified as of moderate or greater complexity for model based motion planning. Throughout this task, we keep the same settings of the physical environment and vary the exploration weight w_e.

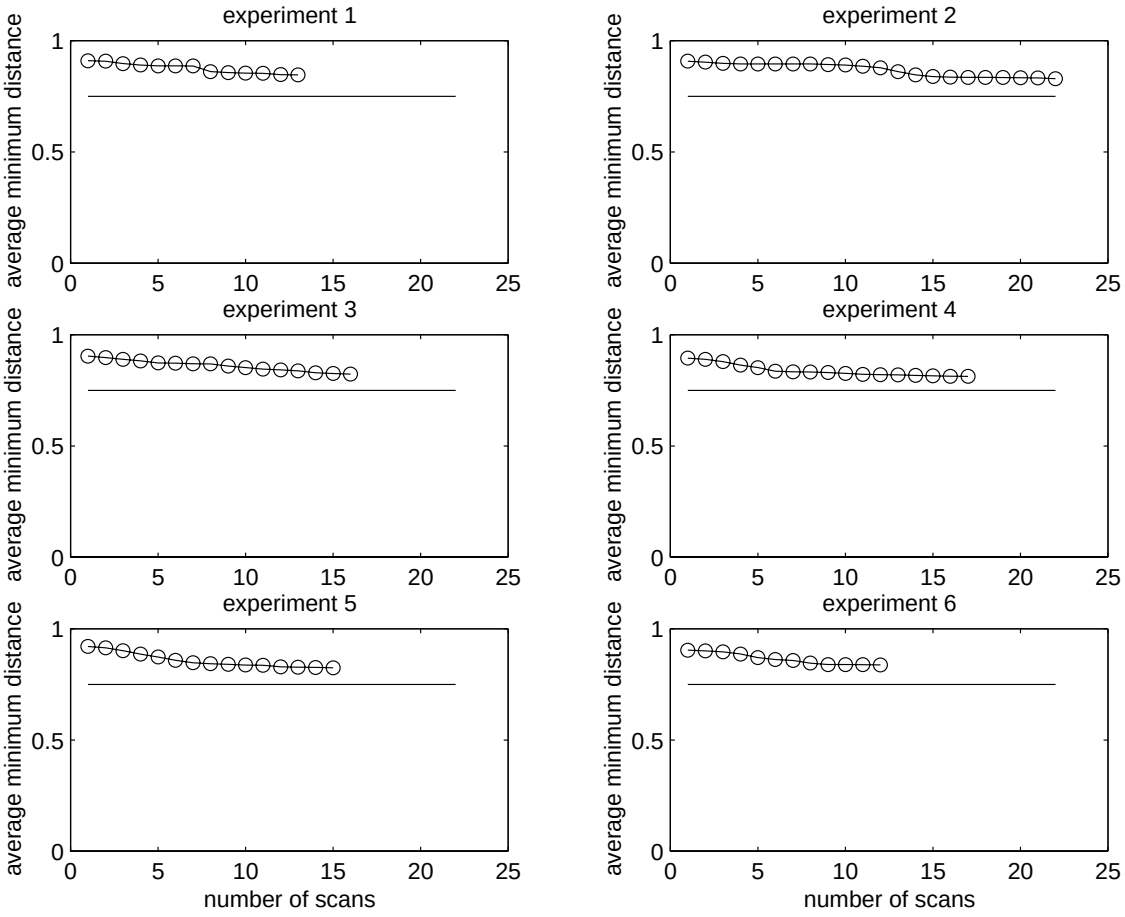


Figure 7.10: Average minimum distance between the landmarks with successive iterations.

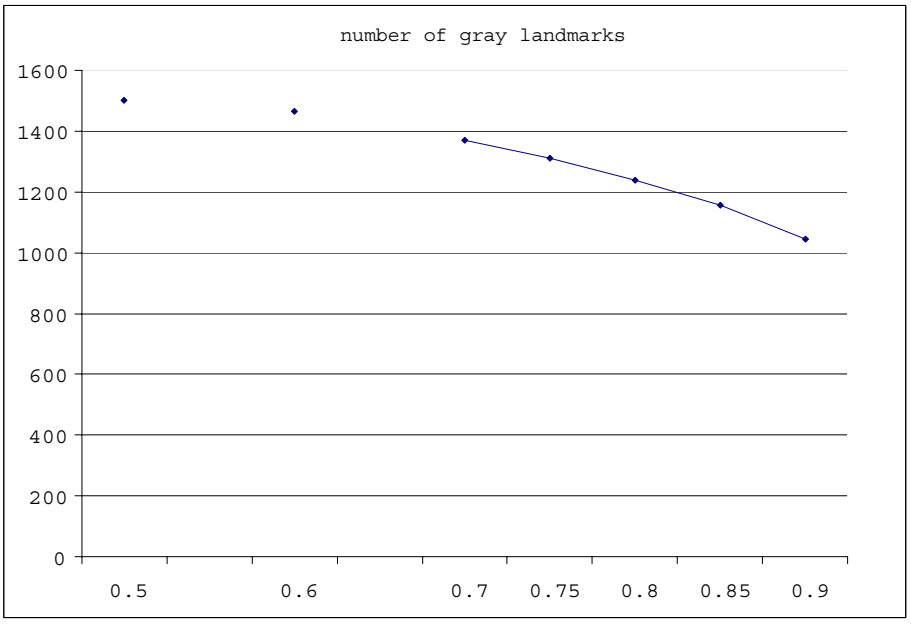


Figure 7.11: Average number of gray landmarks (after 7 iterations) as the m parameter changes.

w_e	average number of scans	failure rate
0.0	12.77	0.10 (2 out of 20)
0.25	14.65	0.14 (3 out of 21)
0.50	16.33	0.09 (1 out of 11)
1.0	18.70	0.09 (1 out of 11)

Table 7.2: Average number of scans before reaching a given goal.

Table 7.2 and Figure 7.12 give the average number of scans before the goal is reached with respect to the exploratory component weight w_e. The failure percentage (without reaching the goal within 30 iterations) is also shown in Table 7.2.

From Table 7.2, we observe, in this experiment, that the heavier exploratory weight increases the number of scans before a given goal is given. The fast speed of C-space exploration is, on average, at a cost of slightly increasing the number of scans to reach the goal. In some situations, however, a larger w_e may lead to fewer scans for reaching a goal. For instance, a larger w_e may help the planner to escape dead ends which were initially believed to be a free path towards the goal.

7.4 Summary of the Experimental Results

From the experimental results of this chapter, we have the following conclusions:

(1) SBIC-PRM is an efficient algorithm for sensor-based motion planning for Eye-in-Hand systems. For a six dof system, it can successfully perform tasks of moderate or greater difficulty within 30 iterations for most of the experiments.

(2) Our information theoretical approach using the novel notation of C-space entropy provides an efficient way of exploring C-space. The MER criterion for choosing scans results in significantly faster exploration than randomly chosen views.

(3) SBIC-PRM maintains uniform distribution of landmarks through out the entire planning process. Roadmap degeneration is avoided.

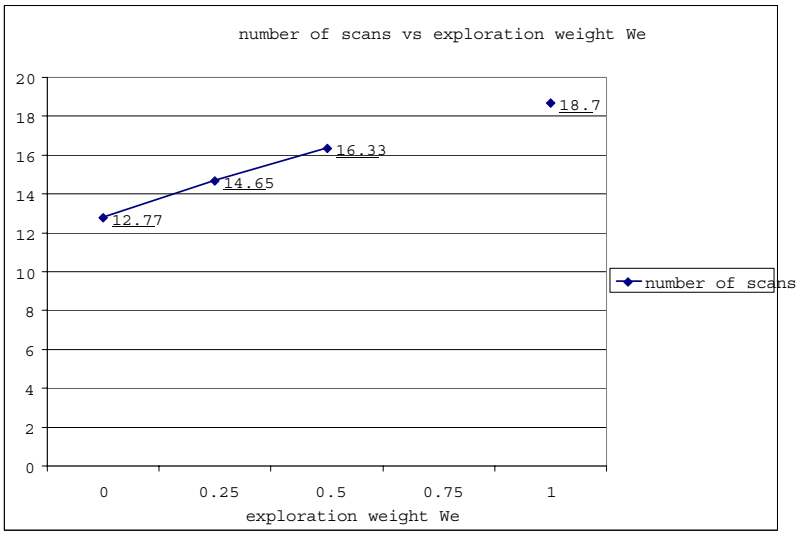


Figure 7.12: Number of scans. The horizontal axis is the exploration weight w_e.

(4) Parameter w_g and w_e provide a nice control mechanism to control the behavior of the algorithm. The C-space exploration rate increases with the weight of the exploratory component w_e.

Chapter 8

Conclusions and Future Work

We presented an implemented approach to sensor-based motion planning (MP) for Eye-in-Hand systems consisting of an articulated arm equipped with an "eye" type range sensor. The approach, Sensor-Based Incremental Construction of Probabilistic Roadmap (SBIC-PRM) incrementally constructs a probabilistic roadmap as the sensor senses new free region in the physical space. The (evolving) roadmap reflects the connectivity of known $\mathcal{C}_{free}$ within which the robot carries out its motion to further sense the physical environment. In order to explore the C-space effectively, a novel view planning approach based on a novel concept of C-space entropy for sensor-based motion planning was implemented.

A series of experiments showed that the SBIC-PRM algorithm works efficiently and is able to find a path in fairly complex unknown environments. View planning, based on maximal entropy reduction (MER) criterion for choosing the next scan, provides an efficient way to explore C-space.

Numerous experiments also showed that the system is a reliable one. We have recently implemented a sensor-based version of ACA (SB-ACA) (similar to the simple planar implementation reported in [2]) on the same eye-in-hand test bed system. The ease of integration of this SB-ACA planner has convincingly shown the convenience of our modular software architecture. We believe that SBIC-PRM and the related work have made the deployment of manipulator arms in unstructured environments one step closer to reality.

8.1 Future Work

This thesis is really a first step toward an "eye" sensor based planning system for robots with non-trivial geometry/kinematics. There is ample scope for future work here, both for algorithmic improvements to SBIC-PRM, and for investigating several broad theoretical issues related to eye-sensor based planning. First we discuss the algorithmic improvements to SBIC-PRM.

- Information gain density function is currently computed under the assumption that a small region is being sensed, and that there is no occlusion. Relaxing these assumptions, i.e., determining the expression for IGDF for a finite sized region (say, a spherical ball of given radius) and IGDF with occlusion constraints would result in more efficient and effective sensing.

- Instead of uniformly choosing landmarks in C-space, more elaborate distributions and schemes, such as OBPRM [4], Hsu's algorithm [27], or Gaussian Distribution [52] could be incorporated. These approaches try to address the well known "narrow passages" in probabilistic approaches by adding more landmarks in difficult regions. This increases the probability that the probabilistic roadmap indeed reflects the connectivity of the underlying free C-space. We have not addressed this problem in the current version of SBIC-PRM.

- To deal with sensor and positioning errors, more elaborate schemes, such as Elfes's [16] probabilistic spatial occupancy and Payeur's [54] probabilistic octree model, can be used instead of our simple strategy of simply shrinking the free space by a certain secure distance.

- It would be interesting to explore other approaches such as ACA [6] or RRT [42] and compare relative advantages/disadvantages vs. PRM.

- One of the factors to determine a view node in Kruse's algorithm is the distance from the current landmark to the view node [38], R_{dist}. This factor becomes important when the cost of moving the robot is high. Combining this factor into our algorithm will also be an interesting topic.

Now we discuss the broad theoretical open issues related to eye-sensor-based planning for robots with non-trivial geometry/kinematics.

- The necessary and sufficient criteria for s-completeness of SBIC-PRM remains an open problem. Our conjecture, that SBIC-PRM is probabilistically s-complete, needs to be proven. A key issue here would be to come up with necessary and sufficient conditions for a "complete" view planner.

- There are implications for design as well. For instance, how should a given eye sensor be mounted on the robot so that navigability (or explorability) is "maximized", either for a given class of environments, or for all possible environments.

- Combining an "eye" sensor with "skin" sensors (such as the one used by Lumelsky et al. [47]) is a promising strategy both theoretically and practically, since the two types of sensors are complementary. "Eye" sensors have the ability to sense a larger range. Skin sensors could give robots the ability of s-completeness. Developing a planning algorithm that effectively utilizes and co-ordinates both types of sensors is a very interesting problem.

Appendix A

Various Parameters in SBIC-PRM

parameter	description	default value
w_e	weight of exploratory component	
w_g	weight of goal component	
	octree resolution	1.5 cm
	information gain density function resolution	same as above
ρ	landmark density	0.765
	max iterations	30
m	thickness of $\mathcal{C}_{free+}(m)$	0.85
m_g	greedy component landmark	0.85
	range image resolution	256×256
	robot (end effector) error	1-3 cm
	thickness of obstacle surface in octree	same as octree resolution
$\mathcal{P}_{free}^{(0)}$	initial free region size	$80cm \times 80cm \times 220cm$
λ	the obstacle density	

Appendix B

Program Psudo-code

B.1 Code for Motion Planning

PhysicalSpace::initialization()
{
$\mathcal{P}_{free}^{(0)} \leftarrow$ initial free region
$\mathcal{P}_{obstacle}^{(0)} \leftarrow$ NULL
}

Program B.1: Physical space initialization.

PhysicalSpace::scan()
{
$startScaner()$
$\partial\mathcal{V}_{\mathcal{S}} = receiveRangeImage()$
$getSensorPosition()$
}

Program B.2: Scan physical space.

{
$\Delta\mathcal{P}_{free}^{(i)} = constructOctree(\partial\mathcal{V}_{\mathcal{S}}) - \mathcal{P}_{free}^{(i-1)}$
construct $\Delta\mathcal{P}_{obs}$
$\mathcal{P}_{free}^{(i)} = \mathcal{P}_{free}^{(i-1)} \bigcup \Delta\mathcal{P}_{free}^{(i)}$
$\mathcal{P}_{obs}^{(i)} = \mathcal{P}_{obs}^{(i-1)} \bigcup \Delta\mathcal{P}_{obs}^{(i)}$
}

Program B.3: Update the data structure of the physical space.

Roadmap::initialization()
{
Initialize $\{\mathcal{N}, \mathcal{E}\}$ $\mathcal{L}_g$, $\mathcal{L}_b$
$l_{cur} \leftarrow l_o$
build $\mathcal{R}^{(0)}$ in $\mathcal{C}_{free}^{(0)}$ with PRM
}

Program B.4: Roadmap initialization.

```
Roadmap::expand()
{
Δ𝓛^(i) ← randomly generated landmarks within Δ𝓒^(i)_free and adjacent regions
𝓛_g ← Δ𝓛^(i) ⋃ (𝓛_g)
/*new landmarks & some in 𝓛_g examined*/
∀ l ∈ 𝓛_g,
   if colorOf(l) = WHITE
      removeFromList(l)
      ∀ n ∈ ℛ /*add new white landmarks to roadmap*/
      if local_planner(q_l, q_n) = TRUE
         E ← E ⋃ (l, n)
   else if colorOf(l) = BLACK
      removeFromList(l)
      addToList(l, 𝓛_b)
}
```

Program B.5: Expanding the roadmap.

```
Roadmap::reachGoal()
{
∀ l ∈ ℛ(l_0), and connected(l_cur, l) = TRUE
   if local_planner(q_goal, q_l) = TRUE
      return TRUE
return FALSE
}
```

Program B.6: Try to reach the goal from landmarks.

Roadmap::getExcursionLandmark(x)
{
$\forall\ l \in \mathcal{R}(l_0)$
  if { {x is visible from sensor} &
    ($v_l(x)$ exists) & $x \in$ (sensing range) &
    (*local_planner*$(q_l, q_{v_l}(x))$ = TRUE) }
      addToList(l, tempList)
if *(empty(tempList))* return *NULL*;
else
  $e = \{l : \min_{l \in tempList}(distance(x, center(\mathcal{V}_{\mathcal{S}}(q_{v_l}))))\}$
  return $v_e(x)$
}

Program B.7: Find an excursion landmark.

ViewPlanner::getWhereToScan(G)
{
$x_{max} = \{x : \max_{x \in \mathcal{P}_{unk}} G(x)\}$
if $G(x_{max}) \neq -INFINITY$
  {
  $G(x_{max}) = -INFINITY$ // tag x_{max}
  return x_{max}
  }
else return *NULL*
}

Program B.8: Find the place to be scanned in physical space.

ViewPlanner::planViewNode()
{
/*pre-calculate objective function *G(x)* */
calculateIGDF()
  $G(x) = w_e G_e(x) + w_g G_g(x)$
repeat
  x = *getWhereToScan(G)*
  if x =NULL return NULL /* indicating failure*/
  e = *Roadmap.getExcursionLandmark(x)*
  if $e \neq NULL$ return $v_e(x)$; // end of repeat
}

Program B.9: Plan next view node.

ViewPlanner::calculateIGDF()
{
$\forall x \in$ *physical space*
  $G_e(x) = 0$ /*IGDF initialization*/
$\forall l_g \in \mathcal{L}_g$ /* for each gray landmark */
  /*calculate $\mathcal{A}_{unk}(q_{l_g})$ robot physical position at landmark l_g */
  $\mathcal{A}_{unk}(q_{l_g}) = \mathcal{A}(q_{l_g}) \bigcap \mathcal{P}_{unk}$
  $p = e^{-\lambda\, vol(\mathcal{A}_{unk}(q_{l_g}))}$ /*Equation 3.13*/
  $g_{q_l} = \lambda(\log \frac{p}{1-p} - p \log p - (1-p)\log(1-p))$ /*Equation 3.14*/
  $\forall x \in \mathcal{A}_{unk}(q_{l_g})$
    $G_e(x) = G_e(x) + g_{q_l}$
}

Program B.10: Calculating information gain density function.

```
compute_node_color(node)
{
Choose an arbitrary voxel p in node,
voxel_status = check_voxel_status(p);
If voxel_status == IN,
   {
   face_status = ⋂_{f∈far−faces} (check_face (f)==IN).
      if face_status == TRUE
         then node.color = WHITE
         else node.color = GRAY
   }
else
   {
   face_status = ⋂_{f∈near−faces} (check_face (f)==OUT)
      if face_status == TRUE
         {
         if x_a is inside node
            then node.color = BLACK
            else node.color = GRAY
         }
      else node.color = GRAY
   }
}


check_voxel_status(p)
{
if D(p) ≤ M(P(p))  return (IN)
else return (OUT)
}
```

Program B.11: Psudo-code for octree construction.

Appendix C

Experimental Setup

C.1 System Component Connection

In the "eye-in-hand" system, two computers are used: (1) an SGI workstation on which the robot server runs. (2) a PC (Pentium450) hosts the sensor server and the planner. These two computers are connected with a switched hub (CentreCom RS710TX). Since there are only two computers in the switched hub, the entire bandwidth, 10mb, can be used for the communications between the two computers (although the PC has 100mb network interface, the speed of communication is limited by the SCI's network interface, which is 10mb).

The robot server is essentially a multiple thread robot control program that runs on a host computer, the SGI workstation. Its application control interface is RCCL (Robot Control C Library). It communicates via a parallel port (National Instruments AT-DIO-32F) with a program (MOPER) that runs on the Unimation controller (an LSI 11). MOPER dispatches the joint setpoint commands to individual (original Unimation) joint servo controllers.

C.2 Sensor Specifications

The sensor uses a 30 mW laser striper (Lasiris SNF-501 series, 45 degrees fan angle). The sensor is configured to sense in the range of 30 cm - 105 cm from the camera

(Sony DC10-5 with a Cosmicar 8.5mm F1.5 lens). Within this measurable range, the sensor is accurate to about 0.5 cm (in all three coordinates). This range and resolution is just adequate for our Puma robot whose positioning accuracy is now roughly 1cm - 3cm, and whose workspace radius is about 1m. The field of view of the sensor in the direction perpendicular to the laser stripe (joint 6 rotating direction) is not limited. However, in the direction of the laser stripe, it is limited to 35 degrees. The scanning time depends on the scanning resolution and the field of view. In our typical application, it takes about 30 seconds to get a 256 by 256 resolution range image.

C.3 Initial Free Physical Space

We assumed that a small cuboid region of $80cm \times 80cm \times 220cm$ surrounding the robot is free. This is $\mathcal{P}_{free}^{(0)}$ (defined in Chapter 2). Note that with the PUMA in its most "compact" configuration, the smallest cube enclosing the projection of the PUMA on the horizontal plane is $67cm \times 67cm$. The minimum distance from $\mathcal{A}(q_0)$, the robot at the start configuration to the boundary of $\mathcal{P}_{free}^{(0)}$, is therefore only $6.5cm$ on each side.

Bibliography

[1] J. Ahuactzin, K. Gupta, and E. Mazer. Manipulation planning for redundant robots: A practical approach. *The International Journal of Robotics Research*, 17(7):731–741, July 1997.

[2] J. Ahuactzin and A. Portilla. A basic algorithm and data structure for sensor-based path planning in unknown environment. In *Proceedings of IEEE International Conference on Intelligent Robots and Systems*, page To appear, 2000.

[3] N. Amato, O. Bayazit, L. Dale, C. Jones, and D. Vallejo. Choosing good distance metrics and local planner for probabilistic roadmap methods. In *Proceedings of 1998 IEEE International Conference on Robotics and Automation*, pages 630–637, 1998.

[4] N. Amato, O. Bayazit, L. Dale, C. Jones, and D. Vallejo. Obprm: An obstacle-based prm for 3d work spaces. In *Robotics: The Algorithmic Perspective*, pages 155–168, 1998.

[5] J. Barraquand and J.-C. Latombe. Robot motion planning: A distributed representation approach. *International Journal of Robotics Research*, 10:628 – 679, January 1991.

[6] P. Bessiere and J. Ahuactzin. The "ariadne's clew" algorithm: Global planning with local methods. In *Algorithmic Foundation of Robotics, A K Peters, Ltd*, pages 39–47, January 1994.

[7] H. Choset and J.W. Burdick. Sensor based planning. i. *the generalized voronoi graph.* In *Proceedings of 1995 IEEE International Conference on Robotics and Automation*, pages 1649–1655, May 1995.

[8] H. Choset and J.W. Burdick. Sensor based planning. ii. *the generalized voronoi graph.* In *Proceedings of 1995 IEEE International Conference on Robotics and Automation*, pages 1642–1648, May 1995.

[9] H. Choset and J.W. Burdick. Sensor based planning for a planar rod robot. In *Proceedings of 1996 IEEE International Conference on Robotics and Automation*, pages 3584–3591, 1996.

[10] P. J. Clark and F. C. Evans. Distance to nearest neighbor as a measure of spatial relationship in populations. *Ecology*, 35:445 – 453, 1954.

[11] P. J. Clark and F. C. Evans. Generalization of a nearest neighbor measure of dispersion for use in k dimensions. *Ecology*, 60:316 – 317, 1979.

[12] C. Connolly. The determination of next best views. In *Proceedings IEEE International Conference on Robotics and Automation*, pages 432–435, 1985.

[13] T. Cormen, C. Leiserson, and R. Rivest. *Introduction to Algorithms*. McGraw Hill, 1989.

[14] C. K. Cowan and P. D. Kovsi. Automatic sensor placement from vision task requirements. *IEEE Transaction on Pattern Recognition and Machine Intelligence*, 10(5):407 – 16, May 1988.

[15] J. Craig. *Introduction to Robotics: Mechanics and Control*. Addison Wesley, 1990.

[16] A. Elfes. Using occupancy grids for mobile robot perception and navigation. *IEEE Computer*, 15(7):46–57, January 1989.

[17] S. Ghosh and J. Burdick. Exploring an unknown polygonal environment with a sensor based strategy. *Technical Report, Tata Institute of Fundamental Research, India*, April 1997.

[18] K. Goldberg. Completeness in robot motion planning. In *The First Workshop on the Algorithmic Foundations of Robotics . A. K. Peters*, pages 419–429, 1994.

[19] M. Greenspan and N. Burtnyk. Obstacle count independent real-time collision avoidance. In *Proceedings of IEEE International Conference on Robotics and Automation*, pages 1073 – 1080, April 1996.

[20] K. Gupta. Sensor-based motion planning for eye-in-hand systems. In *IEEE International Conference on Robotics and Automation (ICRA) Workshop – Integrating Sensors with Mobility and Manipulation*, 2000.

[21] K. Gupta and A. P. del Pobil. *Practical Motion Planning in Robotics: Current Approaches and Future Directions*. John Wiley and Sons, 1998.

[22] K. Gupta and Y. Yu. On eye-sensor based path planning for robots with non-trivial geometry/kinematics. Submitted to ICRA2001.

[23] J. Harvey and S. Payandeh. Rubber band paradigm: A method for environment reconstruction for global planning of a mobile robot. In *Proceedings of IEEE International Conference on System, Man and Cybernetic*, pages 1016 – 1020, 1995.

[24] V. Hayward and R. Paul. Robot manipulator control under unix: Rccl, a robot control library. *International Journal of Robotics Research*, 5(4):94 – 111, April 1986.

[25] S. Hedberg. Robots cleaning up hazardous waste. *AI Expert*, pages 20 – 24, May 1995.

[26] K. Hirai, M. Hirose, M. Haikawa, and T. Takenaka. The development of honda humanoid robot. In *Proceedings of IEEE International Conference on Robotics and Automation*, pages 1321–1326, 1998.

[27] D. Hsu, L. Kavraki, J. Latombe, R. Motwani, and S. Sorkin. On finding narrow passages with probabilistic roadmap planners. In *Algorithmic Foundation of Robotics, A K Peters, Ltd*, pages 141–153, 1998.

[28] S. Hutchinson and A. Kak. Planning sensing strategies in a robot work cell with multi-sensor capabilities. *IEEE Transaction on Robotics and Automation*, 5(6):765 – 783, December 1989.

[29] Y. Hwang. Completeness vs. efficiency in real application of motion planning. In *IEEE International Conference on Robotics and Automation Workshop, "Practical Motion Planning In Robotics: Current Approaches and Future Directions"*, April 1996.

[30] S. Ihara. *Information Theory for Continuous Systems*. World Scientific Publishing, 1993.

[31] D. S. Jones. *Elementary Information Theory*. Oxford University Press, 1979.

[32] D. Jung and K. Gupta. Octree-based hierarchical distance map for collision detection. In *Proceedings IEEE International Conference on Robotics and Automation*, pages 454–459, 1996.

[33] D. Jung and K. Gupta. Octree-based hierarchical distance maps for collision detection. *Journal of Robotics System*, 14(11):789 – 806, 1997.

[34] I. Kamon and E. Rimon. Range-sensor based navigation in three dimensions. In *Proceedings of IEEE International Conference on Robotics and Automation*, pages 163–169, 1999.

[35] L. Kavraki, M. Kolountzaki, and J. Latombe. Analysis of probabilistic roadmaps for path planning. *IEEE Transactions on Robotics and Automation*, 14:166 – 171, Feb 1998.

[36] L. Kavraki, P. Svestka, J. Latombe, and M. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 12(4):556 – 580, Aug 1996.

[37] S. Kovacic, A. Leonardis, and F. Pernus. Planning sequences of views for 3-d object recognition and pose determination. *Pattern Recognition*, 31(10):1407 – 1417, January 1998.

[38] E. Kruse, R. Gutsche, and F. Wahl. Effective, iterative, sensor based 3-d map building using rating functions in configuration space. In *Proceedings of IEEE International Conference on Robotics and Automation*, pages 1067 – 1072, 1996.

[39] K. Kutulakos, V. Lumelsky, and C. Dyer. Vision-guided exploration: A step toward general motion planning in three dimensions. In *Proceedings of IEEE International Conference on Robotics and Automation*, pages 289–296, 1993.

[40] J. C. Latombe. *Robot Motion Planning.* Kluwer Academic Publications, 1991.

[41] S.L. Laubach and J. Burdick. An autonomous sensor-based path-planner for planetary microrovers. In *Proceedings of IEEE International Conference on Robotics and Automation*, pages 347 – 354, 1999.

[42] S. LaValle and J. Kuffmer. Randomized kino-dynamic planning. In *Proceedings of IEEE International Conference on Robotics and Automation*, pages 472 – 479, 1999.

[43] A. Li and G. Grebbin. Octree encoding of objects from range image. *Pattern Recognition*, 27(5):727 – 739, May 1994.

[44] T. Lozano-Perez. Spatial planning: A configuration approach. *IEEE Transactions on Computers*, 32(2):108–120, 1983.

[45] V. Lumelsky. A comparative study on the path length performance of maze-searching and robot motion planning algorithms. *IEEE Transactions on Robotics and Automation*, 7(1):57– 66, January 1991.

[46] V. Lumelsky and K. Sun. Three-dimensional motional planning in an unknown environment for robot arm manipulators with revolute or sliding joints. *International Journal of Robotics and Automation*, 9:188– 198, April 1999.

[47] V.J. Lumelsky and E. Cheung. Real-time collision avoidance in teleoperated whole-sensitive robot arm manipulators. *IEEE Transactions on Systems, Man and Cybernetics*, 23(1):194–203, Jan.-Feb 1993.

[48] J. Maver and R. Bajcsy. Occlusions as a guide for planning next view. *IEEE Transaction on Pattern Analysis and machine Intelligence*, 12(5):417 – 433, May 1993.

[49] C. Nissoux, T. Simeon, and J-P. Laumond. Visibility based probabilistic roadmaps. In *Proceedings of IEEE International Conference on Intelligent Robots and Systems*, pages 1316 – 1321, 1999.

[50] H. Noborio, S. Fukuda, and S. Arimoto. Construction of the octree approximating three dimensional object by using multiple views. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 10(6):769 – 781, Nov 1988.

[51] H. Noborio and T. Yoshioka. Sensor-based navigation for a mobile robot under uncertain conditions. In *Practical Motion Planning in Robotics, Current Approaches and Future Directions*, pages 346 – 361, 1998.

[52] M. Overmars and P. Svestka. Probabilistic learning approach to motion planning. In *Algorithmic Foundation of Robotics, A K Peters, Ltd*, pages 19–37, 1996.

[53] A. Papoulis. *Probability, Random Variables, and Stochastic Process (third edition)*. McGraw-Hill, Inc, 1990.

[54] P. Payeur, D. Laurendeau, and C. Gosselin. Range data merging for probabilistic octree modeling of 3-d workspaces. In *Proceedings of IEEE International Conference on Robotics and Automation*, pages 3071–3078, 1998.

[55] M. Ponamgi, D. Manocha, and M. Lin. Incremental algorithms for collision detection between solid models. *IEEE Transactions on Visualization and Computer Graphics*, 4(1):61–64, 1997.

[56] A. Portilla. *Planificacion de trayectorias de un brazo robot en un ambiente desconocido*. Universidad de las Americas-Puebla, Mexico, 1999.

[57] A. Pruski and A. Atassi. Sensor information space for robust mobile robot path planning. *Robotica*, 18:415 – 421, 2000.

[58] P. Renton, M. Greenspan, H. Elmaraghy, and H. Zghal. Plan-n-scan: A robotic system for collision free autonomous exploration and workspace mapping. *Journal of Intelligent and Robotic System*, 24:207 – 234, 1999.

[59] E. Rimon and J.F. Canny. Construction of c-space roadmaps using local sensory data – what should the sensors look for? In *Proceedings of IEEE International Conference on Intelligent Robot and System*, pages 117–124, 1994.

[60] H. Samet. *The Design and Analysis of Spatial Data Structures*. Addison-Wesley Publishing Company, 1989.

[61] C. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, pages 379–423, July 1948.

[62] C. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, pages 632–656, Oct 1948.

[63] R. Shu and S. Kankanhalli. Efficient linear octree generation from voxel. *Image and Vision Computing*, 12(5):297 – 303, May 1994.

[64] C. E. Smith, S. A. Brandt, and N. P. Papanikolopoulos. Eye-in-hand robotic task in uncalibrated environment. *IEEE transaction on Robotics and Automation*, 13(6):903 – 914, Dec 1997.

[65] G. Soucy, F. Callari, and F. Ferrie. Uniform and complete surface coverage with a robot-mounted laser range finder. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robot and System*, pages 1682 – 1688, 1998.

[66] D. Stoyan and W.S. Kendall. *Stochastic geometry and its applications*. J. Wiley, 1995.

[67] P. Svestka and M. H. Overmars. Probabilistic path planning. In *Robot Motion Planning and Control*, pages 255 – 306. Springer, 1998.

[68] K. Tarabanis, P. Allen, and R. Tsai. A survey of sensor planning in computer vision. *IEEE Transaction on Robot and Automation*, 11(1):86 – 104, Feb 1995.

[69] C. J. Taylor and D. Kriegman. Vision-based motion planning and exploration algorithms for mobile robots. *IEEE Transaction on Robot and Automation*, 14(3):417 – 426, June 1998.

[70] D. Vallejo, C. Jones, and N. Amato. An adaptive framework for single shot motion planning. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robot and System*, 2000.

[71] Y. Yu and K. Gupta. An efficient on-line algorithm for direct octree construction from range images. In *Proceedings of IEEE International Conference on Robotics and Automation*, pages 3079 – 3084, 1998.

[72] Y. Yu and K. Gupta. On sensor-based roadmap: A framework for motion planning for a manipulator arm in unknown environments. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robot and System*, pages 1919 – 1924, 1998.

[73] Y. Yu and K. Gupta. Sensor-based roadmaps for motion planning for articulated robots in unknown environments: Some experiments with an eye-in-hand system. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robot and System*, pages 1707–1714, 1999.

[74] Y. Yu and K. Gupta. An information theoretic approach to view point planning for motion planning of eye-in-hand systems. In *Proceedings of 31st International Symposium on Robotics*, pages 306– 311, 2000.

[75] Y. Yu and K. Gupta. Sensor-based motion planning for manipulator arms: An eye-in-hand system. In *IEEE International Conference on Robotics and Automation video session*, 2000.

[76] Y. Yu and K. Gupta. Sensor-based roadmaps probabilistic roadmaps: Experiments with an eye-in-hand system. *Advanced Robotics*, To appear in 2000.