Toward Autonomous Cleaning with Manipulator Arms

Junmei Cao
B.Sc. Beijing (China)

February 1999

A thesis submitted for the degree of Master of Engineering of The Australian National University

Department of Engineering
Faculty of Engineering and Information Technology
The Australian National University

Acknowledgements

I would like to express my heart-felt appreciation to my supervisor, Dr. Jon Kieffer, for his excellent and thorough guidance during my Masters program. Without his help, support and encouragement, it is impossible to have done the research. Also, I would like to thank all people in the Department of Engineering who helped me. Finally, I would like to thank my beloved husband for his understanding and supporting.

Statement of Originality

The information contained in this thesis is the result of original research except where reference is made. And has not been submitted for the purpose of obtaining a higher degree at any university or institution.
The research described in this thesis is the result of a collaborative effort with my supervisor Dr. Jon Kieffer.

Junmei Cao
February, 1999

Conference Paper:

[C1] J. Kieffer, K. Yu, J. Cao and K. Maw, "Toward blind grouping: an approach to manipulation in unknown sensor-hostile environments," *Proceedings of the International Conference on Field and Service Robotics*, pp.449-455, Canberra, December, 1997.

Abstract

This thesis is to develop techniques to enable a manipulator arm to perform a cleaning task in an unknown obstacle-filled environment using only force sensing.

Our investigation focusses on reactive control, graph-based modelling for manipulator arms and graph-based motion-planning algorithms.

Reactive control strategies have produced significant results in improving the ability of wheeled and legged mobile robots to cope with unfamiliar environments and uncertainties. However, it seems that very few attempts have been made to apply these strategies to manipulator arms. In this thesis, we extended the purely reactive controller [10] with the ability to plan, and used only force sensing. The controller was applied to an Industrial Scientific SCARA robot to explore an unknown obstacle-filled environment. The task is to cover all free space to simulate cleaning.

The motion planning problem for manipulator arms is different from mobile robots. The arm kinematics allows each task position to be reached in a number of configurations. Our graph-based modelling for manipulator shows manipulator motion planning problem can be formulated as a graph search.

To work in unknown environments, robots have to re-plan motions on-the-fly as new information is acquired. In the thesis, we present a new algorithm to minimise re-planning computation on uniform graphs by incrementally repairing data. It is essentially a simplified version of Stentz's D* algorithm [8], but, it is easier to implement, and more efficient for uniform graphs searches.

We also investigate two graph-based motion planning algorithms in the context of sensor-based motion planning: an optimistic shortest path algorithm and a depth first search algorithm. We compare the DFS algorithm with the OSP and show that it achieves similar average-case performance if an original technique that we call "tree-

pruning" is used. This contradicts widely held perceptions that shortest path algorithms are significantly better than depth-first searches. Simulations suggest that the developed algorithms are effective for manipulator arm motion planning.

Contents

List of Figures

Chapter 1

Introduction

1.1 General Background

The development of specialised service robots [1] was brought about by the increasing demand to economise on several kinds of services. A representative application field is floor-cleaning. Our work is directed toward the development of an autonomous vacuum cleaning system to perform floor cleaning in unknown obstacle-filled environment using only force sensing.

Motivation for the autonomous cleaning problem comes from the materials handling in heavy industry. BHP, for example, spends about 8 million dollars per annum to collect raw materials that fall from conveyor belts in the Port Kembla Steel works. It is generally done by hand, using shovels and scrapers, exposing workers to heavy manual labour in dusty and unhealthy conditions. So developing automated cleaning systems can potentially not only decrease costs for heavy industry, but also free the workers from hostile and unhealthy environments.

To move around, find a path, or do a job like cleaning seems very simple for a human. However, the human operator can not be replaced easily by "artificial intelligence" [1]. One reason is the complexity of cleaning task. The difference between robots and animals is that animals use their brain to solve a problem, and we start to understand the complexity of these "simple" tasks when we analyse the complexity of the brain [2]. Another reason is the limitations of the actual technology. This field needs smart systems that are well adapted to their environment through their shapes, sensors, actuators and behaviours [3]. Therefore, most of the developed cleaning machines

primarily assist a human operator in large, fixed cleaning areas with known structure. The lack of results is common to a large range of applications in autonomous robotics.

In the last decade, new approaches have tried to bring "artificial intelligence" and the computer science community back to the real world to get better results in the field of autonomous mobile robotics [4]. Reactive control, for example, have produced significant results in improving the ability of wheeled and legged mobile robots to cope with unfamiliar environments and uncertainties [10, 18, 22]. Reactive systems are a relatively recent development in robotics that has redirected artificial intelligence research. This new approach grew out of a dissatisfaction with existing methods for producing intelligent robotic response and a growing awareness of the important of looking at biological systems as a basis for construction intelligent behaviour [5].

In addition, the development of force sensing technology brings good results in heavy industry environment. It offers the advantages of being well developed, rugged, reliable and economical.

In the thesis, we present some foundational research toward the development of an autonomous vacuum cleaning system to work in an heavy industrial environment based on reactive control and force sensing with suitable motion planning algorithms.

1.2 Summary of Contributions

The general objective has been to develop techniques to enable a manipulator arm to perform a cleaning task in an unknown obstacle-filled environment using only force sensing. In summary, this research has made the following contributions:

1. Experimental implementation of a reactive controller for a manipulator arm to perform surface coverage. This controller extends the purely reactive controller [10] with ability of planning, and uses only force sensing. The experiment is limited to end-effector force sensing rather than the whole arm sensing.

2. Graph-based modelling for manipulator motion planning. The motion planning problem for manipulators is different from mobile robots. The arm kinematics allows each task position to be reached in a number of configurations. Our graph-based modelling for manipulator shows manipulator motion planning problem can be formulated as a graph search.

3. Efficient graph marking scheme to minimise re-planning computation on uniform graphs by incrementally repairing data. It is essentially a simplified version of Stentz's D* algorithm [8], but, it is easier to implement, and more efficient for uniform graphs searches.

4. Based on the efficient graph marking scheme, we develop efficient implementations of two algorithms for graph-based motion planning: an optimistic shortest path algorithm (OSP) and a depth first search algorithm (DFS). Also, we develop an innovative pruning rule for our DFS algorithm, which keeps the robot from exploring subgraphs that are obviously void of goals.

5. A study of OSP vs DFS algorithms in sensor-based motion planning. This shows that our DFS achieves similar average-case performance with OSP algorithm. This contradicts widely held perceptions that OSP algorithms are significantly better than DFS searches.

1.3 Overview of Thesis

Chapter 2 presents the conceptual description of the manipulator arm cleaning problem. It is an overview of the problem. Enabling technologies are discussed and a general literature review is given. Chapter 3 develops a reactive motion planning algorithm for a manipulator arm in our laboratory. Experiment results are given to demonstrate complete coverage in an unknown obstacle-filled environment. Chapter 4 talks about modelling. It presents graph search models for our cleaning problem. Chapter 5 gives sensor-based motion planning algorithms. The algorithms are more general and more realistic for practical application than the one considered in chapter 3. Chapter 6 describes the details of our incremental repair algorithm (similar to Stentzs D* algorithm) which improves the computational efficiency of sensor-based graph searches. Chapter 7 gives an evaluation of algorithms though simulation. Chapter 8 concludes this work and presents further research.

Chapter 2

Conceptual Description of the Manipulator Arm Cleaning Problem

2.1 Introduction

This chapter presents an overview of the cleaning problem. It begins by describing the cleaning problem. In section 2.3, we talk about enabling technologies that we feel will be important to develop the system. A literature review is given in section 2.4.

2.2 Description of the Problem

Path planning problems are nearly as old as mankind: they were necessary early in our history for basic activities such as finding food and other necessities. Here, we consider autonomous coverage of unknown terrain in a heavy industry environment.

Robots for field applications should be low-cost, rugged and reliable. Especially, cleaning in heavy industry requires that machines are insensitive to dirt and dust. In addition, they should also be designed to operate effectively in unknown and sensor-hostile environments. This applies to their mechanical design as well as their sensing and control strategies.

For collecting raw material we envision an autonomous system comprising mobile robot with a dexterous arm that manipulates a vacuum tool to perform cleaning. Rather

than mobile robot navigation problem, we focus on controlling a manipulator arm to perform floor cleaning in unknown obstacle-filled and sensor-hostile environment. We highlight our problem as follows:

1. A manipulator arm is to accomplish the cleaning.
2. The manipulator arm works in a heavy industrial environment.
3. The environment is filled with unknown obstacles.
4. The obstacles have unknown shape.
5. Obstacles can interfere with the links of the manipulator arm as well as the end-effector.
6. At the beginning, the robot know nothing about the environment.

Autonomous cleaning seems feasible if the following requirements can be met.

- Sensing

  Measurement of environment parameters is fundamental to the successful application of robots. For a manipulator arm to be autonomous it must be able to sense obstacles and traverse paths through its environment. In addition, the sensing system should work correctly regardless of the shape of the obstacles.

- Control

  The control system need to achieve multiple goals in the environment: it is trying to reach a target while avoiding obstacles. And it must be responsive to high priority goals. For example, avoiding obstacles is much more important than reaching a goal. Furthermore, to respond quickly, the system should be as simple as possible instead of an extremely complex control system.

- Motion planning

  We are interested in two kinds motion planning: surface coverage and find goal. Surface coverage describes an action by which all free space in an environment are covered in a systematic and efficient manner. Find goal focusses on finding a path from a start location to a goal location. Based on the two types of motion

planning, we can not only ensure a complete sweep of all free space, but also perform some targeted cleaning in particular areas.

- Reliability

  The system must be rugged, reliable and insensitive to dirty and dust.

The usual approach to this task would be to start by building a three dimensional model of the environment. Then we would employ an algorithm to plan a trajectory to cover all free space or from the start point to a target position, and then pass this path to a servo system to give us precise control of the robot's joints.

Forget the problems of implementing such a complex control system for the moment. What seems to be the trouble is that it is difficult to get the reliable environment information from sensors. The approach described above relies on accurate world modelling. To build a proper three-dimensional model, we need to take multiple images from different directions and then fuze them into a single coherent description [12]. Unfortunately, the sensors needed to support this endeavour are typically expensive, difficult to implement and problematic in dirty and dusty environment. Also, some of these sensing techniques require special lighting or impose restrictions on the surface properties of objects [28]. In addition, it is difficult to get clean data from the sensors mounted on a moving vehicle. Furthermore, the undeveloped sensor fusion techniques prevent us getting an accurate world model.

However, without an accurate world model, it is impossible to implement the robot navigation algorithm described above. As Leonard and Durrant-Whyte said in [33] to answer the question of why robot and reliable autonomous mobile robot navigation remains such a difficult problem:" In our view, the reason for this is clear. It is not the navigation process *per se* that is a problem – it is the reliable acquisition or extraction of information about navigation beacons, from sensor information, and the automatic correlation or correspondence of these with some navigation map that makes the autonomous navigation problem so difficult."

To overcome this limitation, we avoid building a three dimensional model and precise manipulator control. Instead, we follow Brooks [10] to adopt a "vertical" decomposition of our control system. We also choose to base our control on force sensing. The uncertain and sensor-hostile nature of the environment under conveyor belts has

led us to the concept of blind groping, and we believe it is a better method to solve the cleaning problem, in term of realizable, low-cost, rugged and reliable.

blind groping: *robotic arm manipulation in unknown environments based primarily on force and proprioceptive sensing.*

2.3 Enabling Technologies for Blind Groping

2.3.1 Mechanical Design

Industrial robot arms are unsuitable for blind groping because they are not designed for force control or to accommodate unexpected impacts with the environment. The philosophy of Whole Arm Manipulation (WAM) [20], however, provides principles and technology for developing such arms. The WAM arm is designed to contact and sense the environment with any of its surfaces. This allows the manipulator arm to learn whether a given motion is possible through trial motions.

According to WAM principles robot arms should be light weight, back drivable and constructed from durable material. In [20], Townsend and Salisbury showed how this can be achieved using cable-drive technology. They also describe how motor current sensing can be applied to determine the location of environmental forces on the arm.

We believe that these principles, along with rugged design standards such as those developed in the mining industry should be a suitable for developing blind groping manipulators.

2.3.2 Sensing

The sensor system is an important part of the robot as it provides information about the environment. This information is used for control loops, finding the location of objects and monitoring the environment for changes that may affect the task. For blind groping, detection and avoidance of obstacles is especially important.

Because remote sensing is problematic in many environments, it is particularly difficult in dirty and dusty environments. we feel that force sensing will play a key role in field applications.

Force sensing technology offers the advantages of being well developed, rugged, reliable and economical. It is also insensitive to the wide variety of noise sources that

effect remote sensing technologies: atmospheric dust, variable lighting, acoustic and electromagnetic noise. In addition, the shape of obstacles is not important for force sensing.

2.3.3 Control

The cleaning problem offers new challenges to robot arm control that are perhaps more familiar to researches in mobile robotics and legged locomotion: the key problem is how to achieve robust and useful operation in unknown or uncertain environments.

Inaccurate sensors , world unpredictability, and imperfect control often cause the failure of traditional planning and navigation methods for real-time robot systems. In contrast, behaviour-based control strategies has been used successfully in improving the ability of mobile robots to cope with unfamiliar environment and uncertainties [10, 22, 26, 27, 28, 29]. To accomplish cleaning we believe that traditional theories for manipulator motion and force control should be integrated into a reactive or behaviour control framework. The traditional controllers are needed to support free motion and contour following and the behaviour control framework is important for organising tasks and for reacting quickly to unexpected events.

2.4 Literature Review

In the subsection, we focus on the literature of behaviour-based control and motion planning. Mechanical design is beyond the scope of our research.

2.4.1 Behaviour-based Control

Reactive robotic systems originate in the cybernetic movement of the 1940s. Grey Walter developed an electronic "tortoise" capable of moving about the world, avoiding perceived threats and attracted to certain goals [5].

With the development of artificial intelligence, a number of people interested in organising intelligence have contributed to this field [10, 15]. There was a requirement that intelligence be reactive to dynamic aspects of the environment, that a mobile robot operate on time scales similar to those of animals and humans, and that intelligence be able to generate robust behaviour in the face of uncertain sensors , an unpredictable

environment and a changing world. There is no generally accepted term to describe this style of work. It has sometimes been called "reactive planning". Sometimes the approach is called "behaviour-based" as the computation components tend to be behaviour producing modules [11].

Brooks [10] is a leader of the reactive robotic paradigm. He believes that:" From an evolutionary stance, human level intelligence did not suddenly leap onto the scene, There were precursors and foundations throughout the lineage to humans. Much of this substrate is present in other animals today. The study of that substrate may well provide constraints on how higher level thought in humans could be organised." [11]

In [10], Brooks designed a control structure, called subsumption architecture, for autonomous mobile robots. He articulated the departure from classical AI and broke away from the sense-plan-act paradigm that dominated AI in the 1970-80s as typified by robots like Shakey that used resolution theorem proving as its primary reasoning mechanism [5]. The basic idea proposed is to decompose the robot system based on its external behaviour rather than internal operation. Reactive robotic control systems are characterised by a tight coupling between sensing and action, typically without the use of any intervening global representations. With the control system, a mobile robot wandered around unconstrained laboratory areas and computer rooms. Sonar data was used for real-time obstacle avoidance because it is useful for low-level interactions and it is cheap and easy implement than visual data.

However, it is recognised that purely reactive robotic systems are not appropriate for every robotic application, especially in situations where the world can be accurately modelled and there is restricted uncertainty. Aiming for the compensation of shortcomings of the subsumption architecture,several approaches have been proposed. Hybrid architecture [14, 16, 23] is the most common approach. It permits reconfiguration of reactive control systems based on available world knowledge, adding considerable flexibility over purely reactive systems. The idea is that the reactive system handles the real-time issues of being embedded in the world, while the deliberative system does the 'hard' stuff traditionally imagined to be handled by an Artificial Intelligence system. In [14], it is shown how a priori world knowledge, when available, can be used to configure behavioural and perceptual strategies in an efficient form. Although Brooks [11] thinks that these approaches are suffering from the well known "horizon effect", it has bought

better performance in their overall system with the reactive component.

It's a complex endeavour and inevitable process to realize the ambition of artificial intelligence. Brooks said:" We sometimes need to step back and question why we are proceeding in the direction we are going, and look around for other promising directions."

Reactive control directly couples real-time sensory information to motor actions without the use of intervening symbolic representations. Arkin [18] lists the following general characteristics of reactive control:

- It is typically manifested by a decomposition into primitive behaviours.
- Global representations are avoided.
- Sensor decoupling is preferred over sensor fusion.
- It is well suited for dynamically changing environments.

In addition, the reactive system also has the following characteristics:

- The global system is robust.
- The system can cope with multiple goals.
- The system is well-adapted for hardware implementation [30].

Behaviour-based and reactive control strategies have produced significant results in improving the ability of wheels and legged mobile robots to cope with unfamiliar environments and uncertainties [10, 11, 21, 26], however, it seems that very few attempts have been made to apply these strategies to manipulator arms [27]. Although robot arms have been traditionally associated with highly constrained and predictable environments, in field robotics applications they will have to cope higher levels of uncertainty. In [34], Connell followed Brooks [10] to implement a reactive controller for an actual mobile robot arm to locate and retrieve empty soda cans in an unstructured environment using a variety of local sensors to get information of environment.

To realize our cleaning task, we are interested in applying the behaviour-based control strategies to our manipulator arm using force sensing only. It is a novel attempt. In chapter 3, we present an experimental reactive controller for a manipulator arm in

our laboratory. It covers all free space in its workspace using only the force sensor to obtain information about the environment.

2.4.2 Sensor-based Motion Planning

Much of the focus of the research effort in path planning for robots has concentrated on the problem of finding a path from a start location to a goal location, while minimising one or more parameters such as length of path, energy consumption or journey time [7, 8]. Surface coverage or sweeping can be defined as: a path of complete coverage is a planned path in which a robot sweeps all areas of free space in an environment in a systematic and efficient manner [17].

One problem of interest is to provide nearly identical algorithms for the two problems, so switching between them is easy and the approach is well-suited to higher-level supervision.

Path planning problems can be classified in two main groups: known environment and unknown environment [8].

There are many research results for the known environment [31]. For the known environment case, a path is generated from a given map of the environment.

In [3], for example, a 2D-map of a prior known walls, pillars, staircase or fixed objects is given. With the map, a rule based planning system generates the motion sequence for an appropriate cleaning path according to robot geometry and kinematic restrictions.

Zelinsky in [17] presents a complete coverage algorithm for an unstructured environment. The distance transform path planning methodology [35] is used in the algorithm. In this method, the planner starts at the goal cell and propagates distances through free space. Like dropping a stone in a pond, the ripples radiate out, bouncing off objects in their path, with the first ripple to reach a designated point taking the shortest path, A distance transform value is calculated for each grid square [32]. Once the distance transforms are generated, the path is planned from start to goal. The algorithm is described as: if the cells around the start cell do not have a lower value than it does, ripples in the distance transform did not reach it and, this, there is no path. If they do have a lower value, the ripples did reach the start cell and there is a path. To find it, follow the valley (lowest values) to the goal. In this way, a path can be found from

any start point to the goal.

Choset and Pignon in [24] use cellular decomposition technique to decompose a free space into cells and an adjacency graph to reflect the connectivity of the individual cells. Motion planning between two points is achieved via a graph search of this adjacency graph. By covering each of the cells of the decomposition, a robot is guaranteed to cover the entire free space. This is done in two steps: a path is found in the adjacency graph that visit each node, and then the robot motions are computed within each cell.

However, there are fewer result for unknown environments. Without a prior knowledge of the world, robots have to replan motions on-the-fly as new information is acquired. This is known as sensor-based motion planning.

In fact, once an unknown environment is modelled by cell decomposition, road maps, terrain grids etc., then path-planning problem reduce to a sensor-based graph search [7, 8, 6, 19]. In these publications, the robot assumes that the terrain is clear unless it knows otherwise. It uses this "freespace assumption" to plan a shortest potentially traversable path though vertices that are known to be open or unknown. Whenever it detects an obstacle that blocks the planned path, the robot re-plans its path. If it reaches a goal vertex, then it stops and reports success. If it fails to find a traversable path from its current vertex to the goal vertex, it stops and reports that the goal vertex can not be reached. In [7], Seven Koenig presents the following "proof" that planning with the freespace assumption is correct.

> *Every time the robot cannot follow a planned path, it has learned about at least one additional blocked vertex. There are only a finite number of them, implying that planning with the freespace assumption terminates in finite time. Planning with the freespace assumption reports success only if it is at the goal vertex and has thus solved the sensor-based planning problem. It reports failure only if no traversable path from its current vertex to the goal vertex exists. Since there is a traversable path from its current vertex to the starting vertex, there is no traversable path from the starting vertex to the goal vertex either. Consequently, reaching the goal vertex is impossible in this case.*

Another benefit of the algorithm is that the shapes of obstacles need not to be considered.

With the freespace assumption and distance transform, sensor-based motion planning schemes, such as [6], rely on the following shortest-path strategy: determine the path from the current position to the goal that appears to be shortest based on current knowledge. Then move along it. When new obstacles are encountered, the "shortest path " may need to be revised, but the policy (to follow it) remains the same. We call this the optimistic shortest path (OSP) strategy. It is often associated with A* or Dijkstra's algorithm which are used to make the shortest path calculations. OSP provide good average-sense performance for finite graphs. However, its worst-case performance is less tightly bounded than any depth-first search(DFS) and there is no evidence that OSP is optimal against any specific criteria. DFS algorithms cover graphs with no more than two traversals of each edge in the spanning tree they generate. This bounds the cost of a goal search to twice the weight of the heaviest spanning tree. No search algorithm can provide a better worst-case performance bound [7].

With cellular decomposition, freespace assumption, the distance transform and some rules, we present DFS and OSP coverage planning algorithm in chapter 5. In chapter 6, we present essentially a simplified version of Stentz's algorithm to minimise re-planning computations, which is easier to implement, and more efficient for uniform graphs searches. In chapter 7, we compare DFS with OSP algorithms, and show that it achieves similar average-case performance against some criteria.

Chapter 3

A Reactive Controller for Manipulator Arm

3.1 Introduction

In this chapter, we present a reactive floor-coverage controller that was implemented experimentally in our laboratory. We did this "proof of principle" experimental work to familiarise ourselves with the basic issues. Firstly, manipulator arms have traditionally worked in constrained and predictable environments, their ability to work in field robotics applications where they must cope with higher levels of uncertainty is not clear. In addition, we are interested in how a reactive controller can achieve coverage task. Finally, the performance of force sensing in blind groping is also of interest to us. The experiment was designed to answer these questions, but it is not general because it is limited to end-effector force sensing rather than the whole arm sensing. The controller followed Brooks [10], but extends the purely reactive subsumption architecture with some ability to plan. The robot worked in an unknown and obstacle-filled environment. Its task was to explore all free space in its workspace and to stop when it knew that better coverage can not be obtained.

Different from traditional centralised sequential controller, a reactive controller is a collection of independent behaviours. Each behaviour only pays attention to those factors relevant to their particular task rather than general-purpose. For example, if the task is to avoid an obstacle then the associated control system does not need to know the shape of the obstacle. To do so, the perceptual burden is distributed, and there is

not complex computation between the sensor data and execution. By tightening the link between sensor data and execution, the robot can cope with uncertainty quickly. It is important to our cleaning task because we expect that blind groping will require the ability to respond quickly.

The different behaviours interact though a suppression mechanism. The mechanism allows higher level behaviours to override lower level behaviours. Each behaviour can realize a small task, but with all these behaviours, the robot can realize a more complex task. The architecture is presented in detail in section 3.2. We used only force-sensor to interact with environment. Of course, we also need encoder to measure the position of the robot. We use a value of force sensor to judge that the arm has collided with obstacles.

Instead of an accurate world map, the workspace of the robot arm is modelled as a grid-like map. The map provides the system with a global perspective on task execution which is lacking in the purely reactive system. The lack of an explicit world model was originally seen as an advantage of subsumption architecture. However, without a world model, a system can not anticipate a targeted action until that action happens to be sensed. For our cleaning task, the robot has to explore all free space with a systematic and efficient manner. The purpose of a world model is twofold: To keep track of the state of the world, and to describe the effect of an action. So the world map can potentially decrease unnecessary actions and ensure that all free space is reached.

Planning is to look ahead and to make decisions based on the outcome of actions. In the past, planning has been studied in isolation. Traditional planning system constructed plans in a centralised planning subsystem. Planners were seen as logical engines which generated a complete, detailed plan. Our planner described in section 3.3 avoids global searches by using only the state of a local patch of the map. The experiment in section 3.4 proves that the planner can cover all free space.

Our work described in the chapter is based on the following assumption and limitations:

1. As long as the value of force sensor is above a threshold, no matter whether the collision is with obstacles or with a pile of granular material, it is treated as meeting an obstacle.

2. Obstacles are stationary. In the cleaning problem, granular material may be spill from conveyor belts at any time. However, we assume that the spilling process is much slower than the cleaning process.

3. Only the end-effector of the robot arm meet obstacles.

3.2 Architecture

The reactive controller followed Brooks architecture [10]. The complete set of behaviours and their interconnections is shown in Fig 3-1. There are three layers in the control system. Each layer with actuators can accomplish a special task. Each box is a module which implements a specific behaviour. Individual layers can be working on individual goals concurrently, and there is no need to make an early decision on which goal should be pursued. The suppression mechanism then mediates the actions that are taken. The objective is to design each behaviour so that the resulting interaction achieves the cleaning task.

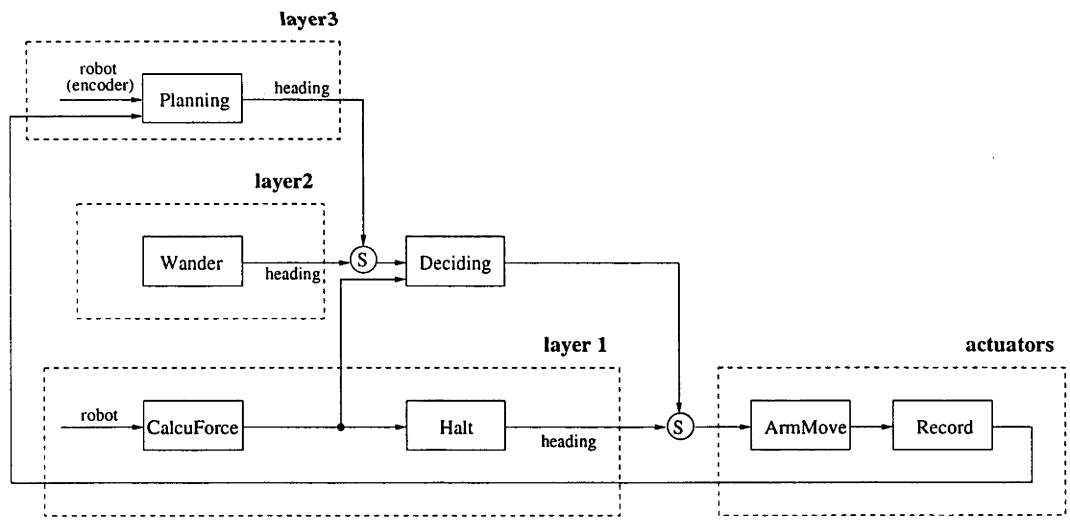

Figure 3-1: The control system for cleaning

3.2.1 Layer 1: Avoiding Obstacles

The lowest level of the control system is for obstacle avoidance. If contact with an obstacle occurs, then the robot arm will halt, then move away to the position given. The behaviour of *Calcuforce* is used to calculate the force from force sensor and get

a force with magnitude and direction. The function also checks whether the force is above a threshold, which implies the arm has contacted an obstacle. If this happens, the *Halt* function will immediately stop the robot moving and send a heading for the arm to move away.

3.2.2 Layer 2: Wander

Combined with the first layer, layer 2 guides the robot to wander and explore its workspace without collision. The function of *Wander* is to generate a heading randomly. *Deciding* takes result of *Calcuforce* and combines with the heading from *Wander* to produce a modified heading.

3.2.3 Layer 3: Planning

In this layer, decision sequences are generated by the planner based on the environmental situation. Combined with layers 1 and 2, the robot will go to the targeted position avoiding obstacles. The planner needs a world model to evaluate alternatives. The model contains enough information to make a decision. With the planner, the robot can achieve an coverage behaviour or find-goal behaviour, and it stops when better coverage can not be obtained.

3.3 An Algorithm for Planning

The section describes a behaviour for covering a 2D grid-like graph that contains unknown obstacles. The algorithm guarantees complete coverage, includes a simple termination condition, and uses a cell-closing mechanism to improve efficiency.

Let the robot's two-dimensional configuration space be divided into a grid of cells. Each cell is either occupied or free. The robot's location is represented by a cell on the grid. The robot starts with no knowledge of which cells are free or occupied and is assigned the task of visiting all cells that are reachable from its starting position. For simplicity, the robots is only able to move to one of the four cells adjacent to its current position, not diagonally.

3.3.1 Definitions and Assumption

1. **Definitions:**

   - neighbouring cell: one of the four cells adjacent to a given cell
   - corner cell: one of four cells diagonal to a given cell. A neighbour to two of the neighbouring cells
   - unreachable cell: a cell that has been found to contain an obstacle or is outside a predetermined search area (workspace)
   - closed cell: a cell that has been visited and is not worth revisiting because it does not lead to any unvisited cell
   - door: the connection between two neighbouring cells
   - unexplored door: a door from a visited cell to a cell whose status in unknown (a cell of unknown status is one that has not been visited and is not known to be unreachable).

2. **Assumption:**

   One failure to move to a cell is sufficient to consider the cell unreachable (a cell's reachability does not depend on the direction it is approached from).

3.3.2 Algorithm

1. **Summary of algorithm:**

   The algorithm looks at the 4 neighbours of the current cell, then eliminates those which do not exist (are not part of the grid), are known to be unreachable, or are closed. Of the remaining cells, the algorithm chooses the cell that has been visited the least number of times. If several cells meet this criteria the algorithm uses an arbitrary preference scheme(e.g. right-front-left-back, north-south-west-east, etc. or even randomly) to choose between them.

   Heuristic proof: the scheme is similar to the potential function approach to motion planning, except that the value of the potential function in each cell increases with the number of visits. The robot can not remain stuck in a local well because visits within the well eventually convert it into a peak. Complete coverage follows from boundedness of the grid.

2. **Termination Condition:**

   The termination algorithm keeps track of the number of unexplored doors and terminates when there are none left. Complete coverage has then been achieved.

   Proof: Only an unexplored door can lead to an unvisited cell. If there are none left, then all reachable cells have been visited.

   The termination condition is implemented by a counter that is initialised to the number of unexplored doors in the starting cell, then incremented or decremented according to the following two rules.

   rule1: when a cell is visited for the first time, the unexplored doors counter is incremented for each door that leads to an unvisited cell and decremented of reach door that leads to a visited cell.

   rule2: Whenever a cell is found to be unreachable (i.e. , when an attempt to move through an unexplored door fails) the unexplored doors counter is decremented for each door leading from the unreachable cell to a visited cell.

3. **Closing cells:** A cell can be closed if it has been visited and its closure does not block access to unvisited cells.

   The conditions above can be satisfied in many ways, but we are only interested in cases, such as the following, that can be determined from local information.

   Criteria 1: If a cell has been visited AND three out of four of its neighbours are unreachable(or closed) then the cell can be closed.

?	X	?
X	*	X
?	?	?

X = unreachable (or closed)
cell * may be closed if visited

   Criteria 2: If a cell has been visited and two adjacent neighbours are unreachable (or closed) AND the opposite corner cell has been visited and is not closed, THEN the cell can be closed.

?	X	?
X	*	?
?	?	V

X = unreachable (or closed)
V = visited and not closed
cell * may be closed if visited

we use the above criteria in following way:

When a cell is entered:

(a) check its neighbouring cells against both criteria

(b) check its corner cells against both criteria

(c) check its neighbouring cells(again) again both criteria

3.4 Experiment

The coverage algorithm was implemented on an industrial Scientific CF-310 SCARA robot shown in Fig.3-2. A JR3 force-torque wrist sensor was mounted between the wrist and a spherical end-effector which was used to encounter partially filled plastic buckets which serve as obstacles. The robot is an industrial robot with four degrees of freedom, and both base and elbow were equipped with an encoder and a tachometre. The reactive controller was implemented by C language on the Vxworks operating system. The sampling frequency for both position and force sensing is 300 HZ.

The task is to explore the unknown obstacle-filled environment and to move the end-effector over the all free space to simulate cleaning. In the experiment, we assume that only end-effector can meet with obstacle rather than the whole arm.

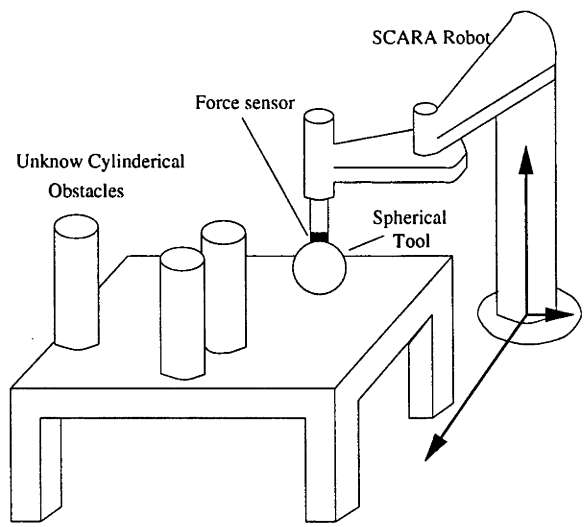

Figure 3-2: Coverage experimental system

1. **empty environment**

   Terrain coverage of an empty workspace works well. Figure (3-3) show the resulting path of coverage. Attention should be paid to cell **a** and **b**. When a robot moved to the cell **a** or **b**, it found a way go to unvisited cell efficiently because of closing cells rules.

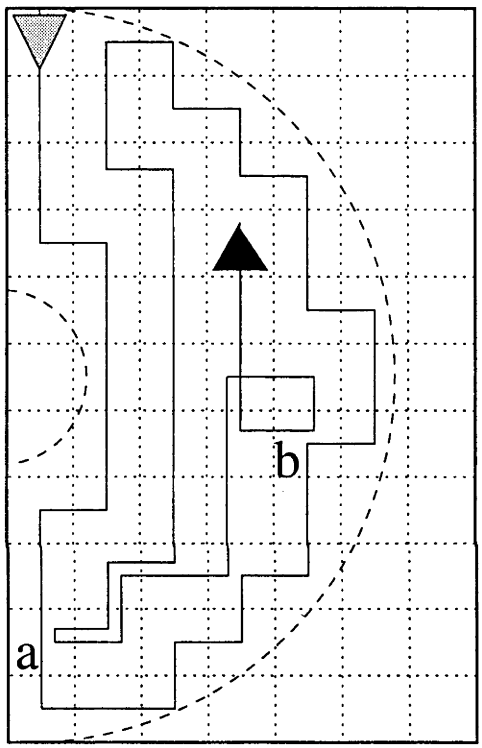

Figure 3-3: Terrain coverage of an empty environment: the robot started at the cell indicated by the light shadowed triangle, and terminated at the dark shadowed triangle. The line indicates the path.

2. **environment with obstacles**

   The figure (3-4) is an example of covering an obstacle-filled environment. Complete coverage was obtained in this case with no more than two visits to any cell.

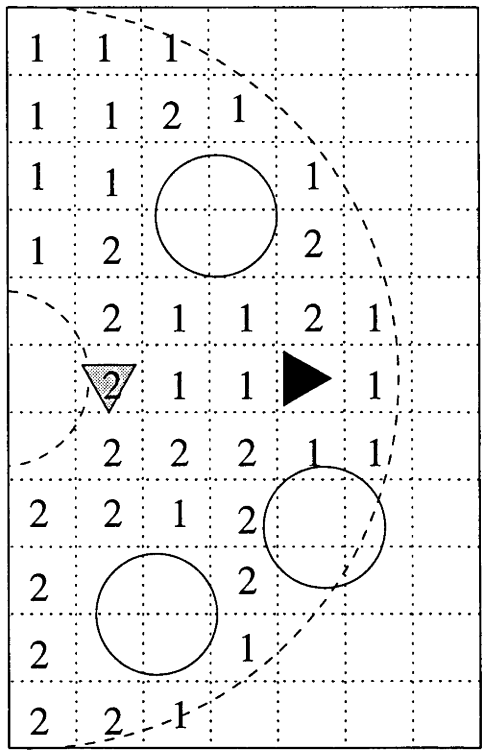

Figure 3-4: Terrain coverage of an obstacle-filled environment: The full circles represent obstacles (buckets) grown by the radius of the spherical end-effector tool. Starting at the position indicated by the down pointing triangle, the manipulation tool explored its workspace using the r-f-l-b preference scheme and cell closing . The tool explored the reachable workspace to the grid resolution shown. The numbers represent the number of visits to each cell. All cells except the last were closed before termination.

3. Conclusion

The results of the experiment support the conclusion that a manipulator arm driven by a reactive controller can work well in an unknown obstacle-filled environment.

Also, a reactive controller with a planner can accomplish a coverage task (to simulate cleaning).

Finally, we conclude that using only force sensing is a reasonable way to solve our cleaning problem.

Chapter 4

Modelling for Graph Search

4.1 Introduction

In this chapter, we present graph search models for our manipulator arm cleaning problem. These models are more general and more realistic for practical applications than the problem considered in chapter 3. The chapter begins with a description of the cleaning problem, then we propose solutions to the modelling problem.

4.2 Description of the Problem

Our objective is to control a manipulator arm to perform floor cleaning in an unknown obstacle-filled environment. To be more realistic for practical applications, our investigation is based on the following assumptions:

1. Obstacles can interfere with the links of an arm as well as the end-effector.

2. The arm can sense contact with obstacles. Here, we provide robot arm a realistic sensing capability. As we describe in section 2.3.1, WAM manipulation arm, for an example, offers a benefit of contacting and interacting with the environment by using any of its surface. Backdrivability and motor current sensing allow the manipulator to learn whether a given motion is possible through trial motion.

Based on these two assumptions and some results from chapter 3, we would like to provide a sensor-based motion planning algorithm for manipulator arms. The cleaning problem is complicated by the following facts.

Firstly, manipulator kinematics brings a new challenge to graph search. For a robot arm, there exist several arm solutions which are corresponding to the same task space location. In an environment where only the end-effector can contact obstacles, a manipulator arm can use either one of them to explore a task position and make a conclusion about whether the location is reachable. But for the problem in this chapter, there maybe exist some solutions which can not be reached because unknown obstacles prevent moving of its links. So, it may be necessary to try each kinematic solution to answer the question of whether the location is reachable or not.

Secondly, exploring task space using each arm solution in joint space is a sufficient condition for complete coverage of the reachable task space vertices, but it is not a necessary condition. Therefore, efficency is also an issue.

To solve the problem, we propose to

1. Build a manipulator joint space graph

   The search task requires a graph on which a manipulator can find a path to a specified goal or report the goal is unreachable. Because of the manipulator arm kinematics, there are several arm solutions for the same task space location. For a redundant robot, the solutions are infinite. It is necessary to build a joint space model that covers all arm solutions.

2. Provide an algorithm to search in that graph for specific task-space goals. The algorithm should meet the following requirements:

   (a) It is easy to implement.

   (b) Its computations can be done efficiently.

   (c) It does not need to make assumptions about the shapes of obstacles.

   (d) It knows to stop when the arm is certain that a graph search task is finished or unreachable.

   (e) It is can be achieve goal search and coverage problem together efficiently.

In sections 4.3 and 4.4, we present examples of how to build joint-space models for a two-link planar(non-redundant) and three-link planar(redundant) robot arm. The

models are undirected and unweighted graph. Algorithms to search on these models in detail are given in chapter 5.

4.3 Model of Two-link Planar Arm

Consider the inverse kinematics solution of the arm shown in Figure (4-1), it is desired to find the joint variables θ_1 and θ_2 corresponding to a given end-effector position and orientation G.

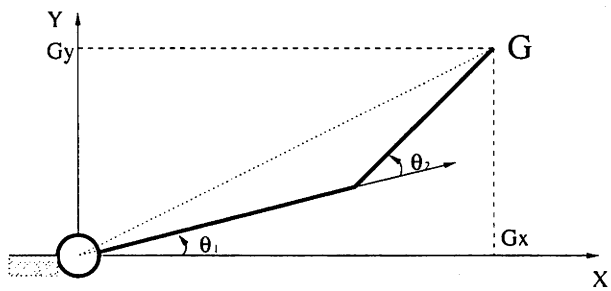

Figure 4-1: Two-link planar arm

For a end-effector location $\mathbf{G}(G_x, G_y)$, the following equations can be obtained:

$$\left.\begin{aligned} G_x &= a_1\cos\theta_1 + a_2\cos(\theta_1 + \theta_2) \\ G_y &= a_1\sin\theta_1 + a_2\sin(\theta_1 + \theta_2) \end{aligned}\right\} \tag{4.1}$$

Squaring and summing the equations yields:

$$G_x^2 + G_y^2 = a_1^2 + a_2^2 + 2a_1a_2\cos\theta_2$$

From which

$$\cos\theta_2 = \frac{G_x^2 + G_y^2 - a_1^2 - a_2^2}{2a_1a_2}.$$

The existence of a solution obviously imposes that $-1 \leq \cos\theta_2 \leq 1$, otherwise the given point would be outside the arm reachable workspace. If this condition is satisfied then two solutions for θ_2 can be computed as

$$\theta_2 = \pm\arccos\frac{G_x^2 + G_y^2 - a_1^2 - a_2^2}{2a_1a_2} \tag{4.2}$$

Substituting into equation (4.1) yields an algebraic system of two equations in which there are two unknown $\sin\theta_1$ and $\cos\theta_1$, whose solution is

$$\sin\theta_1 = \frac{(a_1 + a_2\cos\theta_2)\,G_y - a_2\sin\theta_2 G_x}{G_x^2 + G_y^2}$$

$$\cos\theta_1 = \frac{(a_1 + a_2\cos\theta_2)\,G_y + a_2\sin\theta_2 G_x}{G_x^2 + G_y^2}$$

Thus for each θ_2 solution a unique value of θ_1 can be computed

$$\theta_1 = atan2\,(\sin\theta_1, \cos\theta_1) \tag{4.3}$$

From the equation (4.2) above we know that for a each end-effector position, there are two arm solutions. We call the one with negative θ_2 'elbow up' and the one with positive θ_2 'elbow down' as shown in Figure (4-2).

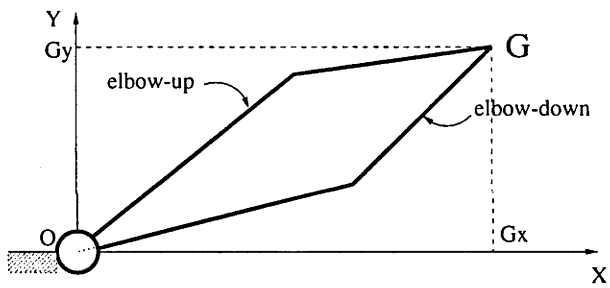

Figure 4-2: Two arm solutions

If we put all arm solutions in a graph, we can get a initial model of joint space graph. Figure (4-3) shows the relationship between the task space graph and joint space graph. The task-space graph **T** is generated by dividing the task space into uniformly-space vertices. Edges connect each vertex to its four nearest neighbour. The joint space graph is then determined with vertices that map onto the task space graph through the robot kinematics. As shown in Figure (4-3), the joint space graph **J** is composed of two sheets that correspond to the two different inverse kinematic solutions: elbow up and elbow down. The vertices on each sheet are connected to four neighbours as in the task-space graph. In addition, there are some edges that connect vertices (e.g. i_1 and i_2), that are not on the same sheet, these edges are determined as follows.

Consider that most manipulators have configurations where the Jacobian is singular, these configurations are often places where two or more branches of the inverse kinematic join together. For the robot arm considered these correspond to task space

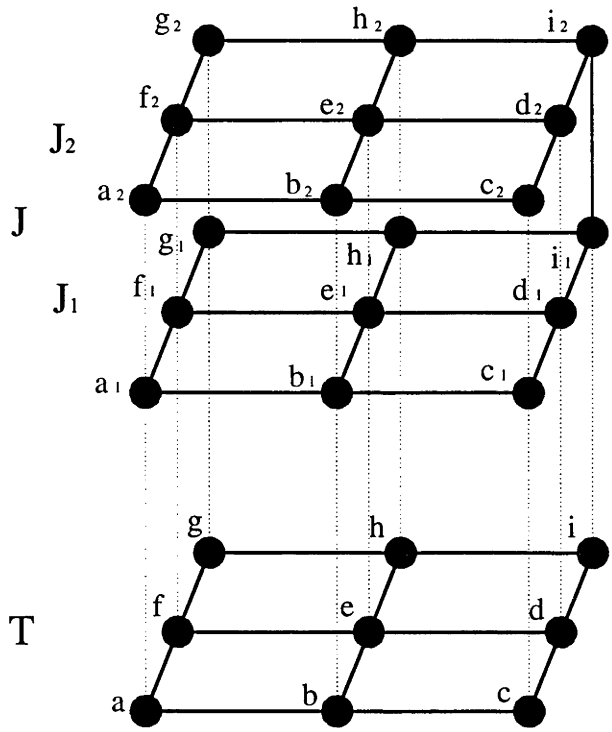

Figure 4-3: Corresponding relationship between task space graph and joint space graph

vertices which only have one solution. An obvious position where this occurs is at the boundary of the workspace of the robot. Our two-link manipulator exhibits two workspace boundary singularities: One when the arm is fully retracted (the angle between the links is zero), and one when it is fully extended (the angle between the links is π). So there is a possibility to connect two 'sheets' $\mathbf{J_1}$ and $\mathbf{J_2}$ together when the two solutions pass though kinematic singularities.

A task space vertex may not be exactly on the boundary. So, for most cases, we assume a motion of robot arm in Figure(4-4), which is from one arm posture to another. Obviously, the motion is easier to perform when a vertex is near the boundary.

For a vertex near outer boundary $\mathbf{G_1}$, for an example, the easiest motion of moving from one arm posture to another is to move from one arm solution vertex to another. In an unknown environment, the same location motion can be achieved when shadow region S between the two posture and near the end-effector area is clear for both robot arm and end-effector. Hence, it is reasonable to assume that: the smaller the area of

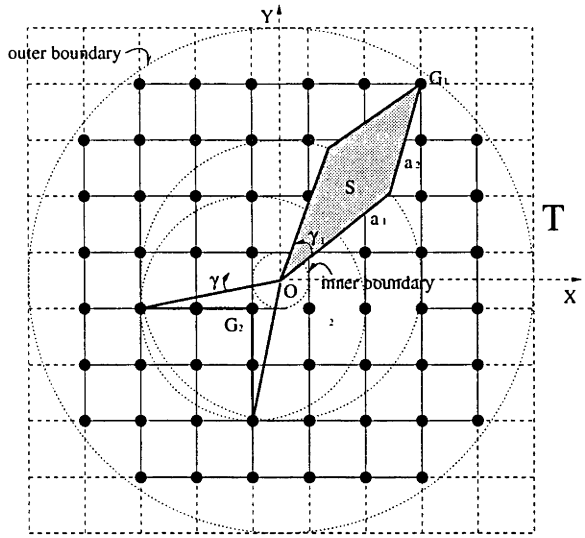

Figure 4-4: Connected two-solution

S, the more possible the motion can be achieved. Because the area of S depends not only on the angle γ_1 but also on the length of the robot arm a_1 and a_2, we define an edge between the two arm solution vertices if the length of distance OG_1 satisfied

$$(a_1 + a_2) \geq \sqrt{x^2 + y^2} \geq 0.8(a_1 + a_2) \tag{4.4}$$

For a vertex near inner boundary, $\mathbf{G_2}$ for an example, if there is no joint limitation, there exist the same location motion as near outer boundary. So we define an edge between the two arm solution vertices if the length of distance OG_2 satisfied equation (4.5). If there is joint limitation, then there don't exist such edges. Here, we only consider the situation $a_1 \geq a_2$.

$$(a_1 - a_2) \leq \sqrt{x^2 + y^2} \leq 1.5(a_1 - a_2) \tag{4.5}$$

Now, we can connect the two 'sheets' $\mathbf{J_1}, \mathbf{J_2}$ of Figure (4-3) according to the equation (4.4), (4.5) and put them into a same 'sheet'. We call it a 'double sheet' because it is formed by two 'sheets'. So we gain a model of a two-link robot arm as a 'double sheet'. The model is presented in the Figure (4-5), which is mapped from a task space shown in Figure (4-6). In Figure (4-6), there are 8 vertices satisfy equation (4.4), 4 vertices satisfy equation (4.5). Therefore, there are 12 edges between elbow-up solution vertices and elbow-down solution vertices.

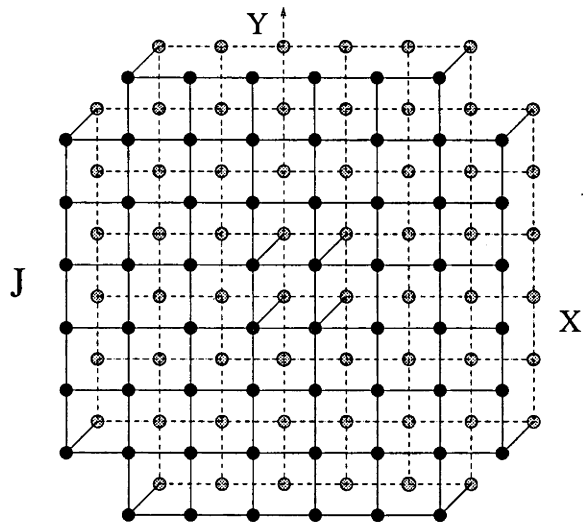

Figure 4-5: Model of two-link planar robot arm

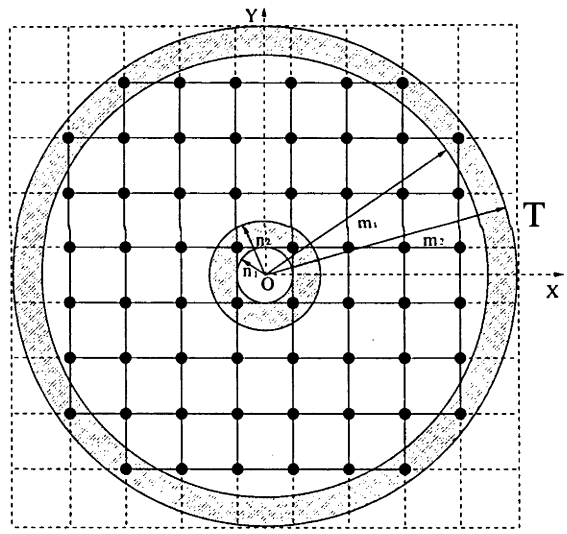

Figure 4-6: The work space for a two-link robot arm

4.4 Model of Three-link Planar Robot Arm

Now, we build a model of three-link planar robot arm in much the same way as building a model of two-link planar robot arm described above.

A three-link planar robot arm is illustrated in Figure (4-7). It is redundant with respect to the task of positioning its tool point **P**. In terms of a minimal number of parameters, it is convenient to specify position and orientation by the two coordinates $\mathbf{G_x}$, $\mathbf{G_y}$ and the angle ϕ, which are related to the tool point position by the following equations.

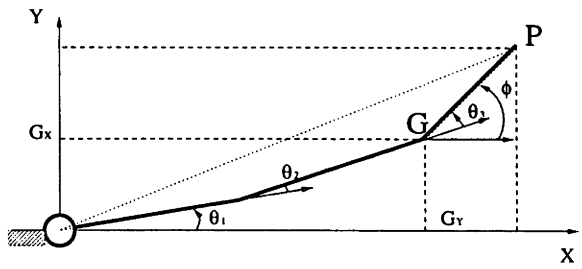

Figure 4-7: Three-link planar arm

$$\phi = \theta_1 + \theta_2 + \theta_3 \tag{4.6}$$

$$\left.\begin{aligned} G_x &= P_x - a_3 \cos\phi \\ G_y &= P_y - a_3 \sin\phi \end{aligned}\right\} \tag{4.7}$$

Therefore, for a given vertex $P(P_x, P_y)$, if ϕ is known, then G_x and G_y is known. Variables θ_1 and θ_2 can then be computed by equation (4.3), (4.2) presented in the section (4.3). Obviously, for a given tool position $\mathbf{P}$, there exist infinite solutions to the equation (4.7) with ϕ serving as an independent parameter. In order to get a joint space graph, ϕ is artificially specified as $\mathbf{0}, \pi/4, \pi/2, ..., 7\pi/4$. For a each ϕ, there may be two arm solutions associating with elbow-down and elbow-up (Here, elbow-down and elbow-up is based on link 1 and link 2). Hence, there may exist 16 arm solutions for a given end-effector position Figure (4-8).

Let us focus careful attention on Figure (4-8). Firstly, there are two parts $\mathbf{J_1}$, $\mathbf{J_2}$ in it. $\mathbf{J_1}$ includes all elbow-down solutions and $\mathbf{J_2}$ includes all elbow-up solutions. Secondly, each part has 8 'sheets' respectively. Thirdly, each 'sheet' is associated with a specified ϕ. In addition, each task space vertex has 16 joint space vertices that map onto it. Vertex **a** is an example. Finally, vertical vertices in each part connect to each other. Table (4.1) presents an explanation of Figure (4-8).

Similar to the way that we built the model of the two-link robot, we put elbow-up and elbow-down solution for the same specified ϕ together and define an edge between the two sheet vertices when they satisfy the condition equation (4.4) and equation (4.5). So we put J_{11} and J_{21} in a 'sheet' and get the first 'double sheet' J_1 in joint space J, put J_{12} and J_{22} in a 'sheet' and get the second 'double sheet' J_2 in joint space J ...

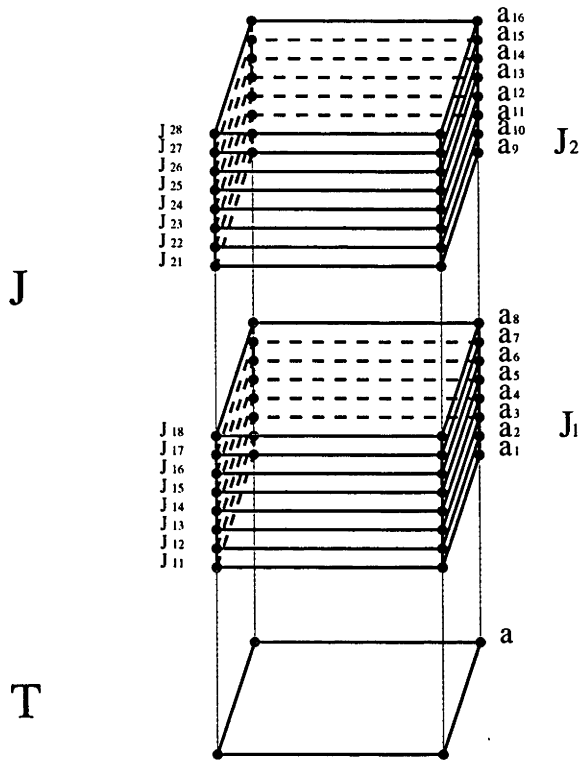

Figure 4-8: Relationship of joint space and task space

'sheet'	ϕ	arm	'sheet'	ϕ	arm
J_{11}	0	elbow-down	J_{21}	0	elbow-up
J_{12}	$\pi/4$	elbow-down	J_{22}	$\pi/4$	elbow-up
J_{13}	$\pi/2$	elbow-down	J_{23}	$\pi/2$	elbow-up
J_{14}	$3\pi/4$	elbow-down	J_{24}	$3\pi/4$	elbow-up
J_{15}	π	elbow-down	J_{25}	π	elbow-up
J_{16}	$5\pi/4$	elbow-down	J_{26}	$5\pi/4$	elbow-up
J_{17}	$3\pi/2$	elbow-down	J_{27}	$3\pi/2$	elbow-up
J_{18}	$7\pi/4$	elbow-down	J_{28}	$7\pi/4$	elbow-up

Table 4.1: Explanation of joint space

put J_{18} and J_{28} in a 'sheet' and get another 'double sheet' J_8 in joint space J. This finally provides a model of a three-link arm.

Figures (4-9)- (4-16) show how each of the 'double sheet' is formed. In these figures, the left figure represents the task space. For a three-link arm shown in Figure (4-7), the two non-shadowed circle rings show the position of **G** which satisfied the equation

(4.4) and equation (4.5). If the robot arm moves its work space with **G** in the two non-shadowed circle rings and keeps the same ϕ, then we get the other two shadowed circle rings which shows the end-effector position **P**. Therefore, if a vertex is in the shadowed circle rings, that means the task space vertex has two connected vertices (elbow up vertex and elbow down vertex) in its joint space graph. The right figure shows its associated 'double sheet'.

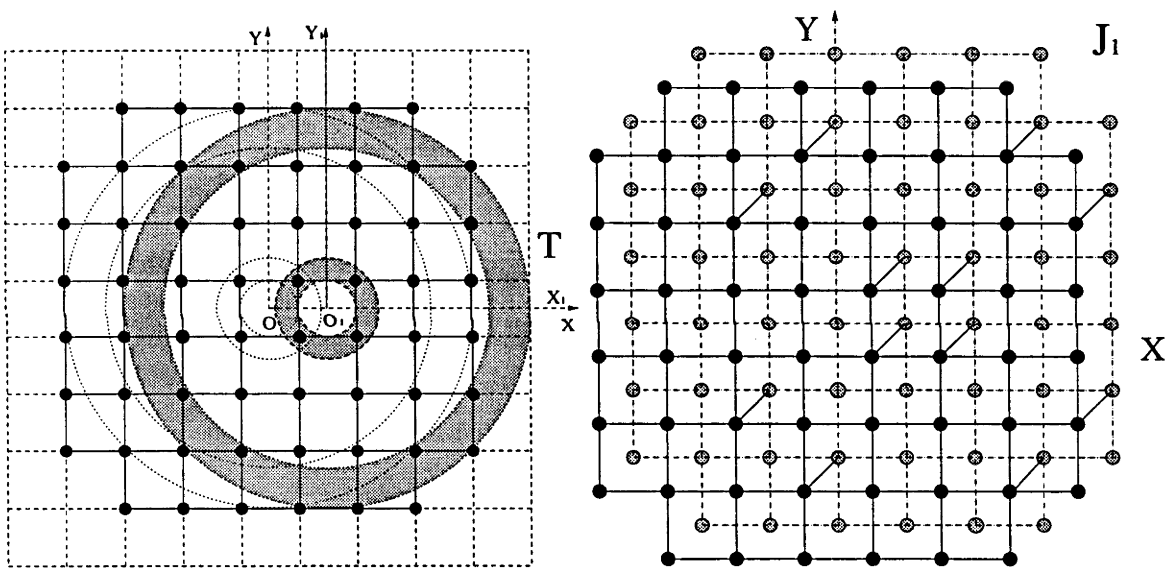

Figure 4-9: Sheet 1, $\phi = o$

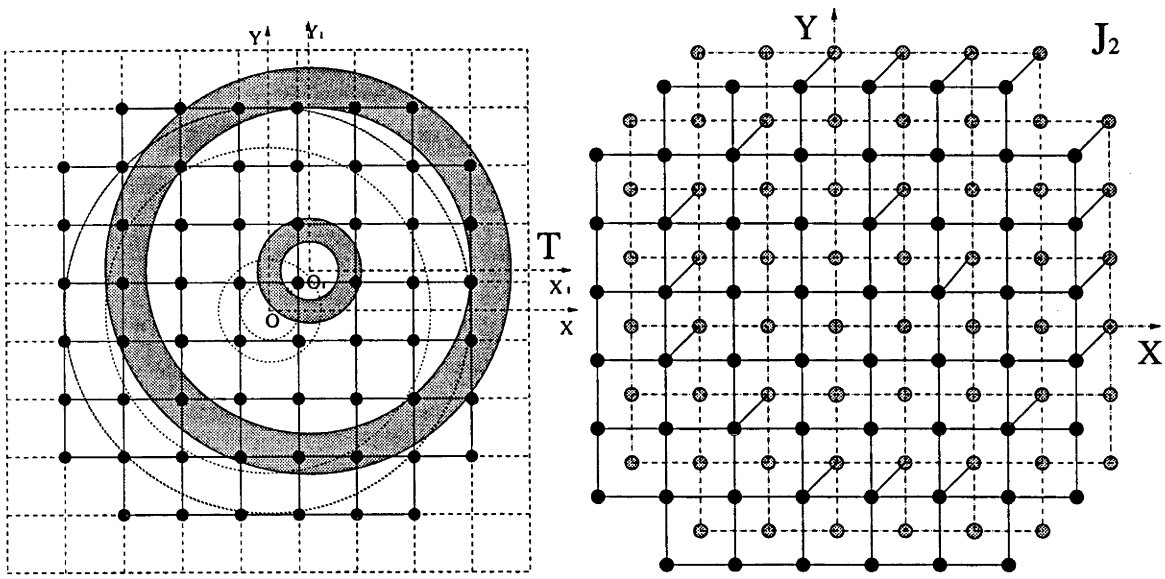

Figure 4-10: Sheet 2, $\phi = \pi/4$

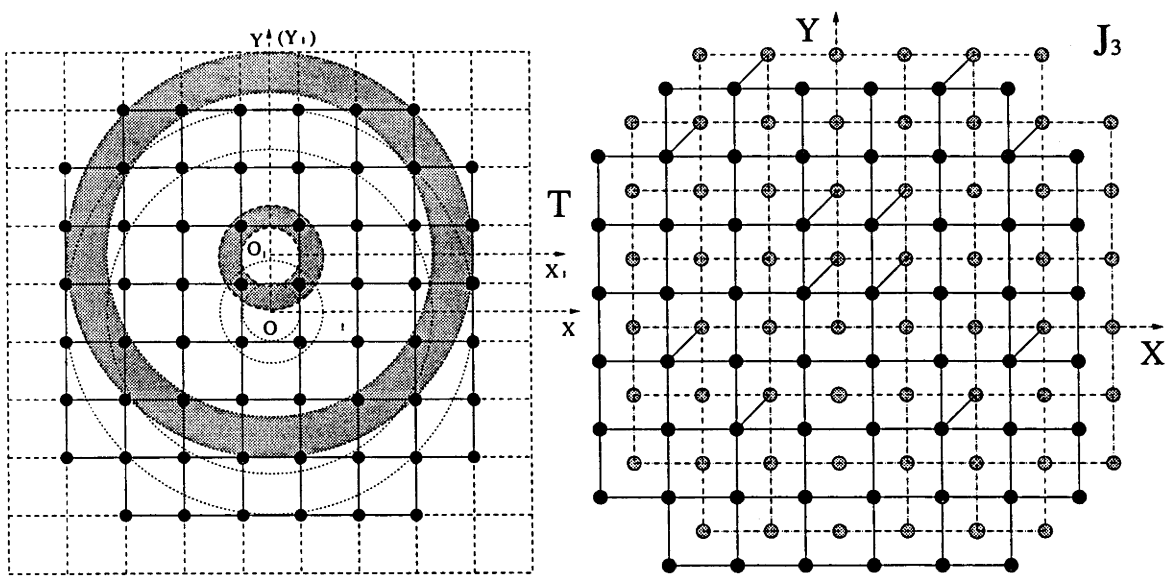

Figure 4-11: Sheet 3, $\phi = \pi/2$

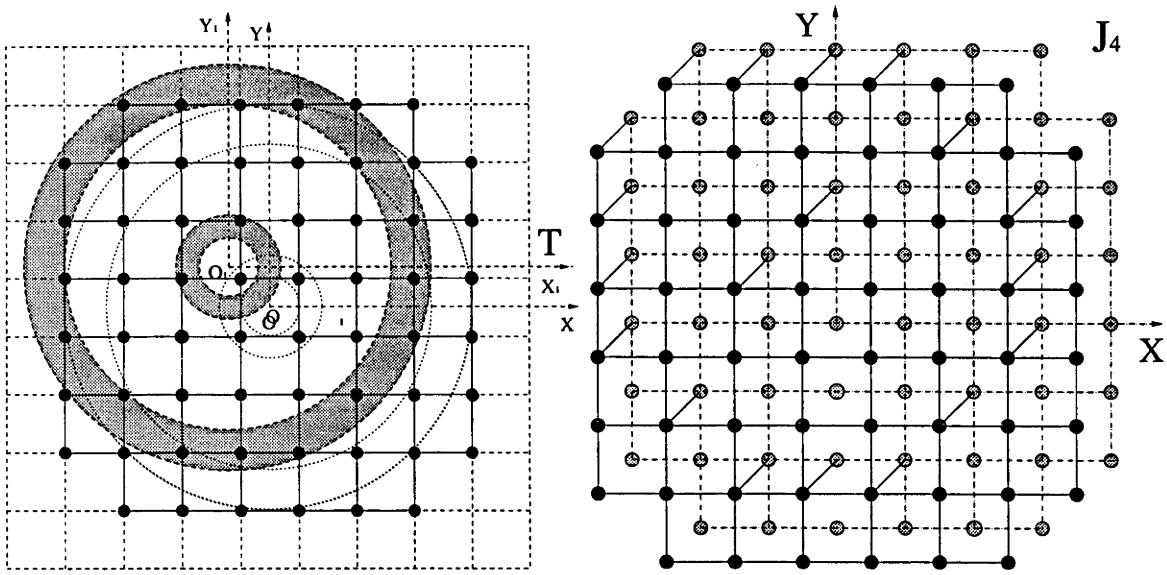

Figure 4-12: Sheet 4, $\phi = 3\pi/4$

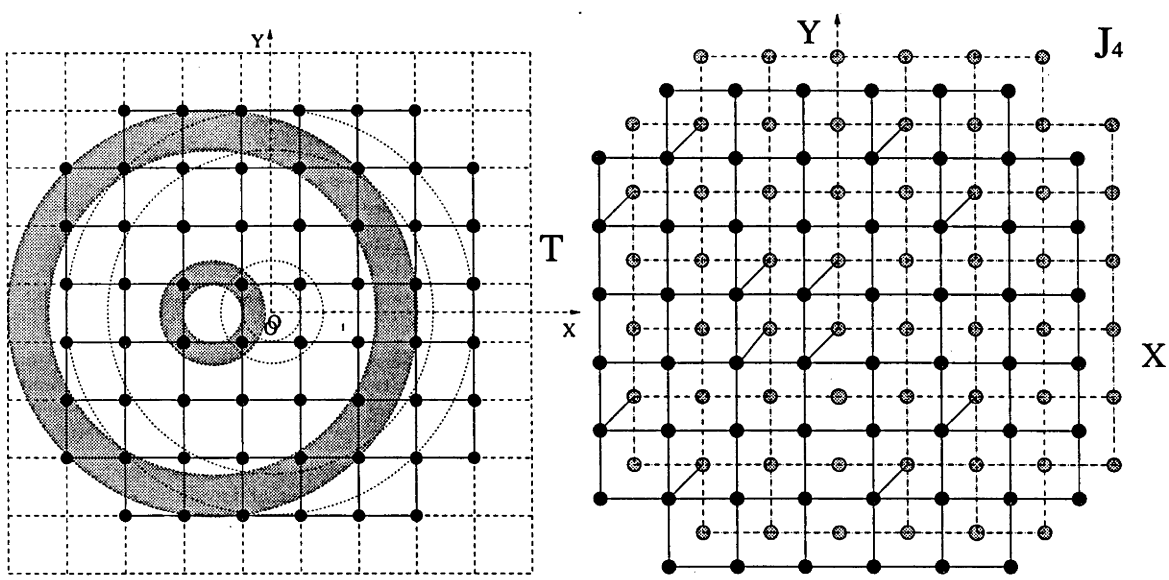

Figure 4-13: Sheet 5, $\phi = \pi$

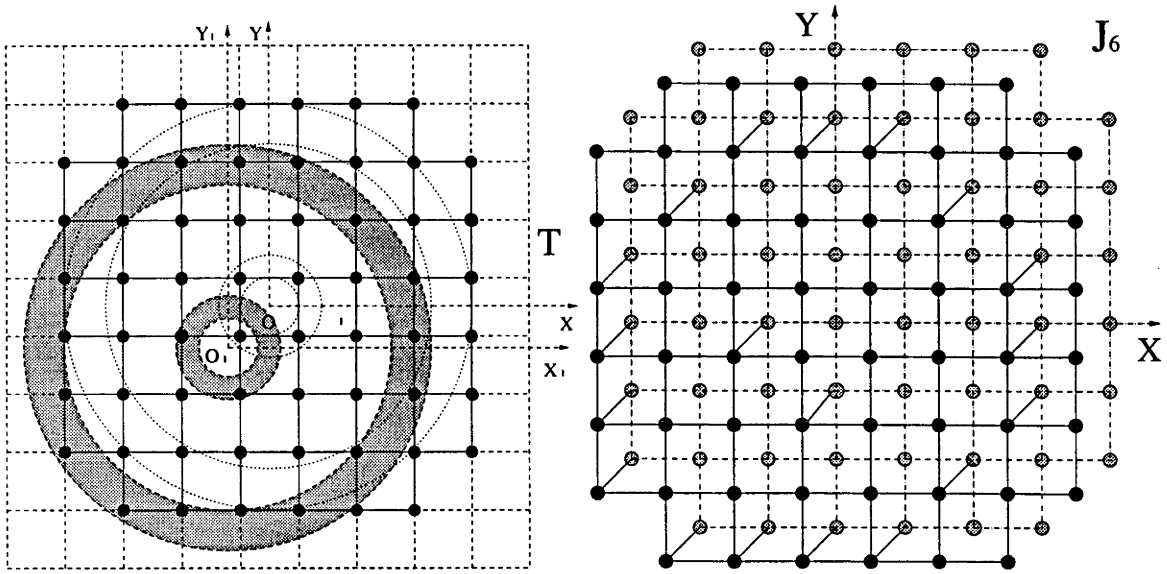

Figure 4-14: Sheet 6, $\phi = 5\pi/4$

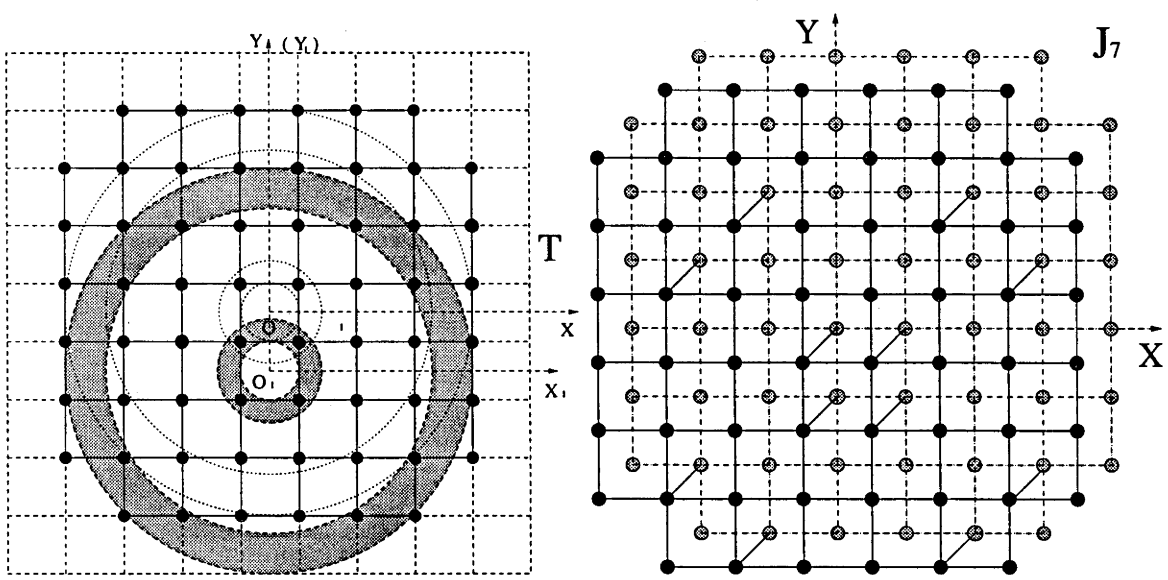

Figure 4-15: Sheet 7, $\phi = 3\pi/2$

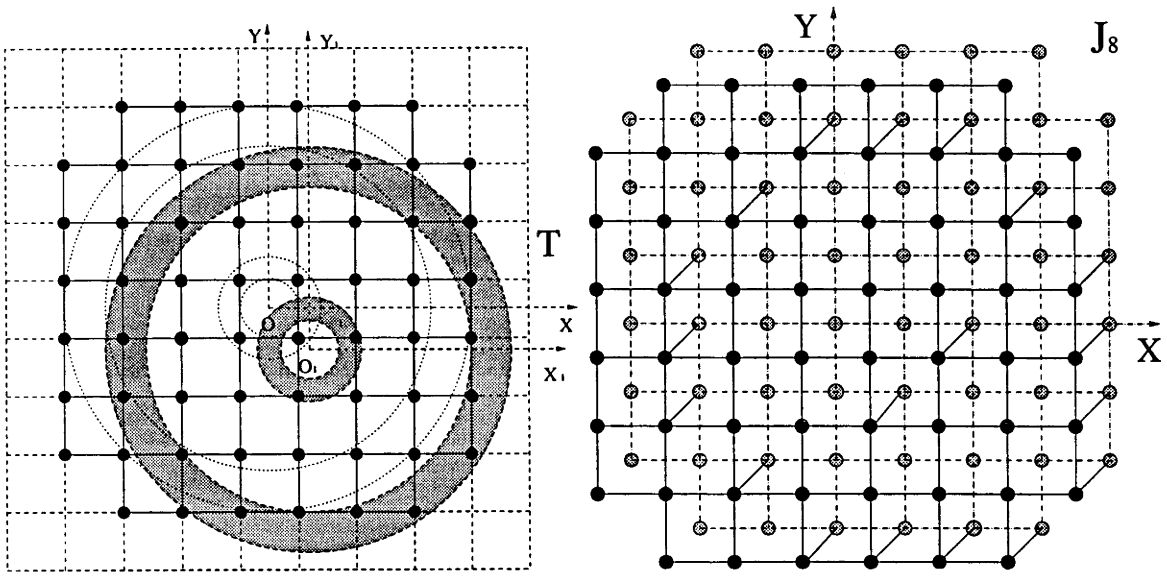

Figure 4-16: Sheet 8, $\phi = 7\pi/4$

4.5 Discussion

In this chapter, we have developed a method for determining joint-space graph models of manipulators. In the models, the vertices represent configurations of the manipulator. The edges represent motions between configurations. And at least two joint space vertices map on a task space vertex. In such a manipulator graph, several vertices map onto the same task space goal. That means when a task space vertex is specified as goal, then a set of joint-space vertices which map on the task space vertex are treated as goals. Once the task space vertex is reached, the other vertices that map onto the same task space position are no longer goals. So, a graph search problem for a manipulator can be formulated as following:

The Find-goal Problem

1. Make a task space graph and identify the goal vertices on it.
2. Use inverse kinematics to identify the joint space model.
3. Identify all joint-space vertices that map onto the task-space goal vertices as goal vertices in the joint-space graph.

4. Initialise the current vertex of the joint-space graph to the initial configuration of the manipulator.

5. Set the status of all vertices to unknown.

6. Apply a goal-seeking algorithm to the joint-space graph

The Cover-goal Problem

1. Make a task space graph and mark all vertices as goals

2. Use inverse kinematics to identify the joint space model.

3. Mark all joint-space vertices as goals.

4. Initialise the current vertex of the joint-space graph to the initial configuration of the manipulator.

5. Set the status of all vertices to unknown.

6. Apply the goal-seeking algorithm to the joint-space graph

7. While reachable goals remain

   (a) Remove the found task space vertex from the set of task space goals

   (b) Re-identify the joint space goal vertices

   (c) Apply the goal-seeking algorithm to the joint space graph

For a two-link planar arm we get a 2D joint space graph, while three-link planar arm we get a 3D joint-space graph. It is easy to imagine that the joint-space graph should be more complicated for 4 or more links manipulator. However, as long as the manipulators motion can be modelled by a graph, then a similar graph-search algorithm can be developed to suit them all. We use neighbours to indicate edges that connect vertices. All neighbours are assumed to be equally close, so the resulting graph is unweighted. If an edge is found closed, then its neighbour is removed. The advantage of the data structure is:

1. It accurately represents models of different shape in 2D or 3D.

2. It can handle both local and global maps.

3. It does not rely on the assumption that the model is true to square, as invariably it is not.

4. It leads to efficient algorithms for graph searching.

5. It is easy to integrate sensor data with simple algorithms for map construction, adaption and extension.

In the next chapter, we describe our algorithms for sensor-based graph searches that are not specific to any particular model.

Chapter 5

Sensor-based Motion planning

5.1 Introduction

Sensor-based motion planning is important for robots to work in unknown environments. Because such robots have no a priori knowledge of the world, they must replan motion on-the-fly as new information is acquired. It is different from ordinary motion planning because the robot does not have full information about the environment. Sensor based planning must weave the latest sensor information into robot's planning process, and each search is not isolated. When a new obstacle is encountered, robots will replan based on the current state of environment which has probably only changed incrementally with respect to its previous state. Algorithms to support such "sensor-based motion planning" can be compared with respect to minimising robot motion and minimising online computation.

In this chapter, we introduced two sensor-based motion planning algorithms: The Optimistic Shortest Path Algorithm (OSP) and The Pruned Depth-First Search (P-DFS).

The OSP algorithm, such as [6] and [8], rely on the following shortest-path strategy: determine the path from the current position to the goal that appears to be shortest, based on current knowledge combined with the free-space assumption. Then move along it. When new obstacles are encountered, the "shortest path" may need to be revised, but the policy (to follow it) remains the same. OSP is popular because it is known to converge for finite graphs and to provide good average-sense performance (not in a worst-case). Our OSP algorithm described in section 5.4 inherits the shortest-path

strategy and focusses on reducing on-line computation by an efficient graph marking and updating algorithm detailed in chapter 6.

The OSP algorithm provides good average-sense performance, however, its worst-case performance is less tightly bounded than any depth-first search (DFS). Furthermore, there is no evidence that OSP is optimal against any specific criteria. DFS algorithms are practical for robots to physically implement because they require no "jumps" by the search agent. DFS algorithms cover graphs with no more than two traversals of each edge in the spanning tree they generate. This bounds the cost of a goal search to twice the weight of the heaviest spanning tree. No search algorithm can provide a better worst-case performance bound. Nevertheless, DFS average performance is thought to be inferior to OSP because, even when guided by a heuristic, an early mistake can force DFS to explore large areas of the graph that are distant from the goal.

In section 5.5 we present a DFS algorithm that we believe has average performance comparable to OSP. The algorithm combines DFS with a shortest-path heuristic that has a self-blocking property that keeps the robot from exploring subgraphs that are obviously void of goals. We imagine that it is "pruning" fruitless branches from the tree search and have named it Pruned DFS(P-DFS). P-DFS is still a depth-first search, but not a complete one. Also, we use an efficient graph marking and updating algorithm to reduce computation, which is the same as the one used in OSP algorithm.

The efficient graph marking and updating algorithm provided to both OSP and DFS algorithms is essentially a simplified version of Stentz's D* algorithm. But it is easier to implement and are more efficient for uniform graphs searches.

This chapter introduces the two algorithms respectively in section 5.4 and section 5.5. Our efficient graph marking and updating algorithm is the same for both algorithms, so we introduce it for both in chapter 6

5.2 Definitions

- **visited vertex**: a vertex that is either the current vertex, or has been the current vertex at one time.

- **unvisited vertex**: a vertex that has never been the current vertex.

- **goal vertex**: any vertex that is desirable to reach (there may be one, or many).
- **unknown edge**: an edge whose motion has not been tried.
- **open edge**: an edge whose motion has been successfully completed.
- **closed edge**: an edge whose motion has been tried and failed.
- **neighbour**: two vertices are neighbours if they share an un-closed edge(an edge with status unknown or open).
- **source neighbour**: the neighbour that the current vertex was first entered from.
- **mark**: an integer assigned for motion planning purposes.
- **high neighbour**: a neighbour with a higher mark
- **low neighbour**: a neighbour with a lower mark
- **level neighbour**: a neighbour with the same mark.

5.3 Summary of OSP and P-DFS

Implementation of these algorithms is very similar. Let each vertex V be marked with the length of the shortest path from the current vertex to the goal through.v. The freespace assumption should be used and, for P-DFS only, visited vertices should be treated as obstacles. Given a correctly marked graph the motion planning algorithm for each step is:

OSP: If one or more neighbours are marked, then try to move to the lowest one. Else stop (goal is unreachable).

P-DFS: If one or more neighbours are marked, then try to move to the lowest one. Else search the trunk of the visited tree for marked neighbours of visited vertices. If found, then backtrack to the first one. Else stop (goal is unreachable).

5.4 The Optimistic Shortest Path Algorithm

We are interested in the OSP algorithm because it has good average-sense performance. Furthermore, although depth-first search has the best worst -case performance bound,

it is arguable that this bound may be impractical for typical problem. In addition, other algorithms may out perform it in an average sense. The main drawback of the depth-first algorithm is that it artificially constrains motion to a tree. If the most optimistic move is to a visited neighbour, the directed depth-first algorithm will not take the most optimistic move until it has exhaustively searched the subtree of each unvisited neighbour. In the OSP algorithm, each motion is the most optimistic for the current state of information. Because motion is not constrained to a tree, optimal worst-case bounds on termination have been sacrificed. However, the algorithm is guaranteed to terminate for finite graphs, and it is reasonable to expect that it will perform well, especially in relatively uncluttered environments.

1. **Marking algorithm**

   - Given: a graph with edges in any state (open, closed, unknown), vertices in any state (visited, unvisited), and a set of goal vertices on that graph.
   - Consider that all vertices are initially unmarked
   - Mark all goal vertices with distance = 0
   - set current_distance =0
   - while there exist unmarked neighbours of marked vertices
     - identify all unmarked neighbour vertices
     - mark them with distance = current_distance + 1
     - set current_distance = current_distance + 1

2. **Motion planning algorithm**

   The OSP algorithm can be described as follows:

```
mark the graph using the marking algorithm
while the current vertex is not a goal vertex
      if current vertex has no marked neighbour
         stop(terminate because no goals can be reached
      else
            go to a lower mark vertex
            if goal reached
               stop
            else update graph with new-found closed edge
            end
      end
end
```

5.5 The Pruned Depth-first Search Algorithm

1. **Three strategies to the P-DFS algorithm**

   Interestingly, the attractive worst-case performance bound that DFS algorithms enjoy in goal seeking is achieved without using knowledge of the goal's location. An undirected depth-first search (using Tarry's algorithm: backtrack if all neighbours of current vertex have been visited, otherwise move to an unvisited neighbour) will stumble on the goal in the process of systematically exploring the reachable subgraph. Furthermore, any heuristic can be added to choose between unvisited neighbours without disturbing the worst-case performance bound.

   To improve average performance, we use three strategies in the P-DFS algorithm: 1) marking only unvisited vertices 2) pruning rule 3) a goal-seeking heuristic.

   (a) **Marking algorithm:**

   The graph marking algorithm is modified to propagate marks only through *unvisited* vertices. Each vertex that can reach a goal through a path of only unvisited vertices (under the optimistic assumption that all unknown edges are open) is marked with a distance. The distance represents the number of edges in the shortest unvisited path to the goal.

- Given: a graph with edges in any state (open, closed, unknown), vertices in any state (visited, unvisited), and a set of goal vertices on that graph.
- Consider that all vertices are initially unmarked
- Mark all goal vertices with distance = 0
- set current_distance =0
- while marked vertices have unmarked and unvisited neighbours
  - identify all unmarked and unvisited neighbours of vertices that have been marked
  - mark them with distance = current_distance + 1
  - set current_distance = current_distance + 1

It is obvious that the marking algorithm has the following characters.

i. If a vertex is unmarked and unvisited, then there is no path from it to any goal vertex (even under the optimistic assumption that all unknown edges will be open).

ii. If any vertex is marked, then all of its unvisited neighbours will be marked. (proof: the marking algorithm propagates marks through unvisited neighbours).

iii. All marked vertices, except goal vertices, have at least one neighbour with a lower mark than itself.

The reason for not allowing visited vertices to be marked is that the tree-structured search does not allow them to be used. Although the joint-space graph will generally contain loops, the chronological-backstepping algorithm refuses to use any. The algorithm imposes an artificial tree structure on the paths that it will move on. For example: when the current vertex moves, it may find new neighbours that have already been visited, This is an opportunity to make a loop, but the algorithm won't try to go to these neighbours because it only allowed to return to previously-visited vertices through chronological back-steps.

The chronological backstepping algorithm produces the tree-structured graph when searching the graph. The search tree is not unique and is not known ahead of time. The search tree evolves starting from the initial vertex (the

tree root) and grows whenever an unvisited neighbour is successfully moved to. Each visited vertex can eventually be associated with a branch of the tree (possibly null) that is linked to the root only through itself. However, this branch is not defined when the vertex is first visited. It is fully defined when the vertex is revisited through backstepping. If the chronological backstepping algorithm returns to a vertex, then it returns from a branch of the search tree that has been exhaustively searched without finding a goal. The reason is because the algorithm only returns to vertices through chronological back-steps.

This marking scheme which avoids loops leads to the possibility that some unvisited vertices(those which are surrounded by visited vertices) may not be marked.

(b) **Pruning rule:** *ignore unvisited neighbours that are cut off from the goal by a subgraph of visited vertices.*

Because a tree search cannot cross itself(except when backtracking), it serves no purpose to explore subgraphs that can only reach a goal through already visited vertices. P-DFS still provides a depth-first tree search, but not a complete one. It avoided searching fruitless branches.

The to-be-ignored neighbours (roots of fruitless branches) are easily identified when computing shortest paths for the goal-seeking heuristic: they are the ones that provide no path to the goal. Also they will be unmarked by the modified marking algorithm.

(c) **heuristic:** *move to the unvisited neighbour that is nearest to goal under the freespace assumption. By nearest we mean the one that is on the shortest path of unvisited vertices from the current vertex to the goal.*

Visited vertices are treated as obstacles because a DFS can only use them in backtracking. This heuristic implies that P-DFS uses the same strategy as OSP. But it does so within the constraints imposed by a tree-structured search. So P-DFS improves performance without disturbing the worst-case bounds.

2. **Algorithm**

The P-DFS algorithm can be described as follows:

```
mark the graph using marking strategy
while the current vertex is not a goal vertex
   if all neighbours of the current vertex are unmarked
      search the tree of visited vertices for marked neighbours
      if none found
         stop(terminate because no goals can be reached)
      else
         back step to first visited vertex with a marked neighbour
      end
   else(the current vertex has at least one marked neighbour)
      try to move to the marked neighbour that has the lowest distance mark
      (if several neighbours share the lowest distance, them choose arbitrarily)
      if successful
         update current vertex
         update graph marking
      end
   end
end
```

The P-DFS algorithm has the following characters.

(a) All moves, or attempted moves are either to marked vertices or are chronological backsteps.

(b) The graph marks have to be updated whenever the robot moves successfully(because an unvisited vertex becomes visited), but not when it discovers a closed edge(proof: the current vertex does not need remarking because it is already an unmarked visited vertex, The marked vertex does not need remarking because its low neighbours remain unchanged by the edge closure)

(c) The algorithm terminates only when a goal is reached, or when the graph marks indicate that all goals are unreachable

(d) Chronological backstepping imposes a tree-structure on the way the graph is traversed. Graph edges that are not consistent with the tree are never tried. (Proof: no attempt is made to move to a visited neighbour, except through backstepping).

(e) If the reachable graph has N vertices, then the algorithm will terminate after no more than 2(N-1) successful edge transversal. (Proof: a tree-structure graph with N vertices has N-1 edges. Chronological backstepping ensures that no edge will be traversed more than twice).

(f) If the reachable graph has N vertices and M edges (including edges that join to unreachable vertices), then the total number moves attempted before termination is bounded by M+N-1. (Proof: The coverage tree of the reachable graph will have N-1 edges. This leaves M-(N-1) edges for unsuccessful moves).

(g) The algorithm can terminate without complete coverage of the reachable graph if marking indicate that all goals are unreachable

Chapter 6

Efficient graph marking and updating

6.1 Introduction

Different from ordinary searches in which each search is isolated, sensor-based planning is a sequence of searches that are very similar to each other. When a new obstacle is encountered, the new search will differ from the previous search only by an incremental change to the environmental model (graph). This implies that there is a potential to reduce computation if data from one search can be reused in the next, rather than regenerated from scratch. Recognising this, Stentz [8] modified the A* search algorithm to incrementally repair, rather than regenerate, its field of optimal-path pointers.

In the chapter, we present a new algorithm to minimise replanning computations on uniform graphs by incrementally repairing data. This algorithm is essentially a simplified version of Stentz's D* algorithm that it is easier to implement and are more efficient for uniform graphs searches. The algorithm is provided to both OSP and DFS algorithms.

The principles we use to reduce marking are essentially the same as Stentz uses in his D* algorithm [8].

1. mark no more than is immediately needed for motion planning
2. when new information changes the graph, update only the vertices that are effected

6.2 Summary of algorithm

We use propagation to initially generate marks, to clear marks, and to remark vertices. Changes are seen as being triggered by events that directly invalidate the mark of a small number of easily identified vertices that we call invalid roots (see below). From the invalid roots, changes can potentially propagate through large regions that we call invalid subgraphs. The correction process has two stages: 1) identify the invalid subgraphs and clear marks from them, then2) re-propagate marks back into these subgraphs from correctly-marked neighbours. To minimise computation the initial marking and subsequent clearing/ re-marking processes are only done to the extent required to plan the robot's next motion (until a neighbour of the current vertex is marked).

6.3 Incremental repair algorithm in detail

The concepts of *dependency* and *support* are useful for describing the algorithm. Any marked vertex (other than the goal) has at least one low neighbour (a neighbour with a lower mark than it has). If a vertex has only one low-neighbour than it *depends* on that neighbour for its mark (recall how marks are generated by a breadth-first search from the goal). Such dependency can propagate through many vertices. Each vertex is *supported* by its low neighbours in the sense that it depends on them for its mark.

Mark corrections are triggered by different events in the two search algorithms, but in either algorithm the invalid rots are easily identified:

OSP: if the current vertex has only one low-neighbour and fails in an attempt to move to it. Then that edge is removed from the graph and the current vertex is left unsupported. It is the invalid root.

P-DFS: if the current vertex moves to a marked vertex, then that vertex becomes visited and its mark must be cleared. This leaves any vertices that depended on the newly-cleared vertex unsupported. They are the invalid roots.

Once the roots of the invalid subgraph have been identified, the correction process is to identify and clear marks from the invalid subgraph, then re-propagate mark from supported neighbours(if any exist) .

Mark clearing is accomplished through incremental propagation. A stack called

clear_stack is defined to keep track of undone clearing.

clear_stack = stack of all marked vertices that have no low-neighbours (low marks on top)

The following algorithm performs one incremental propagation of the clearing process. Note that in addition to clearing marks, clear_mark identifies supported neighbours of the newly-cleared vertices and places them in a stack called mark_stack.

clear_next

```
clear_frontier = pop lowest vertices from clear_stack
more_to_clear = dependent neighbours of clear_frontier
marked_boundary = other( non- dependent) marked neighbours
                  of clear_frontier
clear marks from clear_frontier
insert more_to_clear into clear_stack
insert marked_boundary into mark_stack
```

Re-marking: Suppose that marks have been cleared from an invalid subgraph and the supported neighbours to that subgraph have been identified and placed in a stack, called *mark_stack.*

mark_stack = stack of all marked vertices that have unmarked neighbours (low marks on top)

Then remarking can be accomplished by repeated application of the following algorithm which provides one incremental propagation of marks.

mark_next

```
mark_frontier = pop all lowest- vertices from mark_stack
remove unmarked vertices from mark_frontier
more_to_mark = unmarked neighbours of mark_frontier
mark more_to_mark with mark( mark_frontier) + 1
push more_to_mark onto mark_stack
```

Interleaving marking and clearing. Now we wish to interleave these algorithms so that marking, clearing, and re-marking is only done to the extent immediately required for path planning. The following function, propagate_marks, incrementally propagates marks once using mark_next, but before doing so it applies clear_next as many

times as necessary to ensure that enough vertices are cleared for one correct incremental propagation of marks. If the graph has been exhaustively marked, propagate_marks returns marking_complete = 1. Function *lowest()* in line 5 returns the lowest mark in the stack.

marking complete = propagate marks()

```
while mark_stack = empty & clear_stack ≠ empty
                    clear_next
end
if mark_stack ≠ empty
        while clear_stack ≠ empty&
       lowest(clear_stack) <= lowest(mark_stack) + 2
        clear_next
        end
        mark_next
        marking_complete = 0
else
        marking_complete = 1
end
```

The condition italicised in line 6 insures that each vertex in clear_stack is at least 3 edges (and 2 clear vertices) away from each vertex in mark_stack when mark_frontier is propagate (see theorem below). This buffer is a little bigger than necessary, but it simplifies the propagation of clear_frontier if there is at least one cleared vertex between it and mark_frontier after mark_frontier is propagated.

Theorem 1: condition (*) insures that there are at least N+1 edges and N clear vertices between any vertex in clear_stack and any vertex in mark_frontier.
lowest(clear_stack) > lowest(mark_stack) + N (*)

Proof of Theorem

1. After re-marking, all marks in an invalid subgraph must increase. (Proof: The free space assumption implies that marks can never decrease. Marks in an invalid

subgraph can not remain unchanged because that would imply a shortest path after correction is the same as one before which contradicts the fact that all shortest paths were previously undermined. Therefore all marks in an invalid subgraph must increase.)

2. Let M = the current mark of all vertices in mark_frontier.

3. 1. implies that the maximum mark that a vertex in mark_frontier could have had before correction is $M - 1$.

4. Condition (*) ensures that all vertices in clear_frontier currently have a mark of M+N. Note the marks in clear_frontier are not yet corrected.

5. 3. and 4. imply that the difference between marks in clear_frontier and mark_frontier, before any marks were corrected, was at least $(M + n) - (M - 1) = N + 1$. Thus there were at least N+1 edges between them.

6. Edge closures never decrease the distance between vertices, therefore there must be at least N+1 edges between clear_stack and mark_frontier if condition is observed.

Integration. The following pseudo-code shows how the marking algorithms are integrated into the sensor-based motion planning algorithms.

OSP & P-DFS with incremental data repair

```
mark all goals with zero
mark_stack = goals;
clear_stack = ()
marking_complete = 0

while goal not found
        while current vertex has no marked neighbours
                and not (marking _complete)
        marking_complete = propagate_marks()
        end
        if current vertex has a marked neighbour
                try to move to the lowest neighbour
                if successful
                        update graph and add invalid roots to
                        clear_stack *
                else
                        update graph and add invalid roots to
                        clear_stack !
                end
        elseif marking_complete & back-search not complete *
                        backtrack to first unvisited neighbour *
        else
                        stop goal are unreachable
        end
end
* applies only to P-DFS
! applies only to OSP
```

Chapter 7

Evaluation of Algorithms through Simulation

7.1 Introduction

Once the sensor-based motion planning problem is reduced to a graph search, algorithms to guide the search can be compared with respect to minimising robot motion and minimising online computation. In chapter 5, we introduced two algorithms, the OSP and the P-DFS, to solve the problem of robot motion planning. The OSP algorithm is popular because it is known to provide good average-sense performance: according to one source [7], it preformed 45% better than the DFS on average. However, DFS algorithm is known to provide better worst-case performance bounds. In this chapter we show that our DFS algorithm, P-DFS, has average-sensor performance comparable to DFS and argue that this is the consequence of the innovative pruning rule. We also evaluate the performance of the scheme for efficient graph marking that was described in chapter 6.

Therefore, in this chapter we try to answer three questions:

1. What is the benefit of dynamic marking and is it different for the two algorithms?
2. What is the benefit of the "pruning rule" ?
3. How much better does the OSP perform than the P-DFS algorithm ?

The first question is addressed in section 7.2. Section 7-3 shows the result of the

benefit of the " pruning rule". The third question is difficult to answer because each algorithm is sometimes better than the other and performance depends on factors that are largely unknown and difficult to study analytically. Nevertheless, we try to answer the question by comparing the performance of the algorithms for a large number of simulations in section 7.4. The comparison is a little messy, because we, according to our floor cleaning problem, conduct several sets of simulations to try to isolate the effect of a variety of factors: graph size (7.4.2), obstacle size (7.4.3), density of feasible paths (7.4.4) and goal search vs. coverage problem (7.4.5). and multiple inverse kinematic solution for manipulator (7.4.6). All results support the conclusion that OSP performs about 5% better than P-DFS, regardless of any of these factor.

7.2 The Benefit of Dynamic Marking

Sensor-based searches are different from ordinary searches in that each search is not isolated, but rather, one in a sequence of search that are very similar to each other. When a new obstacle is encountered, the new search will differ from the previous search only by an incremental change to the environmental model (graph). This implies that there is a potential to reduce computations if data from one search can be reused in the next, rather than regenerated from scratch. This section shows the benefit of our scheme for incrementally repairing data on uniform graphs search (chapter 6). Recall that this algorithm is essentially a simplified version of Stentz's D* algorithm that it is easier to implement and more efficient for uniform graph searches.

To evaluate the reduction in marking compared with remarking from scratch, we counted the number of marks made for 1000 random searches on graphs of 5 different sizes. Figure 7-1 shows the on-average factor of improvement for each graph size. It is clear that the two algorithms reduce marking by a large factor that increases with graph size. For graphs ranging in size from 100 to 3600 vertices, dynamic marking reduced marking by a factor about 20 to 200 times. This result is consistent with results presented in [8] for the D* algorithm.

Figure 7-2 compares the average number of marks made by the efficient version of the OSP to those made by the efficient version of the P-DFS for the same set of graph searches. Results show that the two algorithms require about the same amount

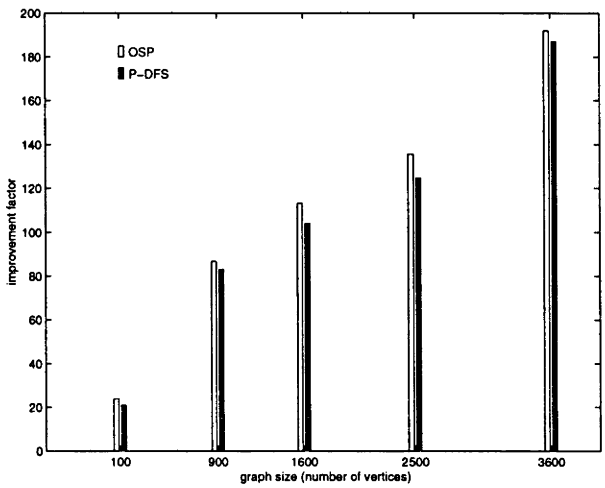

Figure 7-1: *Shows that the proposed algorithms greatly reduce marking compared with re-marking from scratch*

of marking , regardless of graph size. Therefore, in term of computation, the two algorithms are the same.

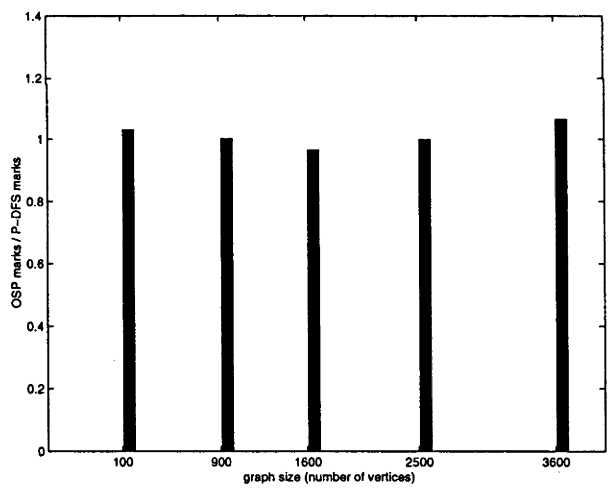

Figure 7-2: *Shows that marking is about the same for OSP and P-DFS when the proposed algorithms are used*

7.3 The Benefit of Pruning

The "pruning rule" allows a robot to ignore unvisited neighbours that are cut off from a goal by a subgraph of visited vertices. Because a tree search cannot cross itself (except when backtracking), it serves no purpose to explore subgraphs that can only reach a goal through already visited vertices. We believe the rule can improve the performance

of a robot guided by the DFS algorithm.

Figure 7-7 illustrates pruning. A robot was searching for goal G starting from S without knowledge of the graph at the beginning. When it reached the vertex **a** in subgraph **g**, the lower-right subgraph comprised by vertices b, c, d, e and their edges was cut off from the goal by the path of visited vertices and became unmarked. (vertices **c, d** and **e** are unmarked before the sub-figure because the algorithm marks no more than is immediately needed for motion planning). Then P-DFS ignored them and backtracked to the vertex which it first come from. A no-pruning DFS would have to explore the subgraph before backtracking.

Figure 7-3 compares the total number of moves (or path length) for P-DFS and DFS applied to 750 randomly generated search problems on a grid of size $20 * 20$. The searches were generated by a random placement of the goal and the robot within a graph generated by cutting various densities of randomly selected edges. The result shows "pruning rule" improves the performance of DFS dramatically: P-DFS performs about 40% better than DFS without pruning on average.

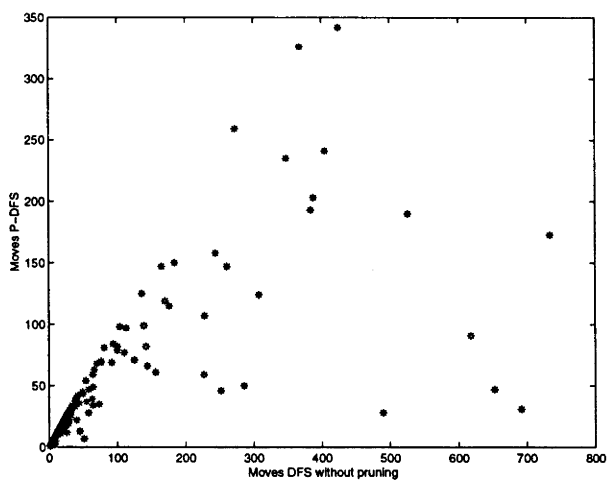

Figure 7-3: Performance of the P_DFS vs the DFS without pruning

Figure 7-4 shows the comparison of the OSP versus the DFS algorithm. The experiment environment and method are exactly the same as Figure 7-3. The OSP presents 45% better average performance than the DFS: the result that we find is consistent with

remarks we have seen in the literature [7]. Combined, these results suggest that pruning dramatically improves the average-sense performance of DFS to a level comparable to OSP.

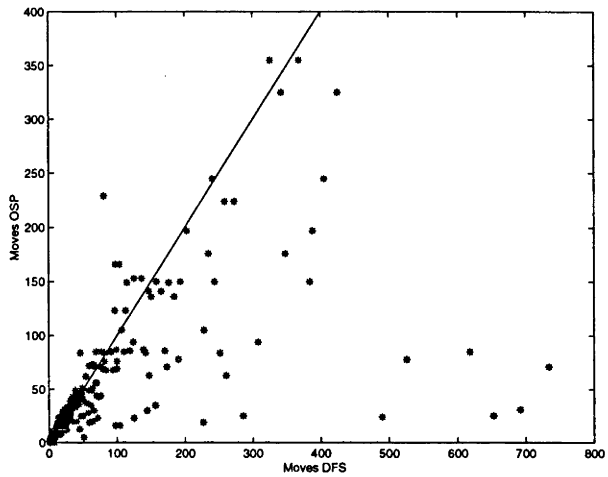

Figure 7-4: Performance of the OSP vs the DFS without pruning

7.4 OSP vs. P_DFS

The results in last section suggest that the P-DFS algorithm presents a comparable performance to the OSP algorithm. In this section, we try to detect whether the performance of the two algorithms vary with graph size, obstacle size and the density of feasible paths. OSP always moves along the shortest path based on current knowledge with the free-space assumption and this offers a good average-sense performance. On the other hand, P-DFS goes to unvisited vertices until a branch is found fruitless and guarantees a covering graph with no more than two traversals of each edge in the spanning tree it generates. However, these advantages are not absolute. In fact, each algorithm can sometimes be better than the other. For this reason, the section begins with examples of better performance to the two algorithms respectively. Then we compared them by a numerical simulations.

7.4.1 Examples of Performance to the Two Algorithms

To illustrate how each algorithm can sometimes be better than the other, a robot is employed to explore two similar environments shown in Figure 7-5. Figure 7-5(b)is

formed by moving one obstacle from Figure 7-5(a). As a consequence, the two algorithms present an opposite result: OSP performs better than P-DFS in the 7-5(a) environment, while it performs worse in the 7-5(b) environment.

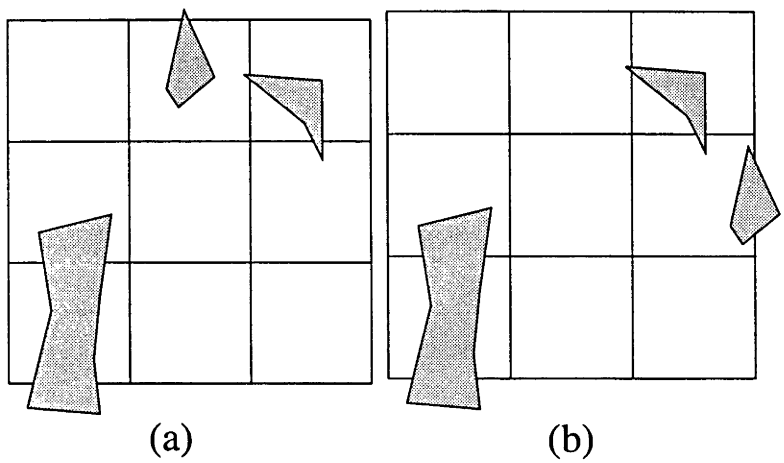

Figure 7-5: Examples of grid-based environment, the shadowed areas represent obstacles. In the two similar environment , the OSP and P_DFS algorithms present different performance

Figure 7-6 and Figure 7-7 show the performance of a robot guided by the two algorithms to explore the environment of Figure 7-5(b). At the beginning , the robot know nothing about the environment (The start position is S, and goal is G). The robot driven by the different algorithm chooses the same path from sub-figure **a** to **e**. However, OSP provides better performance due to going back to the shortest path at sub figure **e**, while the P-DFS find a path to goal only after visiting all of the unvisited tree branches.

Conversely, the P-DFS algorithm performs better than the OSP algorithm in right environment shown in Figure 7-5(b). Figure 7-8 shows a robot guided by P-DFS succeed to find a path in the branch on which it stand to the goal, while one driven by the OSP in Figure 7-9 reaches the goal after several trail searches.

7.4.2 Performance Variation with Graph Size

To investigate whether the performance of the two algorithms vary with graph size, they are applied to 1000 randomly generated search problems on graph size 10*10, 20*20 and 30*30 respectively. The start vertex and the goal are also placed randomly. Figure 7-10, Figure 7-11 and Figure 7-12 present nearly the same graph and the average difference

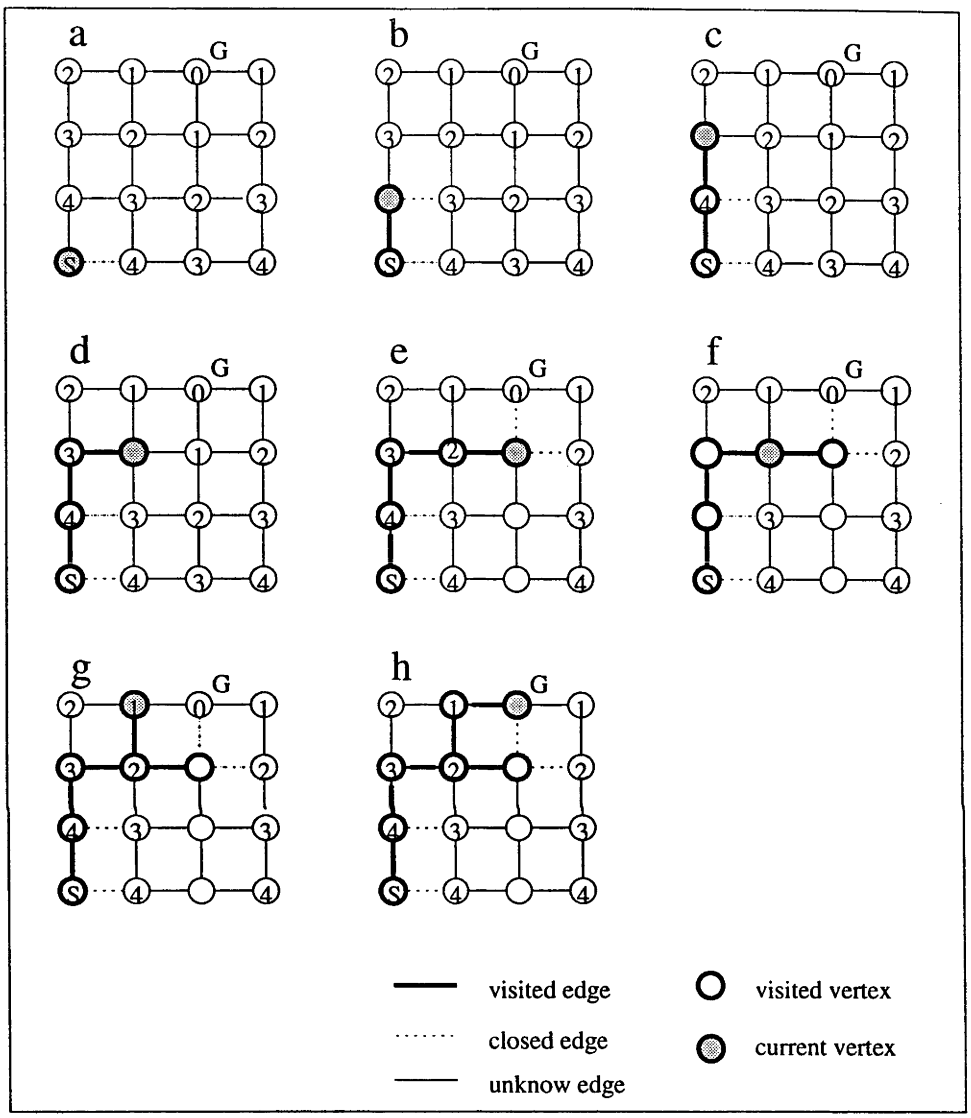

Figure 7-6: An efficient example of graph search with OSP *The robot guided by OSP algorithm always go to the shortest path based on the current knowledges with the free space assumption. Therefore, the robot goes back to the visited vertex from sub figure* e *to* f.

is 3.97%, 5.23% and 5.66%. The results show the difference of the two algorithms is less than 5% in the small graph, such as size $10 * 10$, and the performance of the OSP is slightly better than the P-DFS with increasing graph size, but the increase is very small compared the total performances. Thus, we conclude that the OSP performs about 5% better than P-DFS on average, regardless of graph size.

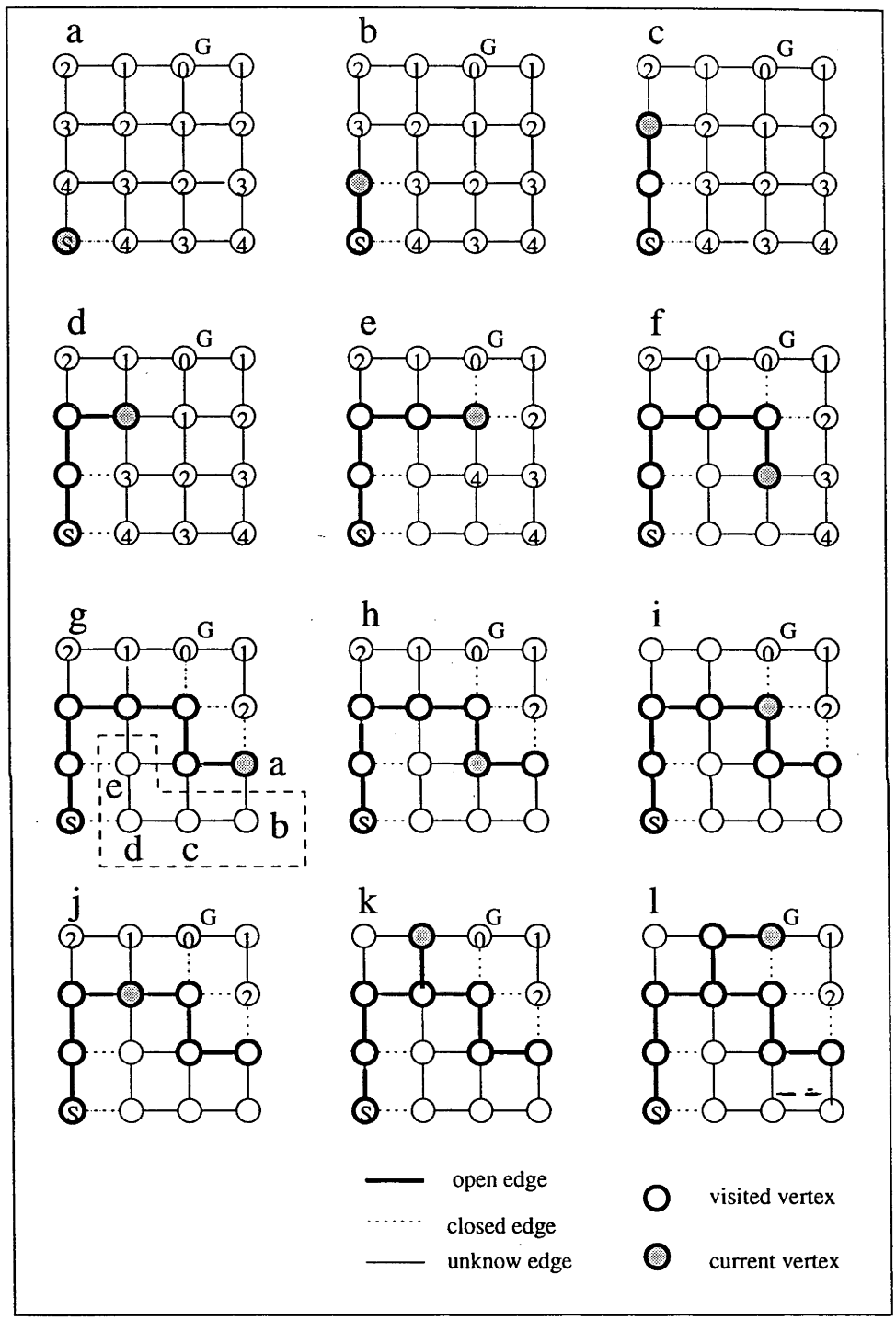

Figure 7-7: An inefficient example of graph search with P-DFS: *The robot driven by P-DFS do not backtrack until the branch is found fruitless. Consequently, the robot chooses a longer path from* **e** *to* **f**. *The figure also shows the robot is kept from exploring subgraph comprising by vertices* **b**, **c**, **d**, **e** *and their edges that are obviously void of goals due to "pruning rule"*

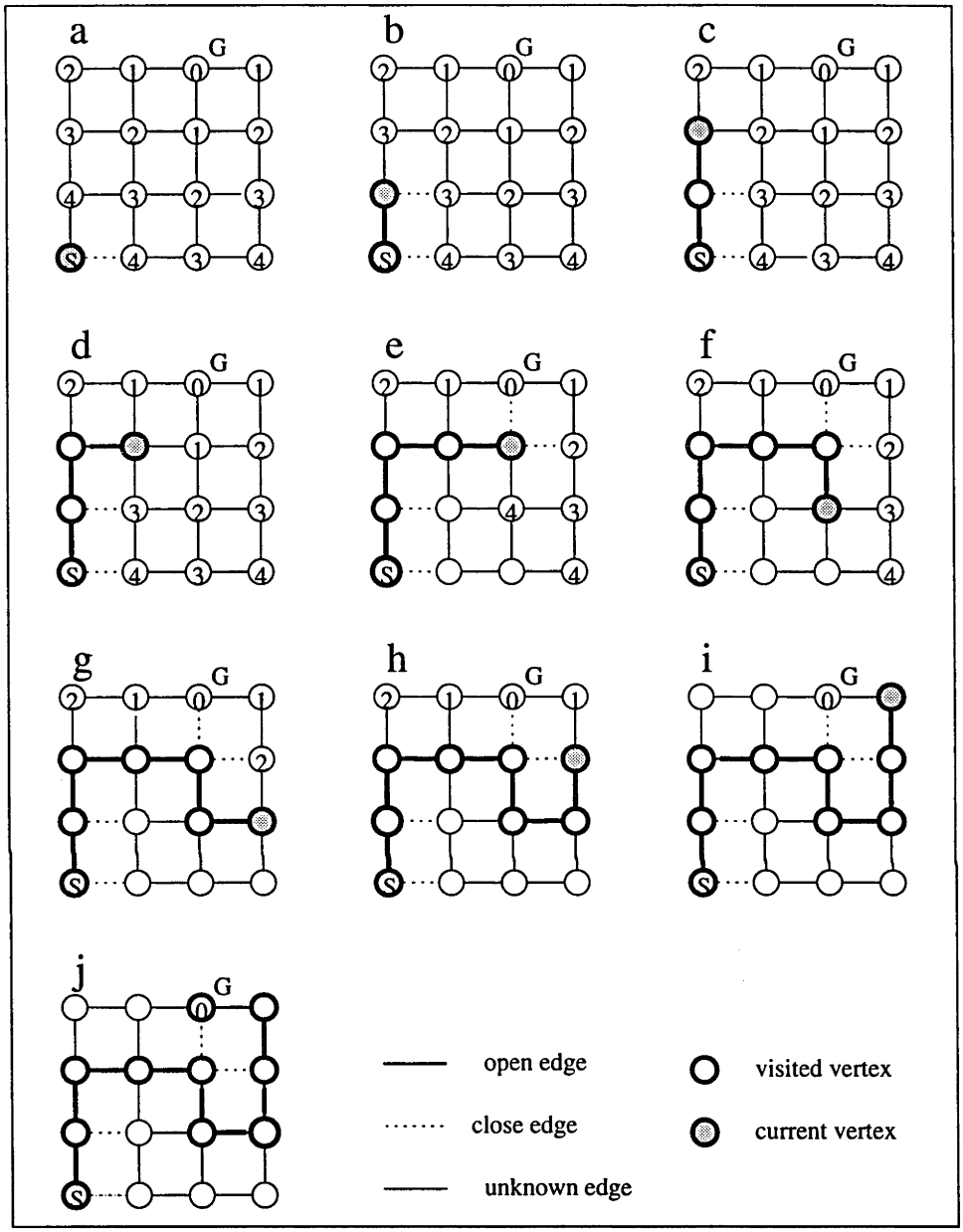

Figure 7-8: An efficient example of graph search with P-DFS: *In the case, the robot happen to stand on a fruitful branch. Accordingly, it result in an efficient search.*

7.4.3 Different Obstacle Size

We are especially interested in developing a manipulator arm to perform floor cleaning by removing spilled granular material from under and around heavy plant equipment such as conveyor. Thinking of the real problem, it is reasonable to imagine that there maybe exist various size of obstacles in the real environment. So we try to answer the question: Is the performance of the two algorithms sensitive to the size of obstacles. The

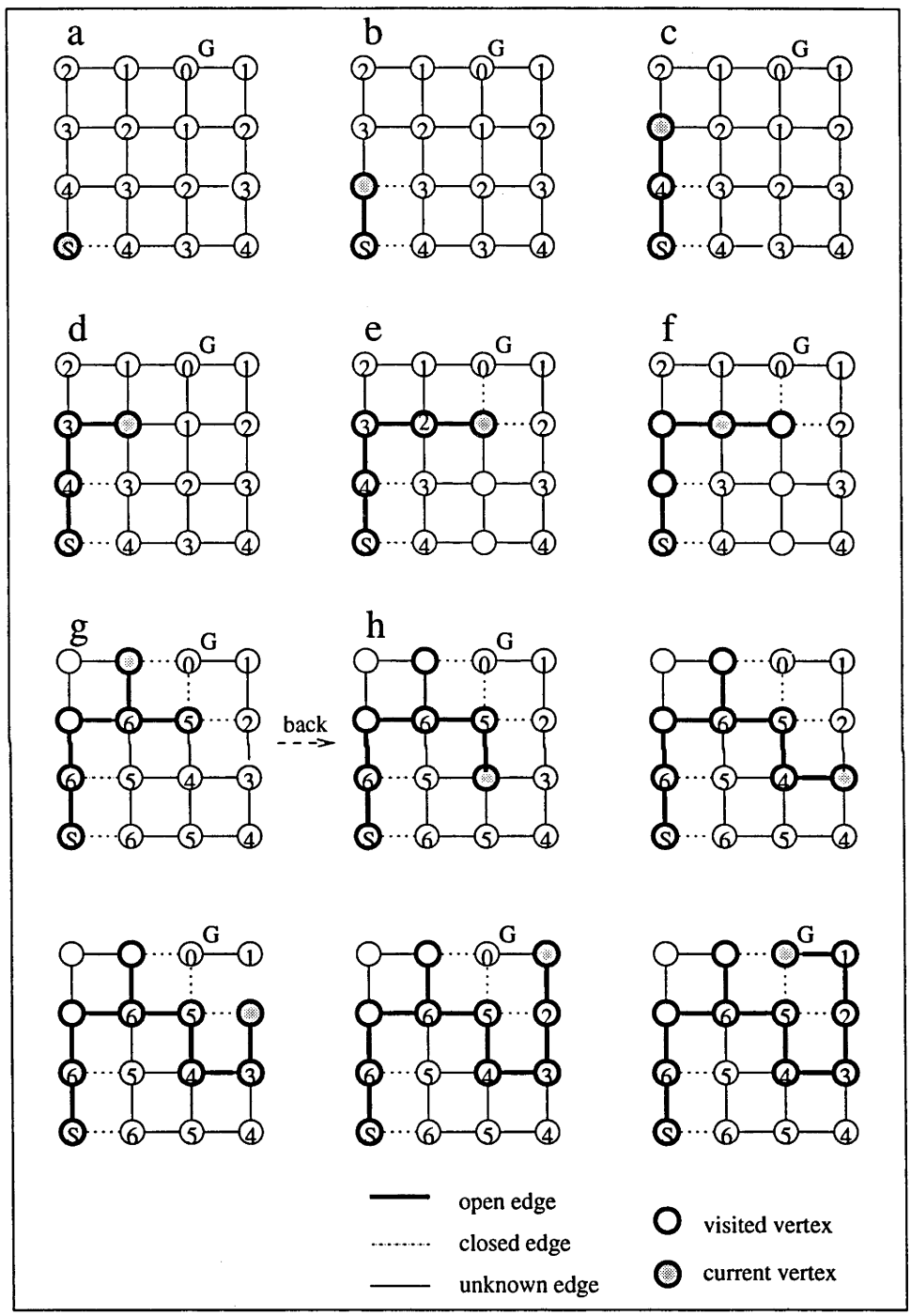

Figure 7-9: An inefficient example of graph search with OSP: *The example shows the feature going to the shortest path of OSP results in inefficient search sometimes. In the case, the robot go back from* e *to* f *to try to follow the shortest path based on the current knowledge with free-space assumption. Unfortunately, the trail search is failed. So the robot go back to* g *again according to the shortest path.*

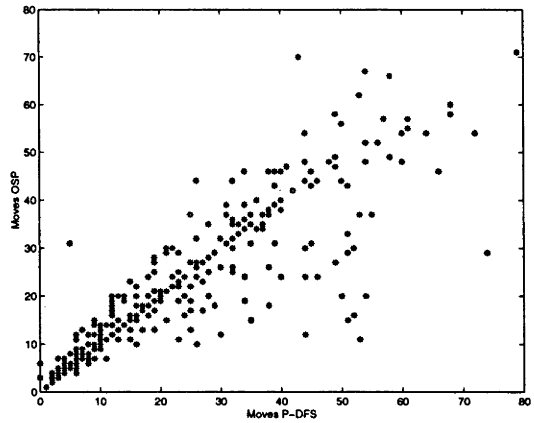

Figure 7-10: Performance of OSP vs P-DFS in a grid of size 10 * 10

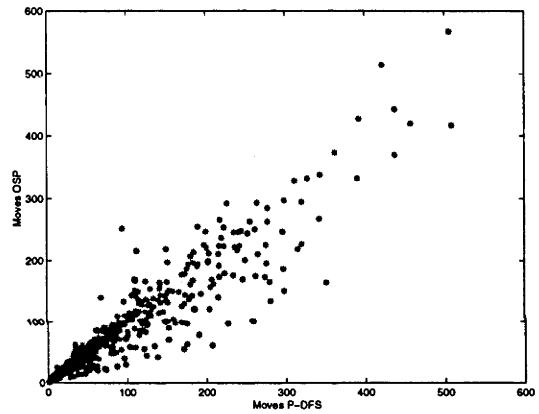

Figure 7-11: Performance of OSP vs P-DFS in a grid of size 20 * 20

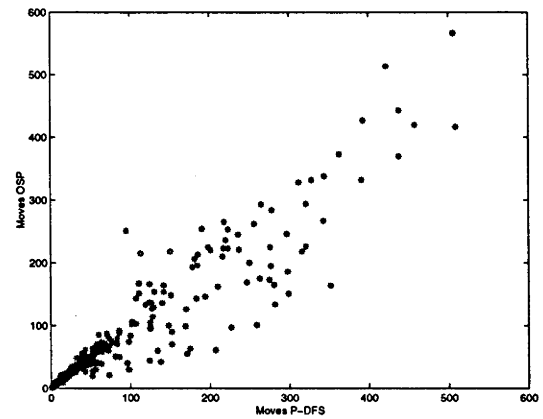

Figure 7-12: Performance of OSP vs P-DFS in a grid of size 30 30

typical comparison environments are given in Figure 7-13 and Figure 7-14. In Figure 7-13, each obstacle cuts only one edge. We called a small obstacles environment. In Figure 7-14, each obstacle cuts several edges. It is called a big obstacles environment.

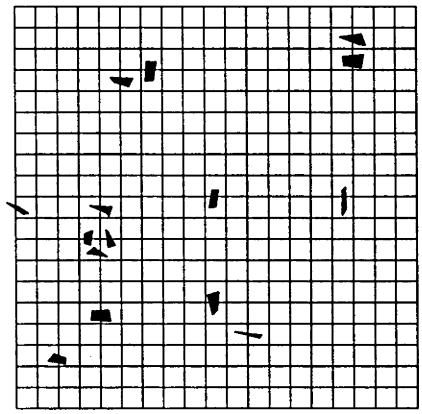

Figure 7-13: Typical comparison environment with small obstacles

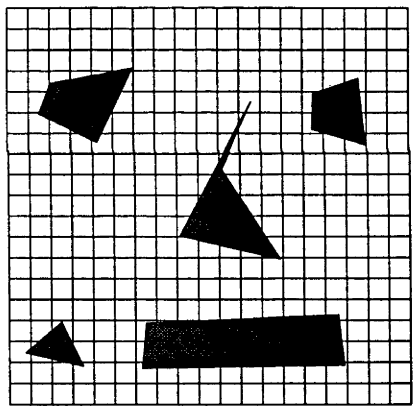

Figure 7-14: Typical comparison environment with big obstacles

Figure 7-15 and Figure 7-16 are obtained from 1000 graph search problems produced randomly on a grid of size $20 * 20$ with different obstacle sizes. They present very similar results. And the average difference of moves is very close: 5.01% and 5.77%. In conclusion, the size of obstacles do not effect the performance of the two algorithms. These results are also consistent with those we have previously found in this chapter: OSP performs about 5% better than the P-DFS on average in the simulations.

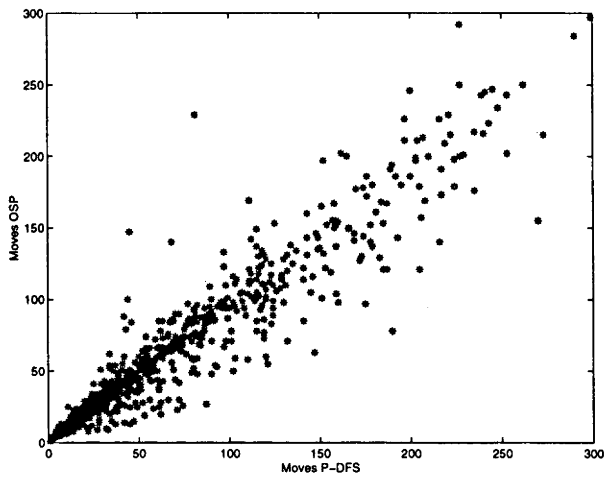

Figure 7-15: Performance of OSP vs P-DFS in small obstacles environment

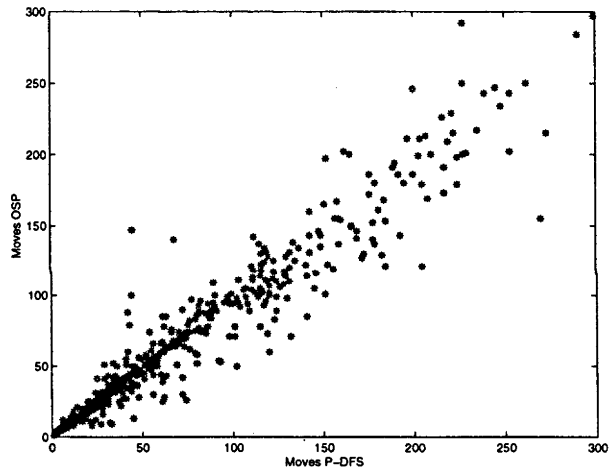

Figure 7-16: Performance of OSP vs P-DFS in big obstacles environment

7.4.4 Different Density of Feasible Paths

So far, we compared the two algorithms by different graph size and different obstacle size. In summary, P-DFS provides comparable performance to the OSP which only performs about 5% better and the results do not vary with graph size and obstacle size. Now, we are interested in another question: In which kind of environments does OSP perform better than P-DFS. In particular we aim to test the hypothesis that P-DFS performs better for graphs that have a low density of feasible paths.

To define "density of feasible paths", firstly, we define two graphs: 1) optimistic graph in which all edges are open and 2) actual graph where all of the edges associated

with impossible moves are closed. Secondly, let the graph be modelled as an electrical circuit with each edge representing a resistor, each goal associated with ground, and the start vertex represents a source voltage. The current in this circuit would flow from the start vertex to the ground using all of the paths available. Furthermore, the ratio of effective resistance in the two graph is a density measure. We think the following ratio density provides a good measure of the density of feasible paths.

$$\text{density of feasible path} = \frac{\text{effective resistance in the optimistic graph}}{\text{effective resistance in the actual graph}}$$

Figure 7-17 was obtained from 1000 graph search problem produced randomly. It shows the exactly the same performance of the proposed algorithms when density is going to 1 (no obstacle) and the P-DFS is slightly better than the OSP with the decrease of density. This supports the hypothesis that P-DFS performs better for graphs that have a low density of feasible paths. In the same way, the average difference of performance to the two algorithms on is only 4.89%, about 5% on average.

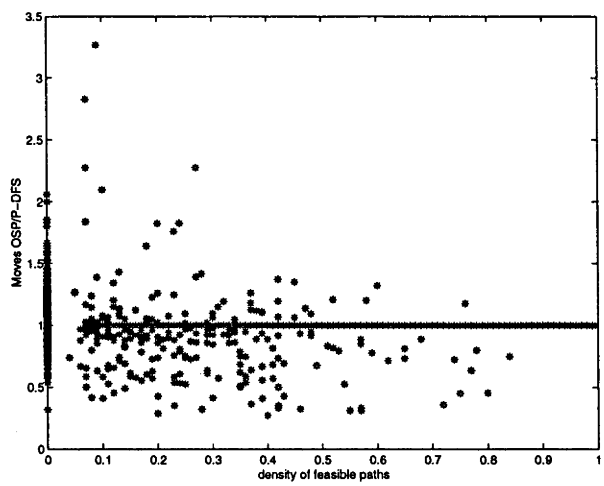

Figure 7-17: : Performance of OSP vs P-DFS with different density of feasible paths: *Shows the P-DFS is slightly better than the OSP algorithm with the decrease of density.*

7.4.5 Coverage Problem

All results so-far presented are for single goal searches. Results suggest that the P-DFS algorithm is comparable to the OSP algorithm in single goal search problem. Coverage problem is another problem we are interested in because our objective is to control a manipulator arm to perform floor cleaning in an unknown obstacles-filled environment. Now, we compare the two algorithms for the surface coverage problem.

Different from a find-goal problem, all vertices but start vertex are regarded as goals in a coverage problem which is demanded to visit all reachable vertices. Therefore, the coverage problem is an extension of a find-goal problem, which has multi-goals. For this reason, we, similarly, measure the total moves to evaluate the performance of the two algorithms. As before, 1000 randomly generated graph search problems in grid size $10 * 10$, $20 * 20$ and $30 * 30$ were applied to the two algorithms respectively. The results, Figure 7-18, Figure 7-19 and Figure 7-20, show they did not overlap with the results of single goal. When the moves are relatively small, the two algorithms present nearly the same results. However, OSP presents an obviously better performance when the moves are bigger than an amount which depends on graph size and the amount increase with the increase of graph size. The average difference of moves are 12.40%, 15.49% and 16.31%. This shows that OSP is significantly better and the difference increase with graph size.

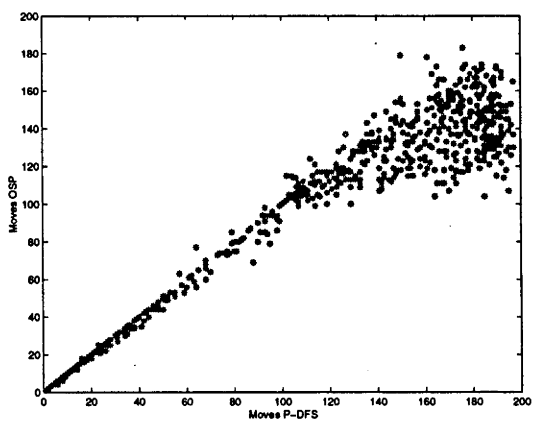

Figure 7-18: The coverage performance of OSP vs P-DFS in grid size of 10 * 10

Why does the OSP algorithm perform so much better than P-DFS in the coverage problem rather than the simple find-goal problem? The reason is that the P-DFS is limited to tree search, so early covering motion can force P-DFS to explore large areas

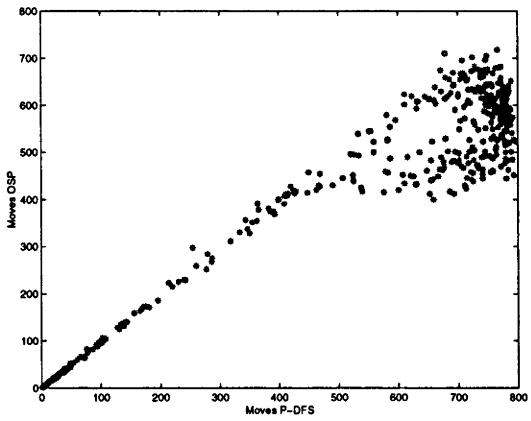

Figure 7-19: The coverage performance of OSP vs P-DFS in grid size of 20 * 20

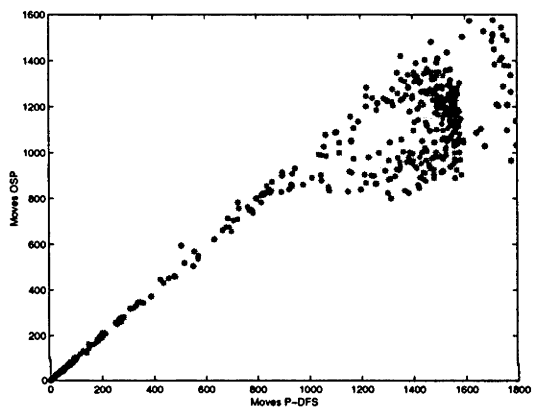

Figure 7-20: The coverage performance of OSP vs P-DFS in grid size of 30 * 30

of the graph that are distance from the goal. Figure 7-21 shows the two algorithms have a similar performance if P-DFS is allowed to treat the coverage problem as a series of new find-goal problems. In other words, let P-DFS forget about its tree whenever a goal is found. In the 1000 randomly produced search problems, the P-DFS and the OSP only perform 1.78% difference on average.

7.4.6 Application to Manipulation Arm

So far, our results have not considered manipulator arm kinematics. To accomplish a cleaning problem by a manipulation arm, the robot has to move in its joint-space graph but the cleaning task occurs in task space. For a find single goal problem for a robot arm, there are at least two joint space vertices marked as goals. If one of these goals is reached, the find goal task is finished. For a coverage problem, all vertices but a set of start vertex are treated as goals. When a goals is reached, all vertices that map onto

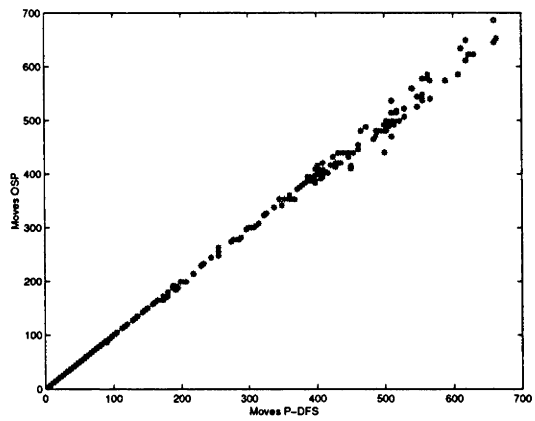

Figure 7-21: Let the P-DFS algorithm treat the coverage problem as a series of new find-goal problems, then it performs nearly the same as the OSP algorithm

the same task-space position in are not goals anymore simultaneously. If no more goals can be reached , then the coverage is finished. So application to manipulation arm also can be treated as a example of multi-goals search. In the section we focus on whether the previous results searched on grid-like graphs still hold for manipulator arm graphs. Our investigation is based on a two planar manipulation arm whose link lengths are 5 and 5, and joint_limit is $[-2/3 * \pi \quad 2/3 * \pi]$.

1000 searches were generated randomly in a task space graph of size $20 * 20$. The base of manipulation arm is placed in the centre of a graph. Figure 7-22 and Figure 7-23 show the results for the find goal problem and the coverage problem respectively.

The result shown in Figure 7-22 is similar to those shown in Figure 7-11 and 7-19, especially 7-22. Besides, the average difference of the moves is also very close to that in Figure 7-22: 5.87% vs. 5.23%. It suggest that the previous results hold for application to manipulator arms.

Figure 7-23 shows nearly the same results as Figure 7-19: the average difference of the two algorithm is 14.3% for Fig 7-19, while it was 15.49% for Fig. 7-23. Thus we conclude that previous results for coverage also hold for manipulator arms.

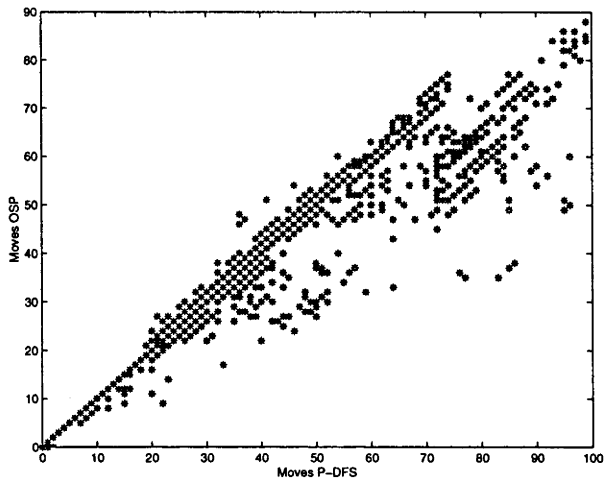

Figure 7-22: The find-goal performance to manipulator arm of OSP vs P-DFS

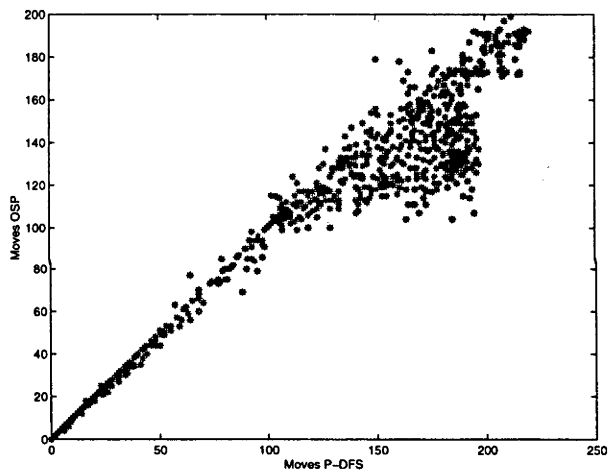

Figure 7-23: *The coverage performance to manipulation arm of the OSP vs the P-DFS, the result is similar as that in Figure 7-19*

7.5 Conclusion

Results suggest that depth-first search algorithms applied to sensor-based motion planning can be more efficient than is generally recognised. In particular, branch pruning seems to improve performance about 40% on-average, to a point comparable to the shortest path algorithm. This is significant because DFS has a better worst-case bound on performance. And the results do not vary with graph size, obstacle size and the density of feasible paths.

Coverage problem is a find multi-goals problem. The result of comparing the two

algorithms suggest that the OSP is significantly better than the P-DFS. However, they have nearly the same performance if P-DFS treat the coverage problem as a series of new find-goal problem.

In addition, simulations suggest that the above results hold for manipulator arms.

Furthermore, the results show that online computation for uniform graph searches can be dramatically reduced by proposed algorithms that reuse data from previous searches.

Chapter 8

Conclusions

8.1 Conclusions

In this thesis, we have investigated three problems to enable the manipulator arm to perform floor cleaning in unknown environments. The approach is based on experiment, modelling and verification by computer simulation. A thorough effort was made to consider theoretical and practical aspects. The work is directed to a practical application and our solutions are based on realistic technologies.

Reactive controller to perform floor coverage (to simulate cleaning).

The experimental work supports the conclusion that a manipulator arm driven by a reactive controller can work well in an unknown obstacle-filled environment and perform coverage motion. Only force sensing was used in the experiment. Results suggest that force sensing provides an useful basis for the system to work well in a heavy industrial environment.

Graph-based modelling for manipulator motion planning.

The modelling results in chapter 4 shows that manipulator motion planning problem can be formulated as a graph search. This provides a foundation for efficient and general motion planning algorithms for manipulator arms.

Graph-based motion-planning algorithms.

In chapter 5-7, we described two sensor-based motion planning algorithms: an optimistic shortest path algorithm (OSP) and a depth first search algorithm (DFS). A scheme to minimise re-planning computation on uniform graphs by incrementally

repairing data is developed for the both algorithms, which is essentially a simplified version of Stentz's D* algorithm that it is easier to implement and are more efficient for uniform graphs searches. In addition, an original technique that we call "tree-pruning" is used for DFS algorithm. With the algorithm, the DFS algorithm achieves similar average-case performance with the OSP algorithm. The study results show that the two algorithms are efficient algorithms to manipulator arm motion planning. It is important in practical application in term of minimising online computation and minimising robot motion.

8.2 Future Research

This thesis has presented some foundational research toward realizing an new type of autonomous cleaning system. There are still many problems to realize our ambition: to free workers from hostile and unhealthy environment.

Firstly, we envision that an manipulator arm be purpose-built according to Whole Arm Manipulation principles.

In addition, the addition of a vacuum mounted on the manipulator arm will bring some new challenges to control.

Finally, research is needed into providing the arm with the ability to know whether it is colliding with obstacles or a pile of granular material when it is cleaning in an unknown obstacle-filled environment.

Bibliography

[1] M. Schraft, M. H., and H. Volz. Service robots: The appropriate level of automation and the role of users/ operators in the task execution. In *Proceedings of the 25th ISIR*, 1994, pages 225-231, Hannover.

[2] I. R. Ulrich, F. Mondada, J.D. Nicoud. Autonomous vacuum cleaner. *Robotics and autonomous systems*, 19, 1997, pp 233-245.

[3] C. Hofner and G. Schmidt. Path planning and guidance techniques for an autonomous mobile cleaning robot. In *Proc. of the IEEE Int. Conf. on intelligent mobile cleaning robots and system*, 12-16, 1994, Neubibery.

[4] R. A. Brooks. Elephants don't play chess. *Robotics and autonomous systems*, 6:3-15, 1990.

[5] R. C. Arkin. Reactive robotic systems. *article in Handbook of Brain Theory and Neural Networks*, ed. M. Arbib, MIT Press, pp 793-796, 1995.

[6] G. Foux, M. Heymann, and A. Bruckstein. Two-dimensional robot navigation among unknown stationary polygonal obstacles. *IEEE Trans. Robotics and Automation*, 9(1): 96- 102, 1993.

[7] S. Koenig, Y. Smirnov. Sensor-based planning with the freespace assumption. *Proc. IEEE Int. Conf. Robotics and Automation*, 3540- 3545, 1997.

[8] A. Stentz. Optimal and efficient path planning for unknown and dynamic environments. *International Journal of Robotics and Automation*, 10(3): 89- 100, 1995.

[9] T. Lozano-Perez. Automatic planning of manipulator transfer movements. *IEEE Transactions on Systems, Man and Cybernetics*, SMC-11, 10, pp. 681-698.

[10] R. A. Brooks. A Robust layered control system for a mobile robot. *IEEE J. of Robotics and Automation*, vol.ra-2,no.1,pp.14-23,March 1986.

[11] R. A. Brooks. Intelligent without representation. *Artificial Intelligence* Vol. 47, pp. 139-159, 1991.

[12] N. Ayache and O. D. Faugeras. Building a consistent 3D representation of a mobile robot's environment by combining multiple stereo views. *Proc. IJCAI-87*, pp. 808-810, 1987.

[13] S. Nakasuka, T. Yairi and H. Wajima. Autonomous generation of reflexion-based robot controller using inductive learning. *Robotics and Autonomous Systems*, 17, pp.287-305,1996.

[14] R. C. Arkin. Integrating behaviour. *The MIT Pree, Cambridge MA*,1990.

[15] P. Maes. Designing autonomous agents: theory and practice from biology to engineering and back. MIT Press,1990.

[16] T. C. Henderson and R. Grupen. Logical behaviours. *J. of Rob. Syst.*

[17] A. Zelinsky, R. A. Jarvis, J. C Byrne and S. Yuta Planning paths of complete Coverage of an unstructured Environment by a mobile robot. 6th International conference on advanced robotics, pp 533-538, 1993, Tokyo.

[18] R. C. Arkin. Behaviour-based robot navigation in extended Domains. *Journal of Adaptive Behaviour*, Vol. 1, No. 2, pp. 201-225, 1992.

[19] I. Nourbakhsh and M. Genesereth. Assumptive planning and execution: a simple, working robot architecture. *Autonomous Robots*, 3(1): 49-67, 1996.

[20] W. R. Townsend, K. J. Saliabury. Mechanical design for whole-arm manipulation. *Robotics and Biological System: Toward a new Bionics?*, Eds. Dario, Sardini, and Aerischer, NATO ASI Series F: Computation and System Science, Vol 102: 153-164, 1993.

[21] E. C. Chalfant. Planning in subsumption architectures. *Conference on Intelligent Robotics in Field,Factory,Service, and space* p.2vol. xv+885, 788-98 vol.2.

[22] R. J. Clark, R. C. Arkin and A. Ram. Learning momentum; on-line performance enhancement for reactive systems. *IEEE Int.Conf. on Robotics and Automation* May 1992.

[23] T. M. Mitchell. Becoming increasingly reactive. *Proc. of Conference on Artificial Intelligence* 1990.

[24] H. Choset and P. Pignon, Coverage path planning: The boustrophedon decomposition. *International Conference on Field and Service Robotics*, Canberra, Australia, 1997.

[25] U. M. Nassal. Motion coordination and reactive control of autonomous multi-manipulator systems. *J. of Robotic Systems*, 737-754,1996

[26] M. J. Mataric. Integration of representation into goal-driven behaviour-based robots. *IEEE Trans. on Rob. and Auto.*, Vol 8, No 3,June 1992.

[27] D. C. Mackenzie and R. C. Arkin. Behaviour-based mobile manipulation for drum sampling. *Inter. Con. on Robotics and Auto*, April 1996.

[28] J. H. Connell. A behaviour-based arm controller. *IEEE Trans.on Robotics and Auto.* ,vol5, no.6,pp.784-791,December 1989.

[29] J. M. Cameron , D. C. MacKenzoe and E. C. Arkin. Reactive control for mobile manipulation. *IEEE Inter. conf. on Robotics and Automation*, Atlanta 1993.

[30] R. Brooks and J. Connell. Asynchronous distributed control system for a mobile robot Mobile Robots. In *Proceedings of SPIE's Canbridge Symposium on Optical and Optoelectronic Engineering, Cambridge, MA, October, pp. 77-84.*

[31] E. Gonzalez, A. Suarez, C. Moreno, F. Artigue. Complementary regions: a surface filling algorithm. Pro. of IEEE Int. Con. on Rob. and Aut., *1996.*

[32] Phillip John Mckerrow. Introduction to robotics. Addision-Wesley Publishin company, *1991.*

[33] J. J. Leonard and H. F. Durrant-Whyte. Directed sonar sensing for mobile robot navigation. Kluwer Academic Publishers, *1992.*

[34] J. H. Connell. A behaviour-based arm controller. IEEE Transactions on Robotics and Automation. *Vol. 5, No. 6, December 1989.*

[35] R. Jarvis and K. Kang. A new approach to robot collision-free path planning. Robots in Australia's Future Conference, *pp. 71-79, 1986.*