The Thesis Committee for Brian Erick O'Neil

Certifies that this is the approved version of the following thesis:

Graph-Based World-Model for Robotic Manipulation

APPROVED BY

SUPERVISING COMMITTEE:

Supervisor: _____

Sheldon Landsberger

Co-Supervisor: _____

Mitch Pryor

Graph-Based World-Model for Robotic Manipulation

By

Brian Erick O'Neil, B.S.E.

Thesis

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

Master of Science in Engineering

The University of Texas at Austin

August, 2010

Dedication

To my parents, Pat and Barb O’Neil.

Acknowledgements

I would like to thank Dr. Mitch Pryor for his guidance and mentorship throughout the research and writing process. I would also like to thank Dr. Sheldon Landsberger for his aid and encouragement. I also offer my heartfelt thanks to my wife, Emily. Without her support, this would not have been possible.

This work was funded by the National Nuclear Security Administration under the University Research Program in Robotics, grant # DE-FG52-06NA25591

August 13th, 2010

Abstract

Graph-Based World-Model for Robotic Manipulation

Brian Erick O'Neil, M.S.E.

The University of Texas at Austin, 2010

Supervisor: Sheldon Landsberger

Co-supervisor: Mitch Pryor

There has been a significant push in robotics research toward robot autonomy. However, full autonomy is currently impractical for all but the most clearly defined tasks in the most structured environments. However, as tasks become less defined and environments become cluttered and less controlled, there is still a benefit to implementing *semi-autonomous behaviors* where aspects of the tasks are completed autonomously thus reducing the burden on the human operator. A key component of a robot control system that supports this functionality is a robust world model to act as a repository of environmental information.

The research community has provided many world-modeling solutions to support autonomous vehicle navigation. As such, they focus primarily on preventing collisions with the environment. Modeling schemes designed for collision prevention are of limited

use to robotic manipulators that must have contact interaction with the environment as a matter of course.

This thesis presents a world-modeling scheme that abstracts the model of the environment into a graph structure. This abstraction separates the concepts of entities in the environment from their relationships to the environment. The result is an intuitive world model that supports not only collision detection, but also motion planning and grasping. The graph-based world model presented can be searched by semantic type and tag values, allowing any number of agents to simultaneously use and update the model without causing failures elsewhere in the system. These capabilities are demonstrated on two different automated hot-cell glovebox systems, and one mobile manipulation system for use in remote contamination testing.

Table of Contents

List of Tables

List of Figures

CHAPTER 1 : INTRODUCTION

Robots have become a common fixture in application areas where tasks are repetitive, dangerous, or require physical capabilities beyond those of human beings. The primary functions of a robot are to augment human capability in such a way as to lower costs, improve performance, and keep humans out of harm's way. One important factor in how well a robot meets these requirements is its ability to perform certain tasks or subtasks autonomously. On a modern industrial factory floor, the level of automation is high. With very little human input, robots assemble automobiles on moving assembly lines. The use of robots improves precision in the assembly process and dramatically reduces human labor costs. This level of automation is possible because the environment is highly structured and the tasks are highly repetitive. However, robots used by the military for explosive ordnance disposal have virtually no autonomous capability. Humans remotely operate all functions of a mobile platform with an integrated robot manipulator to clear explosive devices. In this case, the environment is unstructured and unpredictable, and the precise nature of the task changes every time the robot is deployed. Therefore, it is not practical completely to remove a human from the task because it requires the superior reasoning ability of an operator. Due to the very large task domain over which robotic manipulators may operate, there is no "one-size-fits-all" approach to robot autonomy. The level of autonomy must be appropriate to the task being performed.

1.1. ROBOT AUTONOMY

Factory floor robots and explosive ordnance disposal robots represent two ends of an autonomy spectrum that is defined by the nature of the task to be performed and the nature of the environment in which the robot must operate. Figure 1-1 provides a

qualitative depiction of the relationship between task structure, environmental structure, and the level of automation that is feasible and appropriate.

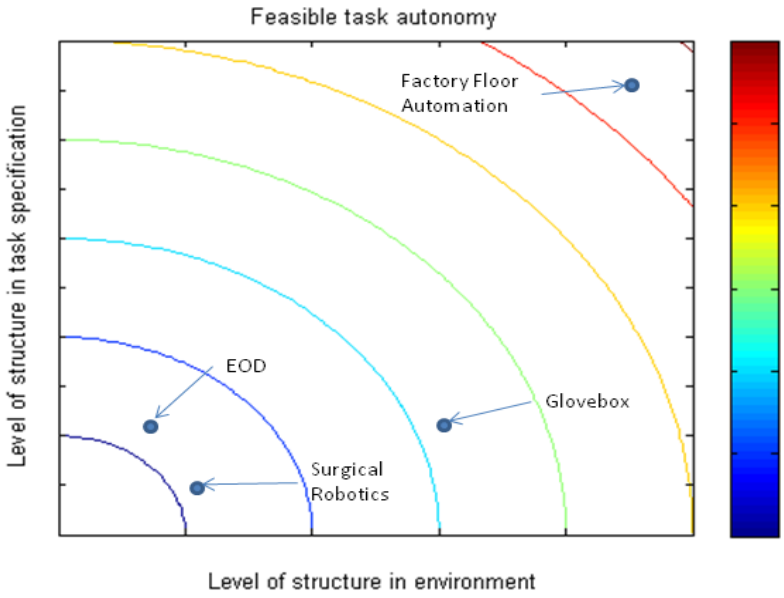


Figure 1-1. Autonomy space. Red indicates high feasibility.

For many applications, full autonomy is not feasible. When this is the case, the human operator's workload can still be reduced by automating certain tasks or subroutines. For example, the Department of Defense explored the idea of a mobile robotic surgical workcell called Trauma Pod. In the Trauma Pod system, a surgical robot was tele-operated by a trained surgeon. A scrub-nurse robot delivered tools to the surgeon and carried discarded items away. The surgeon directed the scrub-nurse by voice command.

Trauma Pod demonstrates the need to identify multiple levels of autonomy. The surgical robot operates inside of a human body where tasks and environments are not well-structured. However, it is possible to automate most of the scrub-nurse functions since it is performing repetitive tasks in a relatively more structured environment. The

system responds to simple voice commands that direct the robot to complete tasks at a higher level of autonomy, requiring less supervision by the operator.

To reduce the operator's workload by using autonomy, care must be taken to identify what tasks or functions can be automated and which require more human input. These tasks and functions are often referred to as semi-autonomous behaviors. This approach to automation is commonly referred to in literature as Human Centered Automation or HCA. Schemes to implement these types of behaviors are often called dynamic or adaptive autonomy schemes.

A layered approach to autonomy has been implemented at Idaho National Laboratory [Bruemmer, et al 2002] in which the robot operates in one of five autonomy regimes at any given time. Table 1-1 provides an overview of the INL's dynamic autonomy scheme.

Table 1-1. INL's Dynamic Autonomy scheme

Autonomy Mode	Defines Task Goals	Supervises Direction	Motivates Motion	Prevents Collision
Tele-operation Mode	Operator	Operator	Operator	Operator
Safe Mode	Operator	Operator	Operator	Robot
Shared Mode	Operator	Operator	Robot	Robot
Collaborative Tasking Mode	Operator	Robot	Robot	Robot
Autonomous Mode	Robot	Robot	Robot	Robot

The lowest level of automation in this scheme is tele-operation. In this mode, a human operates the robot directly. The robot takes no initiative of its own. The next level of automation is "Safe." In safe mode, the robot will attempt to detect and prevent collisions. The next level is "Shared" which indicates that the robot may plan and complete certain motions to ease the burden on the operator. These motions may be for

obstacle avoidance or reactive path planning. For example, if the human operator is moving an object in the workspace, the robot might move the object around a sensed obstacle even if the operator commanded a straight-line path through the obstacle. The fourth level is "Collaborative Tasking." Below this mode, the robot is still controlled primarily by tele-operation. At this level, the robot now completes entire behaviors with little or no human input. In this mode, the robot may pick up and move an object from one place to another, obeying a high-level command from the operator. The scrub-nurse robot mentioned previously operates at this level of autonomy. The final autonomy mode is full autonomy. As discussed above, this level of autonomy is typically only appropriate for very well-defined tasks in highly structured environments where it is safe to omit human reasoning entirely.

1.2. MANIPULATOR WORLD-MODELING

For any autonomous behavior to be possible, the robot must possess some amount of information about the environment in which it is operating. Figure 1-2 demonstrates how the world-model fits into a generic robot control architecture.

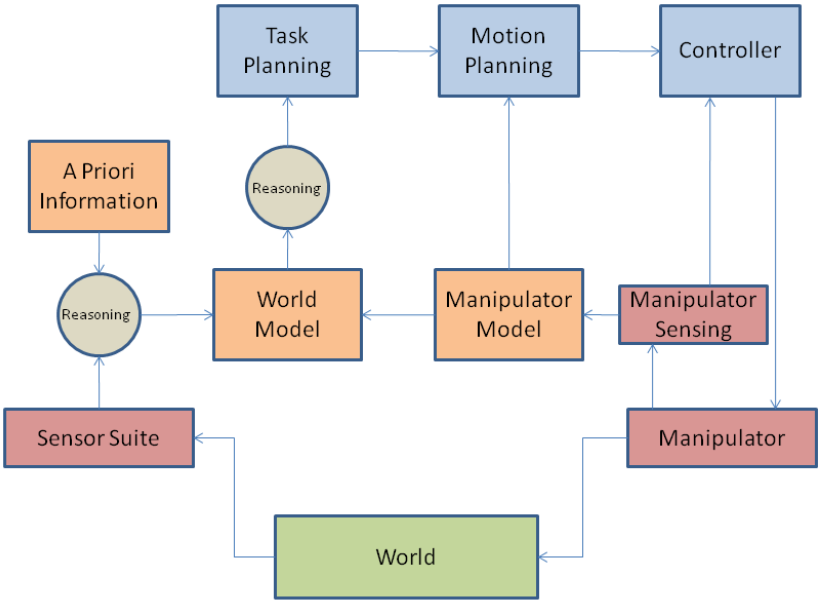


Figure 1-2. Robot Control Architecture

The world-model's basic function is to store parsed data from pre-modeling and sensor data in such a way that it is readily available in a useful format to downstream controller components. Inputs to the world-model include *a priori* knowledge and parsed sensor data. The reasoning block upstream of the world-model in Figure 1-2 must make sense of incoming sensor data, compare it with current model information, resolve inconsistencies, and ultimately keep the model current. The downstream reasoning provides information about the robot's environment to various controller components in a useful form. To preserve generality, these reasoning functions should be kept separate from the model itself in the control architecture. This allows the same world-modeling scheme, or perhaps even the same world-model, to be used by any number of robotic systems without regard to robot configuration or sensor equipment.

For highly structured environments, creating a world-model can be relatively straightforward. As the level of uncertainty in the environment increases, the model must be able to account for unexpected changes in the environment. A heightened interest in

autonomous *mobile* systems has driven significant research in world-modeling. [Angelopoulou, 1992], [Hong, 2002] Unfortunately, these techniques are not necessarily useful for robotic *manipulation* applications. World-modeling for unmanned vehicles serves the performance requirements of a robot designed for non-contact tasks (i.e. visualization, transportation, etc.) within its environment. A mobile robot's primary requirements for a world-model are to support global navigation, collision detection, and obstacle avoidance. World-models devised for robotic manipulation look very different from those that support robotic navigation. However, they typically perform the same function, which is to detect and avoid unplanned collisions. [Knoll and Tesar, 2007] Such collisions could include contact with other robots in the environment or self-contact. The result of this view of world-modeling is that most world-models contain only geometric information. Different modeling techniques represent this information in different ways, but the type of information required in the model is assumed to be geometric. The location, size, and shape of objects in the environment are certainly crucial parts of a world-model. However, inclusion of only geometric data is very limiting in a model designed to support robotic manipulation.

For a manipulator to be useful, it must be able to make certain types of contact with objects in its environment. This suggests that a model designed primarily to support collision avoidance lacks a great deal of functionality that could be very useful to the robot's control system. For example, in a collision-focused model, it is likely that object geometry is simplified to avoid long computation times in collision checking computations. However, an algorithm to compute an appropriate grasp would yield a very poor solution if it were passed this simplified geometric data. Such an algorithm would require high-fidelity geometric information, which is at odds with the collision-checking algorithm's requirement of fast computation time. Furthermore, the grasping computation

may benefit from information about the object of interest's mass distribution as well as compliance and friction properties. None of this information can be contained in a purely geometric model.

1.3. MOTIVATING APPLICATIONS

World-modeling is important to any robotic manipulation system, and the current research aims to be as general as possible. However, it is useful to identify some specific applications that would benefit from a model designed to support manipulation. The following application areas all involve non-repetitive tasks in environments that are not perfectly structured and therefore stand to benefit by the improved autonomy facilitated by the current research.

1.3.1. Explosive Ordnance Disposal

Commonly abbreviated EOD, Explosive Ordnance Disposal is the process of neutralizing an explosive hazard. Strictly speaking, explosive ordnance refers to unexploded munitions like artillery shells and mines, but colloquially EOD has come to include neutralization of other types of hazards such as Improvised Explosive Devices (IED) and other home-made bombs. The military and police departments are the primary consumers of EOD robots, which have become one of the primary tools for EOD operations due to the very high risk to human life. Figure 1-3 shows a typical robot used in EOD operations.

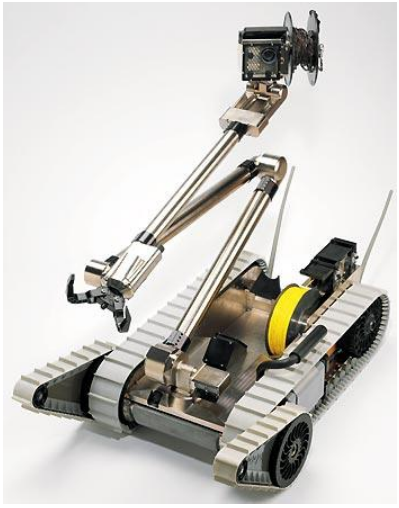


Figure 1-3. iRobot PackBot with EOD kit

The specifics of the techniques used to render explosive devices safe are not made publicly available in order to prevent potential bomb-makers from defeating the EOD tactics. However, the robotic manipulation tasks involved generally include removal of a hazardous explosive to a safe location where it can be detonated in such a way as to minimize its explosive yield. Therefore, the primary function of the EOD robot is to approach and move a hazardous or suspicious device. This must be done in a highly uncertain environment with little prior knowledge about the device itself. In terms of autonomy, the high degree of uncertainty in EOD operations requires that a human operator remain in the control loop. Currently, EOD robots are tele-operated either in joint-space or end-effector space. There is virtually no autonomy.

EOD operations would benefit greatly from improved automation. In a collaborative tasking situation, the operator would be able to select the device through a human interface, and its retrieval would be automated. Because the device in question may be detonated remotely by the enemy at any time, improved speed would be the most tangible benefit to this type of automation. EOD operations are also very stressful for the operator who would therefore benefit from the reduced work load. In order for this type

of automation to become possible, the relevant properties of the device must be determined from sensing and used to perform grasping analysis. These technologies are not yet mature enough to support fully this type of automated grasp, but they are active areas of research. The world-modeling techniques discussed here perform the vital role of organizing the sensor data and making relevant data readily available to the grasping algorithm.

1.3.2. Nuclear facility decontamination/decommissioning

Nuclear reactor facilities present unique challenges when they are to be decommissioned. Some areas of decommissioned reactor facilities have very high radiation levels. This makes the use of robotics a natural choice to comply regulatory requirements to keep human exposure levels "as low as reasonably achievable" (ALARA). The efficacy of mobile robotic manipulators in decontamination and decommissioning of nuclear facilities has been demonstrated at Argonne National Laboratory's Chicago Pile 5 research reactor. [Noakes, 1998] Additional demonstrations have been done with Pacific Northwest National Laboratory's Pit Viper robot, shown in Figure 1-4 [Roeder-Smith 2002].

Figure 1-4. INNL PitViper

Typical manipulation tasks associated with D&D activities include sectioning large fixtures, transport, unbolting, sawing, and packaging. [Noakes, 1997] These tasks involve a high degree of task uncertainty in a cluttered and/or constrained environment. This has limited the use of autonomy in D&D operations. The current research would be an integral part of a system capable of allowing certain D&D tasks to be partially automated. For example, a proper manipulator-focused world-model could contain mass properties for heavy objects, and not allow an operator to attempt to lift objects outside of the robot's design limitations. Clutter could be moved by a few mouse clicks in a graphical user interface instead of direct tele-operation. These improvements would greatly reduce operator workload and improve safety.

1.3.3. Nuclear facility glovebox automation

Handling of radioactive materials in nuclear facilities can be extremely hazardous to humans. Gloveboxes are a common solution to shield humans from the radiation associated with nuclear materials handling. These systems allow humans to handle materials by placing their hands in shielded gloves mounted to the walls of a large

shielded work space. This system presents significant risk in that a failure of any shielding material such as a glove tear can cause significant injury to the radiation worker. More automation is required in order to minimize these risks, and comply with regulatory requirements to minimize human radiation dose in nuclear facilities.

In some applications, it may be impossible or impractical to remove the human from the glovebox entirely. When a human must be present at the glove ports, improved automation can allow human/robot collaboration. Under the direction of the operator, robots in the glovebox can act as extra hands by moving materials in and out of the technician's workspace or by providing force-assist with heavy or bulky items. For this type of collaboration to be feasible, it must be demonstrated safe. The robots must use advanced motion planning algorithms in order to stay out of the human operator's way at all times. In order to accomplish this, the robot control system must provide motion planning algorithms with a robust real-time model of the state of every item in the glovebox, including the human hands.

1.3.3.1. The ALARA principle

The Nuclear Regulatory Commission requires that nuclear facilities keep radiation exposure to workers as low as reasonably achievable. This is known as the ALARA principle. Title 10 of the Code of Federal Regulations, Part 20 [2007] contains the Nuclear Regulatory Commission's (NRC) regulations regarding standard for protection against radiation. It defines ALARA as "making every reasonable effort to maintain exposures to radiation as far below the dose limits in this part as is practical consistent with the purpose for which the licensed activity is undertaken, taking into account the state of technology, the economics of improvements in relation to state of technology, the economics of improvements in relation to benefits to the public health

and safety, and other societal and socioeconomic considerations, and in relation to utilization of nuclear energy and licensed materials in the public interest." As robotics technology matures, robots become less costly and more reasonable to implement. Robots are therefore becoming an important part of satisfying the ALARA requirement.

There are three components to ALARA: time, distance, and shielding. This idea suggests that workers should limit duration of their exposure, maintain as much distance as possible from radioactive sources, and shields should be used whenever possible to limit exposure to humans. Traditional gloveboxes only address the shielding requirement. Workers must still spend significant amounts of time within arm's reach of very hazardous materials. Replacing the humans with robots would allow the other two ALARA components to be addressed. Human operators could operate robotic gloveboxes from any distance and exposure time would be limited to periodic maintenance of the automated glovebox.

1.3.3.2. The Advanced Recovery and Integrated Extraction System (ARIES)

The Advanced Recovery and Integrated Extraction System (ARIES) is a program developed at Los Alamos National laboratory to support the United States' commitment to reduce its stockpile of weapons-grade plutonium. The ARIES system was created to demonstrate the feasibility of an integrated process for dismantling nuclear weapons, reclaiming the plutonium and converting it to a form suitable to disposition and long-term storage. The system is designed to be modular and transportable so that it could be moved to other Department of Energy facilities. One of key aspects of the technology developed under the program has been significant automation of many glovebox functions. ARIES is an example of a robotic system for nuclear materials handling glovebox automation.

This type of system is a strong motivator for the current line of research. This section is a brief description of the automated ARIES components. [Coonley, 2008]

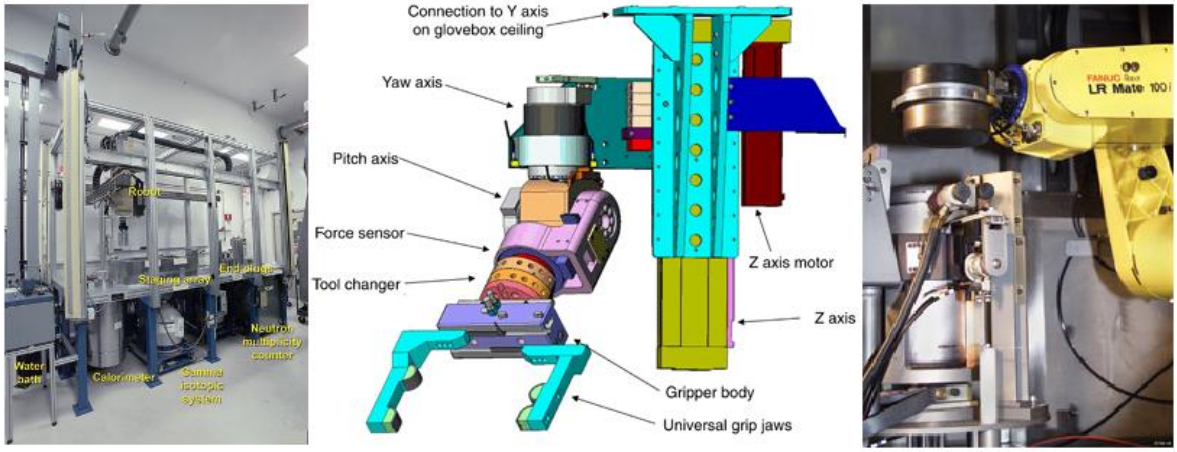


Figure 1-5. Automated ARIES modules

The first ARIES module to be automated was the non-destructive assay (NDA) module, shown in Figure 1-5 (a). NDA is the final step of the ARIES process. Cans containing plutonium oxides are accepted by NDA module that performs calorimetry, isotopic analysis, and neutron multiplicity measurements. This process allows the plutonium oxides to be characterized and accounted for prior to being placed in long-term storage. A single three Degree Of Freedom (DOF) gantry-operated robot moves standard-size canisters from an input station to a staging location, from the staging location to the various instruments, and finally to an output station when assay is complete. These tasks consist only of moving canisters from one pre-defined location to another and the environment is tightly controlled with low uncertainty. This makes automation relatively easy which is why the NDA module was the first to be automated.

The next ARIES module to be automated was the pit disassembly module (PDIS). PDIS contains two robots in a glovebox environment. The first is the ARIES robotic lathe

(ARL). The second is a gantry-mounted robot similar to the NDA robot, but with an additional two-DOF pitch/roll wrist. The PDIS robot is pictured in Figure 1-5 (b). The PDIS module accepts standard canisters and spherical weapons pits that contain special nuclear materials (SNM). The function of the PDIS module is to dismantle and separate these metal objects for classification and conversion. The ARL robotic lathe is similar to a machine-shop lathe, but is automated and uses tools specific to pit-disassembly operations. Its primary purpose is to bisect weapons pits into hemi-shells as the first step in extracting and converting the plutonium. It is also used to open sealed storage cans for processing other types of nuclear materials. The ARL uses several interchangeable fixtures for holding pits and canisters. These fixtures are mounted and dismounted by the PDIS gantry-robot. The gantry-robot also moves items to and from the lathe, and aids in dismantlement. The PDIS module does require a human at the glove ports for some tasks. The robot uses a safety-interlock system so that none of the automated components is in motion while the gloves are in the box. The PDIS module's automated components have a total of ten DOF and are all controlled by a single control system.

The third of the automated modules in ARIES is the robotic integrated packaging system module (RIPS or ICAN). ICAN receives a container of fissile material and packages it in a welded storage container for future disposition. They system removes any external contamination from the container, and verifies that that it is hermetically sealed. The module itself consists of three separate chambers. Material enters the system on the hot side chamber. The packaged material then enters an intermediate decontamination chamber. In the cold side chamber, the welded and decontaminated canister is surveyed before being released to a radiological control technician. The hot side chamber contains a commercially available 5-DOF Fanuc LR Mate 100i robot (Figure 1-5(c)), an automatic welding system, and a leak-check system. The robot

receives a can of fissile material, places it in a standard canister and transfers it to the welding system. The robot then transfers the welded canister to the leak-check system to verify the weld is sealed before passing the canister to the decontamination chamber. The decontamination chamber contains the fluid processing equipment necessary to remove external contamination electrolytically from the welded canister. The decontamination chamber also serves as an airlock between the hot and cold sides of the ICAN module. The cold side chamber contains a robot identical to the one on the hot side. This robot removes the canister from the decontamination chamber and processes it though a series of radiological surveys to verify that the exterior of the can has been successfully decontaminated. The robot then transfers the canister to the cold-side leak check system for a final verification that the weld is hermetically sealed. The canister is then released to a technician for a final survey.

ICAN is the most complex of the automated ARIES components with a total of 20 degrees-of-freedom. ICAN is fully automated, but always attended. This is possible because it is in many ways similar to factory floor automation. While the nature of the materials being processed presents some unique design considerations, there is little uncertainty in the environment or the task specification.

The automated ARIES modules demonstrate the usefulness of robotics in nuclear materials handling. The Department of Energy and the national laboratories have expressed a desire to increase the amount of automation in gloveboxes, and have committed to pursue the advanced technologies required to support this goal. In order to provide robotic solutions to more general-use gloveboxes, the control systems will need to respond quickly to new task requirements, and be capable of operating safely in less structured environments. For example, in small-batch experimental mixed-oxide fuel fabrication, the process will be a little bit different for each batch. The robotic system

must be able to accommodate some ambiguity in the task definition. Since many human-operated gloveboxes are already in use, it may make more sense to add robots to existing gloveboxes than to build a new system from the ground up. This means that robots could be sharing workspace with humans, and must be able to handle the unpredictable movements of a human operator safely. The current research aims to support the safe use of robots in such uncertain environments.

1.4. IMPORTANT TERMS AND DEFINITIONS

Research in robotics is spread over many disciplines, each with its own lexicon of terms and definitions. This can lead to confusion among similar sounding technical terms and incorrect assumptions. Before proceeding, it is important to define clearly some important concepts as they are used in the remainder of this work.

Path, motion, grasp, and task planning

The various distinctions between planning algorithms can be confusing and unclear. Planning, generally, is the process of mapping or translating between a given initial state and a desired goal state. Path planning, motion planning, grasp planning, and task planning can be differentiated based upon the types of states that they map. All are briefly summarized here.

Path planning maps a collision free position trajectory between and initial position state and a desired goal position state. Path planning algorithms are commonly associated with mobile robotics, and are used primarily for finding a collision-free path. Path planning on its own is not often referred to in this work, as most manipulator control architectures incorporate the collision-free path generation as part of the motion planner.

Motion planning adds higher-order kinematics to simple path-planning in order to generate a complete motion trajectory between two sets of kinematic states. A motion

plan typically provides at least a position, velocity and acceleration trajectory, although it is possible to include higher-order kinematics as well where very smooth motion is required. In addition to providing a collision free path, a motion plan provides a smooth motion profile and can enforce limits on actuator velocities and accelerations.

Grasp planning is the process of determining an appropriate end-effector state given the state of the object to be grasped. Exactly what properties of the object are important and what states of the end-effector must be determined vary widely by implementation. In addition, the analysis required can be quite complicated and is an active field of research in its own right. Much of the difficulty lies in that it is very difficult to generalize an approach because objects can be any shape, can be compliant or stiff, and might be fragile, slick, or have any number of other properties that are important in determining an appropriate grasp. In addition, end-effectors can range from simple parallel-jaw grippers to anthropomorphic hands with many degrees-of-freedom. Grasp planning and analysis is discussed in more detail later, but it is important to note that the result of a grasp plan includes a reachable pose for the manipulator's end-effector that can be passed to the motion planner.

Unlike a motion plan, the initial and goal states for **task planning** do not describe kinematic states of the robot, but rather the initial and goal states of something in the environment. The task plan must then describe a series of robot motions that will translate the environment from the initial state to the goal state. A task plan for a manipulator can be considered as an aggregated set of motion plans and grasp plans, which when performed in a particular order will leave the environment at the desired goal state.

An illustrative example would be transfer of a container from an input station to an output station. The task planner would break the task down into its constituent motions. The resulting task plan would look something like this:

1) Request a grasp plan from grasp planner

2) Move to a retrieval pose

3) Execute grasp

4) Move to a release pose

5) Release container

6) Move to stow pose

It is important to note the hierarchical relationship between task planning and motion planning. The robot control system will need to use a motion planner and a grasp planner in order to execute each step in the task plan.

Collision detection and avoidance

Collision Detection algorithms determine when a collision is perceived to be imminent. In their simplest form, they may calculate the minimum distance between objects and warn of a collision when that distance is less than a threshold value. They may also calculate the first and second derivatives in order to characterize the closure rate between objects. This can avoid detection of false collisions where the objects are moving away from or tangential to each other.

Collision avoidance algorithms change the commanded path of the robot in order to prevent a collision while still meeting the performance requirements of the task. Collision avoidance algorithms are one sense more powerful than collision detection in that they relieve the operator of some additional burden. However, they should be implemented carefully since they do cause the robot to follow a trajectory other than what was commanded.

World-model

The word "model" is somewhat vague. A model can be physical, mathematical or representational. Physical models can be built to check the fit of parts in an assembly. Mathematical models are more familiar to engineers. Fluid flows are modeled mathematically to be solved computationally. Collision detection algorithms rely on some mathematical description of the object in the environment. In this work, the term "world-model" is used in the representational sense. A world-model here is meant to be an abstraction of the actual world where its important attributes are catalogued and stored in some type of computer data structure. Elsewhere in the literature, the term "world-model" is used in the more mathematical sense, and may refer to a particular type of collision detection or avoidance scheme.

1.5. SCOPE AND OBJECTIVE OF RESEARCH

1.5.1. Scope

The purpose of this research is to provide a world-modeling technique to support articulated robotic manipulation. In this report, particular emphasis is given to applications in nuclear material handling. The demonstration provided in chapter 5 demonstrates suitability in a robot-augmented general radiochemistry glovebox. However, the modeling techniques presented are meant to be extensible and adaptable to any number of other application areas.

The robotic glovebox stands to benefit most immediately from this type of research because of the nature of the environment. The locations of objects and materials in a radiochemistry glovebox have a moderate degree of uncertainty, which necessitates adequate sensing and world-modeling in order to assure safety. However, the set of

objects, materials, and locations of interest is likely to be well-known in advance. This simplifies an implementation because the world-model can be easily populated with a significant amount of *a priori* knowledge.

1.5.2. Building on previous work at RRG

At the University of Texas, The Robotics Research Group (RRG) has developed several key technologies in robotic manipulation that this work aims to support and extend. The Operational Software Components for Advanced Robotics (OSCAR) (now available from Agile Planet as Kinematix [2009]) is a set of robotics libraries that has served as a repository of the robotics control technology developed at RRG. In many ways, it serves as a springboard for the current research. OSCAR provides most of the functions that the current work aims to support including advanced motion planning, and collision detection/avoidance. OSCAR also provides a method for work cell modeling. The current work aims to extend OSCAR's workcell modeling framework into a much more general world-modeling framework.

1.5.3. Objective

The objective of this research is to develop a world-modeling scheme that supports robotic manipulation and grasping. This report identifies what types of information such world-model must capture in order to support motion planning, collision detection/avoidance, and grasp analysis algorithms. This report also identifies and discusses relevant sensing technologies in order to determine what types of data are available to populate a world-model for robotic manipulation. The current research proposes a structured scheme for containing the identified information that improves real-time motion planning, grasping, and safety.

1.6. ORGANIZATION OF THE REPORT

The remainder of this report will be organized as follows:

Chapter Two is a literature review of current world-modeling techniques, as well as of the control system functions that a manipulator world-model must support. Relevant works discussing motion planning, collision detection/avoidance, and grasp analysis are discussed in order to provide the performance requirements of a world-model for manipulation.

Chapter Three presents an analysis of relevant sensing technologies. Common sensors used in robotics are investigated in order to gain a familiarity with updating and maintaining a world-model with sensor data.

Chapter Four presents the proposed world-model: a graph-based scheme for containing relevant information about the world that meets performance requirements as well as a review of current sensing technologies available for populating such a model.

Chapter Five summarizes a demonstration of the proposed modeling scheme. The demonstration provides a basis for evaluating how well the proposed solution improves the capability of robotic manipulators in a particular application domain.

Chapter Six summarizes the contribution of this research and proposes areas of future work.

CHAPTER 2 : LITERATURE REVIEW

The following review is intended to examine how certain elements of a typical manipulator control architecture are implemented in practice. A familiarity with these algorithms is essential in order to formulate a world-modeling scheme that supports these functions. The specific functions examined are collision detection, manipulator motion planning, and grasping. This is not an exhaustive list, but these three functions are chosen to focus the world-modeling research because they

- Are essential to effective manipulator control
- Require significant use of world-modeling information
- Reflect the spectrum of autonomy introduced in chapter one

Collision detection must be part of a robotic control structure at any level of automation down to basic tele-operation. Advanced motion planning becomes necessary as soon the level of autonomy moves beyond tele-operation. Grasping analysis for objects that cannot be pre-modeled is necessary to move true manipulator automation into unstructured environments. Since these functions require significant information about the environment and represent incremental advances in autonomy for both structured and unstructured environments, they provide an ideal springboard to motivate a world-modeling scheme to support manipulation across the entire autonomy spectrum.

2.1. COLLISION DETECTION

Collision detection algorithms report when contact occurs or is imminent between geometric entities. The problem of collision detection can be described as mathematically determining whether two volumes intersect in both space and time. The volumes ideally are 4D volumes created by extruding an object's solid geometry along a time-dependent

trajectory. The collision detection problem is then to determine if and where such 4D volumes intersect. Jimenez [2001] points out that while this solution is elegant in principle, in practice, generation of extruded volumes for objects undergoing both translation and rotation is a computationally expensive exercise. It is therefore unsuitable for real-time computation.

All collision detection algorithms exist essentially to avoid the 4D computation above. Lin and Gottschalk [1998] provide a review of collision detection algorithms categorized according to the type of geometric representation used. Figure 2-1 shows their taxonomy of 3D modeling techniques used for collision detection.

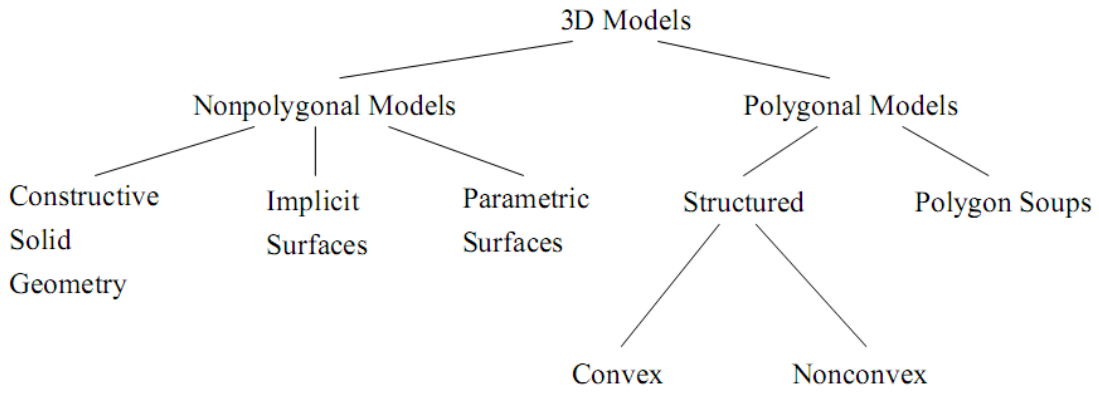


Figure 2-1. Taxonomy of 3D Collision Models

This type of classification is particularly appropriate for the current research since it examines collision detection algorithms according to the type of geometric representation used, and it is precisely this kind of information that the current research proposes to support.

2.1.1. Geometric modeling for collision detection

Polygonal models

Polygonal models model objects as meshes of polygons. The most general polygonal model is simply a list of polygons with no connectivity or topology information to relate the polygons to each other. This is often referred to in the literature as a "polygon soup" [Lin 1998]. In practice, most polygonal representations consist of a structured mesh of polygons, usually triangles, that form a closed manifold that closely approximates the geometry of the modeled object. This manifold constitutes a polyhedron with an arbitrary number of faces. A polyhedron is said to be convex if for every pair of points the line segment with endpoints *a* and *b* is contained entirely within *K*. Figure 2-2 shows an example of both a convex and non-convex polyhedron.

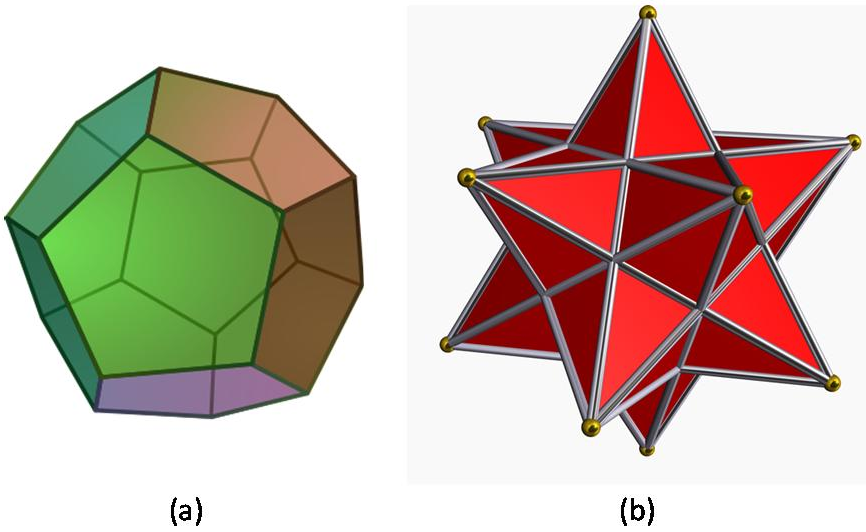


Figure 2-2. (a) Convex polyhedron (b) Non-convex polyhedron.

The computational geometry literature provides many algorithms for collision detection among convex polytopes. [Lin 1991][Jimenez 2001] Algorithms that support other geometries can run significantly faster if the objects are modeled as convex

polyhedra. For example, the Lin-Canny Algorithm has been demonstrated to run in linear time for the worst case. [Lin 1991] Pre-processing of the convex polyhedra can improve the speed to near constant time. [Dobkin 1990]

Constructive solid geometry

Constructive solid geometry (CSG) describes a modeling technique where a model is built by performing Boolean operations on solid geometric primitives such as blocks, spheres, cylinders, cones, and tori. Figure 2-3 shows an example of two CSG objects created by addition and subtraction of a cube and a sphere.

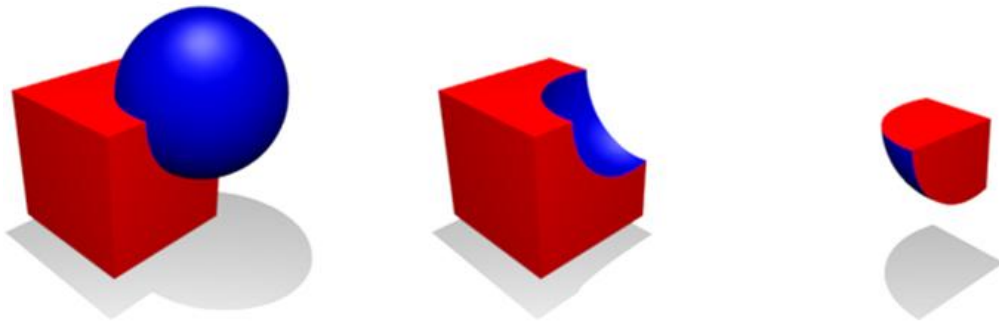


Figure 2-3. CSG object created from a cube and a sphere. (a) Boolean sum (b) Boolean difference (c) Boolean Intersection

Because these primitives and their Boolean combinations have precise mathematical descriptions, CSG objects are "clean" in that all edges are smooth and well defined, and there are no surface gaps caused by improper polygonal meshing. This makes them well suited to applications requiring high-precision. However, for objects with complicated shapes, the number of geometric primitives becomes quite high, and the mathematical description of the object becomes quite complex. This can make it hard to perform interference detection on CSG objects if they must be modeled with high fidelity.

Implicit surfaces

Implicit surfaces are also mathematically well-defined. An implicit surface consists of all points P(x,y,z) that satisfy Many primitives used in CSG modeling are implicit surfaces. Implicit surfaces have a few convenient mathematical properties. They are generally closed manifolds, which is good for solid modeling. Furthermore, the inside and outside of the manifold is straightforward to determine. If the point lies inside the manifold. Likewise, if the point lies outside. If the defining function is a polynomial in x, y, and z then the surface is said to be algebraic. If the algebraic surface polynomial is second order, the surface is said to be quadric. Quadrics can define many surfaces including spheres, cylinders, cones, ellipsoids, paraboloids, and hyperboloids. Algorithms exist for finding the intersection of quadric surfaces. [Farouki 1989] More control of the surface shape can be achieved by letting the polynomial function in x, y and z be of arbitrary order. Surfaces defined this way are called superquadrics. Superquadrics can be used to approximate shapes with sharp edges as rectangular solids or octahedral. Separation distances between superquadrics are more difficult to calculate. Sreedhar and Tesar [1990] were able to determine them numerically with a Newton-Raphson technique, but the solution did not converge reliably.

Parametric surfaces

Parametric surfaces are defined by mapping x and y values from a plane into a third dimension. Parametric surfaces do not form closed manifolds, and so solid geometry is often created by patching together several parametric surfaces. This technique is used in many CAD packages, manifested as Non-uniform rational b-spline (commonly called NURBS) surfaces as well as Bezier curves.

2.1.2. Collision detection at RRG

Early work

The RRG first approached the problem of collision detection with Non-polygonal models. A detailed account of this approach is provided in Knoll and Tesar [2007] and is summarized briefly here. The original approach modeled a robotic workcell as a single moving robot operating among static objects in the workcell. The approach was to model everything in the environment with geometric primitives that can be represented by parametric surfaces. The robot was modeled with cylispheres. A cylishpere is the Minkowski sum of a sphere and a line segment, or the result of extruding a sphere along a line segment. Figure 2-4 shows a Mitsubishi robot modeled with cylispheres.

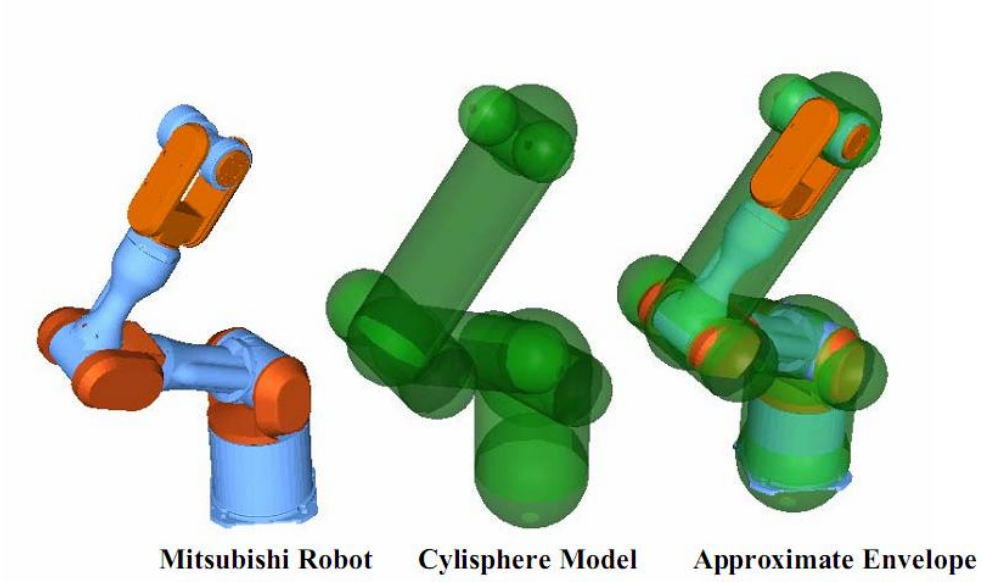


Figure 2-4. Cylisphere modeling

This type of modeling allowed the use of closed-form solutions for separation distance between cylispheres. In addition, an improved numerical solution was developed to determine the minimum distance between a cylishpere and a superquadric that gave

reliable convergence. [Perry and Tesar 1995] These formulations for the minimum distance between the robot and a given obstacle allowed a distance function to be expressed in terms of the robot joint angles,

(2.1)

Where d is a vector of distances of each link to the nearest obstacle. This can be differentiated to provide the closure rate between each link and the nearest obstacle.

— — —— (2.2)

This can be differentiated once more to provide the second order distance between each link and the nearest obstacle.

— (2.3)

The G matrix and H tensor in equations 2.2 and 2.3 represent the first and second order kinematic influence coefficients respectively. [Thomas and Tesar, 1982] These values are dependent only on the joint configuration of the manipulator, allowing the relatively simple computation of the zero through second order distances.

Recent work

The formulations discussed above rely on primitive solids, namely cylispheres, to model the manipulator. This has the advantage of straightforward closed-form distance formulae that allows fast computation times. However, the approach is very limiting in practice. The accuracy of the distance formulations is dependent upon how well the manipulator can be represented with cylispheres. For pure collision detection, it is permissible to accept this error since it is easy to ensure that the model fully envelops the actual geometry. In the worst case, the error in the distance calculation is simply additional safety margin. However, this artificially limits the usable workspace of the

robot. More importantly, a manipulator is of little use if it cannot get very close to some objects, which requires a much higher fidelity representation of the robot than cylispheres can provide.

Knoll and Tesar [2007] recognized that polygonal models were required to achieve sufficient fidelity in a manipulator's geometry. The ability of polygonal meshes to represent solid geometry is evident in the field of computer graphics that relied almost exclusively on triangulated mesh representation. An added benefit of using polygonal representations for robotics is the wide availability of commercial software packages that make it very easy to create and export models. Any modern CAD package can export a triangulated model. A discussion of collision detection algorithms for polygonal models follows.

2.1.3. Gilbert-Johnson-Keerthi distance algorithm

The Gilbert-Johnson-Keerthi distance algorithm (GJK) is well suited to computation of the minimum distance between polygonal models although it is not strictly necessary that the models be polygonal. It is, however, necessary that the objects be convex. For non-convex models, it is necessary to section the model into convex parts in order to perform GJK distance computation. The GJK algorithm is an iterative technique for computing the distance between two objects, A and B. The distance is given by

(2.4)

It is more useful in motion planning to know between precisely which two points on A and B this minimum distance occurs, i.e. and for which

. The GJK algorithm can be used to provide this set of point in addition to the distance between them.

The distance between A and B is determined by the computing the distance from the configuration space origin to the point on the configuration space obstacle (CSO) closest to the origin, i.e.

(2.5)

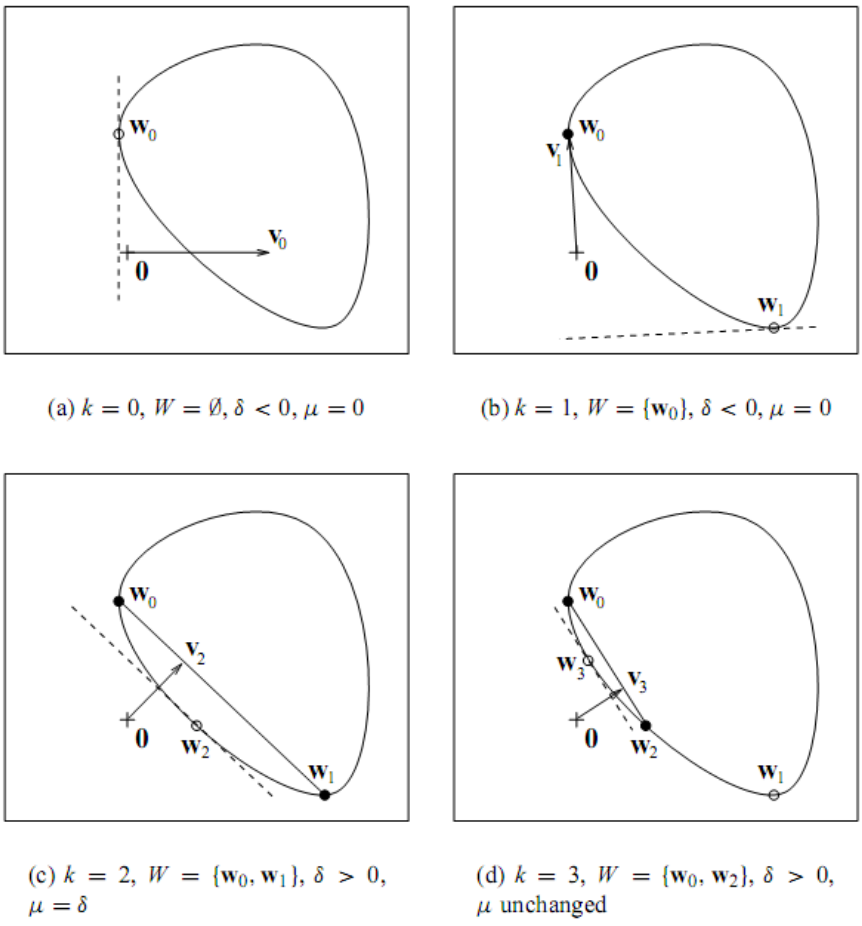
Where the Minkowski difference $A - B$ is the CSO and is the point in C closest to the origin, i.e,

and (2.6)

For each iteration, GJK constructs a simplex that lies in the CSO that lies nearer to the origin than the simplex generated at the previous iteration. A simplex is defined as an n-dimensional polytope with n+1 vertices. Simplices in zero, one, two and three dimensions are points, line segments, triangles, and tetrahedra respectively. For the following description of how GJK works, let W_k be the set of vertices in the simplex at iteration k and $\boldsymbol{v}_k$ a vector from the origin to the point on the simplex that is closest to the origin. For each $\boldsymbol{v}_k$, a support point is found by applying a support mapping $s_c(\boldsymbol{v})$ such that

and (2.7)

The result of this operation at iteration k is a point, which is then added to W_k. Figure 2-5 gives an example of the first four iterations of the GJK algorithm.



(a) $k = 0$, $W = \emptyset$, $\delta < 0$, $\mu = 0$

(b) $k = 1$, $W = \{\mathbf{w}_0\}$, $\delta < 0$, $\mu = 0$

(c) $k = 2$, $W = \{\mathbf{w}_0, \mathbf{w}_1\}$, $\delta > 0$, $\mu = \delta$

(d) $k = 3$, $W = \{\mathbf{w}_0, \mathbf{w}_2\}$, $\delta > 0$, μ unchanged

Figure 2-5. Four iterations of the GJK algorithm. [Van Den Bergen, 2004]

Figure 2-5 (a) shows the CSO, the origin, and the initial state of the calculation. W contains no points, and $\boldsymbol{v}_0$ points to an arbitrary point in the CSO. The value w_0 is the support mapping of $\boldsymbol{v}_0$. Figure 2-5 (b) shows the first iteration where w_0 has been added to W and is the only vertex in the simplex. Therefore v_1 must necessarily point to w_0 and w_1 is the support mapping of $\boldsymbol{v}_1$. Figure 2-5 (c) shows the second iteration. The simplex now contains w_0 and w_1, and $\boldsymbol{v}_2$'s length is the minimum distance between the origin and the simplex. Thus, w_2 is the support mapping of $\boldsymbol{v}_2$ which is then added to the simplex for the third iteration. It is clear in Figure 2-5 (d) that $\boldsymbol{v}_k$ is getting closer and closer to the point

on the boundary of the CSO closest to the origin. Unconditional convergence is demonstrated in Gilbert and Foo. [1990]

Software Library for Interference Detection

A robust implementation of the GJK algorithm is available in the Software Library for Interference Detection (SOLID). [Van Den Bergen, 2004] The SOLID library is distributed both commercially and open-source under the generic public license. Like many implementations polygonal collision detection techniques, the SOLID implementation exploits temporal coherence. Temporal coherence means that the states of objects in the environment do not change rapidly with respect to time. This allows certain aspects of previous collision calculations to be cached and reused in subsequent computations. In the description of GJK above, the initial vector v points to an arbitrary point in the CSO. However, for all but the first computation in time, the number of iterations to convergence can be reduced significantly by using the converged v from the previous time step. It is unlikely that v_t is very far from $v_{t+\Delta t}$. Additionally, objects that are not moving toward each other at time t need not be checked for collision at time $t + \Delta t$.

The SOLID GJK implementation also uses bounding volumes to decrease computation time. Rather than checking the exact geometry of all objects for all computations, the algorithm checks for intersections between Axis-Aligned Bounding Boxes (AABBs). The AABBs are hierarchical in nature. An object of any geometry can be contained in a large box aligned with the Cartesian axes. The SOLID algorithm checks for overlap of the bounding boxes before resorting to the more computationally expensive GJK algorithm. A consequence of this scheme is that the computation time is very fast

when objects are not near collision, but it slows considerably as the collision density increases.

2.2. MOTION PLANNING

Motion planning is the process of generating a trajectory or set of trajectories that move the robot from one state to another state. For robotic manipulators, this typically results in a discrete trajectory of joint states that will move the end-effector from an initial state to a desired final state. This trajectory is calculated to satisfy any number of performance criteria. At the most basic level, the trajectory must assure that no link of the robot makes undesirable contact with the environment. However, many other criteria can be optimized by a well-constructed motion plan, particularly where input redundancies can be exploited. For example, joint torques can be kept as low as possible, or the highest possible motion capability at the end-effector can be optimized. Because the goal of motion planning is to produce joint trajectories, it is often useful to map the motion planning problem to the manipulator's joint space. In the literature, the input space of the robot is referred to as the robot's configuration space or c-space.

Motion planning problems can be broadly characterized as either static or dynamic. A static motion planning algorithm presumes that the performance criteria do not change during execution of the planned trajectory. A dynamic motion planning algorithm is capable of handling changing performance criteria, such as avoiding moving obstacles in the environment. The approaches to motion planning in the literature can be divided into four categories [Aggrawal 2009]:

- Skeleton approach
- Cell decomposition
- Potential field

- Mathematical programming

Each of these categories is discussed briefly below with an emphasis on what each requires in a world-model.

2.2.1. Skeleton approach

The skeleton approach can be thought of as planning a trip on a road map. The road-map, or skeleton, is created by mapping the free space of the robot onto a network of one-dimensional lines. The initial and goal states in the configuration space are mapped onto this network. This mapping satisfies the most basic motion planning requirement of obstacle avoidance by assuring that only trajectories in free space are available. The motion planning problem is then reduced to a graph search that is done in order to satisfy any number of additional performance criteria. This is analogous to planning a road trip on a road map. First, the departure point and destination are located on the map. Since it is assumed that a car must remain on the road, the road map represents the entire free space. Then any number of routes can be identified that allow travel between the initial and final states. A route will be selected in order to optimize some combination of performance criteria such as minimum time, minimum distance, or best fuel economy.

The details of mapping the free space to a one-dimensional network are of course non-trivial. Methods for performing this mapping are presented in [Canny, 1988] and [Hwang and Ahuja, 1992]. The details are not presented here, but it is worth noting that this mapping provides an abstract model of the free space of the robot and can therefore be considered a type of world-model.

2.2.2. Cell decomposition

In this approach, the configuration space of the robot is subdivided into cells. The cells can be object-dependent, which means the cell boundaries match obstacle boundaries. Object-independent cells are typically grids of a predetermined resolution. Depending on the application, the decomposition may occur in physical space or in the robot's configuration space. In either case, the cells that contain portions of obstacles are identified in some way. The cell decomposition of the object-dependent model is computationally expensive, as are intersection computations. The object-independent formulation is far more efficient, but limits the resolution of the obstacle representations to the resolution of the grid. This approach is popular for mobile robotics where the grid need only be 2-dimensional and high resolution is typically not important. It has the additional benefit that the grid is easy to populate with many commonly used sensors. [Thrun 2002] In mobile robotics literature, object-independent cell decomposition techniques are is referred to as grid occupancy algorithms. For manipulators in three dimensions, cell-decomposition algorithms are characterized by relatively slow computation times, and are typically limited to static environments.

2.2.3. Potential field algorithms

Potential field algorithms use a physical representation of the environment to guide the motion of the robot. The environment is modeled as the sum of several force potential functions. Obstacles are modeled by high potential areas and goals are modeled as areas of low potential. The robot is then influenced by the artificial forces imposed on it by following the negative gradient of the sum of the potential functions in the environment. This process of following the gradient of a potential function is frequently referred to in the literature as hill climbing. [Kuipers, 2000] Khatib [1996] and others in the literature applied this method in a way that maps the artificial forces on the

manipulator to joint space as artificial torques. Khatib defined the following obstacle and goal potential functions, U_O and U_{xd}, respectively.

(2.8)

(2.9)

In equation 2.8, η is a scaling constant, ρ is the distance between the manipulator link and an obstacle, ρ_0 is the maximum distance at which the obstacle should be considered. In equation 2.9, $(x-x_d)$ is the distance to the goal. Figure 2-6 shows an example of the potential function associated with a triangular obstacle.

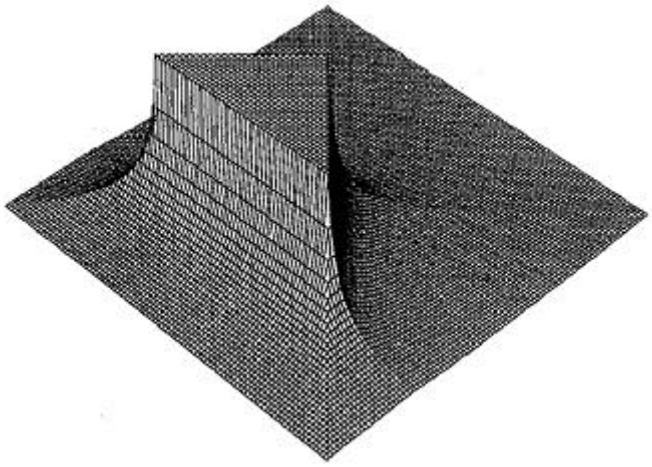


Figure 2-6. Potential field associated with a triangular obstacle [Hwang and Ahuja, 1992]

The forces associated with these potential functions are computed by calculating the gradient of the potential function. As ρ approaches zero, the repulsive force, . Likewise as the distance from the goal approaches infinity, .

These artificial forces can be made to act on the manipulator by mapping the forces into the manipulator's joint space. This is done using a matrix of first-order kinematic influence coefficients. [Spencer and Tesar, 2007]

(2.10)

Where represents the magnitude of the artificial forces, and is a matrix of kinematic influence coefficients relating the first order distance to the obstacle (or goal) to the joint velocities. The result of this mapping is a vector of joint torques that will move the robot along a collision-free path to a goal position. Figure 2-7 shows an example of a seven-DOF robot moving around a spherical obstacle to a goal pose.

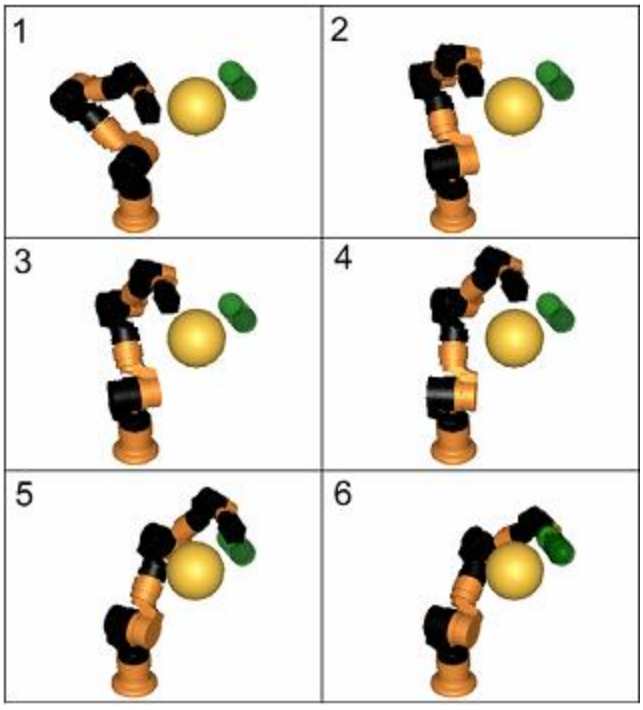


Figure 2-7. Spatial manipulator avoiding an obstacle by artificial joint torques.

The most common problem associated with potential field algorithms is the existence of local minima. When the potential functions are summed, it is possible that

the resulting potential contour will have local minima that are not at the desired goal location. This can result in the robot following the gradient into the minimum and becoming trapped. The artificial potential field approach is typically not suited to global motion planning due tendency to select less-than optimal paths. The algorithm will not move through a passage between closely spaced obstacles because the sum of the potential functions in the passage creates a gradient that drives the robot away from the passage.

2.2.4. Mathematical programming

Mathematical programming treats obstacle avoidance as a mathematical optimization problem. The obstacle avoidance constraints are imposed as a set of inequalities on the configuration parameters. The algorithm determines a path between initial and final states by minimizing a particular scalar quantity. The optimization is highly non-linear, so the optimization is performed numerically. Mathematical programming methods have been implemented and are characterized by long computation times and suitability for only very simple cases. [Aggrawal, 2009]

2.2.5. Motion planner implementations

Aggrawal performs a very comprehensive survey and analysis available motion planners in his thesis. He evaluates several motion planner implementations based on the following metrics:

- **Speed and efficiency** – the computational efficiency of the motion planner.
- **Completeness** – Describes the ability of the motion planner to find a solution if one exists, regardless of computation time.

- **Ability to incorporate multiple performance criteria** – Describes whether the algorithm can be extended beyond basic obstacle avoidance.
- **Generality** – Describes the ability of the motion planner to operate without simplifying assumptions. A very general motion planner would work on any number of manipulators, each with of any number of links. It would also work in both static and time-varying environments.

Several trends are apparent when different algorithms are evaluated by these metrics. Aggrawal's analysis is condensed and summarized in Table 2-1.

Table 2-1. Motion planning comparison

Approach	Speed	Completeness	Task-Based Criteria	Generality
Skeleton	May be efficient for single manipulators in static environments	Yes	Only on less efficient implementations.	Single manipulators in static environments
Cell Decomposition	Moderate: typically 1-10 minutes	Yes for most recent implementation	Only simple criteria	Efficient for single manipulators in static environments
Potential Field	Efficient	Yes	Yes	Any manipulator, but only for local motion planning
Mathematical Programming	Very Slow	Only in simple cases	In some implementations	Any manipulator.

Table 2-1 suggests that no single approach to motion planning is suitable for online computation on its own. Computation time is simply too long for efficient online global motion planning. In terms of modeling, this suggests that it may be useful to maintain a database of motion plans between locations of interest in a particular environment, thus avoiding the costly motion planning computation. Potential field algorithms could then locally augment the cached motion plan to account for any previously unmodeled obstacles.

2.3. GRASPING

Robust and efficient grasp analysis is essential to autonomous manipulator behavior in unstructured environments. Grasping is a difficult problem because the problem space is so vast. The number of objects that a manipulator might grasp is

essentially infinite. There are numerous types of graspers available that each have different kinematic and dynamic capabilities. A simple parallel jaw gripper is easy to control but lacks flexibility. Different jaws must be fitted to match the objects to be grasped. The human hand has over twenty degrees-of-freedom, and is often considered the *ne plus ultra* of small, flexible graspers. Many attempts to emulate the human hand's capability in robotic hardware have been made [Bigliotti, Lotti, Melchiorri, & Vassura, 2004.] [Shadow, 2003]. The flexibility offered by such a grasper introduces redundancies that give more choices for suitable grasps. While additional choices are always desirable, it should be recognized that the redundancy resolution and control of a hand with twenty or more degrees-of-freedom is a difficult problem in itself [Rosell, Suarez, Rosales, & Garcia, 2009]. The following is a review of some grasping basics and some recent approaches taken to the grasping problem. An emphasis is made on the modeling techniques required of various techniques. Much of the analysis can be found in more detail in Bicchi and Kumar's review paper [2000].

2.3.1. Grasp analysis

Before any meaningful discussion about grasping can take place, some consideration must be given to the question of what constitutes a "grasp." In order for an object to be properly grasped, the robotic gripper must make contact with it in such a way as to keep it in static equilibrium with respect to the gripper. This concept is referred to in the literature as closure, and any proper grasp must be closed in this way.

Force and form closure

Consider an object grasped at N contacts. The contacts are typically modeled as point contacts. The point contacts can be categorized according to whether or not the friction and compliance between the gripper and object are included in the model. A

frictionless contact is only capable of exerting a force along the common normal. A frictional contact can also exert a tangential force along any direction perpendicular to the common surface normal. A soft contact can also exert a moment about the common normal. With the force capability of the contacts defined, the grasp can be analyzed to determine its closure. A matrix representing these forces can be constructed and is referred to in the literature as a wrench matrix. For each contact, a wrench matrix W is constructed of the unit wrenches corresponding to the contact forces, ${}^{i}\boldsymbol{w}_{N}$, ${}^{i}\boldsymbol{w}_{T}$, and ${}^{i}\boldsymbol{w}_{\theta}$. The wrench matrix is multiplied by an intensity vector, $\boldsymbol{c}$ to represent the magnitudes of the forces and moments in the wrench. If represents an arbitrary external wrench, the grasp is said to be force closed if at each contact an intensity vector λ exists that satisfies all frictional contact constraints and the following equality:

(2.8)

Another way to think of this is that one could grab the grasped object and attempt to translate and rotate it about any axis without the object moving relative to the gripper. A special case of force closure exists when the following equality is satisfied at each contact:

(2.9)

Under this condition, force closure is satisfied regardless of the magnitude of the external wrench applied. In other words, the grasp no longer relies on friction to maintain static equilibrium. This special case is called form closure, and is a much stronger grasp than the more general force closure.

Measures of grasp performance

Two commonly used measures of grasp quality are noted here that reflect the ability of the grasp to resist and exert forces. A measure of the ability of a grasp to resist

external forces is grasp stability. Stability describes the ability of a grasp to reject disturbances of external wrenches. This can also be described as the stiffness of the grasp. Grasp stability can be evaluated by finding a grasp stiffness matrix by combining the stiffness matrices at each contact. The signs of the eigenvalues of the stiffness matrix indicate the stability of the grasp. Figure 2-8 provides an intuitive understanding of grasp stability.

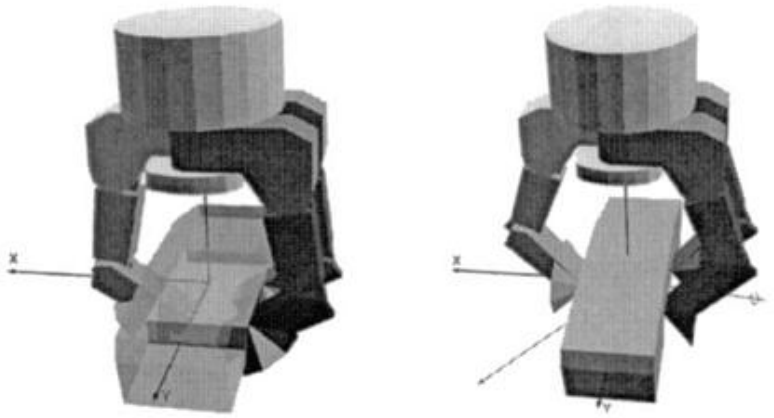


Figure 2-8. (Left) Unstable grasp. As the fingers close, the object will slip. (Right) Stable grasp [Miller and Allen, 1999].

Miller and Allen [1999] approach the problem by finding the convex hull of the space of all possible wrenches given the contacts and associated constraints. If the convex hull contains the origin, the grasp is stable. A measure of the grasp quality can then be made by examining the worst-case wrench rejection capability. This is done by expanding a 6D ball from the origin until it intersects the convex hull of the wrench space. The radius of such a ball, ε, is a good indication of grasp stability.

The second measure of grasp quality is the ability of the grasp to exert wrenches on the object. Clearly, this measure is related to the first measure. Miller and Allen suggest the volume of the convex hull of the wrench space as a metric. This gives an idea

of the average ability of the grasp to exert forces. The direction in which the ability to exert forces is at its minimum is exactly opposite the worst case scenario for external wrench rejection that is measured above. Using the volume of the convex hull acknowledges the ability of the grasp to exert forces in other directions.

2.3.2. Grasp synthesis

The discussion above provides a framework for understanding grasps and evaluating the quality of a known grasp. The problem of synthesizing a grasp, however, is much more difficult. Conceptually, the synthesis problem consists of finding the intersection of the force closure space of the object with the configuration space of the gripper. Both of these spaces are very large and complicated, making a brute-force space-search intractable. Much of the recent literature attempts to mimic the human cognitive approach to grasping. Humans use a limited set of generic hand configurations that are then adapted slightly to the type of object to be grasped. Successful algorithms based on this idea have been demonstrated.

Miller, Knoop, Christiansen and Allen [2003] demonstrate a method where the objects are modeled as geometric primitives. The algorithm uses a rule-based approach for selecting a set of pregrasp configurations of the hand depending on the type and configuration of primitives in the model. They then use their own GraspIt! software to evaluate the candidate pregrasps and select the best one based on the GraspIt! simulation. The algorithm successfully identifies adequate grasps. Figure 2-9 shows the grasps computed for the Barrett hand on a flask.

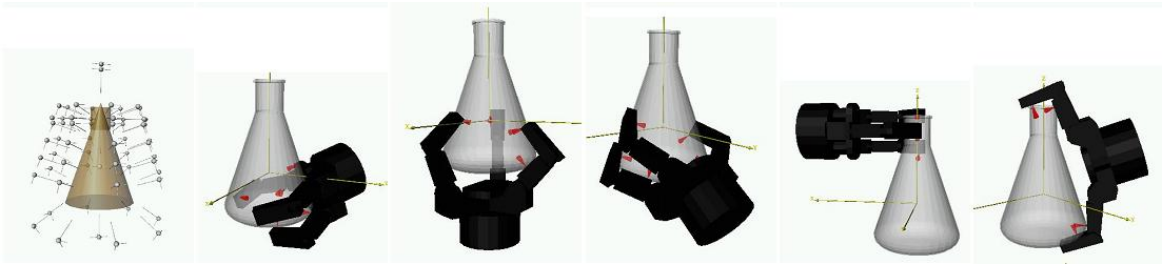


Figure 2-9. Primitive model of flask and grasps computed by Miller et al. algorithm.

It takes from several seconds to several minutes to determine a grasp. This may or may not be acceptable for a given application. In addition, the algorithm requires primitive representations of objects. These models may be difficult to construct from machine sensing, and therefore must be pre-modeled. If the objects must be pre-modeled, an appropriate grasp can quite easily be pre-determined, obviating the need for online grasp synthesis.

Berenson and Srinivasa [2008] demonstrate a method that utilizes grasp preshapes, but does not require a primitive representation of the object to be grasped. They describe a grasp as a preshape and a pose. The preshape is a set of joint values of the hand, and the pose is a 6D hand position and orientation (HPO). A grasp is considered valid if it

1) Does not collide with other objects in the environment during execution of the grasp.
2) Provides force-closure.

They use the method of Li, Jiaxin, and Fu [2007] to determine a grasp preshape. The method still requires a 3D model of the object to determine a preshape. However, it is permissible to use a sampling of surface points on the object. This type of data is much easier to obtain via sensing than the primitive shape representation required by [Miller, et al, 2003]. With preshapes determined, the algorithm searches for an HPO that provides a

grasp that satisfies the validity criteria above. The HPO validation step is relatively expensive computationally, but generates a collision free, force-closed grasp for a given gripper and environment in a few seconds. The method was successfully demonstrated on a WAM arm with a Barrett hand.

In terms of modeling, one attribute of both methods above is that both require a complete 3D model of the grasped object. In practice, this can be very limiting in uncertain environments where this information must be acquired via sensing. Saxena and Driemeyer [2008] present a grasp synthesis technique that does not require 3D modeling of the grasped object. If multiple images of an object are taken from different positions, it is possible through image processing to determine distances to points in the images. This is discussed in more detail in chapter 3. The method presented by Saxena and Driemeyer uses this idea, but not to model the entire scene, or even the entire object. Instead, each of two or more images is searched for a good grasping point. The grasping point is determined by teaching the robot good grasp locations in a set of teaching images. The robot then learns to identify good grasping points in a 2D image. If a suitable grasp point is located in two images taken from different positions, the location of the grasp point can be located in three dimensions. Only the local area of the grasp needs to be triangulated, thus avoiding the computational expense of modeling the entire scene.

The algorithm was demonstrated on the STAIR (STanford AI Robot) robot, equipped with a webcam and a parallel jaw gripper. The ability to synthesize a grasp without 3D modeling data is a strong advantage of this approach. However, it should be noted that this method determines a single grasping point for a parallel gripper, and it is not clear how this approach could be generalized to more complicated, flexible grippers.

2.4. WORLD-MODELING

The previous sections have discussed some functions of a manipulator control system that must be supported by the world-model. This section examines some world-modeling techniques developed previously, and discusses how well they support manipulator control.

2.4.1. Workcell modeling at RRG

The current world-modeling techniques used by RRG were developed for the Trauma Pod surgical workcell, and implemented in the Operational Software Components for Advanced Robotics (OSCAR) libraries. The world-model is contained in an extensible markup language (XML) file, and is designed to support multiple robots working in the same work space. XML provides a format that is easily shared across several systems. This was an important consideration for Trauma Pod, which consisted of multiple robots run by different control systems. XML also provides a robust data structure because the structure and content of the file are validated by the XML schema. The structure of the OSCAR workcell model is hierarchical in nature. At the top of this hierarchy is the workcell. The workcell contains manipulators, tools, frames-of-interest, and obstacle models. Each of these contains its own set of properties. The structure of the XML file is defined by an OSCAR schema and is presented in Figure 2-10.

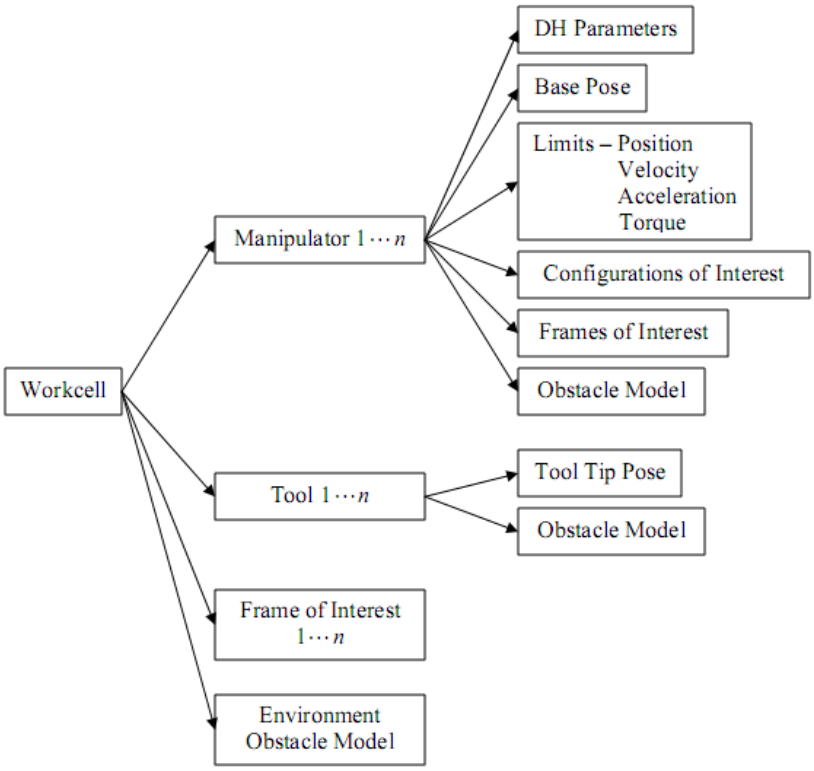


Figure 2-10. Structure of OSCAR workcell model [Knoll, 2007]

The workcell manipulators are defined by their base pose and Denavit-Hartenberg (DH) parameters that describe the manipulator geometry. Obstacle models of each manipulator link are attached to the appropriate DH frame. A primitive (cylisphere, sphere, box etc.) is stored for obstacle avoidance, and a polygonal mesh is stored for collision detection. The same obstacle modeling is used for tools and obstacles in the environment. Additional properties included are the kinematic and dynamic joint limits, as well as configurations-of-interest and frames-of-interest. *Configurations-of-interest* are sets of joint values that the robot is likely to re-use many times. Likewise, *frames-of-interest* represent positions and orientations in the environment that the manipulator is likely to visit frequently. Because XML is extensible, other manipulator properties could be included as well if a particular implementation required them.

Tools are defined external to the manipulators in the hierarchy. This is because for manipulators equipped with tool changers, it is possible that a tool will be used by multiple manipulators. A tool in the workcell model is defined by a tool tip pose and the tool's obstacle model. The tool tip pose is relative to the last DH frame of the manipulator, so if a tool is to be used by different manipulators, a tool tip must be stored for each manipulator that will use the tool. The model does not include information about the tool's compliance, friction, or inertia properties, but the extensible nature of the XML schema allow them to be added if they are important for a particular implementation.

In addition to the frames-of-interest associated with each manipulator, global frames-of-interest are also defined for locations used by multiple robots. These frames are also used to position various subcomponents of the workcell in the same global reference frame. This allows the system to determine their relative positions so that the system can be properly calibrated.

Figure 2-11 displays a simulation image of the Trauma Pod workcell in operation. The entire image is created from data contained in the workcell model. The frames-of-interest are visible, and both the geometric primitive and polygonal mesh obstacle data are pictured. The figure gives a good idea of how well the OSCAR workcell modeling scheme captures relevant information about the environment.

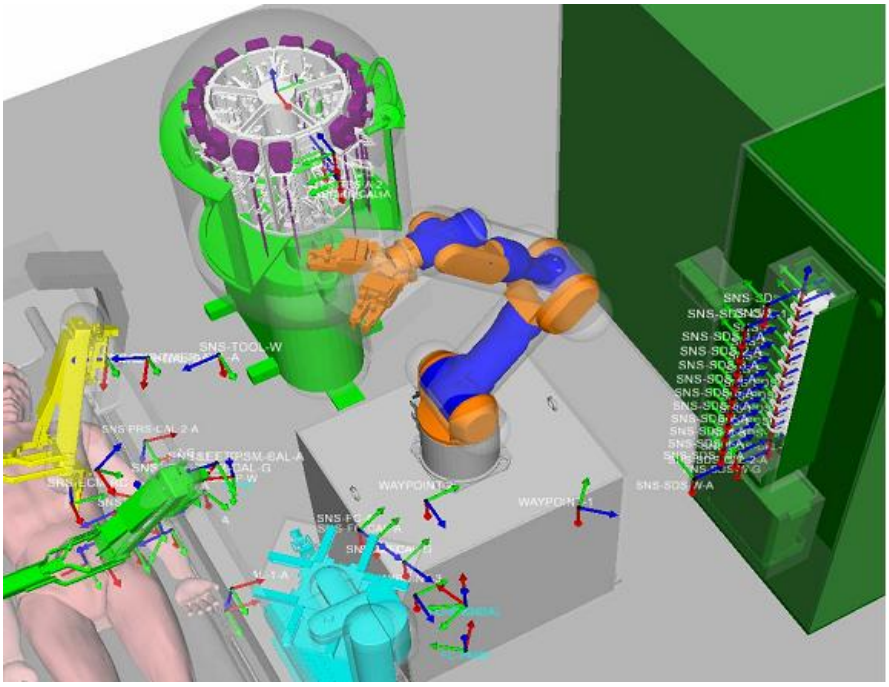


Figure 2-11. Image created from an OSCAR workcell model [Knoll, 2007]

The OSCAR workcell model has many desirable features in a manipulator world-modeling scheme. XML's extensibility allows the model to accommodate additional data on an application specific basis. XML is easily used across multiple platforms, and provides inherent robustness through the OSCAR workcell model schema. It permits obstacles to be modeled in more than one way in order to support different elements of the control system, which improves computational efficiency by assuring that an algorithm receives the most efficient geometric representation available.

The OSCAR workcell model fails to capture relationships between model elements well. The strict hierarchical structure can lead to representational redundancies, and there is some data that may be useful which cannot easily be added. For example, a frame of interest must either by contained by a single manipulator or added globally. There is no obvious way to globalize a frame of interest if it becomes interesting to other manipulators. There is no information about which manipulators can reach a given frame

of interest. Frames-of-interest may themselves be related to each other. An example of a useful relation between frames or configurations-of-interest would be the motion plan that moves a manipulator from one frame or configuration to another. In the discussion about motion planning above, it is apparent that if such trajectories could be stored in the model, considerable computation time could be saved. There would be some advantage in recognizing that relations between modeled objects may not follow a strict hierarchy, and it may be useful to identify a relationship between any two elements in the model. Closed-headed arrows indicated dependencies. Open-headed arrows indicate information flow without dependency.

2.4.2. Spatial Semantic Hierarchy

The Spatial Semantic Hierarchy (SSH) is proposed as both a model for the human cognitive map, and as a method for mobile robot exploration and mapping [Kuipers, 2000]. The SSH consists of five different levels of representation of the environment, arranged in a hierarchy, with higher levels depending on lower levels. Figure 2-12 shows the five levels, the types of information contained in each, and some relationships between parts of the model.

The *sensory* level is the interface to the robot's sensor suite. The *control* level describes a set of control laws that relate the robot's behavior to the environment. The *causal* level can also be thought of as the planning level. This is where appropriate control laws are determined and passed down to the control level. The *topological* level captures the relationships between places, paths, and regions. These relationships may be connectivity, order, or containment. It is this topological information that is not well captured by the workcell modeling in OSCAR. The *metrical* level represents a global

geometric map of the environment. This level applies mostly to mobile robotics where it is useful to relate several local frames of reference to a global frame of reference.

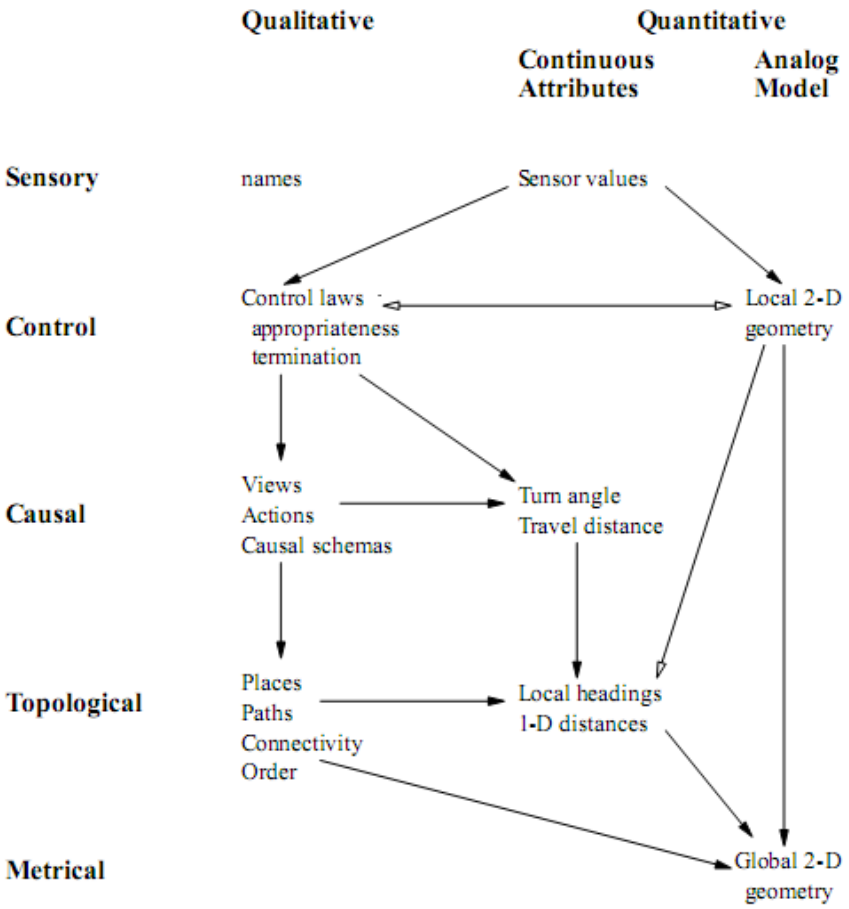


Figure 2-12. The Spatial Semantic Hierarchy.

The hierarchy presented in Figure 2-12 is significantly different from the control system schematic presented in Figure 1-2. The control architecture presented in Chapter 1 does not consider sensing, control and planning to be part of the same spatial model as topological and metrical information. Also, since the SSH was developed primarily for mobile robotics, it does not explicitly support the idea of contact-interaction. It works well for mobile system control and mapping, but it is not clear how it could be useful to

manipulation. However, the idea of capturing topology as well as geometry is important to both mobile robots and robotic manipulation. Kuipers suggests that for human cognition and mobile robotics, topology may be more important than specific geometry of the environment. For non-contact interaction, this can be seen with manipulators where the existence of a collision-free path can be determined with only very coarse geometric information. If this topological relation between manipulator states were captured in the world-model, significant computation time could be saved by not re-computing the path each time motion is required between the two states.

2.4.3. Mobile Manipulation Database

Philippsen, Nejati, and Sentis [2009] suggest a world-model for mobile manipulators that they call the Mobile Manipulation Database (MMDB). The stated role of the MMDB is to collect information relevant for the interaction between various components of a robotic system. This fits nicely with the architecture presented in Figure 1-3, where the world-model serves as a repository of environmental information. The MMDB is a graph-based searchable database that allows the robot control system to quickly locate and use environmental data. A brief description of a graph data structures, and how they are used in the MMDB follows.

Graphs as data structures

The OSCAR workcell model schema is an example of a tree data structure. Each node in a tree has an arbitrary number of daughter nodes, but can only have one parent node. As mentioned in the discussion of the OSACAR workcell, this approach is limited in its ability to represent the topological relationships between nodes. A graph consists of a set of nodes that are joined to each other by edges or links. The links can either be directed, meaning that they can only be traversed one direction, or they can be

bidirectional. What separates a graph from a tree is that any node of a graph can be linked to any other node in the graph without any inherent hierarchical nature. A tree hierarchy can be considered a specific type of graph. Several efficient algorithms exist for traversal of a graph [Collins, 2005]. The MMDB graph is made up of *links* and *nodes*. Each link and node is assigned a unique numerical *ID*, and is identified with a *type* and *tag*. Figure 2-13 shows a room and the underlying graph structure used to model it. Figure 2-14 shows how the abstracted graph, with the node ID, type and tag clearly identified. The MMDB nodes and links are discussed in more detail below.

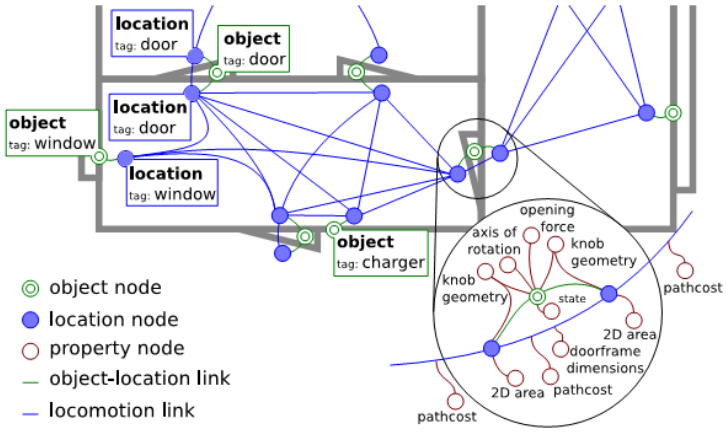


Figure 2-13. MMDB model of a room with doors, windows, and charging station.

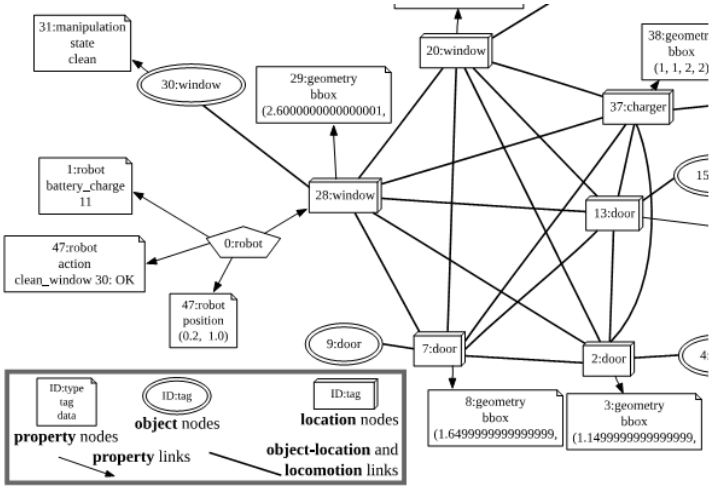


Figure 2-14. Abstracted MMDB model.

MMDB nodes

Nodes in the MMDB are pieces of information that are useful to the robot. They are categorized as object nodes, property nodes, or location nodes. The entities modeled in the OSCAR workcell. For example, a manipulator is an object node. Its base pose and any frames-of-interest are location nodes, and its DH parameters and obstacle models are property nodes. The tag of each node is nominal data, and identifies what the node is meant to represent. The type and tag allow efficient searching of the graph. For example, it is relatively straightforward to find all properties of the robot modeled in Figure 2-14, because only nodes connected to the robot that are of type: property and tag: robot need to be searched.

MMDB links

Links capture the relationships between nodes. In the MMDB, they are categorized as an **object-location**, **property**, or **Locomotion**. An object-location associates an object with an area. It is bidirectional so that the model can be queried to return what location an object is in, or to return what objects are in a particular location.

Property links are directed from an object to the property. This allows a query to the model to report an object's properties. The locomotion link is bidirectional and exists between areas that are known to be reachable from each other. Figure 2-13 shows that property nodes can also be attached to locomotion links. This is useful for storing information about the path between locations. A path cost is pictured, but there is no reason why more information could not be provided in this same way.

2.5. LITERATURE REVIEW SUMMARY

This chapter examined the modeling information that must be captured by a world-model in order to support collision detection, motion planning, and grasping. For collision detection, geometric information is required about all potential obstacles. This information can be in many forms, but common algorithms give strong support to polygonal meshes that are easily obtained from CAD models. For motion planning, geometric information is required to compute a free path between manipulator configurations. The algorithms do not need the high resolution provided by CAD models, so simpler geometric primitives may be sufficient and can reduce computation time. Computation time is a significant consideration in motion planning, so it may be beneficial to provide a means of caching pre-computed motion plans. Grasp synthesis typically requires a complete geometric description of the object, and the computation to determine the grasp is expensive. The model can be constructed of geometric primitives, or a sampling of surface points. Methods exist that do not require complete 3D geometry, but they are not currently generalized to a wide variety of grasping end-effectors.

The chapter then examined some previous approaches to world-modeling. Previous work at RRG has produced a hierarchical workcell model tree that has been

tested on the Trauma Pod system. The model is focused almost entirely on collision detection and obstacle avoidance, and does not fully capture the topology of the environment. The Spatial Semantic Hierarchy recognizes the importance of topology, but is not ideally suited to manipulation where interaction with the environment is necessary. Mobile Manipulation Database captures environmental topology, and recognizes the importance of interacting with the environment. However, it does not contain the richness of manipulator modeling that is present in the OSCAR workcell model. An ideal world-model for manipulators would preserve the wealth of data in the OSCAR workcell, but would abstract the details of the model into a framework that preserves environmental topology.

CHAPTER 3 : REVIEW OF SENSOR TECHNOLOGY

The previous chapter examined several functions of a manipulator control system to determine what types of information should be *captured* by a manipulator world-model. This chapter focuses on the front end, examining what types of information can be used to *populate* a world-model. Several types of sensors are surveyed, and the principles of operation are discussed in sufficient detail to identify each sensor's relative merits. In most cases, raw sensor data cannot be fed directly into a world-model. Some amount of parsing and processing must occur in order to glean useful information. The discussion of each sensor type includes a brief treatment of how the data it provides has been used in other work to provide useful information about a robot's environment. The sensors are categorized below according to the type of information they provide.

3.1. RANGE SENSORS

Range sensors provide distance information. If the direction and range from a sensor to some point in the environment can be sensed, it is possible to locate the sensed point in any frame of reference. Range sensors are therefore well-suited to collecting 3-dimensional position data from the environment. There is a large variety of commercially available range sensors in terms of both price and performance. Very inexpensive ultrasonic sensors provide range information in a single direction, with marginal accuracy while state-of-the-art laser scanners can generate 360 degree high-fidelity point clouds at impressive data rates. This section examines several types of range sensors and characterizes their resolution, accuracy, bandwidth, and cost.

3.1.1. Ultrasonic

Ultrasonic range detection is accomplished by emitting a high-frequency sound pulse and listening for the echo. A typical sensor will raise the voltage of an output pin when the pulse is emitted, and will lower the voltage again when the echo is detected. The pulse width of the signal on the output pin is therefore proportional to the distance travelled by the sound pulse. A typical low-cost ultrasonic transducer can provide distance information and 5-10 kHz. Resolution and accuracy of ultrasonic sensors is difficult to characterize generally because they are significantly affected by the size and shape of the target. Figure 3-1 shows an example of the effective beam width of an inexpensive ultrasonic sensor for both a 3.5 inch cylindrical target and a 12 inch square planar target.

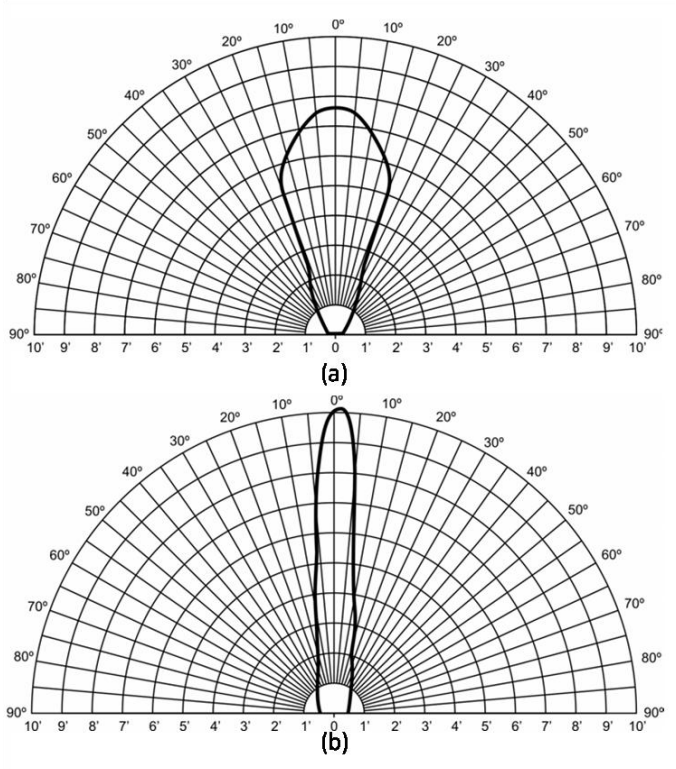


Figure 3-1. Ultrasonic Sensor Beam widths. (a) 3.5" cylindrical target, (b) 12" square planar target [Parallax, 2006]

The test data is much better for the planar target. In both cases, the sensor can measure ranges of up to ten feet with accuracy on the order of several inches. Ultrasonic range finders can be purchased for less than thirty dollars, making them an economical solution where only coarse position resolution is required.

3.1.2. Laser rangefinding

Common laser rangefinders function on the time-of-flight principle. The idea is similar to ultrasonic range detection, except using a pulse of laser light instead of a sound pulse. The use of a laser allows a much more focused interrogation, yielding much better accuracy and resolution as well as allowing measurements to be taken at very long distances.

The sensor detects distance by emitting a pulse of laser light at a specific wavelength, triggering the start of a timer. When the leading edge of the returned light signal reaches the detector, timing is stopped. Assuming a constant speed of light *c*, the distance can be computed according to —. Figure 3-2 diagrams the time-of-flight principle. It also shows the rotating mirror that is often used to emit a fan-like planar array of pulses that allows the sensor to quickly generate several range measurements that define a two-dimensional contour of the target in the two dimensional plane of the mirror's rotation. The spinning mirror permits a wide field of view, with some sensors offering a full 360 degrees.

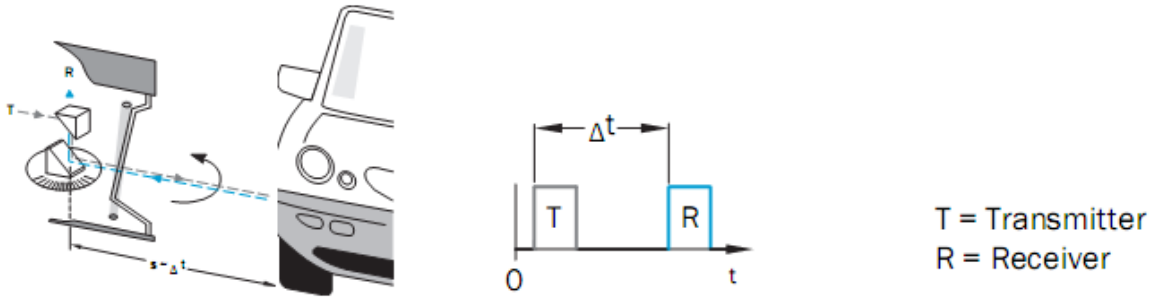


Figure 3-2. Laser Time-of-flight [SICK, 2010]

In practice, these sensors can generate three dimensional data if mounted to a mechanical oscillator that sweeps the sensing plane through some angle.

3.1.3. Infrared time-of-flight cameras

Another type of time-of-flight sensor uses infrared light emitting diodes (LEDs) to emit a light pulse instead of lasers. Each pixel in a Charge Coupled Device (CCD) array measures the time-of-flight of the pulse independently. Because of the use of a CCD pixel array, these sensors are often referred to as time-of-flight cameras. Figure 3-3 shows a SwissRanger infrared time-of-flight camera. The array of infrared LED's that provide the pulsed illumination required for the range measurement can be seen surrounding the camera lens.

Figure 3-3. SwissRanger SR3000 infrared time-of-flight camera

Time-of-flight CCD cameras can generate three-dimensional range data at very fast rates compared to laser sensors without the need for any mechanical components

such as a spinning mirror or vertical oscillator. However, the effective range of measurement is much shorter since the emitted pulse is much less coherent than a laser. Most time-of-flight camera manufacturers recommend their sensors for indoor use only whereas many laser-based sensors can be used outdoors. The field of view is limited to a theoretical maximum of 180 degrees. In practice, the field of view is typically less than 50 degrees. [Mesa Imaging, 2009]

3.1.4. Stereo vision

Stereo vision or stereopsis can measure distances passively by taking multiple images of a scene from different locations. This it most commonly accomplished by taking images with two CCD cameras simultaneously. A point located at the intersection of the respective camera lines-of-sight would appear at the same location on each camera image. Points not at this intersection will appear in different locations in the respective images. This difference is called the image disparity. The cameras in a stereo vision system typically are oriented with their lines of sight parallel, intersecting at infinity. The relationship between disparity *d* and range *r* is given by where b is the baseline distance between camera centers and f is the focal length of the cameras. Figure 3-4 plots this relationship for a Point Grey Bumblebee 2 stereo vision system, pictured in Figure 3-5. The Bumblebee 2 has a baseline of 120 mm. The plot was generated for a model that uses 6 mm focal length cameras. The plot demonstrates a lack of sensitivity at long distances where the magnitude of — is very small. In addition, hypersensitivity is seen at very short distances where the magnitude of — is very large and the measured range much too sensitive to small changes in pixel disparity. A stereo vision system's performance is best at some intermediate range that is most easily controlled by variation of the baseline parameter, b.

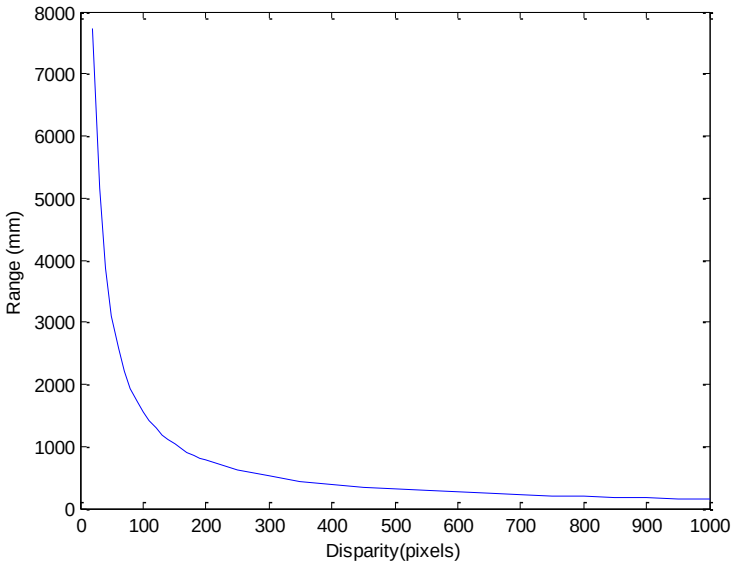


Figure 3-4. Range vs. Disparity

Figure 3-5. Bumblebee 2 Stereo Vision System

The plot demonstrates a lack of sensitivity at long distances where the magnitude of — is very small. In addition, hypersensitivity is seen at very short distances where the magnitude of — is very large and the measured range much too sensitive to small changes in pixel disparity. A stereo vision system's performance is best at some intermediate range that is most easily controlled by variation of the baseline parameter, b.

A perceived disadvantage to using a stereo vision system is that it is computationally much more expensive than other range sensing methods. A block

diagram showing the processing steps required between raw images and range information is presented in Figure 3-6.

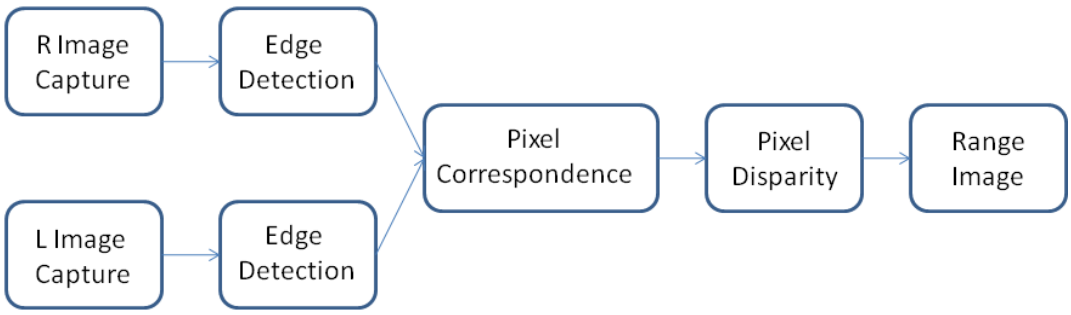


Figure 3-6. Stereo Image Processing

A Bumblebee 2 system operating on a 2.8 GHz Intel Pentium IV processor with 1 GB of RAM can generate range images at a rate of 83 Hz if the resolution of the cameras is reduced to 320 x 240 pixels and the processor devotes all cycles to the computation [Point Grey, 2004]. This speed is certainly sufficient if one can devote an entire processor to generating range images, but can be too expensive if run on the same computer as the robot control system. Most of the computation is in the pixel correspondence analysis, so performance can be improved by further reduction of the effective resolution. The TriClops stereo vision processing SDK that ships with the Bumblebee 2 allows an application developer easily to set the effective resolution.

3.1.5. Range measurement summary

Table 3-1 presents a summary of the range sensing technologies described above.

Table 3-1. Range Sensor Summary

	Field of View	Angular Resoution	Accuracy	Operating	Bandwidth	Cost
Ultrasonic (Parallax Ping)	Directional	> 10°	Widely Variable	0 - 1.8 m	5000 kHz	$30
Laser (SICK LMS-200)	Up to 360°	0.25°	± 20 mm	0 - 80 m	75 Hz	$5,000
Stereo Vision (Point Grey BumbleBee 2)	43°/66°/97°	0.07	± 15 mm	0.5-4.8m	48 Hz	48 Hz
IR Time-of-Flight (SwissRanger SR4000)	43.6°	0.23°	± 15 mm	0.8 - 8.0 m	54 Hz	54 Hz

*Operating at reduced resolution of 320 x 240 pixels

From this table, it is clear that ultrasonic range sensing belongs in a different category than the other three methods presented. It provides only one-dimensional data with poor resolution, accuracy and range. Their fast speed and low price make them suitable for some collision detection applications, but as a source of data for world-modeling, ultrasonic sensors are a poor choice. The laser rangefinder evaluated in Table 3-1 must be placed on an oscillating mount to provide three-dimensional data. The cost of such a mount is not included in the table. The laser sensor is most appropriate for high-speed autonomous vehicle navigation where its long range and wide field of view allow it to scan wide areas at long distances.

For most non-vehicle robotics applications, an IR time-of-flight camera and/or a stereo vision system is most appropriate. The stereo vision system requires that the scene being measured have enough texture that pixels from each image can be correlated to each other. It is also subject to errors caused by shiny surfaces. The reflections from shiny objects appear differently to each camera, which causes pixel correlation errors. The IR time-of-flight camera is not recommended for outdoor use. It suffers from errors relating to multiple reflections. If the emitted pulse reflects off more than one surface before returning, the time-of-flight is overestimated. There currently is no ideal solution

for gathering real-time three-dimensional information for world-modeling. However, several relatively low-cost sensors provide adequate performance under the right conditions. If the environment in which a robot operates can be controlled well enough to maintain sensor performance, a stereo vision or IR time-of-flight camera system can effectively be used.

3.2. MANIPULATOR FORCES

The forces acting both internally and externally in the manipulator can be a useful source of information for world-modeling and general safety. Some commercially available robots allow direct or indirect measurement of the torques in each joint. Industrial six-axis force-torque sensors can also be mounted at the end-effector to provide forces along and torques about the end-effector x, y, and z axes.

3.2.1. Joint torques

Joint torque transducers must be included in the actuators at the time of manufacture in order to have access to direct joint torque data. Access to accurate joint torque information allows torque control of the robot. Equation 3-1 defines the total torque in each actuator as a function of the inertia matrix , the power matrix, , the Jacobian at each link's center of gravity, , and the Jacobian at i^{th} point *e*, .

(3-1)

Where Lg,i is the gravitational force acing on link i and Le is the external force acting at point e. If the torque vector in equation 3-1 is known, then for a robot with a known location in contact with the environment, the force vector acting on the end-effector can be computed. This allows for tactile sensing of obstacles in the environment. Since the location of a point on the -effector can be known with a high degree of accuracy, sensing contact with the environment provides very low-uncertainty position

data. In practice, it is not practical to survey an entire scene in this manner, but this method can provide refinement to an existing model when high accuracy is desired.

Where torque transducers are not installed, it may be possible to estimate the torques by monitoring the current supplied to the actuator. However, this depends upon the success of such a method is strongly dependent upon physical parameters of the actuator such as any gearing, friction, stiction, etc.

3.2.2. End-effector force/torque sensing

A six-axis force/torque sensor at the end-effector also allows tactile sensing. Such a sensor can provide force and torque data at the end-effector with accuracies of ±0.0139 N and 3.53 x 10-4 Nm respectively. [ATI 2010] In addition to the refinement of position data described above, it is possible to use this tactile sensing to determine other important modeling parameters. UT RRG has successfully demonstrated using tactile sensing to determine a door's radius without any *a priori* knowledge of the door. While grasping the door latch, the end-effector's position is perturbed a small amount. The position in then corrected in order to maintain zero lateral force at the end-effector. This process repeats many times, which causes the end-effector to trace the door's swing radius. Forward position kinematics are used to map the swing in Cartesian coordinates. A non-linear circular regression then provides the two-dimensional coordinates of the hinge axis and the radius of the door. From the perspective of world-modeling, capturing such data allows much more efficient opening of the door on subsequent attempts.

3.2.3. Fingertip pressure transducers

Other applications for tactile sensing in world-modeling would be friction and compliance information. This information would be best collected by transducers located at the gripper's fingertips. Waseda University's Twendy One and Willow Garage's PR2

manipulators both use pressure-sensitive pads to provide tactile pressure data with a 0.1 psi resolution A similar system is available as a retrofit for the Barrett Hand BH-8. [Pressure Profile Systems, 2009].

3.3. HUMAN AS SENSOR

An important sensor at any level of autonomy less than full autonomy is the human operator. The human's ability to collect and process sensory information is unmatched by any modern robotic system. A human feedback can be included in the control loop through using a Human-Machine Interface (HMI). It is difficult to characterize a human sensor by its bandwidth, resolution or accuracy but it is adequately characterized as poor. However, a human can survey a scene and parse it into a world-model at near instantaneous speed. Where possible, it is advisable to poll a human operator to provide data that other sensing mechanisms cannot generate with sufficient certainty.

An example of this would be to present the user with an image of the scene and request that he identify or classify objects in the scene. This semantic information can be passed to the world-model or even compared to other data sources to refine the geometric model. This obviates the need for the currently slow, complex, and error prone computer perception algorithms, but limits the autonomy of the system and possibly introduces human error that can be difficult to predict.

3.4. CHAPTER SUMMARY

This chapter examined several types of sensors that can be used to provide real-time data to a world model.Table 3-2 summarizes the types of sensor discussed in this chapter and the data primitives they provide.

Table 3-2. Sensor Technology Summary

	Data Primitive	Uncertainty	Use in world modeling
Range Sensor	3D point or point cloud	±15 mm	Geometry of objects in environment. *
Joint Torque Sensors	Transmitted Torque	±0.10%	-Tactile resolution of geometric uncertainty -Mass Properties of grasped object
End-effector Force/Torque Sensor	Contact forces and torques	±0.0139 N 3.53×10^{-4} Nm	-Tactile resolution of geometric uncertainty -Mass Properties of grasped object
Contact Pressure Sensor	Force normal to sensor	±0.445 N	-Compliance properties of grasped object -Fingertip tactile resolution of geometric uncertainty

The most basic type of information contained in a world-model is geometric position data of objects in the environment. Range measurements transformed into 3D Cartesian points provide this geometric data. Several types of range sensor are available, although no single sensing technology provides an ideal solution. For indoor robotic manipulation in reasonably structured environments, some combination of stereo vision and CCE-based infrared time-of-flight ranging provide adequate data. Outdoors or at long distances, multiple laser rangefinders or a single vertically oscillating laser rangefinder provide accurate data over large areas. With additional processing, the point-clouds generated by these sensors can be parsed to identify specific objects within the environment, although this technology is not yet robust enough for broad industrial use. Given the distance at which the measurements are made, the centimeter-scale accuracy is impressive, but still limiting for precision manipulation.

Tactile sensing can provide a world-model with refined position data and may be feasible for measuring friction and compliance properties that are essential to proper

grasp analysis. This can be accomplished on robots equipped with joint torque transducers or a six-axis force/torque sensor by transformation of the sensed forces and/or torques to the contact point. Tactile sensing at the fingertips can be accomplished by means of pressure-sensitive pads installed on the contact surfaces of grippers. These pads have the advantage of obviating the need for transformation calculations and the associated error propagation.

The next chapter proposes a data framework for a manipulator specific world-model. It proposes a graph-based world-model that takes advantage of the strengths of graph data structures, but retains all of the work conducted previously on OSCAR's world-modeling. Chapter five then explores one implementation for a model and provides a few demonstrations of the graph-based world-model's support for manipulator.

CHAPTER 4 : PROPOSED WORLD-MODELING STRUCTURE

Chapter Two discussed the current state of the art in three areas relevant to manipulator world-modeling: collision detection, motion planning, and grasping. It also examined some current approaches to robotic world-modeling and evaluated their suitability to robotic manipulation. Chapter Three surveyed currently available sensing technologies that may be used to populate a world-model. This chapter identifies the requirements of a manipulator world-model based on the review conducted in Chapter Two. Of the modeling techniques discussed in Chapter Two, the OSCAR workcell model is the best adapted to robotic manipulation. It captures a great deal of information relevant to manipulator control, and is useful over a wide domain of manipulator applications. This chapter proposes a manipulator world-model that preserves the strengths of the OSCAR workcell model, but extends its ability to support manipulator collision detection, motion planning, and grasping.

4.1. WORLD-MODEL REQUIREMENTS

Some performance requirements must be defined to motivate and guide the development of a suitable manipulator world-modeling structure. Based upon the literature survey conducted in Chapter Two, the model presented in this chapter satisfies the following requirements:

- Architecture agnosticism
- Representational efficiency
- Ability to capture topological structure.

4.1.1. Architecture agnosticism

Architecture agnosticism is necessary in order to preserve generality. It is important that the modeling structure be capable of supporting a wide variety of systems as well as changes to existing systems. Therefore the modeling techniques discussed in this chapter aim to be architecture agnostic. This agnosticism is characterized by model extensibility, support for multiple manipulators, and the ability accommodate multiple representations of relevant data.

It is difficult to know *a priori* all of the possible different types of data that one may eventually wish to represent in a world-model. Therefore, a manipulator world-modeling scheme must be able to accommodate new types of information as they become relevant in a given application. A model that meets this requirement is said to be extensible.

It is inefficient for each manipulator working in a shared workcell to maintain its own representation of the world. It is much more efficient to maintain a single model of the workcell. This also assures that each manipulator has access to the same information, avoiding possible conflicts. Thus, a manipulator world-model must be able to accommodate any number of manipulators working in the same space.

Different representations of a single entity may be required in order to support different manipulator control functions. An example would be collision detection and potential-field based collision avoidance. A polygonated mesh is more appropriate for collision detection whereas geometric primitive modeling is more appropriate for collision avoidance. The world-model should not assume in advance what form the data should be stored in to support downstream control functions. In fact, it may have to store the same data in multiple forms simultaneously.

Another implication of architecture agnosticism is that to be truly architecture agnostic there should be no assumed global reference frame. In practice, removal of a global reference frame makes it difficult to represent the positions of objects in a coherent way. The assumption of a global reference frame is only a problem for mobile systems where the "workcell" does not stay in the same global position. This would be the case for manipulators mounted on mobile systems. For a single mobile manipulator, a global frame can be established on the mobile platform, and all positions measured relative to that frame. Therefore, the only case where the assumption of a global reference frame might become a problem is for systems of multiple mobile manipulators. Because the assumption of a global reference frame is reasonable for most robotic workcells, the methods proposed here assume a global reference frame. It is possible that this work can be generalized to remove this assumption with very little modification, but this is left for future work.

4.1.2. Representational efficiency

The next requirement of a manipulator world-model is that it maintains the most efficient representation possible. At first thought, this seems like it may conflict with the ability to maintain multiple forms of the same data. It may be useful to minimize the number of different forms of similar data that the model must store, but this is an application-level concern that must be addressed once the particulars of the system architecture are known. Here representational efficiency requires rather that a single entity in the environment be represented as a single entity in the model. That entity may have multiple representations available to the robot control system, but the entity itself is only represented once in the model. As an example, a manipulator link in the OSCAR workcell model has both a triangulated mesh representation for collision detection and a

geometric primitive model for motion planning. This is not a redundant representation. However, identical frames-of-interest may exist as independent children of different manipulator entities, or may exist entirely separate from a manipulator. In this case, one physical entity may exist as several different entities or nodes in the model. This is inefficient and can cause confusion.

4.1.3. Topological structure

As suggested in Chapter Two, one important feature of a manipulator world-model is the ability to capture topological relationships between entities in the environment. In the OSCAR workcell, the links that define the hierarchy in the model carry no information. All information is contained in the nodes. However, it makes sense to separate entities in the environment from their relationships to the environment. To capture this information, the links in a model should have meaning. A link between two physical objects could represent their adjacency in space and be weighted by a path cost. A link between a manipulator and an object in the environment could indicate that the object is in the manipulator's workspace. It could also conceivably contain the grasp parameters required for the manipulator to grasp the object. The ability of the model to represent not only objects in the environment, but also how they are related in the context of robotic manipulation is very powerful.

4.2. WORLD-MODEL STRUCTURE

To meet the requirements outlined above, the strict hierarchy of the OSCAR workcell model's tree data structure needs to be more general. A graph-based structure allows the flexibility to assure architecture agnosticism, prevent representational redundancies, and to capture important information about relationships among entities in the model. This section explores graphs as data structures in more detail than Chapter

Two. It then proposes a specific architecture for a graph-based world-model for robotic manipulation and explains its utility for collision detection, motion planning, and grasping. The practical consideration of how such a model is accessed to either retrieve data or populate the model is discussed briefly, as is the importance of maintaining measures of uncertainty.

4.2.1. Graph structures

In mathematics, a graph is a symbolic abstraction that consists of a set of vertices, V, and a set of edges, E. Each edge in E is described by a pair of vertices (u,v) in V. A graph is itself a pair, (V,E) composed of all vertices in V, related to each other by the edges in E. Vertices are also commonly referred to as *nodes* or *entities*, and edges may also be referred to as *links* or *relations*. This description is somewhat vague, but this vagueness means that many physical situations can be abstracted into graphs. Social networks are often described by graphs where individuals are represented as nodes and their acquaintanceship with one another is captured by links. A road map is a graph with cities represented as nodes and roads represented as links. The human cognitive map has been described as a graph with objects as nodes and relations among them as links. Figure 4-1 presents an example of a graph $G=(V,E)$ where:

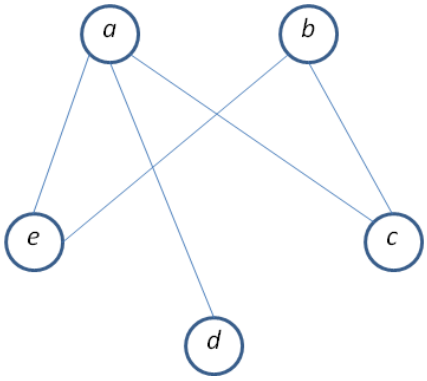


Figure 4-1. Undirected graph

The graph pictured in Figure 4-1is an example of an *undirected* graph. In this type of graph, the links have no direction, the edges are unordered pairs, and the vertices are considered to be linked bidirectionally. In the case of a road map, an undirected graph would assume that each road permits two-way traffic. An alternative would be a map on which some roads are one-way only. This situation would require a *directed* graph. Figure 4-2 is similar to Figure 4-1, but with directed links. The graph depicted in Figure 4-2 is G=(*V,E*) for:

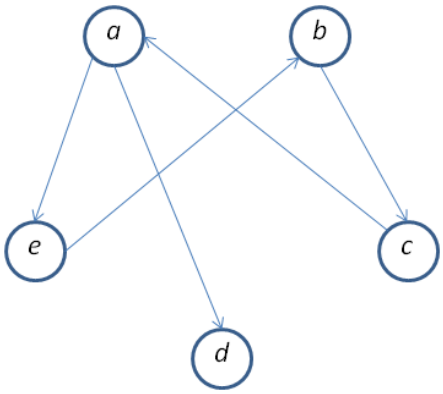


Figure 4-2. Directed Graph

The directed graph is a more general representation than the undirected graph, and it allows representation of additional information about the relationships between entities. The decision to use directed or undirected graphs involves the nature of the physical problem as well as a tradeoff between efficiency of representation and efficiency of graph traversal. If most relations among entities are bidirectional, the representation is simplified by use of an undirected graph since each bidirectional relation in a directed graph requires two links. However, an algorithm traversing an undirected graph must consider each link both ways. If directed links sufficiently model the problem, they make graph traversal more efficient by not requiring an algorithm to backtrack unnecessarily.

To avoid confusion, several other terms should be clearly defined when discussing graphs. For an edge (u,v), v is said to be *adjacent* to u. In Figure 4-1, a is adjacent to e and e is adjacent to a since for an undirected graph (a,e) and (e,a) are equivalent. However, in Figure 4-2, e is adjacent to a but a is not adjacent to e since (e,a) is not a member of E. The relations between edges and vertices are described differently for directed and undirected graphs. For undirected graphs, an edge (u,v) is said to be *incident* on both u and v. For directed graphs, (u,v) is an *out-edge* of u and an *in-edge* of v. For either a directed or undirected graph, v is said to be the target vertex of (u,v) and u is said to be the *source vertex* of (u,v). It is important to note that for an edge in an undirected graph, both vertices are source vertices and target vertices. A *path* is a sequence of edges such that the target vertex for the previous edge is the source vertex for the next edge. If a path exists that starts at vertex v and ends at vertex u, then u is said to be *reachable* from v.

Graph traversal

Consider a tree structure such as the one used in the OSCAR workcell model. Because of the single-parent restriction on tree structures, searching nodes in a tree structure is simple. In order to return a list of all manipulators, an algorithm needs only to start at the top of the hierarchy and search all daughters. In a graph structure, there is no assumed parent-daughter relation. This means that multiple paths from the start node to a particular node may exist, and care must be exercised not to count any node twice. Furthermore, there are multiple ways of moving through the graph and examining the nodes. The process of visiting all nodes in a graph for data access or modification is called graph *traversal*. A graph traversal algorithm locates all nodes that are accessible from a given start node. There are two graph traversal methods: Breadth-first search (BFS) and Depth-first search (DFS).

The BFS will locate all nodes accessible from the start node, discovering the closest nodes first. If a traversal algorithm is thought of as an explorer who maps out a graph, the BFS algorithm walks each edge to see to what node it leads. It then returns to the original node and explores the next edge. Once it has visited the nodes at the end of each edge, it will walk out to the next node and repeat the process, never exploring a level deeper until all closer nodes have been visited. The result is a tree structure where only the shortest path between nodes is retained. Figure 4-3 shows a graph that has been traversed by BFS. The bold edges in the graph represent the BFS tree. Edges not in the BFS tree are grey.

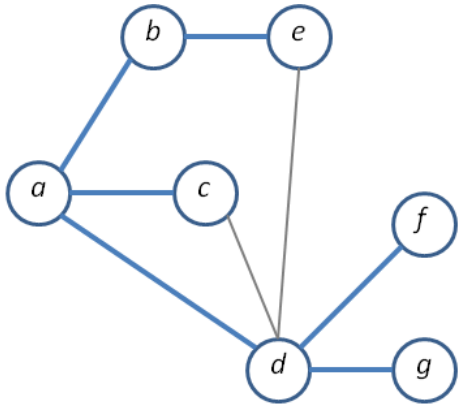


Figure 4-3. Breadth-first search tree

Because the traversal results in a tree structure, some of the same language from tree structures appears when discussing traversal algorithms. When a BFS algorithm discovers a node *v*, the edge (*u*,*v*) is called a tree-edge, and node *u* is called the parent of *v*. It is important to note that in traversing a graph this way, not all edge information contained in the graph is accessed. In Figure 4-3, the connectivity between nodes *c* and *d*, as well as between nodes *e* and *d* is not considered by the BFS algorithm.

The hypothetical graph-explorer performing a DFS search will always choose to explore an edge leading deeper into the graph. When it reaches a node from which no more edges can be explored, it will return to the previous node and explore another out-edge from that node until it reaches another dead-end. When all edges have been explored, the algorithm will start the process again at an unvisited node. This assures that any nodes not connected to the start node are also visited. The result is a set of trees referred to as a *forest*. Figure 4-4 shows the same graph as Figure 4-3, but traversed with a DFS algorithm.

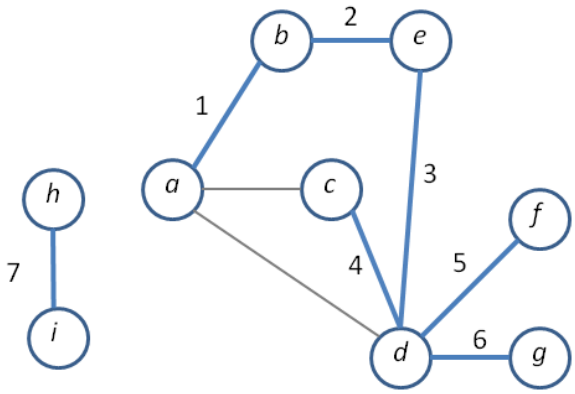


Figure 4-4. Depth-first search forest

The edges in Figure 4-4 are labeled with the numerical order in which the edge was explored. The trees resulting from DFS are much different from those constructed by BFS. Like BFS, DFS does not consider some of the graph's connectivity.

4.2.2. Graph-based world-model

The previous section covered some basics about graphs as data structures, and examined how traversal algorithms visit nodes in a graph. This section proposes the specifics of how to build a graph-based world-model for robotic manipulation. In this model, entities in the world are represented as nodes in the graph and the relationships of entities to other entities are represented as links. Each node and link in the model has a unique numerical ID number that allows any node or link to be accessed directly. The proposed graph-based world-model allows three types of nodes and three types of links.

Nodes

Similar to the Mobile Manipulation Data Base (MMDB) proposed by Philippsen, Nejati, and Sentis [2009], the model presented here allows several types of nodes. The model proposed here allows types of **manipulator**, **location**, and **object**. In addition to its type, every node is identified with a tag. The tag is nominal data and describes the content of the node, enabling semantic classification of nodes. For example, an object

node's tag might identify it as a door. This would allow a search of the model to return a list of all doors in the model. Unlike the MMDB, there is no provision for a property node. An object or manipulator's properties are contained within the object or manipulator node. Since properties are always specific to a single physical entity in the model, there is no need to represent them as separate nodes in the graph.

Manipulator nodes contain all the relevant information about the manipulator. Like the OSCAR workcell model, the manipulator node must contain the robot's DH parameters, joint position, velocity, and acceleration limits, configurations-of-interest, and obstacle representations. Unlike the OSCAR workcell, this model prohibits inclusion of frames-of-interest as a manipulator property. The properties contained in a node should be internal to the node. A frame of interest is best modeled a separate location node. Table 4-1 shows the data that a manipulator node should contain along with the type of data primitive used to store it.

Table 4-1. Manipulator node data

Data	Primitive
DH Parameters	*a, d,* α, and θ values for each joint axis
Joint Limits	Positive and negative position, velocity, and acceleration limits.
Geometric Models	Solid geometric primitives or convex polytopes
Configurations of Interest	Arrays of joint values that define the configuration

A location node need only contain a 6 dimensional transformation from the global reference frame. This locates a specific position and orientation in space. Modeled objects should be linked to a location node in order to establish their position relative to the global frame. This will be discussed and clarified further in the discussion of links. In

addition to providing a means to locate modeled objects in space, a location node would also be used to define a location of interest where objects are often placed or from which they are retrieved. Table 4-2 shows the data that a location node should contain along with the type of data primitive used to store it.

Table 4-2. Location node data

Data	Primitive
6-D Global transform	4 x 4 transformation matrix

The object node is used to model any physical object in the environment that is not a manipulator. Like the manipulator, all internal properties of the object should be modeled within the node. Such properties include its collision detection and collision avoidance representations, and any significant inertial properties that may be of use to the control system. The object node's location should be established by linking to a location node rather than modeling its location within the node. Table 4-3 shows the data that an object node should contain along with the type of data primitive used to store it.

Table 4-3. Object node data

Data	Primitive
Geometric Models	Solid geometric primitives or convex polytopes
Mass/Inertia Information	Total mass and center of mass in local frame *or* Inertia matrix

Links

The proposed world-model allows three types of link as well. The allowed types are **object-location**, **grasp**, and **reachability**. In the proposed world-model, the links represent meaningful relationships between entities in the model, and therefore can contain significant amounts of data. Most manipulator control functions will access specific nodes in the graph rather than traversing the entire graph, so efficient traversal is

not a major concern. All three link types are bidirectional which is beneficial because it permits reciprocity between nodes. For example, the control system may wish to know at what location an object is, but it may also wish to know what objects exist at a given location. Similarly, it may be useful to return a list of all objects a manipulator can reach, or a list of all manipulators that can reach a given object. The proposed graph is undirected, and the links are therefore bidirectional. Because most links would be bidirectional and efficient graph traversal is relatively unimportant, the proposed graph is undirected, and the links are bidirectional. The following paragraphs describe each type of link.

The **object-location link** must be between an object or manipulator node and a location node. It contains the transformation between the object's frame and the location node's frame. As mentioned above, all object nodes should be linked by to a location node by an object-location link at all times. It may seem easier to model an object's position as an internal property. However, modeling the location externally allows the object to be placed at a location of interest by linking to that location's node, avoiding a redundant or conflicting positional representation. Furthermore, it provides a mechanism for tracking uncertainty in an object's position. The positional uncertainty is contained in the object-location link. The importance of tracking uncertainty is discussed later. Table 4-4 shows the data that an object-location link should contain along with the type of data primitive used to store it.

Table 4-4. Object-location link data

Data	Primitive
Positional Uncertainty	tranlational and angular uncertainty of position
6-D global transform	4 x 4 tranformation matrix

The **grasp link** represents the relationship between a manipulator's gripper and an object that it may grasp. It must contain a Boolean active grasp flag that is used to indicate whether the item is currently being gripped by the manipulator. It also contains a relative position and orientation of the gripper to the object and a set of gripper joint parameters that defines an appropriate grasping configuration. It may also contain compliance or friction information between the gripper and the object that could be used for online grasp synthesis. Bicchi and Kumar [2000] state that no suitable analytical compliance model exists that is suitable for grasp synthesis. Therefore, the details of storing a compliance model are left open here. Table 4-5 shows the data that a grasp link should contain along with the type of data primitive used to store it.

Table 4-5. Grasp link data

Data	Primitive
Active grasp flag	Boolean to indicate wheter object is currently grasped.
Grasp HPO	4 x 4 transformatin matrix
Gripper Joint Values	1 dimensional array that specifies the gripper configuration
Friction information	Static friction coefficients for all contact points *or* Friction cone dihedral angles.
Compliance Information	No suitable model available

The **reachability link** must be between a manipulator node and a location node. The link indicates that the location can be reached by the manipulator. The reachability link does not necessarily need to contain any data. However, it may be useful to store a manipulator joint configuration and some measure of the motion potential at the end-effector in that configuration. This would potentially benefit motion and task planning algorithms. Table 4-6 shows the data that a reachability link should contain along with the type of data primitive used to store it.

Table 4-6. Reachability link data

Data	Primitive
Joint Configuration	An array of joint values that places the end effector at the location node
Motion potential	Measure of manipulability *and/or* Joint Range availability

Collision detection support

In order to support collision detection, the graph-based world-model needs to maintain a geometric model of each manipulator and object in the environment. The OSCAR Workcell model does this well with its XML implementation. To preserve this functionality in a graph-based model, the OSCAR hierarchy can be used directly. Because the tree structure is a specific type of graph structure, any tree is easily represented as a graph. For collision detection, it is necessary to traverse the graph to search for obstacles. This can be done with either a depth-first or breadth-first search, since either one will visit all of the nodes in the graph, resulting in a complete obstacle model. To assure inter-operability with existing OSCAR software, it is only necessary to provide some translation software that can parse an XML file into the graph, and can produce an XML workcell file that OSCAR can use for collision detection.

Motion planning support

In addition to collision detection, the OSCAR workcell model also supports motion planning by allowing multiple geometric representations of objects. Since the graph-based world-model is compatible with the OSCAR workcell model, the Cylisphere representations used for potential-field motion planning can be translated into the graph-based world-model.

In addition to supporting the OSCAR motion planning functions, the modeling scheme here introduces the idea of modeling reachability and dexterity. This information can be used for task and motion planning by allowing a planning algorithm to place objects in locations where some metric of motion capability is optimized. This information is particularly useful in systems with multiple manipulators since it provides criteria for selecting the best manipulator to complete a given task. This allows a planning algorithm to make informed choices determining which manipulator is most appropriate for a given task. This is more difficult under the strict hierarchy of the OSCAR workcell model since a frame of interest is modeled as a property of a single manipulator.

Grasping support

The graph-based approach to modeling is particularly useful in supporting grasping. In a hierarchical model, every node belongs to exactly one parent node. Grasp parameters cannot be categorized as a property of either the manipulator or the object. This makes it very difficult to capture grasp information that by nature represents a relationship between two entities: the gripper and the object being grasped. The graph-based model allows the grasp to be modeled as the link between the manipulator and object nodes. Online grasp synthesis is currently not fast or robust enough for most applications. However, it is an active research area, and the proposed world-model is designed to support it. For a grasp synthesis algorithm to evaluate grasp quality, it must have some information about friction and compliance in order to assure force closure. Without this information, it must try to achieve form closure, which is a much more restrictive case. Compliance and friction are dependent upon both the gripper material

and the object material and therefore cannot be modeled hierarchically, but are easily added to a grasp link in the graph-based model.

Real-time update

To support update of the model as new information becomes available, fast access is required to individual nodes and links in the model. This is facilitated by the use of numerical ID, type, and tag values associated with each node and link. The numerical ID provides random access. If the numerical ID of the node or link to be updated is known, it can be accessed directly without the need to traverse the graph. If the numerical ID is not known, the graph can be efficiently searched by use of the type and tag values. For example, suppose a canister has been moved in the environment. If the robot is responsible for moving the canister, then its numerical ID is almost certainly known. The adjacent location node can easily be found by looking for an object-location link and following it to the location node. The location node and the object-location link can then be accessed and updated.

Suppose, however, that without any action on the part of the control system, a canister is detected where none existed previously. The control system may wish to check the world-model against the new sensor data to see if a canister has gone missing from a known location, indicating that a canister has been moved. The graph can be traversed and the locations of all nodes of type **object** and tag **canister** returned. This would allow a comparison of their modeled locations with sensor data to identify which canister was moved between sensor measurements.

Uncertainty management

One important feature of the proposed modeling structure is the tracking of positional uncertainty. This is an important consideration, particularly if sensor data is

used to maintain the positions of objects in the environment. Since measurement errors are propagated through whatever algorithms are used to construct position and geometry, it is necessary to make sure that the manipulator and end-effector can tolerate the amount of uncertainty in the position measurement. As an example, assume the jaws of a gripper open to a maximum 5.0 cm, then the control system should not attempt to grasp a 4.5 cm wide object with a positional uncertainty of ±0.75 cm since an accidental collision is likely. If the manipulator is equipped with some sort of force/torque sensing, the uncertainty measure may also be used to suggest haptic probing to resolve high uncertainty. The manner in which the uncertainty is calculated is dependent upon the specific sensors and processing algorithms used. For a modeling technique to function in unstructured environments, uncertainty information must be retained in the model.

4.3. EXAMPLE APPLICATION

In order to illustrate how the graph-based world-model might be used in an actual system, a simple example is presented here. A manipulator installation has been proposed in a neutron radiography facility at the Nuclear Engineering Teaching Laboratory. Neutron radiography is an imaging technique analogous to x-ray imaging that can be used to image materials that have low photon cross sections but high neutron cross sections. The gamma radiation field in the beam port during operation is on the order of kilorads per hour. This means that the reactor must be shut down and the facility allowed to cool before a human can enter to change out or reposition samples. Installation of a robot could dramatically speed up sample throughput and reduce the risk of accidental exposure.

During the image exposure, the manipulator is stowed behind a radiation shield to protect its electronic components from radiation damage. Between exposures, the robot is

deployed either to re-position samples or to move samples to and from the sample storage area. Both of these scenarios are presented in Figure 4-5.

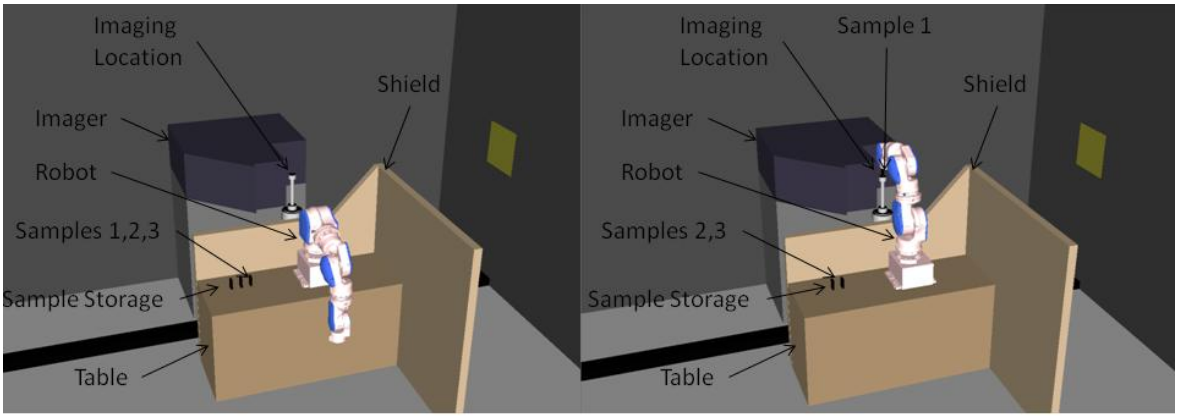
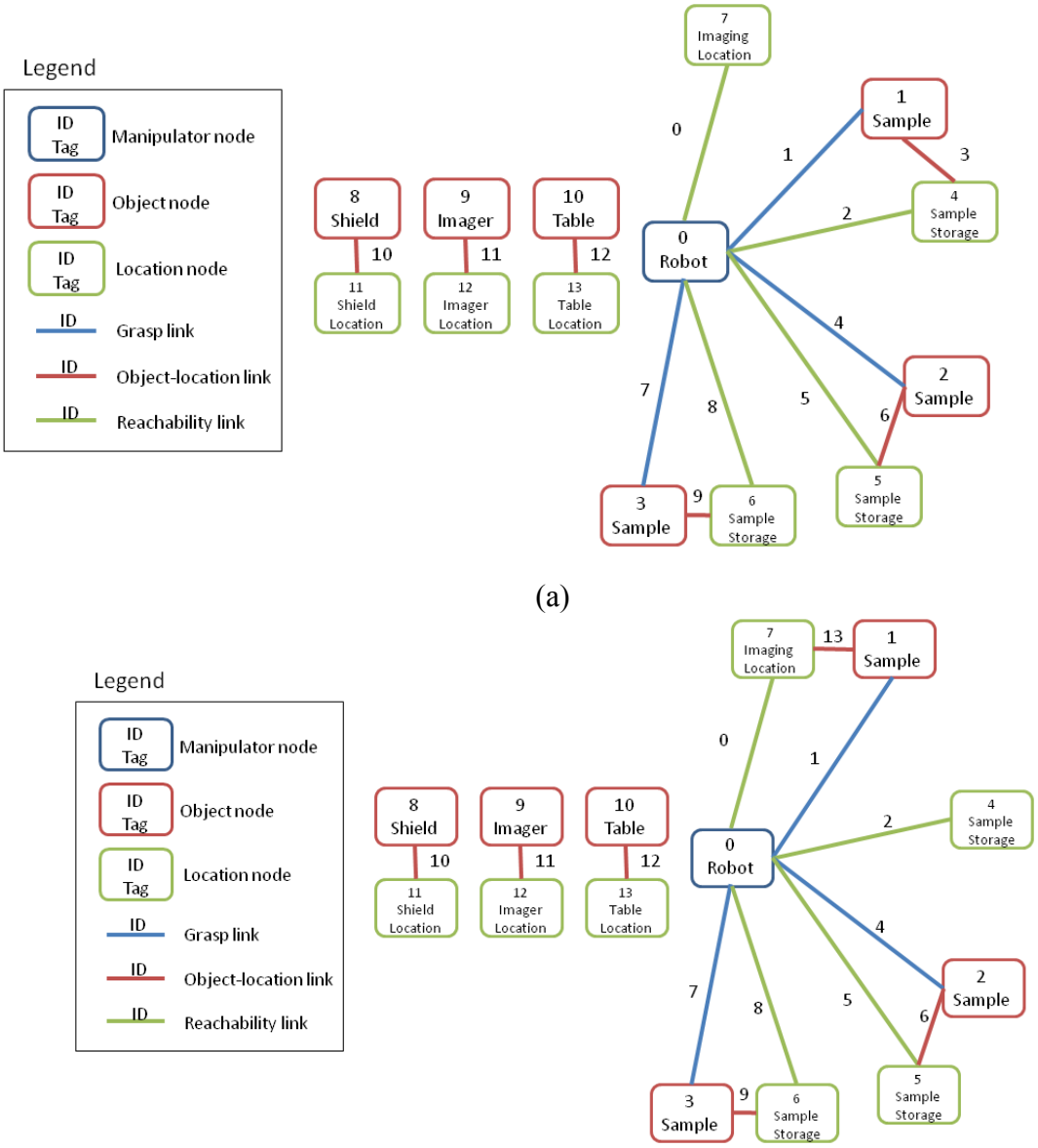


Figure 4-5. Manipulator installation in NETL neutron radiography facility. (a) All samples stowed. (b) One sample in imaging location.

In Figure 4-5, everything that would appear in the world-model is labeled. The imaging location and sample storage locations are considered locations of interest. The shield, imager, and table are modeled for collision detection and motion planning purposes. The samples themselves are modeled both for collision detection and because they are objects that handled by the manipulator.

Figure 4-6 shows the graph-based models of this workcell corresponding to the configurations shown in Figure 4-5. In Figure 4-6 (a), Sample 1 is linked to its sample storage location. Internally, the active grasp flag in link 1 would be set to false. In Figure 4-6 (b), Sample 1 becomes linked to the imaging location, and the active grasp flag in link 1 would be set to true since the sample is being handled by the manipulator.



(a)

(b)

Figure 4-6. Graph Representations of the neutron radiography workcell. (a) Samples stowed (b) One sample in imaging location

4.4. CHAPTER SUMMARY

This chapter presented a brief background on graphs as data structures. It then proposed a graph-based world-modeling structure that consists of an undirected graph

containing manipulator, object, and location nodes connected by object-location, grasp, and reachability links. Table 4-7 lists each type of link and node and shows what information is contained in each.

Table 4-7. Graph-based world-model summary

Manipulator	DH Parameters Joint Limits Geometric Model(s) Configurations of Interest	Object-Location link establishes base pose. Reachability Links identify frames of interest. Grasp links contain grasp information.
Object	Geometric Model(s) Mass/Inertia Info.	Object-Location link establishes pose. Reachability Links not valid. Grasp links contain grasp information.
Location	6-D global tranform	Object-Location link identifies object at the location. Reachability link identifies available manipulator. Grasp links not valid.

(a) Node Types

Link Type	Internal Information	Notes
Object-Location	Positional Uncertainty 6-D transformation	Links an object or manipulator node to a location node.
Grasp	Active grasp flag Grasp HPO Gripper joint values Compliance info. Friction info	Links a manipulator node to an object node.
Reachability	Joint Configuration Motion potential	Links a manipulator node to a location node.

(b) Link Types

The internal information listed in Table 4-7 is meant to demonstrate how a graph-based model supports collision detection, grasping, and motion planning. However, the proposed modeling scheme is meant to be extensible, and there is no reason why additional information should not be included should a particular application require it.

The strength of the graph-based approach compared to other approaches is in the ability to capture important relationships between entities. This is possible because nodes can be linked without the restrictions of a tree structure and because the links themselves contain information regarding the relationship between the nodes they link. This allows a more intuitive representation of the world by separating physical entities in the world from their relationships to the world. This strength is particularly apparent in its ability to model grasp information. A grasp is essentially a set of parameters that define a relationship between a gripper and an object. A graph provides an easy and intuitive structure to capture this information. The use of a graph also allows use of well-known graph traversal algorithms to search for and access data contained in the model. Data search and access is further facilitated by the incorporation of a nominal tag that provides a searchable semantic representation of each node.

In addition to its graph-based nature, another important feature of the proposed model is that it captures a great deal of non-geometric data. Where most world-modeling schemes are primarily concerned with geometric representation of the world, the proposed model more closely approximates the human cognitive map, which concerns itself more with the overall structure of the world, of which geometry is an important part.

The next chapter will present an example implementation of the proposed model that demonstrates its compatibility with OSCAR's workcell model, and its support for motion planning and grasping.

CHAPTER 5 : IMPLEMENTATION AND DEMONSTRATIONS

The previous chapter proposed a world-modeling scheme that abstracts the environment into a graph. This method models entities in the environment as nodes in a graph and their relationships as links in the graph. It suggested three types of nodes and three types of links, and outlined the data structures included in each link and node type. This chapter describes a software implementation of the model presented in Chapter 4, along with three examples that demonstrate how the model supports collision detection, motion planning and grasping.

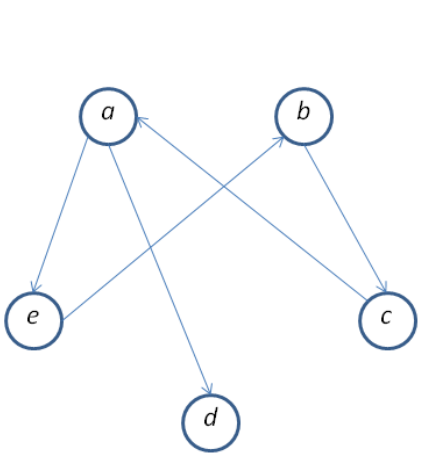
5.1. IMPLEMENTATION

The implementation presented here makes use of the OSCAR manipulator control libraries and the Boost Graph Library (BGL). [Siek, 2002] The OSCAR libraries provide convenient representations of the types of data that must be stored in the model and assures inter-operability with applications developed previously within the RRG. The BGL is part of the Boost C++ libraries that are distributed free of charge for both commercial and non-commercial use. The BGL provides peer-reviewed, robust, and generic graph data structures similar in function to the C++ Standard Template Library (STL) provides robust and generic data containers such as the `vector` and `map` containers. [Siek, 2002] A brief description of the BGL follows to aid understanding of the implementation presented below. In this chapter, C++ classes, objects, and functions will appear in the `Courier` font to enhance clarity.

5.1.1. The Boost Graph Library

The BGL provides two classes that model graphs in the manner presented in Chapter 4: the `adjacency_list` and the `adjacency_matrix`. Each of these

classes stores the graph differently in memory, and therefore the efficiency of some operations is different between the two. Figure 5-1 shows the `adjacency_list` and adjacency_`matrix` representations of the graph originally pictured in Figure 4-2.



a	e	d
b	c	
c	a	
d		
e	b	

Adjacency list representation

	a	b	c	d	e
a	0	0	0	1	1
b	0	0	1	0	0
c	1	0	0	0	0
d	0	0	0	0	0
e	0	1	0	0	0

Adjacency matrix representation

Figure 5-1. Adjacency representations

Each representation captures the same data, but for graphs containing few edges relative to the number of vertices, an adjacency matrix contains mostly zeroes. This uses $O(V^2)$ memory to store very little adjacency data. In addition, traversal of all out-edges in an adjacency matrix requires $O(V^2)$ time versus $O(V+E)$ time with an adjacency list. In the context of the modeling techniques presented here, the graphs are small enough to make the choice of representation arbitrary. However, the developers of the BGL recommend use of the `adjacency_list` class unless the number of edges in the graph is $O(V^2)$, so that is what has been done here.

Descriptors and iterators

To access vertices and edges in a graph, the BGL provides *descriptors* and *iterators*. Functions that add or access vertices typically return a `vertex_descriptor` that provides a unique "handle" that describes a vertex.

Similarly, each edge in the graph has an associated `edge_descriptor`. Iterators are incrementable pointers that can be used for sequential access to edges or vertices. For example, if the application programmer wishes to access each out-edge of a vertex, an `edge_iterator` is used. De-referencing an edge or vertex iterator provides an edge or vertex descriptor. Descriptors and iterators provide low-level access to the vertices and edges in the graph and are used in the higher-level traversal functions provided by the BGL.

Traversal and visitors

The BGL provides functions for both the depth-first search and breadth-first search. Simply travelling the graph is not of much use to the application programmer. Typically, the programmer wants the traversal function to *do* something at each vertex it visits. To this end, the BGL provides a visitor class concept.

Visitor classes can be written that are derived from a default visitor class. Functions within objects instantiated from these classes can be passed as arguments to the `breadth_first_search` or `depth_first_search` functions. The traversal then performs these functions as it visits each vertex.

In addition to the traversal algorithms, BGL provides functions to access all vertices or all edges in a graph. These functions provide more efficient access should an application programmer want to access all edges or vertices and the order they are visited is unimportant.

5.1.2. Software structure

The previous section described the BGL's representation of a graph, and the mechanisms it provides for accessing vertices and edges. This section describes how this framework is extended and customized to represent the modeling techniques proposed in

Chapter 4. This graph-based world-model implementation effort has resulted in a new `WorldModel` class. In this class, the three types of nodes proposed in Chapter 4 are represented as vertices in a BGL `adjacency_list`. The links of the world-model are represented as BGL edges.

Each type of node and each type of link has its own associated class that contains the information shown in Table 4-1, and provides functions for accessing and modifying that data. The node classes are called `maniplatorNode`, `objectNode`, and `locationNode`. The link classes are called `graspLink`, `locatorLink`, and `reachLink`. The `WorldModel` class contains three STL map structures that map BGL `vertex_descriptors` to `manipulatorNodes`, `objectNodes`, and `locationNodes` respectively. Similarly, the `WorldModel` class contains three additional STL maps that map BGL `edge_descriptors` to `graspLinks`, `locatorLinks`, and `reachLinks` respectively. These six maps are called *property maps*. They allow the properties of a given node or link to be accessed with their associated vertex or edge descriptors. This structure is pictured in Figure 5-2. The `nodeTypeMap` and the `linkTypeMap` allow lookup of a node or link type by its vertex or edge descriptor.

Typically when searching the model, an algorithm will be looking only for a particular type of node or link. However, the BGL traversal and access functions will return all adjacent vertices or incident edges indiscriminately. The type maps allow returned vertices to be checked by the algorithm and skipped if not of the correct type. For example, when looking up an object's location, an algorithm will examine adjacent vertices to find a location node. Since the object node may be adjacent to any number of nodes that are not location nodes, the search algorithm checks each adjacent node against

the `nodeTypeMap` until it finds a `locationNode`. Figure 5-2 shows how the nodes in the map are organized to facilitate this type of search. The map structures for the links are organized identically.

Node Type Map	
descriptor	type
vertex_descriptor 0	manipulator
vertex_descriptor 1	object
vertex_descriptor 2	object
vertex_descriptor 3	location

Manipulator Map	
descriptor	Node
vertex_descriptor 0	manipulatorNode 0

Object Map	
descriptor	Node
vertex_descriptor 1	objectNode 0
vertex_descriptor 2	objectNode 1

Location Map	
descriptor	Node
vertex_descriptor 3	locationNode 0

Figure 5-2. World-model map structure

Once a node or link has been located by its type, it can be located in the appropriate type-specific map. The six type-specific maps provide access to the graph's node and link objects, indexed by their BGL vertex and edge descriptors. In the example above, once the location node adjacent to the object is found, the six-dimensional global transformation associated with the object's location can be accessed by looking up the `locationNode` object in the location map.

5.1.3. Class descriptions

The implementation presented in this chapter makes heavy use of the OSCAR libraries since they provide convenient representations of much of the data that must be contained in the library. This section describes each of the node and link classes to demonstrate how the information from Table 4-1 is stored and accessed in the demonstrations described in this chapter. However, the BGL implementation could

conceivably be used with any user-defined node and link classes with very little modification.

Manipulator node class

The manipulator node class contains all of the data that the OSCAR workcell offers with the exception of frames-of-interest. The data contained in each node class is summarized in Table 5-1. The manipulator node contains the Denavit-Hartenburg (DH) parameters to define the manipulator's kinematic configuration. The DH parameters are stored in a two-dimensional (DOF×4) vector array. The joint limits are stored similarly in a DOF×2 array to accommodate the positive and negative position limits of each joint. Collision detection and obstacle avoidance are supported with the inclusion of an obstacle model that allows an arbitrary number of cylispheres to be attached to each link of the manipulator. Using geometric primitives requires that the node class contain seven parameters for each cylisphere: the radius, and endpoints. For collision detection between polytopes, a string can be stored that associates a stereolithography file with each link. The transformation between the first reference frame of the manipulator and the global reference frame is stored as an `OSCAR::Xform` object, as is the orientation of the end-effector tool to the final frame of the manipulator. Configurations-of-interest are stored as a vector of `OSCAR::JointVector` objects, and their names are stored in separate vector of `std::strings`.

Object node class

Objects in the current implementation contain only collision model data. The mass and inertia data suggested in Table 4-1 are easily added should they be required in a future application. For collision and obstacle avoidance, the object node represents an object as a set of extruded-sphere rectangles and cylispheres. For cylispheres, the radius

and the endpoints are stored. For extruded-sphere rectangles, ten parameters are required: the radius and three of the rectangle's corner locations.

Location node class

As suggested in Table 4-1, the only necessary information is the 6-D global transformation that defines the location. This is stored as an `OSCAR::Xform`, a standard 4×4 spatial transformation object.

Grasp link class

The grasp link contains a Boolean to indicate whether the grasp is active or not. A true value indicates that the manipulator is currently holding the object to which the grasp it is connected. Recall that a grasp consists of a hand position and orientation (HPO), and the joint values for each gripper joint. In the grasp link class, the HPO is stored as an `OSCAR::Xform` and the gripper joints are stored in an `OSCAR::JointVector` with a dimension equal to the number of controllable degrees-of-freedom of the gripper.

Locator link class

The locator link class contains an `OSCAR::Xform` that describes the relative position of an object's reference frame to the location node to which the locator link connects it. It also contains a double-precision value to describe the positional uncertainty associated with the relative position.

Reachability link class

The reachability link contains an `OSCAR::JointVector` that describes a robot configuration that places the end-effector at a given location. A double-precision value is stored that provides some measure of the robot's motion potential in that configuration such as a measure of transmissibility or measure of manipulability.

World-model class

The world-model class allows the application programmer to easily create and use a graph-based world-model. The essential components of the `WorldModel` class are the `boost::adjacency_list` that captures the graph structure, and the 8 `std::maps` mentioned in section 5.1.2:

- Node type map
- Link type map
- Manipulator map
- Object map
- Location map
- Grasp map
- Locator map
- Reachability map

The class contains functions for adding and deleting nodes and links from the graph. Access to the nodes and links is provided by standard `get/set` functions that accept a `boost::vertex_descriptor` or `boost::edge_descriptor` as an argument and return a reference to the node or link object to which the descriptor maps. A `tagSearch` function allows the application programmer to retrieve a node's `boost::vertex_descriptor` handle if its type and tag are known. Node addition, deletion, and access are completely architecture agnostic. Except for the node's type and tag, these functions assume nothing about the node's implementation.

The `WorldModel` class also contains some functions that will only work on a system built in OSCAR. `SetLocation` takes an `OSCAR::Xform` and a `boost::vertex_descriptor` as arguments. The function finds the location node to which the object is connected and changes its stored `OSCAR::Xform` to the one passed

as an argument. `GetClosestLocation` accepts xyz coordinates and a `boost::vertex_descriptor` as arguments. The function replaces the passed `vertex_descriptor` with a handle to the location object nearest xyz input location. Similarly, `getClosestObject` provides a handle to the object node nearest the input xyz location.

To assure operability with current OSCAR conventions, the `WorldModel` class includes an `exportWorkcell` function. This function parses the graph-based world-model and exports a properly formatted XML file that can be read directly into an OSCAR application. Because the graph-based world-model does not permit frames-of-interest to be stored as manipulator properties, the XML export will move any frames-of-interest to global scope which may not be what the OSCAR application expects. If the OSCAR application attempts to load a manipulator's frames-of-interest from an exported workcell, it will encounter an error. To resolve this, the `WorldModel` class includes an `exportFramesOfInterest` function that emulates OSCAR's `ManipulatorData::GetFramesOfInterest` function. The `exportFramesOfInterest` function allows all frames-of-interest in the world-model to be loaded into an OSCAR application and requires only that the `ManipulatorData::GetFramesOfInterest` call be replaced with the `WorldModel::exportFramesOfInterest` call.

5.1.4. Implementation summary

Table 5-1 summarizes the current graph-based world-model implementation. It lists each class created as part of the implementation along with the member data and functions available in each.

Table 5-1. Implementation Summary

Class Name	Member Data	Member Functions
WorldModel	nodeTypeMap linkTypeMap manipulatorMap objectMap locationMap graspMap locatorMap reachMap	addManipulator addObject addLocation deleteNode addGraspLink addLocatorLink addReachLink getManipulator getObject getLocation tagSearch setLocation getClosestLocation getClosestObject exportWorkcell exportFramesOfInterest
ManipulatorNode	DHParams positionLimits tag minimumMOT basePose toolPose configurationsOfInterest configurationNames obstacleModel Joint Limits Geometric Model(s) Configurations of Interest	manipulatorNode (cTor)
ObjectNode	tag primitiveTypes cylispheres planes	ObjectNode (cTor)
LocationNode	location	LocationNode (cTor)
graspLink	gripperJoints HPO	graspLink (cTor) getGripper getHPO
locatorLink	relativePosition uncertainty	locatorLink (cTor) getReltivePosition getUncertainty
reachLink	jointPose MOT	reachLink (cTor) getPose getMOT

5.2. DEMONSTRATIONS

To demonstrate how the graph-based world-model supports manipulator control, the implementation from the previous section was applied on three different manipulator systems. Each system provides validation of the graph-based world-model for a separate manipulator control function. The following demonstrations show that the model supports collision detection, motion planning, and manipulator grasping, respectively.

5.2.1. Collision detection support

Collision detection is an essential feature of a manipulator system for unstructured tasks. The graph-based world-model's support of this feature is demonstrated on a simulated system pictured in Figure 5-3. The system consists of two 7-DOF serial manipulators operating in a glovebox similar to those used for nuclear materials handling. The green objects represent the geometric primitives used in the collision model for this demonstration.

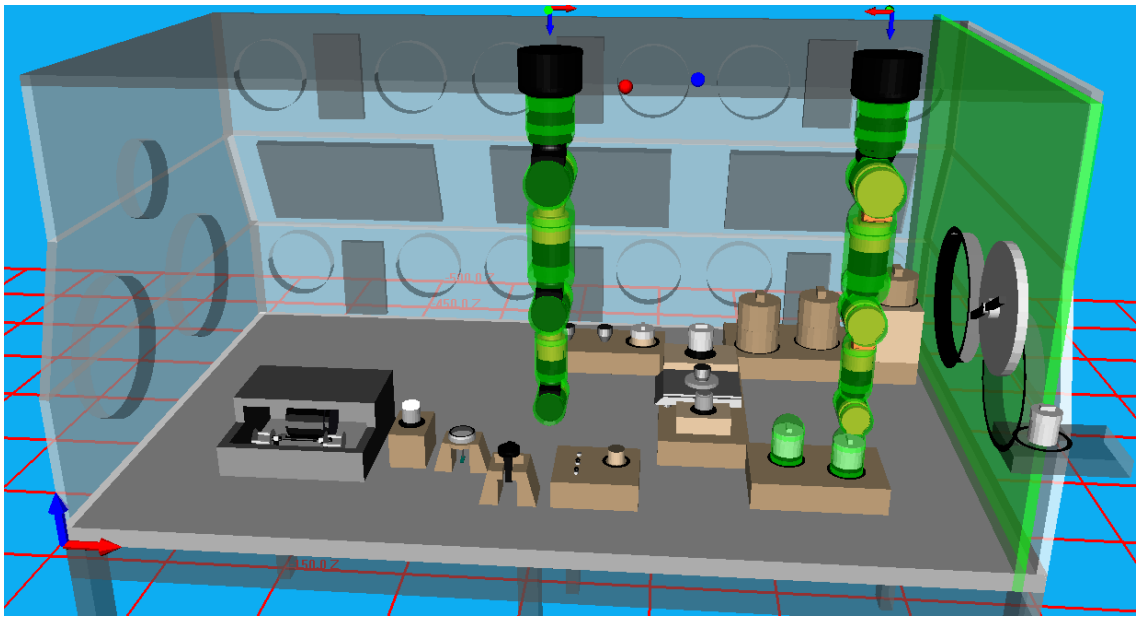


Figure 5-3. Dual-Arm Glovebox Simulation

All of the relevant environmental information for this simulation was built into a graph-based world-model as described in the previous section. Figure 5-4 shows a schematic representation of the world-model constructed for this demonstration. For clarity in presentation, this is only a partial representation of the model that would be used to represent all locations and objects in the system. In addition, in this particular demonstration, no grasping information is stored. Grasping support is demonstrated on a different system and is discussed later.

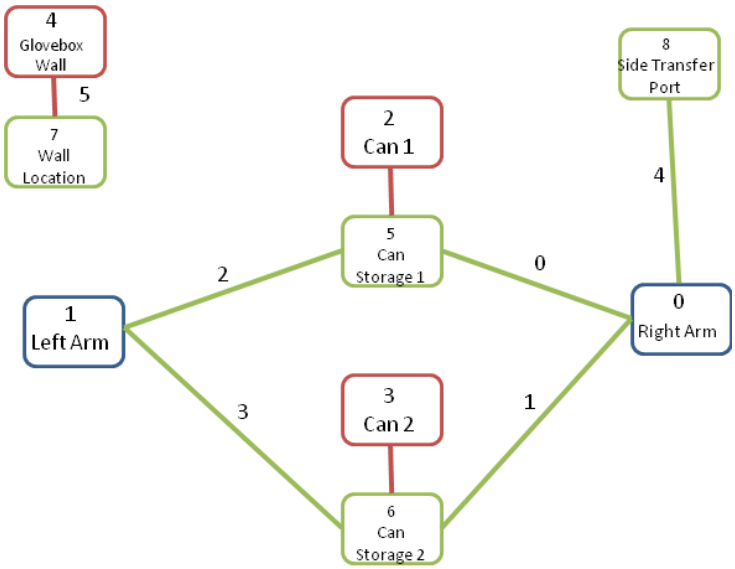


Figure 5-4. Collision detection world-model

Because OSCAR collision detection requires that the collision model be stored in an XML `OSCAR::Workcell`, the `WorldModel::exportWorkcell` function is used to generate the XML file. This XML file is used to load all of the necessary environmental information into the OSCAR application.

In this simulation, the user types the name of a pre-defined frame of interest in order to task the end-effector to that frame. As the manipulator moves, smallest distance calculations are performed on all of the geometric primitives in the collision model. The

application does this through use of obstacle avoidance features available in OSCAR. When a collision in imminent, the red and blue spheres in the simulation environment are moved to the points on the geometric primitives that are nearest each other. The images in Figure 5-5 show the various near collisions as the manipulator moves from an initial state to a location near a port in the side of the glovebox.

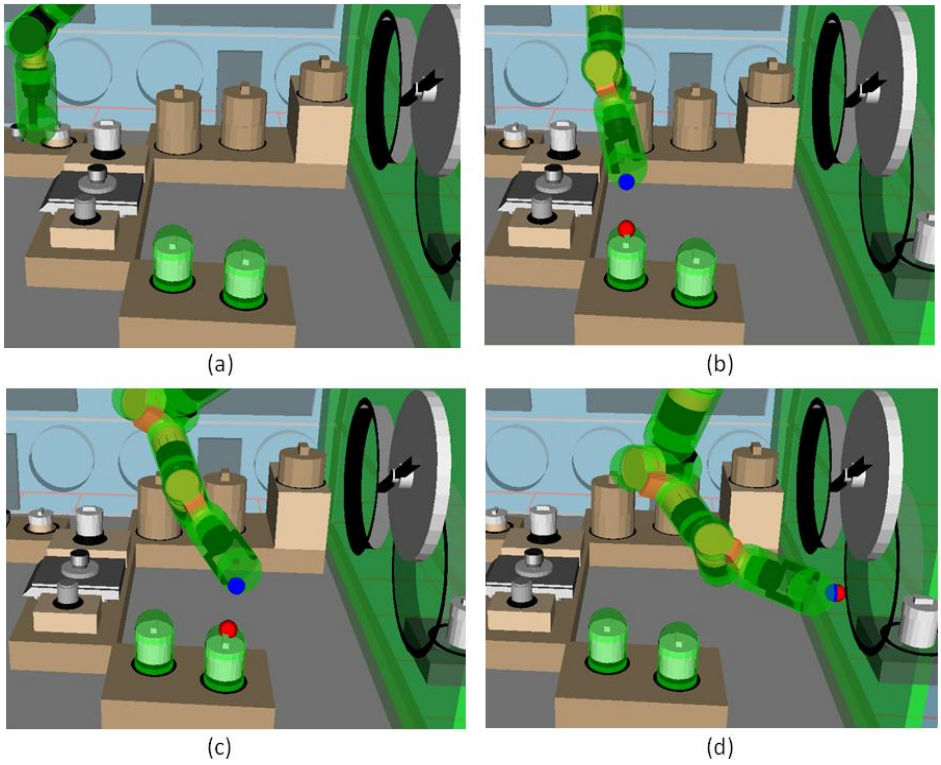


Figure 5-5. Collision detection during a manipulator move.

The simulation presented in Figure 5-5 demonstrates that the workcell's control system can perform the minimum distance calculations from the information contained in the graph-based world-model. The ability to capture this information was already available in the `OSCAR::Workcell`. This simulation is important to demonstrate that the graph-based world-model presented here maintains inter-operability with existing

RRG software without compromising essential control functions, and that it can support multiple manipulators.

This demonstration also shows how the world model provides frames-of-interest. In the simulation presented here, the side port that the manipulator is reaching toward in Figure 5-5 is contained in the world-model as a frame of interest. Since the strict global handling of frames-of-interest is an important difference between the graph-based world-model and the OSCAR Workcell, it is important to demonstrate that no functionality is lost when using frames-of-interest in an OSCAR application. The translation of the OSCAR workcell into a graph-based model followed by the re-exportation to an `OSCAR::Workcell` XML file results in manipulator frames-of-interest being moved to global scope. This had no effect on the manipulator's ability to plan and execute an end-effector move into the transfer port.

5.2.2. Motion planning demonstration

The graph-based world-model's support for motion planning is demonstrated in a glovebox used for nuclear materials handling. The system consists of a modular 7-DOF Amtec Powercube robot mounted in one of the glovebox's side transfer ports. The glovebox and robot are pictured in Figure 5-6. The mounting fixture for the robot is visible in the photograph of the glovebox.

Figure 5-6. Glovebox and robot

This system is being developed in collaboration with Los Alamos National Laboratory to demonstrate the benefits of flexible automation in nuclear materials handling. One system application is the size-reduction of metallic hemispherical components. This operation consists of sectioning 10-16 inch outside diameter hemispheres into pieces that fit into a crucible with a 2.5 inch opening. A device that performs the actual size reduction is under concurrent development at the University of Texas at Austin. The robot must move the metal hemisphere from an initial receiving location to the size reduction device. Because the physical system is still under construction at the time of this writing, the demonstration is performed in simulation.

The graph-based world-model greatly simplifies planning and execution of robot motions. This is facilitated by the inclusion of searchable semantic information to describe objects and locations in the environment. The actual location of an object or frame of interest need not be known by the control application in advance in order to plan a grasping motion. To plan a motion to grasp an object that exists at a particular frame-

of-interest, the control system can retrieve from the frame-of-interest's location node the transformation between the global frame and the frame of interest. It can retrieve the position of the object relative to the frame of interest from the locator link that connects the object node to the location node. The grasp HPO relative to the object can be retrieved from the grasp link that connects the object node to the manipulator. The final desired position of the end-effector for motion planning is then the product of these three matrices. At no time does the control system need to track any of this information. As long as all elements in the system update the world-model with current information, all agents can retrieve pertinent data when it is required.

The grasp link in the world-model serves an important role in task and motion planning as well. Since the grasp HPO does not change relative to the object in links, it can generate global approach poses. It is not typically sufficient to plan a motion directly to the end-effector pose that will be used to grasp an object because a collision with the object is likely. It is preferable to first move to some approach pose and then to move along the final DH frame's z axis with an open gripper. This approach pose is easily generated by applying an offset to the z translation of the HPO prior to performing the matrix multiplication described above. This method is used in the demonstration described below, and is illustrated in Figure 5-8 below.

Figure 5-7 shows the simulation and the graph-based world-model that describes the initial state of the system for the size-reduction demonstration.

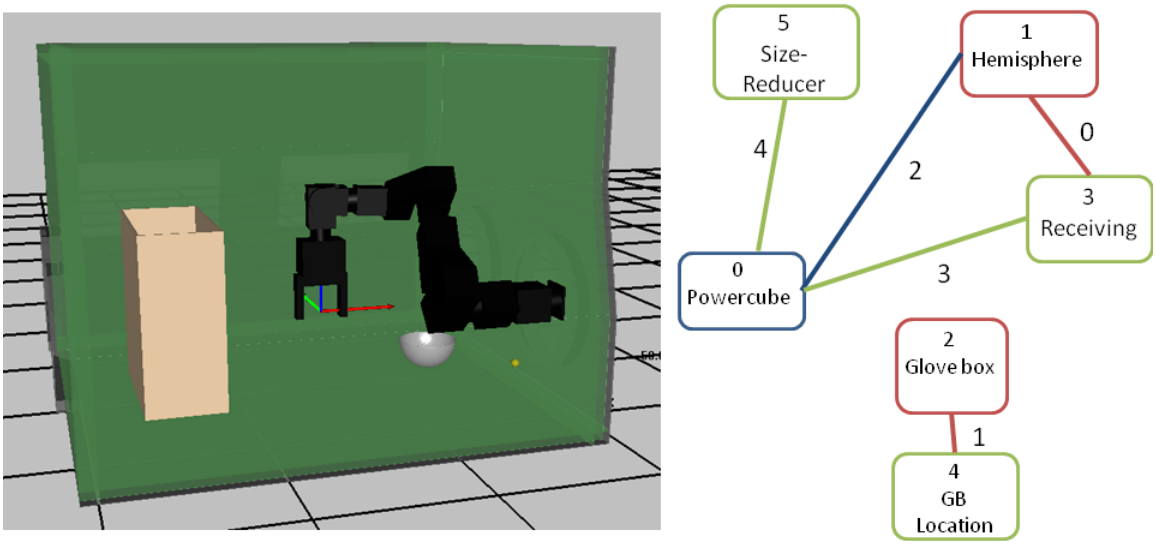


Figure 5-7. Glovebox system initial state.

The metallic hemisphere has been placed in an initial location just inside the transfer port adjacent to the robot. The size-reduction device is the box at the other end of the glovebox.

Figure 5-8 is a succession of images showing the robot's motion from its initial state to grasping the hemispherical object. The final end-effector orientation of this move was planned by the method described above, and the resulting transformation matrix was passed to the motion planning algorithms available in OSCAR to plan and execute the motion.

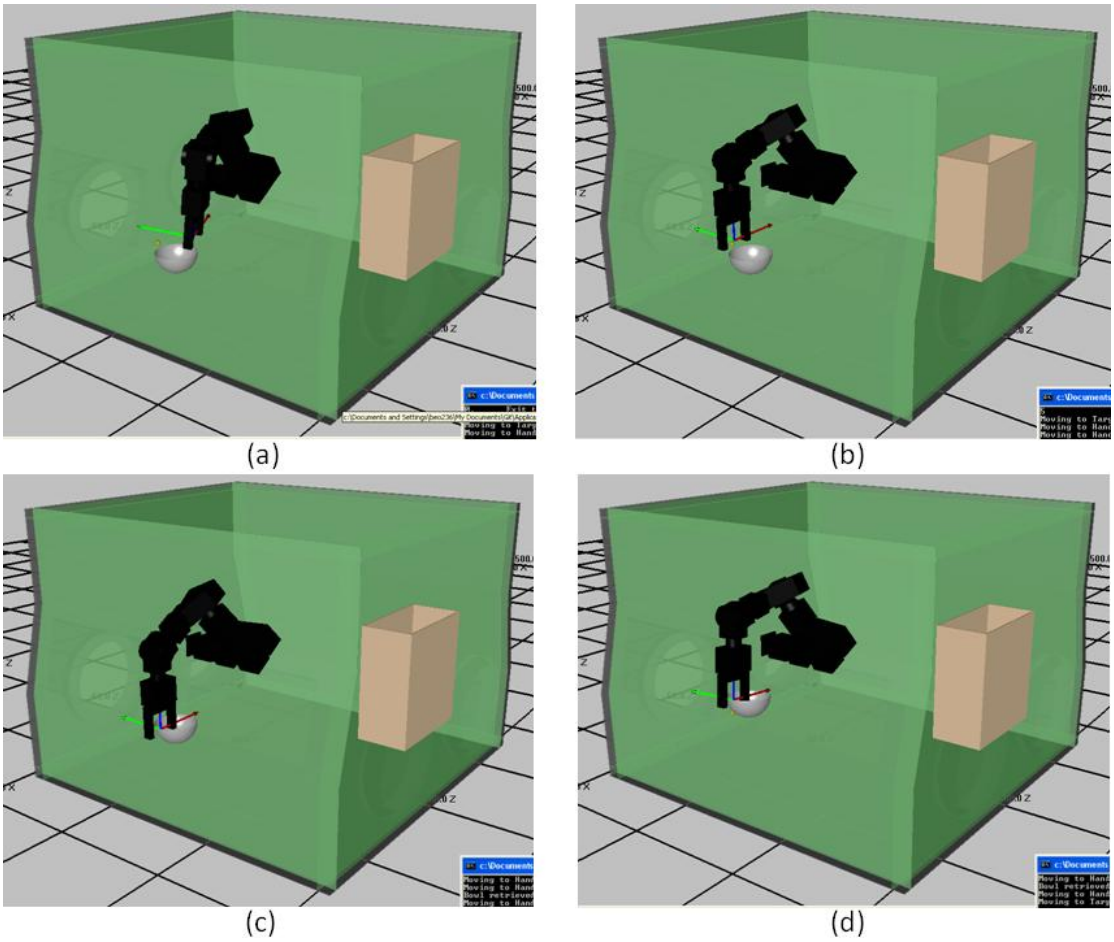


Figure 5-8. Planned motion to grasping position. (a) Start of motion. (b) Approach pose. (c) Grasp pose. (d) Hemisphere retrieved.

This motion requires no knowledge of the hemisphere's actual location. If the hemisphere were moved by another system component, or perhaps by a human working in gloves, the application code that grasps the bowl would function properly as long as the hemisphere's position is accurately updated in the model via machine vision or other sensed data.

Figure 5-9 shows the next move in the task. The series of images shows the hemisphere being moved from its initial position into the size-reduction device.

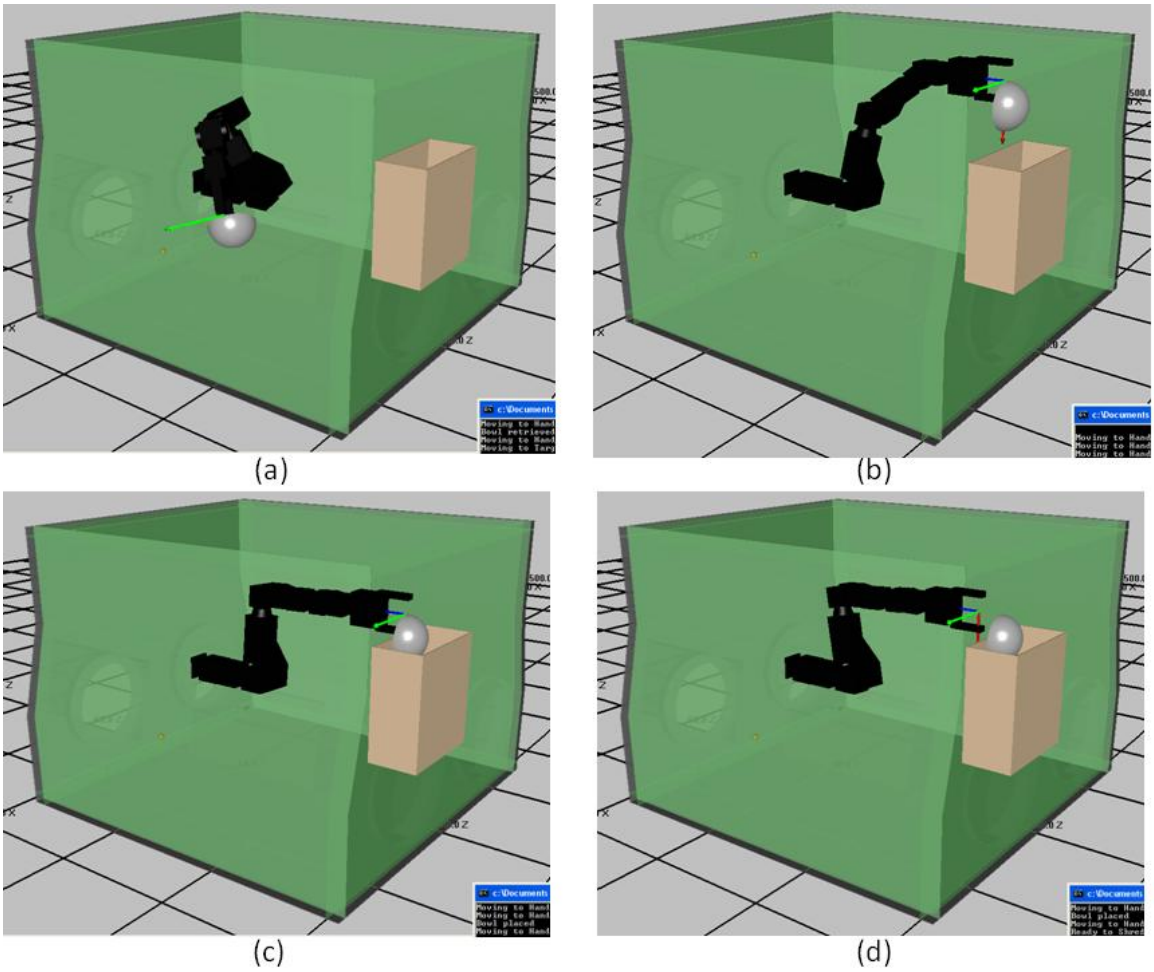


Figure 5-9. Motion to the size-reduction device. (a) Start of motion. (b) Approach pose. (c) Hemisphere in size-reduction device. (d) Hemisphere released.

In this case, once the object is placed in the size-reduction device, it is removed from the model. The final position for this move is determined by the same matrix multiplication used to generate the grasping position. An approach pose is generated to assure that the hemisphere enters the size-reduction device vertically. However, this motion requires the application to know that this is the correct approach angle. In the current implementation, there is no provision for specifying an appropriate approach direction to a frame-of-interest. However, the existing solution for this problem is to move the end-effector through a set of frames-of-interest and motion plans that compel a

particular direction of approach. Secondary dependant frames could be generated from sensor data as well.

5.2.3. Grasping demonstration

Perhaps the most significant contribution of the graph-based world-model to manipulator control is grasping support. This support is demonstrated on a mobile manipulation system constructed to perform remote contamination testing in radiologically hazardous environments. The system consists of a Segway RMP400 mobile platform, a Schunck LWA3 7-DOF manipulator, and a BarretHand BH-8 end-effector. Its sensor suite includes two laser rangefinders and a SwissRanger Infrared camera.

System description

The system is designed to operate under the dynamic autonomy scheme described in Table 1-1. A graphic user interface was developed to support high-level collaborative manipulator tasking in unstructured environments. A SwissRanger infrared time-of-flight camera provides a cloud of 3D Cartesian data points in the environment. The interface presents the operator with a 2D triangulated mesh of these 3D points. The result looks similar to a typical monochrome video image. The operator may then select objects in the scene and perform actions on them simply by clicking them and selecting actions from a drop-down menu.

In the original implementation of this visual tasking interface, the world was modeled poorly. The only information available to the control system was the set of 3D points measured by the SwissRanger. When a point was selected by the user, the manipulator could be tasked to that physical location, but without any knowledge of the actual geometry of the object that existed at the location. The Barrett Hand uses a clever

hardware design that allows closure of the fingers around objects of uncertain geometry. This allowed surprisingly robust grasping of simple objects despite a lack of geometric information.

In the next iteration of the interface, some rudimentary image processing was done to identify object properties (height, diameter, and central axis) of cylindrical storage containers. This allowed more intelligence to be built into the system. The system could notify the operator if the requested object were too large to be grasped. It also enabled use on systems equipped with less-sophisticated grippers. This capability was demonstrated with a SwissRanger in a simulated glovebox environment on a 6-DOF manipulator with a parallel jaw gripper. The gripper opening could be set with sufficient precision based upon the cylinder geometry extracted from the SwissRanger data. The precision can be improved even more if the geometry of the selected cylinder is compared to a catalogue of cylinders that are likely to be in the environment. Such look tables are very reasonable in the more controlled environments associated with glovebox and other nuclear research and manufacturing facilities.

Demonstration results

Implementing a graph-based world-model on this system allows much more intelligent grasping. The system is capable of grasping objects that require complicated grasps, and can do so with high repeatability. Recall that a grasp consists of two parts. The first is a Hand Position and Orientation (HPO) that represents the orientation of the hand with respect to the object being grasped. The second is the set of values for each controllable DOF. Through manual experimentation, HPO and joint values were determined for a cordless electric drill and a roll of tape used to seal cans containing hazardous nuclear materials. In previous iterations of this system, the BarretHand was

operated as if it has only a single DOF, and this DOF was controlled in a binary fashion. It was either opened or closed. The electro-mechanical design of the hand allowed it to be safely closed on objects, but the resulting grasp was unpredictable and there was no guarantee that the resulting grasp would be force-closed. The graph-based world-model allows storage of the HPO and gripper information, allowing better use of the Barrett Hand's four controllable DOF, and assuring repeatability of the closure state of the grasp.

To demonstrate these advantages, the electric drill and the roll of tape are placed in known initial locations in the environment. Figure 5-10 shows the system and environment in its initial state.

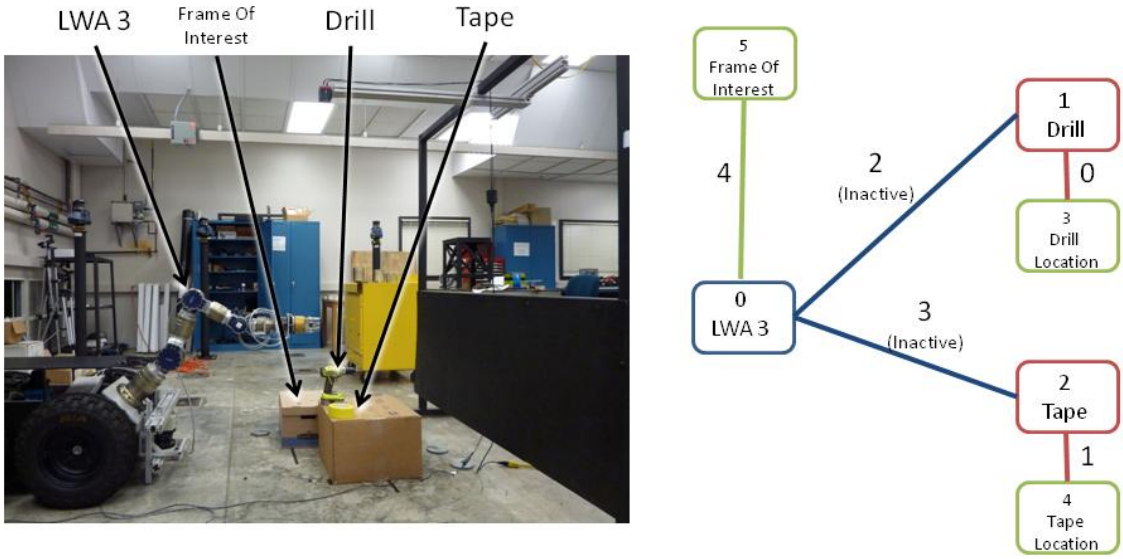


Figure 5-10. Environment and world-model in initial state

The `WorldModel`'s `GetClosestObject` and `GetClosestLocation` functions are used with the SwissRanger interface to select objects and locations in the environment. These functions allow a tolerance parameter to be passed in to allow the application programmer to account for a reasonable amount of position measurement uncertainty. Figure 5-11 shows how the operator selecting and retrieving the drill. Figure

5-12 shows the drill in its grasped state. Note the precise placement of the fingers on the drill. Two fingers grasp the handle and the third is on the trigger.

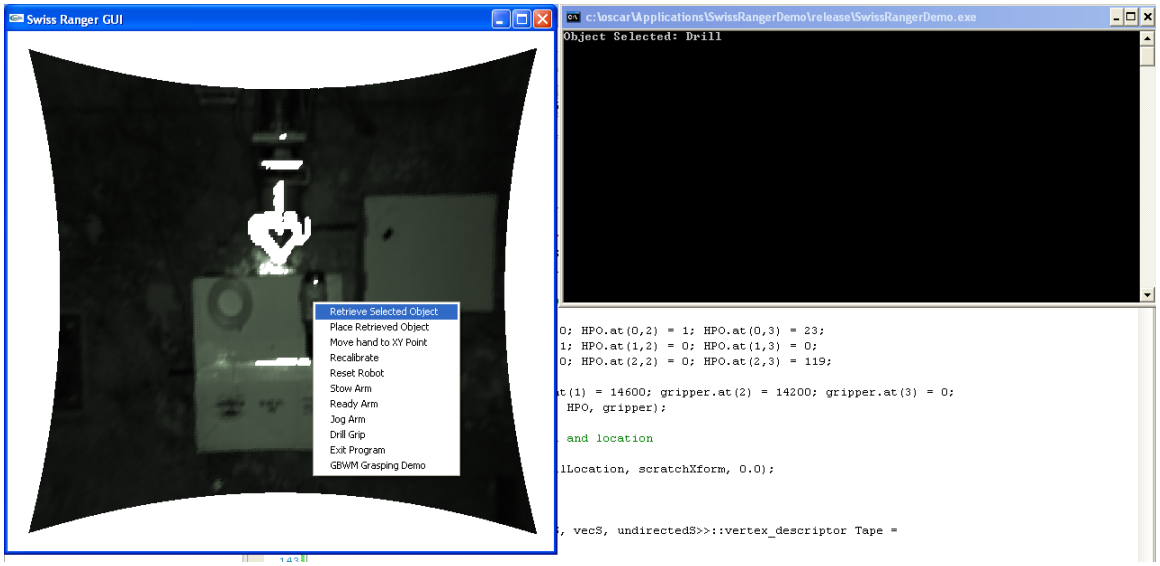


Figure 5-11. User interface showing selection and retrieval of the drill

Figure 5-12. Drill grasp

After retrieving an object, the operator can select a destination position from the image to move the object. The system allows selection of frames-of-interest that are known explicitly and objects can therefore be placed precisely. In this event, the object

node representing the grasped object is linked to the location node representing the frame of interest. Figure 5-13 shows the drill placed at a frame of interest. The origin of the frame of interest is indicated by the black marker. The origin of the drill's object node is at the heel of the drill's handle. Note the precise placement of the drill.

Figure 5-13. Placement of drill at frame of interest

If the operator selects a location that is not a frame of interest, the SwissRanger-measured location can be used as the target location, but the operator is informed that the resulting motion is subject to the positional uncertainty of the SwissRanger. In this case, the `OSCAR::Xform` of the location node attached to the object node is updated to reflect its new position. This is demonstrated by moving the tape to the drill's original location, which is not identified as a frame of interest and is not stored the world-model. Figure 5-14 shows the manipulator after retrieving the tape. Note that the grasp makes use of all of the gripper's degrees-of-freedom. Figure 5-15 shows the tape being released at the selected position.

Figure 5-14. Tape grasp

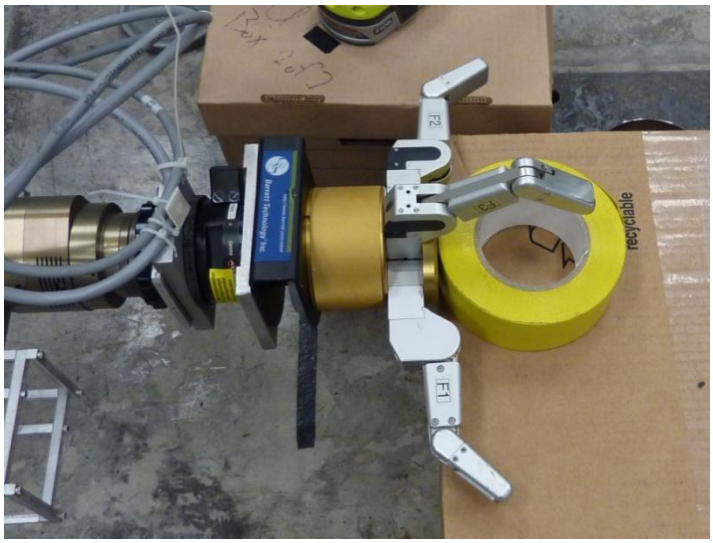


Figure 5-15. Placement of tape in an arbitrarily selected location in the environment

The control system remembers where objects have been placed by updating the world-model as described above. Since the grasp HPO is relative to the object and both the HPO and location are stored as transformation matrices, correct positioning of the end-effector can be achieved by multiplication of the HPO `OSCAR::Xform` and the location node's `OSCAR::Xform`. The result is a predictable force-closed grasp regardless of the position of the object to be grasped. Figure 5-16 shows the manipulator

grasping the drill again after moving the tape. Note that the same grasp is achieved despite the drill being in a different position and orientation.

Figure 5-16. Re-grasping the drill

The above demonstration shows that robust grasping is achieved through use of the graph-based world-model as it has been implemented here. Multiple objects requiring complex grasps can be moved around in a loosely structured environment through high-level collaborative tasking on the part of the operator. This was not possible in the OSCAR::Workcell model. It would have been possible in a single-manipulator system to model the HPO and gripper joint parameters as object properties. However, this solution would break down if a second manipulator were introduced to the system. The graph-based model allows properly closed grasps to be stored and used between an arbitrary number of manipulators and objects.

5.3. IMPLEMENTATION AND DEMONSTRATION SUMMARY

This chapter presented an implementation of a graph-based world-model along with three demonstrations that show how the graph-based world-model supports collision detection, motion planning, and grasping. Building these demonstrations on three

separate systems demonstrates how the graph-based world-model satisfies the architecture agnosticism requirement expressed in Chapter 4. Each of the three demonstrations shows some aspect of the representational efficiency and topological structure requirements as well.

The implementation was built upon the Boost Graph Library. The model provides classes that contain all the information for the node and link types suggested in Table 4-7, as well as a world-model class that contains the graph itself all of the property maps required to provide access to individual node and link objects, and various functions for access and management of the nodes and links in the model.

Collision detection was demonstrated in a hypothetical general radiochemistry glovebox used for experimental mixed-oxide fuel fabrication. An OSCAR workcell XML file was generated from a graph-based world-model. This XML file was then used to load the collision model for the entire system. Nearest distance calculations are demonstrated in Figure 5-5, showing that collision detection is supported and relevant information for obstacle avoidance algorithms is readily available. This demonstration was performed on a system with two manipulators. Prior to the graph-based implementation, world-modeling on this system modeled frames-of-interest as manipulator properties, resulting in redundant representation of several locations in the glovebox. The graph-based approach eliminated this redundancy.

The motion planning demonstration showed the value of modeling the world as a set of objects and a separate set of relations among the objects. Modeling the environment in this way allowed a control system to determine an appropriate approach pose and grasp pose for planning a motion to pick up an object. It also demonstrated the value in a model that captures topological structure. The use of searchable semantic type and tag values allow the control system to use the abstraction present in the world-model

at the command level. In other words, the manipulator can be tasked to "pick up the hemisphere" as opposed to executing a rigid sequence of motion commands that will fail if the object is not in the position that the sequence expects. This ability is a direct result of the manner in which the graph structure abstracts relationships among entities in the environment.

The grasping demonstration also showed the value in capturing relations among entities in the environment. A visual point-and-click interface generated from 3D sensor data allowed the operator to select points in 3D that could be compared with world-model data to identify objects modeled in the environment. Data stored in the grasp link object provided the control system with the necessary information to grasp different types of objects with known force-closed grasps. Dynamic update of the world-model data allowed the objects to re-selected and grasped even after being released in arbitrary locations.

The next chapter examines some areas that still need to be implemented in software and identifies areas that warrant further analysis with respect to the proposed modeling techniques.

CHAPTER 6 : CONCLUSIONS AND FUTURE WORK

6.1. RESEARCH SUMMARY

This document presents a graph-based world-modeling scheme that supports transitional levels of autonomy in robotic manipulation. **Chapter 1** discussed the importance of providing varying levels of autonomy to improve operator ergonomics, efficiency, and safety. It discussed the importance of this autonomy in the use of robotics for explosive ordnance disposal and nuclear materials handling, application domains which will benefit in the near term even from incremental improvements in autonomy.

Chapter 2 identified three manipulator control functions essential to autonomous or semi-autonomous manipulator behaviors. These functions are:

- Collision detection/avoidance
- Motion planning
- Grasping

The chapter then examined how these functions are currently performed in the literature to assure that the proposed world-modeling scheme meets the requirements of modern control algorithms.

Collision detection relies solely on a geometric description of the robot and its environment. This geometry may consist of Boolean combinations of geometric primitives, or it may be derived from CAD models and take the form of polygonated meshes. Nearest-distance and closure rates for cylispheres can be computed using first and second order kinematic influence coefficients of the manipulator according to equations 2.1 – 2.3. Collisions between convex polytopes, including polygonated meshes from CAD data can be computed with robust implementations of the GJK algorithm such as the SOLID collision detection library.

Motion planning for robotic manipulators is the process of computing a discretized trajectory between an initial and final joint or end-effector state. At its most basic, this process consists of computing inverse kinematics for the manipulator at each discrete point along a straight line between initial and final states. For kinematically redundant manipulators, however, the desire to avoid collisions or optimize a particular parameter can make the trajectory-generation quite complex. Chapter 2 presented several approaches to this problem, none of which was well-suited to online global motion planning. A primary requirement of a motion plan is that it avoids collisions, and thus the same geometric information required by collision detection algorithms is required by motion planning algorithms. However, motion planning algorithms may also aim to optimize additional space-dependent parameters such as manipulability or transmissibility. Thus, a world-model should be capable of capturing such information.

In order to provide any meaningful autonomy, a manipulator control system must be capable of using appropriate grasps when manipulating objects. Chapter 2 introduced the concepts of force and form closure, and covered some measures by which these properties can be evaluated. In addition to grasp analysis, a few methods for grasp synthesis were also examined. Current grasp synthesis algorithms are slow and typically require a high-fidelity geometric representation of the object to be grasped. This makes robust online computation of grasps impractical with current sensing technologies. However, if objects that the manipulator is likely to encounter are known *a priori*, grasp parameters can be pre-computed. Regardless of whether the grasp is computed online or pre-computed, the Hand Position and Orientation (HPO) and end-effector joint parameters are essential to manipulator task and motion planning.

Current world-modeling techniques have focused primarily on collision prevention. As a result, they typically contain only geometric information. However, the

motion planning and grasping algorithms described above require information about the environment that is non-geometric. This information may be impossible to express as a property of a single entity in the environment. This makes strict hierarchical schemes such as the OSCAR workcell poorly suited to capture such information.

Chapter3 surveyed several available sensors that can provide the primitive data required to populate a world-model.

Chapter 4 proposed a graph-based world-modeling technique that retains the richness of the OSCAR workcell, but also captures relationships among entities in the environment. The primary contribution of this research is the abstraction of the robot's environment into a graph structure that separates entities in the environment from their relationships to the environment. Entities in the environment are modeled as nodes in a graph, and relationships among entities are modeled as links between the nodes. The result is an abstract topology of the environment. This abstraction, combined with the ability to search the graph with semantic information allows control systems using the model to plan and execute tasks without explicit *a priori* knowledge about the environment. This is essential if the robot is to operate in an uncertain environment where modeling information is generated or updated in real time. This approach is intuitive because it more closely approximates how humans think about their environment. If asking a human to retrieve a pencil from a table, one simply says, "Please pick up the pencil from the table." The exact locations of the pencil, table, and the relative position of a hand to the pencil are all abstracted away in this simple request. The graph based world-model enables similar functionality in manipulator control.

Chapter 5 presented an implementation of the graph-based world-model built upon the Boost Graph Library and uses many of the data structures available in the

OSCAR manipulator control libraries. This implementation offers the following types of nodes and links.

- Manipulator node
- Location Node
- Object Node
- Grasp Link
- Locator Link
- Reachability Link

Table 4-1 describes the information captured by each of these nodes and links.

The graph-based world-model's capabilities were demonstrated on three systems relevant to the nuclear engineering domain. These demonstrations showed that the proposed modeling scheme supports collision detection, motion planning and grasping, while maintaining architecture agnosticism, representational efficiency and making use of topological relationships between entities in the environment. Table 6-1 summarizes the three demonstrations and the capabilities demonstrated in each.

Table 6-1, Demonstration Summary

Demonstration System	Task Description	Capabilies Demonstated
Dual-arm automated MOX fuel fabrication glove box.	Move to side transfer port	-Support for OSCAR collision detection -Support for multiple manipulators
Automated general nuclear materials handling glove box with 7-DOF Powercube	Move a metallic hemisphere from a receiving location to a size-reduction tool	-Support for motion planning algorithms -Autonomous approach pose generation -Command-level task abstraction
Mobile LWA3 Arm with BarrettHand BH-8 and SwissRanger based user interface.	Retrieve and place multiple objects at frames of interest and at arbitrary locations	-Appropriate grasp selection -Frame-of-interest modeling -Use of world modeling data to provide operator interface -Dynamic model update

6.2. RECOMMENDATIONS FOR FUTURE WORK

The implementation of the graph-based world-model presented here supports and extends several manipulator control functions. However, a few areas warrant further work and investigation. Many of these investigations are only possible now that a graph-based modeling technique has been implemented. This section recommends a few areas where the modeling scheme could be improved, and suggests some future work beyond the scope of this research that would allow this modeling scheme to advance manipulator control through greater levels of autonomy.

6.2.1. Grasping support

Because a grasp is a relationship between a manipulator's end-effector and the object being grasped, the graph-based approach is well-suited to grasping. However, the proposed technique and its implementation only support a single grasp for an object. It may be necessary to grasp an object in multiple ways in many situations. For example, in the size-reduction task described in Chapter 5, after the hemisphere is size-reduced, the pieces are deposited in a crucible. This crucible must be lowered into an oven in the floor of the glovebox. It is essential that the crucible remain upright during transport. This means that after retrieving the crucible, the manipulator will have to set it down and pick it up from the top in order to lower it into the oven. The current world-model does not permit this additional grasping configuration to be stored because graph data structures usually only allow a single edge in each direction between vertices. A special type of graph called a multi-graph exists which does permit multiple edges between vertices. Support for multiple grasp types could be added with this type of graph. Support for multi-graphs is provided by the Boost Graph Library.

Another weakness of the current grasping support is that it assumes that the grasp is described by a unique Hand Position and Orientation (HPO) and associated set of joint

parameters. However, it is possible that there are free variables in the HPO that could be used to optimize placement of the end-effector. The hemisphere in the motion planning demonstration is a good example. Because it is axisymmetric, the rotation of the end-effector about the hemisphere's vertical axis is unimportant. In the current implementation, it is possible that attempting to use the modeled grasp will result in the manipulator not being able to reach the hemisphere when in fact it could reach if gripped at a different location around the hemisphere's edge. Grasping support would be significantly improved if some provision existed for identifying any degree-of-freedom that may exist in the grasp HPO.

6.2.2. Motion planning

Planning motions in Cartesian space results in more predictable motion of the robot. This is particularly important in the constrained environments of the glovebox systems shown in Chapter 5. Unfortunately, for large motions, it is very likely that a Cartesian trajectory is not possible. In order to plan a Cartesian trajectory, each discrete end-effector pose along the trajectory must have an inverse kinematics solution. In less constrained environments, this is easily avoided by using joint-space planning for large motions.

It would be possible to use graph-based world-modeling and graph traversal to plan a series of Cartesian moves between end-effector poses in order to avoid the unpredictable motions of joint-space planning. This could be done by indicating when a Cartesian trajectory between two location nodes is possible. If enough locations in the environment are modeled, the result would be a web of location nodes. A graph traversal would yield a path from the initial to final locations, and the control system could follow only Cartesian trajectories between the points in the path. This might be accomplished

with an additional link type that could connect location nodes between which a Cartesian move is possible. However, the trajectory is specific to a particular manipulator, so the manipulator node must somehow be included. In keeping with the convention of using nodes to represent entities and links to represent their relationships, the Cartesian trajectory presents a problem. It must link together three nodes instead of two, and there is no provision for a "y-link" in the graph structure. An alternative might be to make the trajectory a node which links to a manipulator and two locations, but this abandons the idea of separating entities from relationships. It also complicates the problem of finding a Cartesian path between points since the traversal would not result in a series of locations, but instead an alternating series of location and trajectory nodes. Despite the difficulties in implementation, it appears that the graph-based world-model could allow better use of Cartesian moves, and an appropriate way to support this function should be investigated further.

One of the advantages of the way that grasps are modeled in the current implementation is the support of automated approach pose generation. Unfortunately, this capability is only possible when approaching an object node and not when approaching a location node. If the location of interest were a box with only one open side, the direction from which it is approached is important. There is no way to indicate this in the current implementation. A provision for adding an approach vector to a location node should be added.

6.2.3. Architecture agnosticism

One of the requirements of a world-model for manipulation articulated in Chapter 4 is that it should remain architecture agnostic. While the diversity of demonstrations presented in Chapter 5 suggests that the system is usable across many different platforms,

one way in which the current implementation does not satisfy this requirement is in its heavy use of the OSCAR libraries. This prevents the model from being used on any system not based on OSCAR. One way to improve the agnosticism of the current implementation would be to re-write the `WorldModel` class using the principles of *Generic Programming*. Siek [2002] provides a brief discussion of Generic Programming, a technique that allows data types to be specified at run-time. As an example, the current implementation relies on the `OSCAR::Xform` to communicate position and orientation data. Instead of specifying the `OSCAR::Xform`, the `WorldModel` could specify a template for any 4x4-transformation matrix type. This would maintain the inter-operability with OSCAR, but would also allow users to define their own transformation matrix types for use on systems not using the OSCAR libraries.

Another approach would be to specify a communication protocol so that the world-model could continue to use OSCAR types internally, but the access functions would pass data in a neutral format. For example, instead of returning an `OSCAR::Xform`, an access function might return an array of 16 doubles that could then be interpreted by the calling function into whatever system it is running. This is how the popular Robot Operating System (ROS) assures inter-operability between different systems. [ROS.org 2010] A system in ROS is a graph of nodes that communicate with each other by subscribing to each other's services. To offer a service in ROS, a message type must be specified that defines the data structures used to transmit information. ROS integration is strongly suggested as an area of future work because it would assure that the implementation is agnostic to the rest of the control system and avoids the expenditure of valuable research efforts implementing functionality that exists elsewhere.

6.2.4. Object recognition

The world-model is a repository of information relevant to robot control. The model implemented in the current research supports high levels of autonomy. This was demonstrated in Chapter 5 where systems were capable of responding to high-level operator commands. Motion planning, collision avoidance and grasp selection were all performed without specific direction from the operator. It is important to recognize, however, that these demonstrations required significant *a priori* data. To achieve more autonomy in unstructured environments, it is necessary that the model be populated with real-time sensor data. Chapter 3 surveyed some of the available sensors that may be used to provide raw environmental data, but the parsing of this data into individual objects is currently an active area of research. This is beyond the scope of the current research, but is nonetheless essential to realizing the full utility offered by the graph-based world-model.

6.2.5. Improved online grasp synthesis

Also beyond the scope of the current research, and closely related to the problem of object recognition is that of online grasp synthesis. Chapter 2 covered some of the current approaches to grasp synthesis but they tend to be slow and require high-fidelity geometric information about objects that is difficult to tease out of sensor data. Right now, these functions can still be incorporated by including a human in the control loop who can identify objects and determine force closed grasps by trial and error. To advance manipulator autonomy, novel objects must be recognized by sensed data and appropriate grasps synthesized online.

6.3. CONCLUDING REMARKS

Chapter 1 opened with a discussion of robot autonomy and presented the idea of a spectrum of autonomous behaviors. The research presented here is meant to provide a world-modeling scheme capable of supporting manipulator control at any level of autonomy. Chapter 2 identified collision detection, motion planning, and grasping as three functions that rely on world-modeling data and represent increasing levels of manipulator autonomy. Chapter 3 surveyed sensing technologies that may be used to provide data that populates the world-model. Chapter 4 proposed a particular modeling scheme consisting of three types of links and three types of nodes, and explored what kinds of data should be contained in the structure. Chapter 5 described one implementation of the model demonstrated on three different systems relevant to nuclear materials handling and remote contamination testing. These systems clearly demonstrated the benefits of the proposed modeling solution.

The major contribution of this research is the extension of the RRG's manipulator world-modeling beyond geometry. By abstracting the environment into a graph representation, the model approximates the human cognitive map. This abstraction can be brought to the command level, enabling high-level manipulator tasking. This high-level tasking enables better human-robot collaboration that in turn improves ergonomics, efficiency and safety.

References

Aggarwal, A. (2009). *Task Based Global Motion Planning of Multiple Manipulators in Time-Varying Environments* (Unpublished master's thesis). University of Texas at Austin, Austin, TX.

Agile Planet: Kinematix. (2009). Retrieved from Agile Planet, Inc. website: http://www.agileplanet.com/products/kinematix

Angelopoulou, E., Hong, T. H., & Wu, A. Y. (1992). World-model representations for mobile robots. *Proceedings of the Intelligent Vehicles '92 Symposium*.

ATI Industrial Automation. (2010). *F/T Sensor: Gamma* [Product information page]. Retrieved from ATI Industrial Automation website: http://www.ati-ia.com/products/ft/ft_models.aspx?id=Gamma

Berenson, D., & Srinivasa, S. S. (2008). Grasp synthesis in cluttered environments for dexterous hands. *IEE-RAS International Conference on Humanoid Robots*.

Biagiotti, L., Lotti, F., Melchiorri, C., & Vassura, G. (2004). How far is the human hand? a review on anthropomorphic robotic end-effectors. In *Tech. Rep.* Spain: University of Bologna.

Bicchi, A., & Kumar, V. (2000). Robotic Grasping and Contact: A Review. *International Conference on Robotics and Automation*.

Bruemmer, D. J., Marble, J. L., McKay, M. D., & Dudenhoeffer, D. D. (2002). Dynamic-Autonomy for Remote Robotic Sensor Deployment. *Spectrum 2002*.

Canny, J. (1988). *The Complexity of Robot Motion Planning*. Cambridge, MA: MIT Press.

Collins, W. (2005). Graphs, Trees, and Networks. In *Data Structures and the Java Collections Framework* (pp. 646-695). New York, NY: McGraw-Hill.

Coonley, M. S. (Ed.). (1st/2nd Quarters 2008). ARIES turns 10. *Actinide Research Quarterly*.

Dobkin, D. P., & Kirkpatrick, D. G. (1990). Determining the Separation of Preprocessed Polyhedra - A Unified Approach. *Proceedings of the 17th International Colloquium on Automata, Languages and Programming*, 400-413.

Farouki, R. T., Neff, C. A., & O'Connor, M. (1989). Automatic parsing of degenerate quadric-surface intersections. *ACM Transactions on Graphics, 8*, 174-203.

Gilbert, E. G., & Foo, C. P. (1990). Computing the distance between general convex objects in three-dimensional space. *IEEE Journal of Robotics and Automation, 6*, 53-61.

Hong, T., Balakirsky, S., Messina, E., Chang, T., & Shneier, M. (2002). A Hierarchical World-model for an Autonomous Scout Vehicle. *SPIE Aerosense Symposium*.

Jimenez, P., Thomas, F., & Torras, C. (2001). 3D Collision Detection: a Survey. *Computers and Graphics, 25*, 269-285.

Knoll, J. A. (2007). *Complete Workcell Modeling for Robot Control and Coordination* (Unpublished master's thesis). University of Texas at Austin, Austin, TX.

Konolige, K. (1999, August). Disparity and the Horopter. In *Technical Notes: Stereo Geometry*. Retrieved April 19, 2010, from Willow Garage website: http://pub1.willowgarage.com/~konolige/svs/disparity.htm

Kuipers, B. (2000). The Semantic Spatial Hierarchy. *Artificial Intelligence, 119*, 191-233.

Li, Y., Fu, J., & Pollard, N. (2007). Data driven grasp synthesis using shape matching and task-based pruning. *IEEE Transactions on Visualization and Computer Graphics, 13*.

Lin, M. C., & Canny, J. F. (1991). A fast algorithm for incremental distance calculation. *Proceedings of the IEEE International Conference on Robotics and Automation, 2*, 1008-1014.

Lin, M. C., & Gottschalk, S. (1998, May). Collision Detection Between Geometric Models: A Survey. *IMA Conference on Mathematics of Surfaces, 1*, 602-608.

Mesa Imaging, Inc. (2009, August 7). *SwissRanger SR4000 Data Sheet* [Brochure].

Miller, A. T., & Allen, P. K. (1999). Examples of 3D grasp quality computations. *International Conference on Robotics and Automation*.

Miller, A. T., Knoop, S., Christensen, H. I., & Allen, P. K. (2003). Automatic grasp planning using shape primitives. *International Conference on Robotics and Automation*.

Nelson, T. O., Bronson, M. C., Lewis, D. K., Massey, P. W., Cremers, T. L., & Kreyer, L. S. (n.d.). *Advanced Recovery and Integrated Extraction System Program Plan*. U.S. Department of Energy, Los Alamos National Laboratory.

Noakes, M. W. (1998, September). Remote Dismantlement Tasks for the CP5 Reactor: Implementation, Operation, and Lessons Learned. *Spectrum 98 International Conference on Decommissioning and Decontamination and on Nuclear and Hazardous Waste Management*.

Noakes, M. W., Haley, D. C., & Willis, W. D. (1997, April/May). The Selective Equipment Removal System Dual Arm Work Module. *Proceedings of the*

American Nuclear Society Seventh Topical Meeting on Robotics and Remote Systems.

Optical Metrology Centre. (2001). *OMC technical brief - laser time of flight* [Brochure].

Parallax Inc. (2006, June 13). *Ping))) Ultrasonic sensor datasheet* [Brochure].

Perry, B. (1995). *The Development of Distance Functions and Their Higher-Order Properties for Artificial Potential Field-Based Obstacle Avoidance* (Unpublished master's thesis). University of Texas at Austin, Austin, TX.

Phillippsen, R., Nejati, N., & Sentis, L. (2009, July). Bridging the Gap Between Semantic Planning and Continuous Control for Mobile Manipulation Using a Graph-Based World Representation. *International Workshop on Hybrid Autonomous Systems*, 77-81.

Point Grey Research Inc. (2004). *Table Showing Triclops Library Performance Over Different Processor and Memory Configurations* [Data file]. Retrieved from http://www.ptgrey.com/products/ triclopsSDK/TriclopsLibraryPerformance3.2a05.pdf

Point Grey Research Inc. (2009). *Bumblebee 2 datasheet* [Brochure]. BC, Canada: Author.

Pressure Profile Systems, Inc. (2009, April). *Robotouch* [Brochure]. Retrieved from http://www.pressureprofile.com/UserFiles/File/Robotics_Brochure_0409.pdf

Roeder-Smith, L. (2002, May/June). Pit Viper Strikes at the Hanford Site: Pit Maintenance Using Robotics at the Hanford Tank Farms. *Radwaste Solutions*, 33-39.

Rosell, J., Suarez, R., Rosales, C., Garcia, J. A., & Perez, A. (2009). Motion planning for high DOF anthropomorphic hands. *International Conference on Robotics and Automation*.

ROS.org Documentation. (2010, June 3). Retrieved from http://www.ros.org

Saxena, A., Driemeyer, J., & Ng, A. Y. (2008). Robotic Grasping of Novel Objects using Vison. *International Journal of Robotics Research, 27*(2), 157-173.

Shadow Robot Company. (2003). Design of a dextrous hand for advanced clawar applications. *Climbing and Walking Robots and the Supporting Technologies for Mobile Machines: CLAWAR*, 691-698.

SICK Sensor Intelligence. (2010, April 15). *LMS-200 Data Sheet* [Brochure].

Siek, J. G., Lee, L.-Q., & Lumsdaine, A. (2002). *The Boost Graph Library: User Guide and Reference Manual*. London: Addison-Wesley Professional.

Sreedhar, R. (1990). *Potential Function Based Obstacle Avoidance Algorithm for Manipulators with Extra Degrees of Freedom* (Unpublished master's thesis). University of Texas at Austin, Austin, TX.

Standards for Protection Against Radiation, 10 C.F.R. § 20 (2007).

Thomas, M. and Tesar, D. 1982 "Dynamic Modeling of Serial Manipulator Arms," *Journal of Dynamic Systems, Measurement, and Control*, Vol. 102, pp. 218-228.

Thrun, S. (2002). Robotic Mapping: A Survey. In G. Lakemeyer & B. Nevel (Eds.), *Exploring Artificial Intelligence in the*. Morgan Kaufman.

Van Den Bergen, G. (2004). *Collision Detection in Interactive 3D Environments*. Morgan Kaufmann Publishers.

Vita

Brian Erick O'Neil was born in Tempe, AZ in 1978 to his parents Pat and Barb O'Neil. He graduated from McClintock High School in 1997. Brian graduated from Arizona State University with a B.S. in Political Science in 2001. He then worked as a professional flight instructor and airline pilot before returning to Arizona State where he earned a second B.S. degree in Mechanical Engineering. While earning his second degree, Brian worked nights at lumber yard, tutored students in math, science, and engineering at Chandler-Gilbert community college, and completed internships at Honeywell Aerospace and Triumph Air Repair. Brian entered the graduate school at The University of Texas in 2008 studying nuclear engineering and robotics. Brian currently resides in Austin with his wife, Emily, and son, Henry.

email brian.erick.oneil@gmail.com

This thesis was typed by the author