Real-Time Motion Planning for Whole-Sensitive Robot Arm Manipulators

Chapter 1

Introduction

Research on robot motion planning today concentrates on endowing robotic systems with the ability to operate autonomously in their environment. This is due to the increasingly complex tasks that future generations of robots will be required to perform in hazardous, hostile or unhealthy environments such as the construction in outer space or unstructured factory work cells. Current industrial robot arms do not have the capability to operate autonomously in a constrained or unknown environment, as robot motions must be downloaded from CAD/CAM systems, or carefully programmed by a human operator using teach pendants. Unfortunately, robots programmed in this manner may collide with unknown objects, including humans which enter into its work area, and may lead to costly damage of the robot or its environment, unacceptable human injury, or loss of valuable productivity. For these reasons it is important to equip robot arm manipulators with the ability to react to unknown or moving obstacles in its environment. Only a sensor-based system capable of real-time generation of continuous purposeful paths among unknown obstacles while guaranteeing collision-free motion for every point of the robot body would be acceptable in these applications. A robot arm equipped with such a system is a safe, productive, and powerful tool that can operate in a highly unstructured or time-varying environment. In addition to the applications mentioned above, uses for robot arms equipped with such capabilities include the exploration and salvage of undersea sites, underground mining operations, maintenance and refurbishment of nuclear facilities, and

similar applications where direct human presence is necessarily limited.

This thesis researches the issues associated with the development of a motion planning system that enables a robot arm manipulator to operate in an unknown environment in a collision-free manner. Current research in motion planning can be divided into two approaches, the model-based and the sensor-based approach. In the model-based approach, complete information about the environment is assumed, and the motion planning occurs by off-line computation of a collision-free path in the environment. Advantages of this approach include its capability to handle, in principle, the most general cases and to deliver optimal solutions. Its drawbacks are the lengthy off-line computations necessary to implement the search for a path, the necessity to have the complete description of the environment, and the inability to incorporate new information and uncertainty in the planned path of the robot arm.

On the other hand, the sensor-based motion planning approach assumes incomplete information about the environment. In this case, the robot has no á priori knowledge about the environment, and is equipped with sensors that notify it of impending collision, or proximity to an obstacle. At frequent intervals while navigating through the environment, the robot decides its next move based on the current sensor information. This approach does not have any of the drawbacks of the model-based approach mentioned above. It does have, however, drawbacks of its own: since a complete description of the environment is not available, the robot is not guaranteed to find the optimal solution, and the approach cannot be used for a general multidimensional case. Furthermore, the method is susceptible to error of the sensor data. The main advantage of this approach, and the reason it was selected for this work, is its ability to handle time varying or unknown environments.

Since the algorithms generated by the sensor-based motion planning approach are well suited for the two dimensional case, much of research in this area has concentrated on roving robots that move using wheeled mechanisms. In the area of robot arm manipulators, the research has concentrated mainly on the theoretical issues of motion planning.

However, so far there has been little research in the problems associated with the actual implementation of sensor systems and the corresponding data and signal processing algorithms for robot arms. This work addresses these problems through the design and construction of appropriate hardware and algorithms for the purpose of obstacle avoidance for robot arm manipulators.

As a first requirement in the sensor-based approach, some form of obstacle sensing along the entire robot arm body is needed. Items of study in this area were: what sensing range was adequate, if the sensor should be a passive or active device, and what sort of sensing radiation should be used. Ideally, the sensor should detect all obstacles, regardless of size, location, shape, color and composition. In our experimental system, the robot arm is covered with a "sensitive skin" sensor, which allows it to detect obstacles in its environment. The skin is covered by 475 active infrared proximity sensors, and the resulting effect is a sensing capability similar to that produced by hair on the legs of some insects. Each sensor is read once every one seventeenth of a second and has a range of about 15cm. The sensor covers the entire surface of the arm including the arm endpoint and joints, and uses the amount of reflected light off the obstacle for the proximity reading. Using the skin together with the sensor data processing and motion planning algorithms, the arm is able to generate purposeful motion while avoiding obstacles in its environment.

Once the sensor readings are gathered from the sensitive skin it must be processed in some intelligent fashion. First, assuming the global direction of motion is given, how do we move the arm locally so that in the next moment it does not collide with nearby obstacles that may be distributed along several locations? This task is handled by the *Step Planning* algorithm, which decides on incremental motions of the arm based on current sensor data, and commands from a higher level algorithm. These incremental motions are designed to guarantee collision free motion of the arm, and cause the arm to slide on the surface of the obstacle or to move in free space.

Once incremental motions of the robot arm can be commanded, in what manner

should the global decisions be made that guide the arm to the desired target position specified by the system user? This task is handled by the *Motion Planning* algorithm, which can take different forms depending on the specific application of the motion planning system. The three variations described in this thesis are

1) Repeller mode: the sensitive skin detects the obstacle(s) and moves the robot arm away from it.

2) Teleoperation mode: input from a human operator - via, for example, a joystick - controls the robot arm motion unless there is a nearby obstacle, in which case the robot arm is commanded to automatically slide along the obstacle, preventing collision.

3) Automatic Motion Planning: the arm is automatically navigated to a desired target position if a path exists, or the algorithm concludes in finite time that the target is not reachable if this is the case.

This system has been developed and implemented at the Yale Robotics Laboratory using a sensitive skin that detects the presence of nearby obstacles, and data processing and motion planning algorithms to command safe and useful motions of the arm. The system allows the robot arm to do real-time motion planning for the entire body of the manipulator, enabling it be used in an environment with unknown or moving obstacles.

1.1 Research Objectives

Our purpose is to design, build, and test a complete motion planning system that would allow a robot arm to move autonomously in unknown environments with obstacles, while still guaranteeing collision-free operation for every point of the robot body. Input to the motion planning system from the system user consists of the desired target position for the robot arm. The task of the motion planning system is to guide the robot arm to the specified target position while avoiding obstacles in the robot arm's environment. In operation, if an obstacle is encountered during its motion towards the specified position,

the robot arm maneuvers around it, without ever making physical contact.

To make such a motion planning system feasible, the following three basic components are to be developed:

1) A physical ability to sense the presence of an obstacle in front of every point of the arm body and to identify such points in the robot reference system regardless of its characteristics. These characteristics, in the order of decreasing importance are obstacle location, shape, size, composition, and color. Those characteristics considered to be of lesser importance can be neglected if they are judged too difficult to implement. Related issues will be whether the sensor should be passive or active, what sensing range is needed, what density of sensors to use, and what type of sensing radiation should be employed.

2) Signal and data processing capabilities for processing large amounts of sensor data in real time. A greater amount of processing power will allow a higher arm velocity. Although the highest attainable arm speed is desirable, the resulting motion of the robot arm must be at least comparable in speed to those common in industrial assembly and space applications.

3) Complementary algorithms for local (low level) and global (high level) planning, to guide the arm maneuvering around obstacles based on the local sensory data and in accordance with the global planning strategy. The required arm motion must be reasonably smooth, and comparable to that produced by a human operator. For flexibility, the motion planning algorithm must be able to accommodate various modes such as teleoperation and automatic motion planning.

Finally, these three components have to be integrated in a common structure. The integration process is vital: for example, depending on the strategy for local planning, more strict or less strict requirements may be imposed on the selection and calibration of sensors and on the strategy for sensor data processing.

1.2 Contributions of this thesis

The contributions in this thesis pertain to the construction of a sensor-based system for motion planning for robot arm manipulators, and are as follows:

1. A sensitive skin sensor has been designed and built based on infrared proximity sensors. The skin completely covers the robot arm and allows detection of single or multiple obstacles in the vicinity of any point of the robot body.

2. A sensor data processing algorithm has been designed that produces incremental arm motions based on current sensor data and commands from the Motion Planning algorithm so as to guarantee protection form collisions.

3. A Motion Planning algorithm has been designed to make global decisions to guide the robot arm to a desired position.

4. A complete integrated motion planning system that includes the above components has been built and tested.

The realization of such a system encompasses a variety of hardware, data processing, and algorithmic issues that, to our knowledge, have not been addressed before. The developed system is the only existing one that allows a robot arm manipulator to do real-time navigation in a cluttered environment filled with unknown obstacles, while guaranteeing collision-free operation for every point of the robot body.

Each of the mentioned components and their integration into the overall motion planning system are discussed in general in the sections below and in detail in the following chapters. The general information flow diagram of the implemented motion planning system, depicted in Figure 1-1, shows the interaction and hierarchy of the various units in the system; in the diagram, the software and hardware items are shown by rectangles with the rounded and square corners respectively.

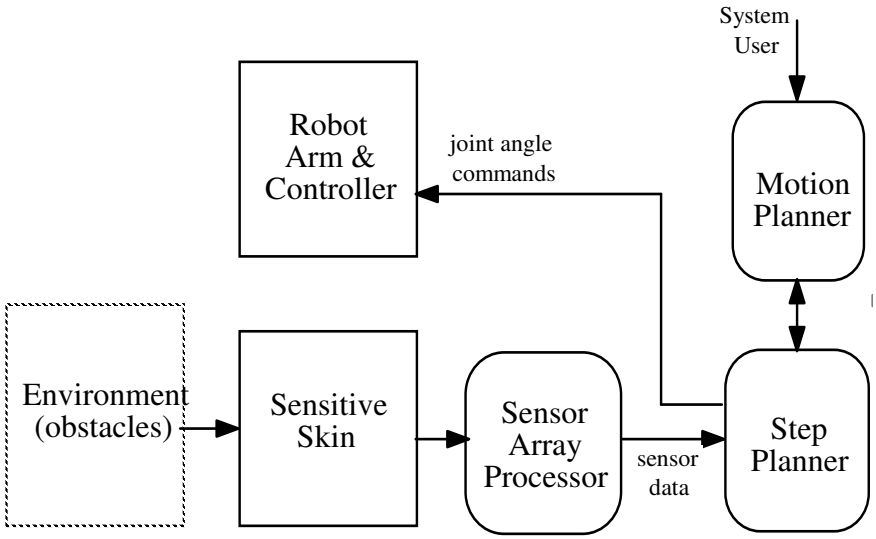


Figure 1-1. Information flow diagram of the arm motion planning system.

1.2.1 Sensitive skin sensor

The individual active optical proximity sensors that cover the skin are organized in the form of a grid on the surface of the arm. The skin covers all the areas of the arm that might in principle come in contact with obstacles. This includes practically the whole arm body, including the endpoint and joints. No actual contact with objects in the environment takes place during the operation, as the sensors use reflected light to sense obstacles.

The light emitted by a sensor is amplitude modulated. If reflected back by an obstacle, it is then synchronously detected by the receiver part of the sensor, to increase immunity to ambient light and to allow operation on multiple frequencies. The output of the sensor is an analog signal proportional to the amount of light reflected off the obstacle. By using two frequencies and parallelizing the polling of the arm, the total time necessary to read all the sensors is reduced. Each sensor is read seventeen times a second by the Sensor Array Processor, see Figure 1-1; the sensing distance of each sensor is up to 15 cm. In the current implementation, 475 sensors blanket the robot arm. The motion planning system operates the arm major linkage which combines the first three arm joints and is responsible for bringing the wrist in the desired location in the workspace. The remaining wrist joints are not controlled directly, although the collision-free motion of the wrist is

still guaranteed.

The sensor pairs are mounted on a Kapton based flexible circuit board that has been processed using standard circuit board development techniques. The circuit board provides both structural support and electrical interconnection for the optical components, and allows the accurate placement of several hundred sensor pairs.

The distribution of infrared sensors on the sensitive skin is such that no approaching obstacle larger than a sphere 3cm in diameter can penetrate the area around the arm within the distance up to 15 cm without being detected by at least one sensor. Given the implemented algorithms for Step Planning (see below), and given the range of velocities, update rates, and accuracies of typical industrial manipulators, this distance is sufficient to guarantee that no collision takes place during the motion. The resulting system is quite robust in that it is insensitive to the variability in characteristics of individual sensors and requires no elaborate sensor selection and calibration procedures.

1.2.2 Step Planning algorithm

The sensitive skin and the robot arm together with the software that control them present the lower level subsystem of the motion planning system. On the next level is the Step Planner, Figure 1-1 - a module that plans incremental steps at every moment which cause the arm to move in free space toward its target position or slide along an obstacle. In the latter case, the Step Planning algorithm must guarantee that, first, no collision with the obstacle takes place, and second, a “contact” with the obstacle is maintained at the end of the step. The term “contact” here refer to the situation when an obstacle is within the sensitivity range of one or more skin sensors. If the arm moves in free space so that no skin sensor detects an obstacle, the consecutive arm positions generated by the Step Planner move the arm directly to the selected target position. What is meant by “directly” depends on the desired trajectory - the one that the arm would go through if it encountered no obstacles on its way. This can be, for example, a straight-line motion of the arm end

effector in the work space, or a straight line in the configuration space, or any other predetermined curve.

Once a skin sensor senses an obstacle, the Step Planner starts generating steps such as to result in the arm safely maneuvering around the obstacle. These steps are taken along the tangent plane to the obstacle, which is found by the Step Planning procedure. To generate the local normal of the tangent plane at the point of contact with an obstacle, the Step Planner takes into account the location of the point of contact on the robot body.

Safe motion is guaranteed at every step of the arm; this is made possible by the sensing range of the sensitive skin. At every step an envelope of certain thickness (as much as 15cm) is certified to be free of obstacles. The next step is calculated such that after its completion, all points of the arm body are still inside the safe envelope. This process then repeats at the new position of the arm.

If more than one point of the arm body are simultaneously in contact with possibly more than one obstacle, all such contacts are evaluated by generating a local tangent plane at each contact. Then, one local tangent plane is chosen such as to guarantee safe motion, based on the general direction for sliding along obstacles and the conditions for leaving the obstacle and proceeding toward the target position that come from the Motion Planner subsystem, Figure 1-1.

1.2.3 Motion Planner

Once the experimental apparatus and the Step Planning algorithm are in place, the last element to be added on the highest level of the motion planning system are algorithms and software for transforming the processed sensor data into global planning decisions that guide the arm's motion. The developed motion planner commands the arm to move in free space towards a selected target position, or if an obstacle is encountered, causes the arm to maneuver around obstacles by sliding along their surfaces. The exact nature of the motion around obstacles depends on the level of automatic control desired of the Motion Planning

algorithm. On the lower level of automatic control, the robot arm can be commanded to move in the direction of the weighted sum of the local normals, which causes the arm to move away from obstacles. This "Repeller mode" can be used to avoid collisions, to "push" the arm into a desired position, and can also be used to test the sensor hardware, local normal generation routines, robot interface, and robot arm motors.

Recently, there has been much work in the area of teleoperation and telerobotics with robot arm manipulators [4,25,40,50]. The arm is commanded to move in real-time by a human operator who is viewing the progress of the operation. Traditionally, both the task of accomplishing the goal and avoiding obstacles that can obstruct the arm are performed in such systems by the human operator. With the help of the sensitive skin the second task - obstacle avoidance - can be performed more automatically. With the Teleoperation mode, motion commands can be passed directly from the human operator to the robot arm, unless obstacles are obstructing the arm. In the latter case, motion of the arm is modified by automatically sliding the arm along the surface of the obstacle to prevent collision.

On the higher level of automatic control, a fully automatic motion planning algorithm is necessary. In this case, the user supplies only the desired target position, and the Automatic Motion Planning algorithm finds a path to the target if one exists. The algorithm produces motion that takes place either in free space, or along obstacle surfaces, similar to other algorithms for motion planning [35,36,37,38,39,58]. Specifically, when maneuvering around the obstacles, the algorithm requires the arm to slide along the intersection of the obstacle and a previously chosen plane, or along the intersection of two obstacles. This corresponds to the arm motion along one or more local tangent plane to the obstacle at the point of contact.

The Automatic Motion Planning algorithm is divided into three phases, each performing a more complex search for a path to the desired target position. The algorithm is based on topology of the configuration space, which is a function of the arm kinematics rather than of obstacle geometry. Consequently, the geometry, number, size or placement

of obstacles that the arm is interacting with at any given moment are not essential for the system operation. The algorithm will find a path to the target position if this exists, or conclude in finite time that there are none if this is the case.

In terms of memory requirements for the Automatic Motion Planning algorithm, as an example, while moving in a rather crowded environment with 4 to 5 obstacles with curved and planar concave and convex surfaces, the robot will need memory to store about 1000 points, each point being a 3D vector. This estimate is quite conservative and thus quite reasonable. As to the time constraints, note that the real-time control is only a function of local interaction with obstacles, and so it does not depend on the complexity of the environment. In other words, a capability to produce fast smooth motion translates into requirements to sensor range and reaction time, raw sensor data processing capabilities, and local normal calculation time, rather than into any constraints on the Motion Planning algorithm. Overall, the speed of arm motion in our system is that of typical industrial arm manipulators.

1.3 Allied Literature

1.3.1 Optical proximity sensors

The ideal sensitive skin should detect all obstacles approaching the robot arm surface, and should be insensitive to the obstacle's shape, size, placement, composition, and color. Of a variety of sensing devices that could be used for the motion planning system, each has its strengths and weaknesses with respect to robust detection of obstacles. They include active and passive sensors, tactile proximity sensors [1,26,47,48], acoustical [17,20,29], inductive [3], capacitance [3] and optical. In the latter category there are several ways to extract the obstacle information, including triangulation [24], time-of-flight [15], amplitude [5,16,44,65], multiple reflection paths [56,57], and extended Kalman filtering [18]. The above sensor types will be reviewed more thoroughly in Section 2.4.1, together with justification of the method chosen for our sensitive skin system.

1.3.2 Step planning

One obvious consideration during the Step Planning procedure is to provide collision-free motion. What is meant by "collision-free" can be defined either probabilistically or deterministically. In the former case, the safety of a contemplated position can be, for example, inferred indirectly, by extrapolating from available information. This is done sometimes in wall following by autonomous vehicles, where the next step is done parallel to what is expected to be the wall, based on the previous steps [19,31,53]. In the application at hand, distances between the arm and obstacles are short, speeds are relatively high, and "wall" shapes are unpredictable. This deems probabilistic techniques undesirable.

In deterministic techniques, the safety of the next position is assured by direct sensing. This can be done in one of two ways. One strategy is similar to that of a blind person with a cane, who first tests all candidate positions, and only then makes a step. This technique is appropriate for computer simulations [59] but is not very practical for our application. An alternative technique, adopted in this work, capitalizes on the information provided by proximity sensing - that an envelope of certain thickness around the arm is free of obstacles. Based on this information, the next step is calculated so that after accomplishing it all points of the arm body are still inside the same safe envelope. Then, a new safety envelope appears, and the process repeats.

1.3.3 Motion planning

As mentioned in the introductory section, current research in robotic motion planning encompasses two major trends. In one approach complete information about the robot and its environment is assumed. A priori knowledge about the obstacles is represented by an algebraic description, such as a polyhedral representation. Motion planning occurs by off-line calculation of a collision-free path in the environment.

Advantages of this approach include its capability to handle, in principle, the most general cases and to deliver optimal solutions. Its drawbacks are the lengthy off-line computations necessary to implement the search for a path, the necessity to have the complete description of the environment, and the inability to deal with feedback and uncertainty in the model of the environment. An overview of research in this area can be found in [51,52,66].

Another approach, considered in this work, assumes incomplete information about the environment. Such a situation takes place, for example, when the robot has no á priori knowledge about the environment, and is equipped with sensors that notify it of impending collision, or proximity to an obstacle. At frequent intervals while navigating through the environment, the robot decides its next move based on the current sensor information. This approach does not have any of the drawbacks of the complete information approach mentioned above. However, since a complete description of the environment is not available, the robot is not guaranteed to find the optimal solution; also, the approach cannot be used for a general multidimensional case. Since the algorithms generated by this approach are well suited for the two dimensional case, much of research in this area has concentrated on roving robots that move using wheeled mechanisms [15,17,19,20,21,30, 31,35,53].

When only two degrees of freedom are used to control the robot motion, as in the above cases, a number of maze searching algorithms can in principle be used to solve the motion planning problem [35]. These algorithms rely on the fact that there are only two directions for the robot to move around an obstacle. This approach can be applied to two degree of freedom robot arms [36,39], by transforming the problem of moving the robot arm into that of moving a point along a certain two dimensional surface. An algorithm of this type had been found successful in the preliminary feasibility study [9,12]. Since only two degrees of freedom were used in that version, only those sides of the arm that could interact with obstacles during planar motion were covered with sensors. It was shown, specifically, that with no previous knowledge about the obstacles and using only sensor

data, the arm was capable of moving to a given target position if this were possible, or conclude in finite time that no path existed if such were the case.

A natural extension to the above work is the application of this sensor-based motion planning approach to three degree-of-freedom spatial robot arms. The difficulty with extending maze searching algorithms to the three-dimensional case is that, in general, there is an infinite number of ways for the arm to maneuver around a 3D obstacle. Previous work on motion planning for 3D robot arms with incomplete information about the environment [38,58] exploit the natural constraints imposed by the kinematics of the arm in order to narrow down the available choices during its navigation. These constraints allow one to infer some global information about obstacles interacting with the arm based on local contact information, and to narrow down the number of choices for motion planning during navigation, while still guaranteeing convergence.

Unfortunately, no such provable planning algorithm is known for the robot arm used in our motion planning system. It turns out, however, that for this arm a planning scheme with proven convergence exists that covers all cases of interaction with obstacles, except in certain cases when obstacles interfere with the third link. In such a worst case the robot may need to carry out a complete search of the surface of the obstacle in order to find a path to the desired target position. The complete planning strategy is described in detail in Section 4.7.4.

1.4 Organization of the thesis

The material below is organized as follows. First, the developed architecture and hardware of the motion planning system are discussed in Chapter 2. The overall relationship of the computers in the system is explained, along with the robot interface scheme and the associated hardware. The chapter also discusses the properties of the ideal sensing skin, the various sensing options, and the chosen sensing method for the constructed sensitive skin sensor. In addition, this chapter describes the proximity

detection method, the design of sensor pairs, a detailed circuit description of the constructed sensor, and its measured performance.

The processing of the output of the sensor hardware by the Step Planner is described in Chapter 3. Also presented here are the procedures for generating the local normals at the contact point for the 2D and the 3D case, the various local normal selection procedures for sliding along obstacles, and the three modes of planning mentioned in Section 1.2.3

Chapter 4 presents the Automatic Motion Planning algorithm. Each of the three phases of the algorithm is treated separately along with explanations of the strategy and scope of the search during the phase.

Chapter 5 describes the software implementing the data processing and motion planning algorithms discussed in the previous chapters. Cycle time information of each parallel process running on the computer system is also presented.

Results of the experiments with the developed system are presented in Chapter 6. Included are plots and photographs documenting experiments with several unknown obstacles.

Finally, the summary of the presented work and an outline of possible future research are discussed in Chapter 7.

Chapter 2

Experimental Apparatus: Computer, Robot Interface and Sensor System

2.1 Introduction

This chapter presents the system architecture and design of the prototype system for sensor-based motion planning of a three-dimensional (3D) robot arm manipulator operating in the work space with unknown obstacles. The sensitive skin sensor, which consists of an array of infrared sensor pairs, covers the surface of the robot arm and enables it to detect the presence of obstacles in its environment.

Assuming that, in general, every point of the arm body is subject to potential collisions, one approach to the sensing problem is to cover the surface of the whole arm with an array of proximity sensors. Obstacles appearing from any direction can then be detected, and appropriate action taken to avoid them. Such a sensor system based on the infrared light sensitive skin, together with sensor data interpretation algorithms, has been developed at the Yale University Robotics Laboratory [11,13]. The prototype system is based on an articulated industrial arm manipulator General Electric Model P5, Figure 2-1. Only the arm major linkage that includes the first three links and joints is controlled by the motion planning system. In addition to the sensor hardware/software, the assembled system includes several microprocessors in a distributed processor arrangement, an IBM-AT computer as the development platform, and a MicroVax workstation as the graphical display device and data logger. Various data handling subroutines implemented in a parallel processing language operate asynchronously from each other to process the data from the sensors and to run the motion planning algorithm. The overall system block diagram is shown in Figure 2-2.

Ideally, the sensitive skin should include the ability to accurately detect the distance

to any obstacle, regardless of surface texture, color and material. It should have a high noise immunity in order to prevent false alarms, and a fast response time to allow the fastest possible motion of the arm or objects in its environment. The entire arm should be covered so that obstacles approaching from all directions can be detected. The density of sensors along the arm should be sufficiently high to provide unambiguous data about the location of obstacles in the arm's path. To be resistant against wear and tear, wires that run across the joints of the arm should tolerate constant flexing. Although it is difficult to build a sensitive skin having all these properties, one can come close enough by careful design of the system.

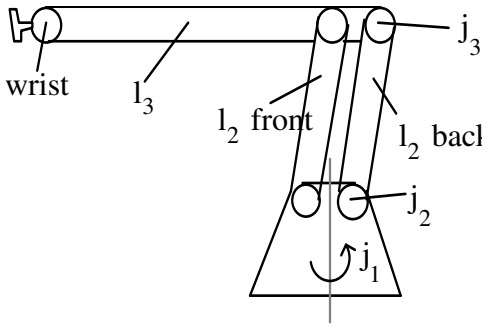


Figure 2-1. Sketch of the robot arm; j_1, j_2, j_3 are arm joints; l_2, l_3 - arm links; link l_1 is of zero length.

The sensitive skin described in detail below uses amplitude modulated infrared light as the sensing medium. The amount of reflected light is used for the proximity indication. More complicated methods such as time-of-flight and triangulation require such complex arrays of interconnecting wires and high speed electronic circuitry that it is doubtful if it is possible to implement this system in the current approach - using a double sided flexible circuit board. A more complete presentation of the various choices of sensor types and previous research using optical sensors can be found below in Section 2.4.1. In the current prototype system, an array of 475 proximity sensors is integrated onto a flexible circuit board which forms the skin surface and is wrapped around the body of the arm.

To eliminate the possibility for interference between links, the frequencies of the light transmitted by the sensors on link l_3 and link l_2 are kept at a constant 2:1 ratio. Using the demodulation system in the sensor processor circuit, the light transmitted by one link's sensors is rejected by the receivers of the other link, allowing parallel sensor operation. In the current implementation, the sensor polling rate is such that the entire skin is polled once every one seventeenth of a second.

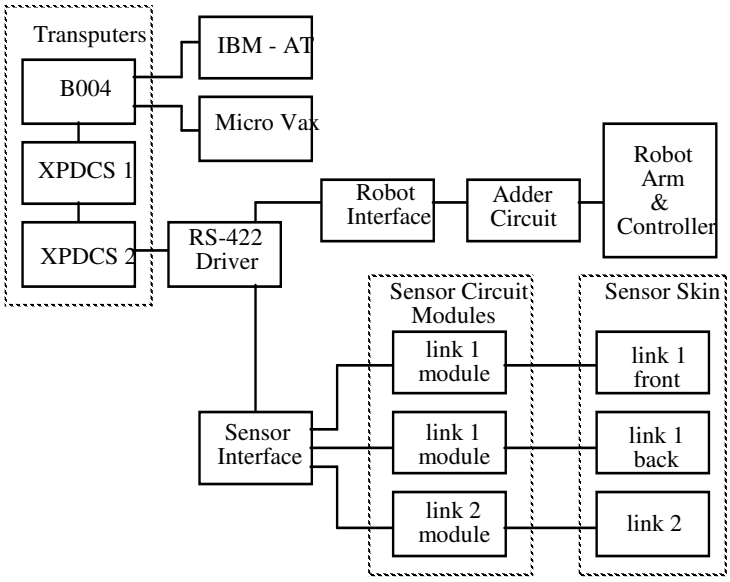


Figure 2-2. Overall system block diagram

The overall system shown in Figure 2-2 can roughly be divided into three major sections: computer hardware, robot arm control system, and the sensor system. Below, the computer hardware is discussed in Section 2.2, the robot arm control system in Section 2.3, and the sensor system - in Section 2.4.

2.2 Computer Hardware

The main source of on-line computational power comes from three transputer boards, labeled XPDCS 1, XPDCS 2, and B004 in Figure 2-2. In addition to the

transputers, an IBM-AT computer is used as the user interface, and a MicroVax workstation is used for real-time monitoring, documenting, and the graphical display.

The transputer board labeled “B004” is a product of the INMOS Corporation, manufacturer of the transputer chips. It contains 2 megabytes of computer memory and a T400 transputer rated at about 100 kiloflops. The development environment runs on this transputer board and communicates with the IBM-AT and the MicroVax. The other two XPDCS boards (XPDCS stands for Transputer Development and Control System) are a product of a Yale Robotics Laboratory affiliate [7]. They each contain 128 kbytes of computer memory and a T818 transputer, rated at about 1.2 megaflops. Each transputer can communicate with four high-speed direct memory access (DMA) serial links that allow the interconnection of transputers in different configurations. The high speed links are also connected to “link adaptors”, also manufactured by INMOS, which convert the serial links into eight bit data buses. Data to and from the sensor and robot control systems are handled by these link adaptors.

The board XPDCS 2 performs the low level sensor and robot interface. Three main subroutines run asynchronously on this board. They handle the sensor polling, robot command refreshing, and information exchange to XPDCS 1 respectively. This latter board asynchronously runs the subroutines that filter the raw sensor data and the sensor data processing (step planning) algorithm. The B004 does the high level motion planning in addition to interface to the development system (IBM-AT) and graphical display station (MicroVax).

The development environment, called Transputer Development System (TDS), is also by INMOS. It includes a text editor, compiler, debugger, as well as other utilities related to managing a network of parallel processors [61]. Programs that run on the transputers are written in OCCAM, a parallel processing language [27].

The MicroVax is connected via a serial connection to the B004 board, and charts the progress of experiments on its display console. A stand-alone software package,

written in C language, takes data from the serial port of the MicroVax and displays the current position of the robot, as well as the history of the experiment. The robot path and its various projections can be viewed from any direction using the utilities available in the software package. The display program also allows hard copy documentation of the experiments.

The programs that run on the various computers implementing the data processing and motion planning algorithms are presented in the following chapters: they are also discussed in more detail in Chapter 5 .

2.3 Robot interface

2.3.1 Interface architecture

The original robot controller in the General Electric P5 robot has no provision for a host computer interface, and programming the arm for a task can only be accomplished by point to point "teaching" of the manipulator using the teach pendant. To implement the sensor-based motion system, an interface between the host computer that houses the planning software and the robot controller has been designed and built.

The connection between the host computer and the robot arm controller is via the 16-bit counters that keep track of the joint angle values of the arm. Signals that increment or decrement these counters are supplied by the incremental encoders mounted on the motor shafts. By inserting a binary adder in the feedback loop of each robot joint, a position command can then be given to the robot arm controller by an outside computer, as shown in Figure 2-3.

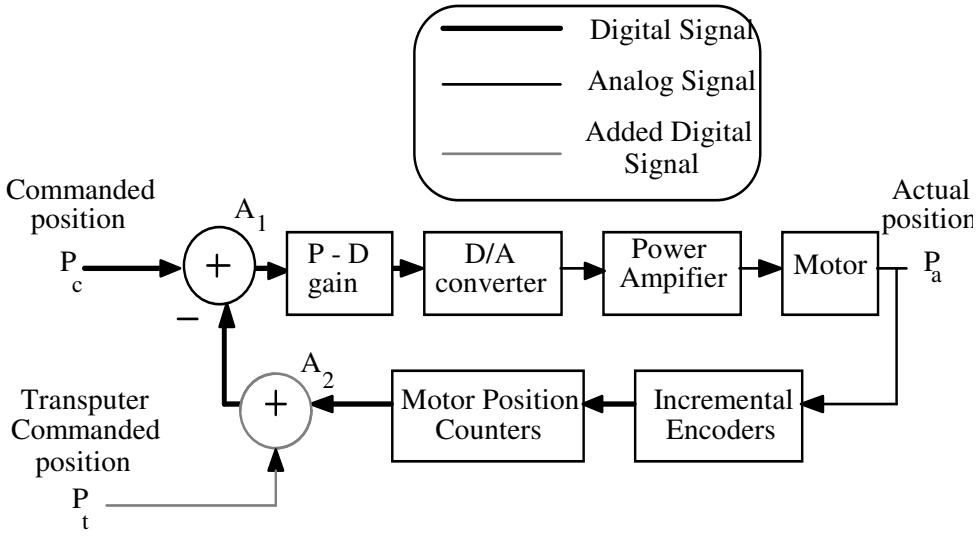


Figure 2-3. Low level motor control loop of each joint. Solid lines relate to the original controller, dotted lines indicate added circuits.

Neglecting a small time delay introduced by the binary adder A_2, Figure 2-3, stability and transient response of the altered system are roughly equal to those of the original system. The total propagation time delay of the introduced electronics is less than 0.05 μsec. This eliminates the need for tuning of the control loop parameters such as the derivative and proportional gains.

Apart from the above advantage of this interface scheme, all of the original hardware in the robot arm controller can be used as before, which simplifies the interface significantly. Its other advantage is a "plug in" implementation, so that the controller can be restored to its original state in a matter of minutes. Furthermore, the added hardware can be made imperceptible using one toggle switch, allowing the controller to be used as if the new electronics were not installed. These features are especially useful at the development stage - e.g. for isolating the sources of malfunctions.

Since only the first three joints (Θ_1, Θ_2, Θ_3) are controlled by the motion planning algorithm, only these control loops are affected by the adder interface. The other arm joints can be controlled by the regular means, such as the teach pendant interface. As far as the transputers are concerned, the position commands are given in an open loop fashion: no

feedback of an actual robot position is returned to the transputer. It is assumed that the original robot control loop will position the joint to the commanded position after some settling time, and that commanded motion velocities, $d\Theta_i/dt$ are relatively low. The adder interface thus effectively transforms the robot into a positioning device. Joint velocities can then be commanded by writing a suitable software driver, which, as our preliminary 2D experiments showed [9,12], are sufficient for our tasks.

To use the interface, the robot is first placed by the robot controller in some known position P_c, see Figure 2-3. When a command P_t is given via the interface, it will cause the CPU to servo the arm to stabilize the control loop, causing the arm to move to the position P_a, where $P_a = P_c - P_t$. If P_c is kept fixed at the origin, $\Theta_1 = \Theta_2 = \Theta_3 = 0$, the arm can be given position commands so that $P_a = - P_t$.

2.3.2 Interface hardware

The purpose of the robot interface is to convert signals from the transputer XPDCS 2 into commands for the Robot Adder Circuit. The overall interface block diagram is shown in Figure 2-4. Commands from the transputers are communicated over twisted wire pairs connected to two link adaptors; these are eight bit bidirectional interface chips by INMOS Corp. One byte (eight bits) received from the transputer is interpreted as the data byte, and the other as the command byte. A sixteen bit Data Bus is generated by demultiplexing the data byte from the link adaptor. Each bit of the control byte, when asserted, triggers an appropriate action in the interface. The functions of each bit are listed in Table 1, and a bit by bit description of the functions of the command byte is given below.

Table 1. Bit functions in the command byte
(bit 0 is the least significant bit)

Bit	Command
0	Link adaptor send
1	Load high byte of Position Data Bus
2	Write teach pendant interface

3	Θ_1 position command
4	Θ_2 position command
5	Θ_3 position command
6	Emergency stop override
7	Not used (spare)

When asserted, bit 0 of the command byte causes the link adaptors to send the sixteen bits present at their collective input lines “Iin” to the transputer, see Figure 2-5. This is done by asserting the IValid input on IC3 and IC4. Although the IValid lines are not used in the system, the input data bus was originally wired to implement sending data from the interface to the transputers.

The next bit, bit 1, of the command byte is used to demultiplex the data byte into the sixteen bit Position Data Bus of the robot interface. When this bit is a “1”, the data byte from the link adaptor is loaded into the octal flip-flop IC7 [22,63]. The output of the octal flip-flop forms the most significant byte of the Position Data Bus. The least significant byte of the Position Data Bus is formed by the data byte itself. Data from the Position Data Bus can be loaded into the teach pendant interface by asserting bit 2 of the interface. This mode of operation is used to control the remaining arm joints if needed.

Bits 3 through 5 cause the data on the Position Data Bus to be loaded into the adders pertaining to joints Θ_1 through Θ_3 respectively, and causes the writing of a position command. The decimal number represented by the data on the Position Data Bus is the two's complement representation of an integer value proportional to the commanded joint angle in degrees. The proportionality constant depends on the encoder pulses per motor revolution and the gearing between the motor shaft and the joint.

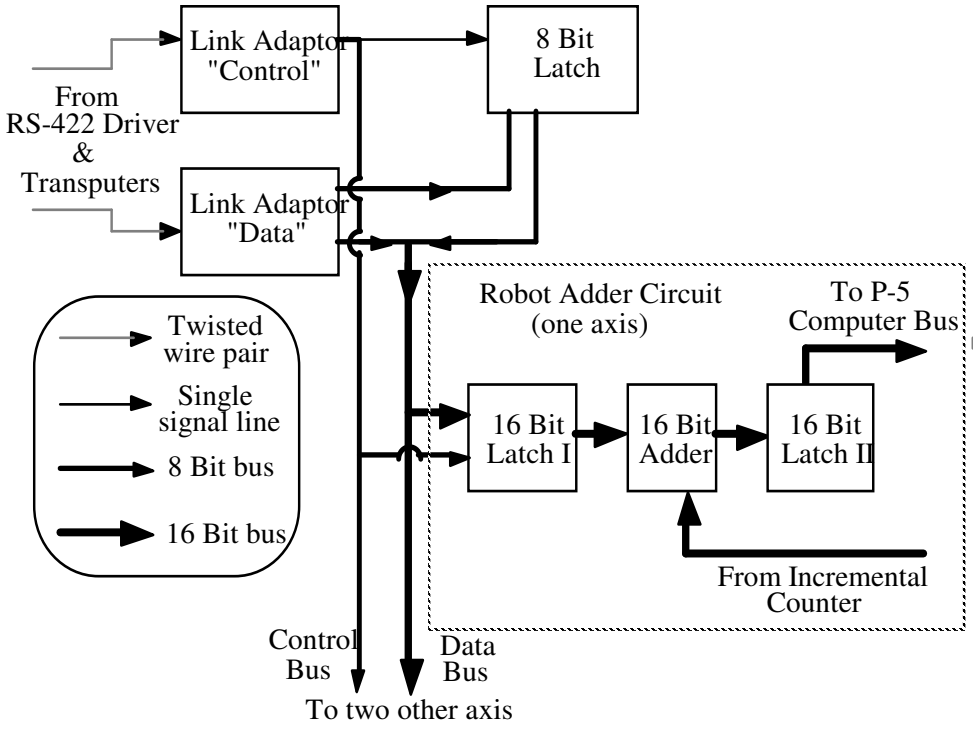


Figure 2-4. Overall robot interface block diagram

The next bit, bit 6, bypasses the automatic emergency stop circuit. When this bit is a “1”, the so-called “watch dog circuit” is disabled. This system will be explained in more detail below.

The circuit that performs the robot interface is shown in Figure 2-5. The circuit is connected to the transputers by a twisted wire pair to increase the noise immunity. The main chips in the interface circuit, link adaptors IC3 and IC4, are connected to IC1 & IC2, which convert the signals from RS422 protocol to single ended TTL level signals. The output of IC3, which forms the data byte, is buffered by IC8, and forms the least significant byte of the Position Data Bus. It is also connected to the input of IC7, which latches the most significant byte of the Position Data Bus when bit 1 of the command byte is asserted. The output of IC4 forms the 8 bit command bus of the interface. Each bit of the command byte controls a particular function as explained above.

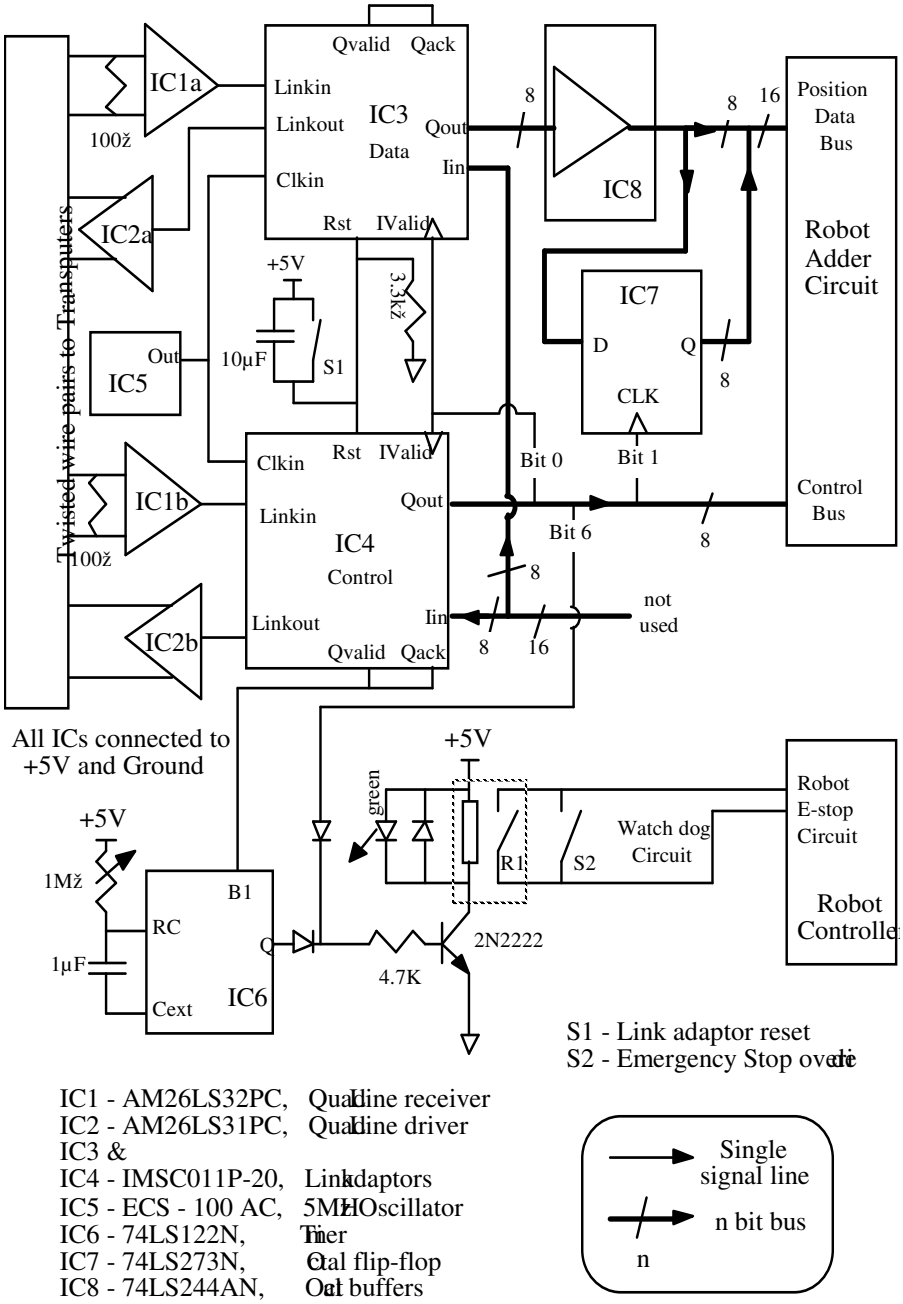


Figure 2-5. Robot interface circuit diagram.

Every 15 msec, the transputer XPDCS 2 writes a position command to the interface; thus all three joints receive a new position command at that interval. At the arrival of a new byte, the link adaptors assert their respective “QValid” signal. Using a suitably

designed circuit, this signal can be utilized for the purpose of monitoring the status of the transputer. This “watch dog” circuit is formed by IC6, a retriggerable multivibrator. Its “Q” output stays high if pulses arrive at its “B1” input that are spaced sufficiently close in time. This minimum spacing is set by the variable resistor connected to its “RC” input. If the “Q” output of IC6 changes state to a low voltage level, relay R1 will interrupt current flow in the robot controller's emergency stop circuit, which causes the servos to switch off, and the brakes of the robot arm to be energized. The arm will stay in this disabled state until the emergency stop circuit is again closed, and a front panel reset switch is pushed. This feature can be bypassed in software by asserting bit 6 of the command byte. In this case, the relay will remain energized even if the transputer connected to the robot interface crashes. In series with R1 and S2 is the so-called “chicken switch” (not shown in the circuit diagram), which can be held in the hand of the robot operator. Actuating this switch interrupts current flow in the emergency stop circuit, and can be used for an emergency stop in the case of a malfunction.

Figure 2-6 shows the circuit diagram of the binary adder part of the robot interface. It converts commands from the Robot Interface Circuit to data which it inserts into the low level robot feedback loops. To “break into” the robot controller we first removed the chips that contain the registers which keeps track of the joint values of the arm. The chips are on the circuit board inside the motor controller called the Counter Board. Then cables were installed in place of the chips, which jumpered all the necessary signals from the Counter Board to the Adder Interface Circuit board. The original registers that were removed from the Counter Board were then placed on the Adder Interface board, with the binary adders inserted in series with the data path. The binary adders used were 4 bit full adders, thus four were used per joint (IC3 through IC6 on Figure 2-6). The registers that were removed from the Counter Board are IC7 through IC8 for the joint shown.

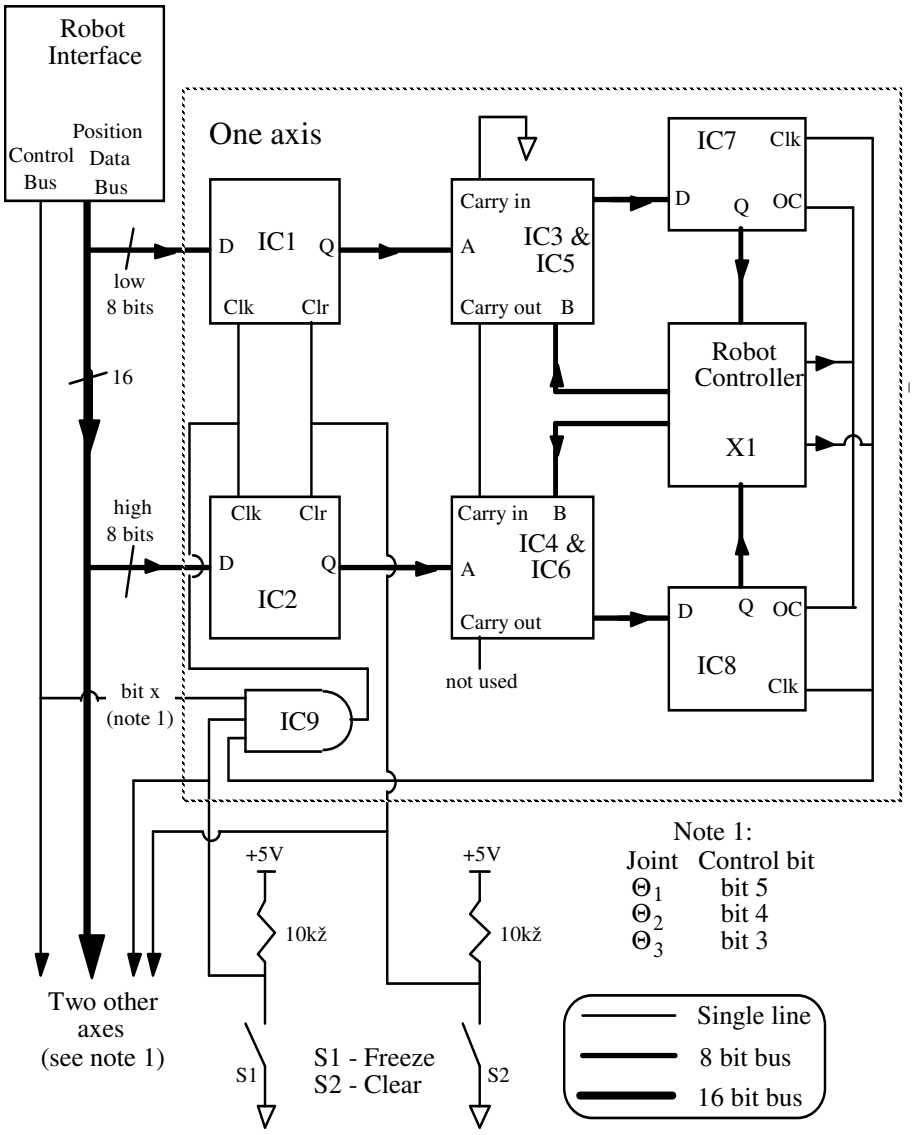


Figure 2-6. Adder circuit diagram.

The following example illustrates how data is loaded into the binary adders and given to the robot controller. The sequence shows the data (in Hex) on the data and command bus of the robot interface during a load of the number 325(Hex) into the adder

for joint Θ_1. This corresponds to a value of about 5^o for Θ_1.

Data bus	Command bus	Operation
03	02	Load "03" into high byte register.
25	10	Load Position Data Bus into Θ_1 adder.

The switches S1 and S2 control overall operation of the interface. The data path from the Position Data Bus can be broken by closing S1, which causes the robot arm to halt at its current position. In addition, the adders can be cleared by closing S2, which functionally restores the robot controller to its original state, without the adders.

2.4 Sensor skin and sensor interface

2.4.1 General

The function of the arm sensors is to provide the motion planning system with two kinds of information - the fact of approaching obstacles and locations on the arm body where these obstructions take place. There are many considerations that can guide selection of an adequate sensing medium. For the purpose of this work, these were divided into two categories -- those that were deemed essential for demonstrating the feasibility of our general approach to motion planning, and those that could be sacrificed at the current stage. For example, a capability to handle various shapes and distributions of obstacles in the workspace was considered to be essential, whereas a capability to handle obstacle materials with a wide range of surface reflectivity - not essential. Below, criteria that guided our design of the sensor system are reviewed in more detail.

Depending on a specific sensor, the area that is being sensed can be large, as when ultrasonic range sensors [17,20,29,43,64] or stereo vision [30,42] is used, or it can be a small local area, such as when proximity sensors are used. Within the latter type, optical sensors are especially popular. Applications include object detection [15,16,24,44,56,57,65], object grasping [5,18] where optical sensors are mounted in the

robot gripper, and tactile sensing [49] where a grid of optical fibers is used for shape detection.

In our case, the system must guarantee that no obstacle approaching the arm remains undetected. Clearly, if any point of the arm body is subject to collision, the only way to guarantee obstacle detection is to supply every point of the body with the sensing capability. One option is to cover the robot arm body with a "sensitive skin", an array of sensors covering the whole arm body. The density of sensors on the skin should be so that no dead zones appear on the arm body; otherwise, if an obstacle approaches the arm in an area of a dead zone, a collision may take place.

What sensing range - specifically, the maximum sensing distance of the sensors - would be adequate? On the one hand, the sensing distance should be limited, to reduce the size and weight of the electronics. Larger range requires more transmitted power and/or more sensitive receivers. This leads to larger components in the transmitter, and more complicated and larger receivers. This in turn will limit the sensor density on the skin and may result in dead zones on the arm body.

On the other hand, the smallest acceptable maximum sensing distance depends on the sensor sampling rate and the maximum linear speed of the arm. The latter depends on the arm specifications and/or intended applications, and is more likely to occur near the arm end effector. Clearly, between two successive scans of all skin sensors no point on the arm body should move more than the maximum sensing distance. This means that the product of the maximum sensing distance and the scan rate must be no less than the maximum linear speed of the arm:

Maximum arm speed < Maximum sensing distance x Sensor scan rate

The factors to be considered are thus the arm speed range, sensor reaction time (which determines the sensor scan rate), and sensor density on the arm skin. The result of

this analysis is that for typical arm manipulators the maximum sensing distance should be about 10-15 cm.

The next choice to be made is between passive sensors, such as tactile or vision, and active sensors that operate by emitting a form of energy and sensing the reflected signal. For the considered application, passive sensors do not seem to present a viable alternative. For example, tactile sensing of obstacles in practice would amount to numerous collisions, whereas covering the arm body with sensors, each of which would provide a vision capability, is not practical[1] . Thus, a viable choice is likely to be among active sensors.

Two major types of active sensing make use of optical or acoustical radiation. In addition, inductive and capacitive methods can be used to accomplish sensing; these methods are generally limited to detection distances of less than 2 cm, and are heavily dependent on the material of the obstacle [3,34]. Commercial inductive and capacitive sensors are mainly used in industrial environments where the applications require durability and involve objects of known composition that have to be sensed at very close range. With acoustic sensing (another popular term - sonar), a burst of ultrasonic energy is transmitted, reflected from an obstacle and then received, making use of the time of flight for distance measurement [64]. A drawback to this type of sensor is that, because the wavelength of sound is relatively long, some obstacles can exhibit specular (mirror like) reflection. Consequently, large flat surfaces may be undetected by the sensor because of the lack of reflected signal. This effect is especially pronounced at shallow angles between the energy beam and the obstacle - a case very common in the considered application. Another drawback is the relatively low speed of sound; commercial ultrasound sensors require as long as 50 msec to acquire a distance reading. The total time needed to poll

[1] Note that if one contemplated a vision system attached to the walls of the workcell, one would need a very large number of “eyes” in order to guarantee that neither the arm itself nor the obstacles would ever occlude any possible contact between the arm links or the arm and an obstacle.

several hundred sensors may be many seconds.

In addition, commonly available acoustical sensors, such as the popular Polaroid sensor, operate poorly when obstacles are placed closer than 15cm from the sensor [43,64], because the same transducer is used as the transmitter and the receiver. This effect can be only partially compensated by the active damping of the transducer [43]. This dead zone necessitates the path planner to restrict the distances of obstacle detection to larger than 15cm from the sensor, which is not realistic in the case of motion planning for industrial arm manipulators. For example, such a system may render the target position unreachable if it can be reached only by passing between two obstacles located at about 30-40cm from each other.

To overcome some of these shortcomings, sources of radiation of shorter wavelength can be used. Among those, sensors based on non-optical electromagnetic radiation were considered less desirable because of their bulkiness: for example, focusing electromagnetic radiation requires the use of focused antennae, such as parabolic dishes. Visible or infrared light seem to present a better choice: the wavelength of visible light is in the range of hundreds of nanometers, tending most surfaces to scatter light equally in all directions and thus appear matte [6].

If a sufficiently matte surface is illuminated by a narrow beam of light, the amount of reflected radiation will depend mainly on the distance between the surface and the receiver. Although this does not produce a perfect distance sensor, the insensitivity to obstacle orientation can be successfully exploited in detecting the presence of objects in a wide range of materials, shapes, and orientations. One drawback to using light for sensing is that objects of very dark color may not be sensed. Also, non matte surfaces and optical mirrors may exhibit specular reflection, possibly resulting in a lack of returned radiation.

Based on these considerations, a sensor system using light as the sensing medium was chosen. Given this, the next issue is extracting from the reflected light distance information needed for the operation of the planning system. Of the several techniques

that can be used for this purpose, five that are relevant for the task at hand are reviewed here.

An approach often used in auto focus cameras is to emit a beam of light and use triangulation to determine distance. The advantage of the triangulation method is that the object color has a reduced effect on the measured distance; on the other hand, optical mirrors continue to be difficult to detect. The disadvantage of using an array of optical triangulation sensors is the need to use optical components such as small lenses and lateral effect photo detectors (detectors that sense the position of a received spot of light). Due to the small size of these diodes (several square millimeters in area), assembly of the sensor array elements requires the placement of many lenses to an accuracy of a millimeter [24]. The sensor system may therefore not be robust enough and be susceptible to vibration of the robot arm.

The concept of time-of-flight can also be used with an optical sensor to determine obstacle distance [15]. Similar to acoustic sonar, some of the light emitted from the transmitting element is then received back after having been reflected off an obstacle. The time that it takes for the burst of light to return is proportional to the obstacle distance. Light travels approximately 0.3 m/nsec, therefore very high speed electronic circuitry is needed to detect obstacles in the range of interest (from zero up to about 15 cm). The complexity of building an array of such sensors is the reason that this approach was rejected.

A method presented in [18] utilizes extended Kalman filtering by using known motion in the sensing direction to extract the proximity data from the intensity of the reflected light. The problem with using this method in our case is the need for a learning and calibration stage for each sensor, requiring known motion of the sensor in the sensing direction. This may not be easy for sensors covering a robot arm with irregularly shaped links, or when the sensing direction is not perpendicular to the arm link. Furthermore, objects of varying size cannot be handled since the method calls for objects of standard

shape and size. A large increase in system complexity to incorporate this method is therefore still insufficient to insure immunity to variations in the obstacles encountered.

The sensor presented in [56,57] uses multiple light reflectance paths to reduce the effect of obstacle surface reflectivity and tilt. By mounting one light transmitter in the center of their sensor, and two concentric rings of receivers around it, the sensor processing computer derives distance to the obstacle that is relatively insensitive to obstacle reflectance. This method, if used in a sensitive skin, requires such a complex array of interconnecting wires on the skin that it is doubtful if it is possible to implement this system in the current approach - using a double-sided flexible circuit board. Unfortunately, this method also can not handle obstacles of varying size and shape.

A more simple way of using the reflected light, which was used in our system, is to use the amplitude of the reflected light for proximity measurement, similar to [5,16,65]. This form of sensor is readily available commercially, although its output is typically digital, going 'high' if the reflected light exceeds a certain threshold that can be set by the user [3,34,54]. Analog output sensors are also available, although they are in general too bulky to be incorporated into a sensor array that is to cover the surface of an arm. The sensor that was eventually built in our laboratory uses the reflected light for proximity measurement, and is specifically designed to fit the surface of a robot arm; its more detailed description is given in the next section.

The transmitted infrared (IR) light used in the sensor system has a wavelength of approximately 880 nm. This light is modulated at a frequency of 135 kHz or 67.5 kHz, depending on which link of the arm it is transmitted from, and is then synchronously detected after reception to improve immunity to other light sources, such as ambient light. The reflected light is then amplified and sent to a digitizer.

In the current design, 475 sensor pairs, each consisting of a PIN diode and an infra red emitting diode (IRED), are mounted on a thin flexible circuit board, which is then wrapped onto the robot arm. To reduce the number of wires connecting the sensitive skin,

the sensor pairs are addressed using a serial protocol. As a result, only six wires are needed to connect the sensitive skin for link l_2 and l_3.

2.4.2 Sensor interface

The purpose of the sensor interface circuit is to allow computer access to the sensor. Its two major components are the analog to digital converter, and the one-shots that control the addressing of a particular sensor. The overall sensor interface block diagram is shown in Figure 2-7, and the circuit that implements the sensor interface is shown in Figures 2-8 and 2-9. Points labeled “A”, “B”, and “C” on Figure 2-8 are connected to the corresponding points in Figure 2-9.

Communication to the sensor system from XPDCS 2 is handled by one link adaptor, which is connected with twisted wire pairs for noise immunity, Figure 2-8. The output byte of the link adaptor at “Qout” is used as the command byte. Similar to the robot interface's command byte, asserting a particular bit of the command byte triggers an action in the sensor interface, which in turn can cause an action in the sensor circuit module. The function of each bit of the command byte is shown in Table 2.

The frequency of the transmitted signal by the sensor skin is generated on the sensor interface circuit board by op-amp IC5a. This signal is distributed to the three sensor processing circuit modules, which are mounted on the surface of the arm, underneath the sensitive skin. The signal is also distributed to other parts of the sensor interface circuit.

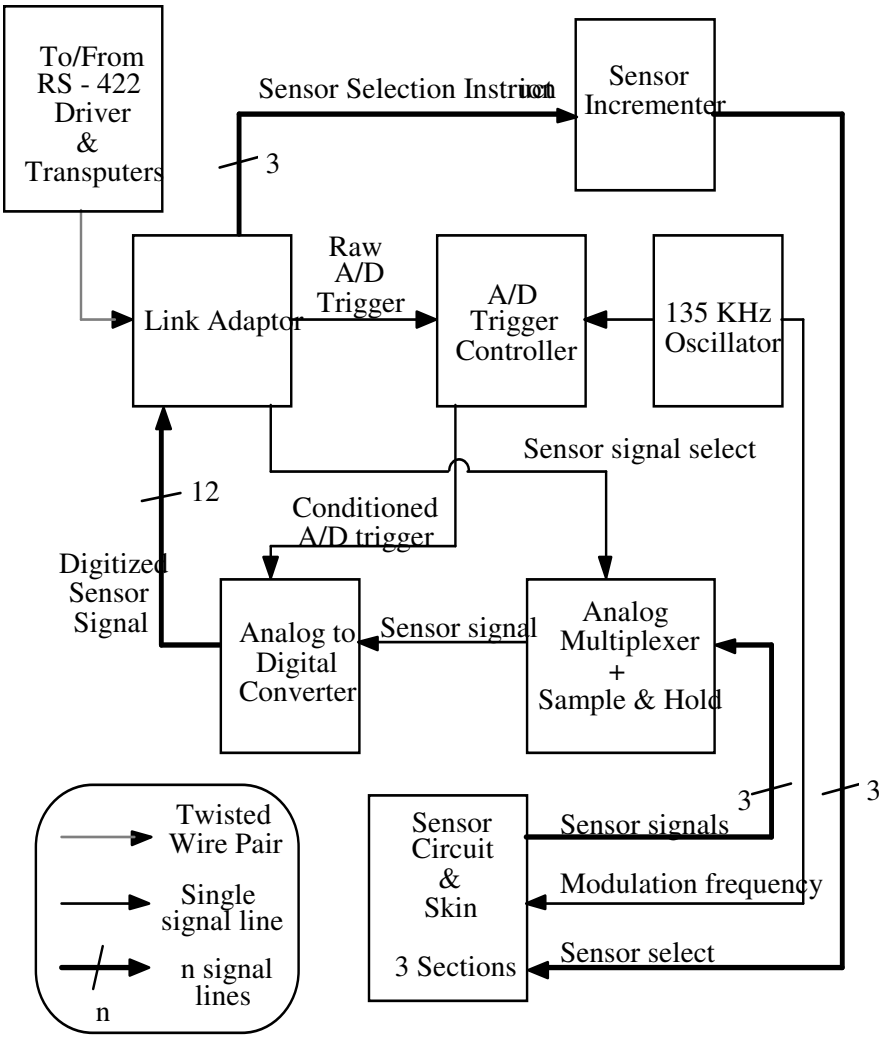
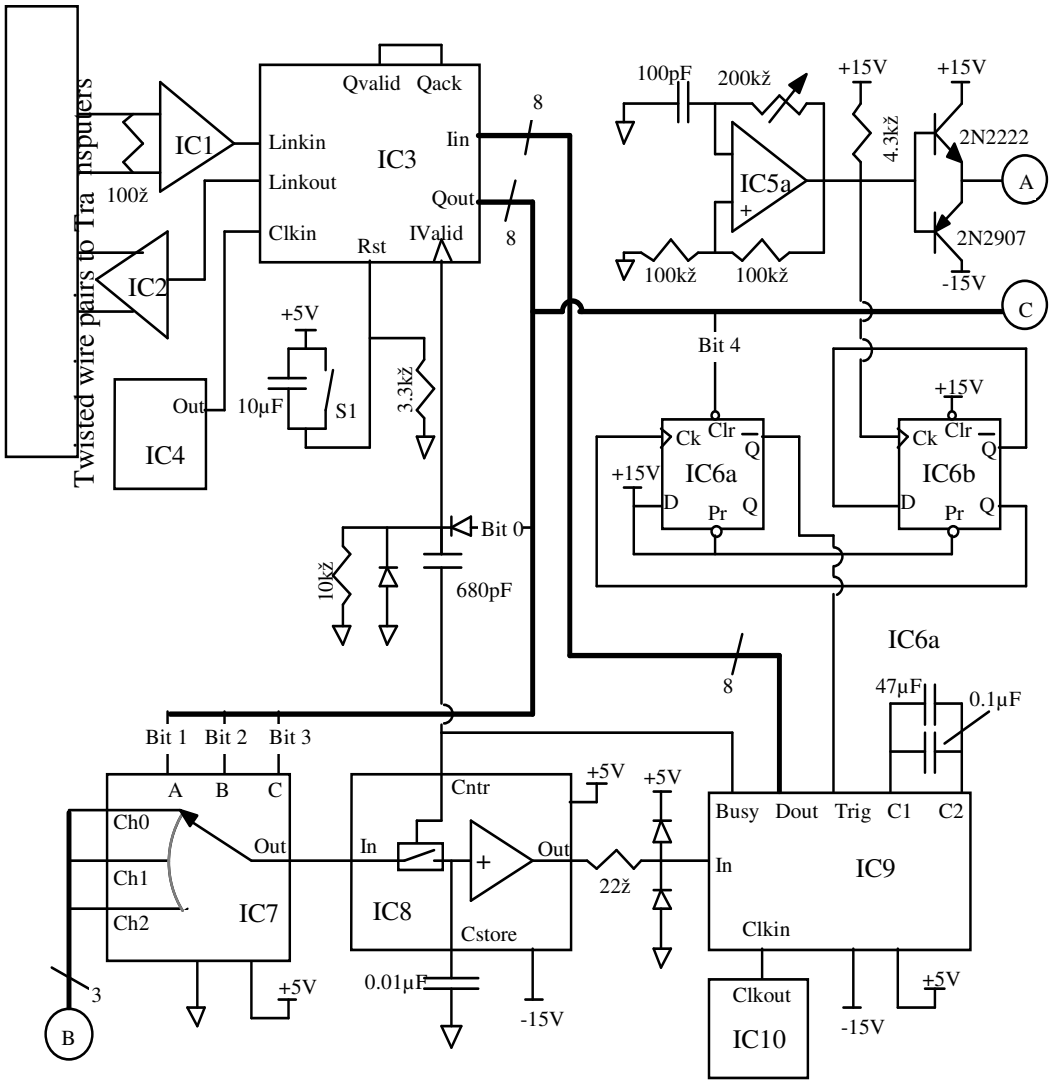


Figure 2-7. Sensor interface block diagram

Table 2. Bit functions of the command byte
(bit 0 is the least significant bit)

Bit	Function
0	Link adaptor send
1	Analog channel select
2	Analog channel select
3	Analog channel select
4	Analog to digital converter trigger
5	Increment l_3 sensor
6	Increment l_2 back sensor
7	Increment l_2 front sensor



Single signal line

n signal lines

S1 - Link adaptor reset

IC1 - AM26LS32PC, Quad line receiver
IC2 - AM26LS31PC, Quad line driver
IC3 - IMSC011P20, Link adaptor
IC4 - ECS 100 AC, 5MHz Square wave oscillator
IC5 - LM339, Quad op - amp
IC6 - 74LS74, Dual D-type flip-flops
IC7 - CD4051, 8 Channel analog mux
IC8 - LM398, Sample and hold
IC9 - MAX162BCN, 12 Bit analog to digital converter
IC10 - ECS 100 AC, 4MHz Square wave oscillator

Figure 2-8. Sensor interface circuit, part 1. Points A, B, C refer to the corresponding points on Figure 2-9.

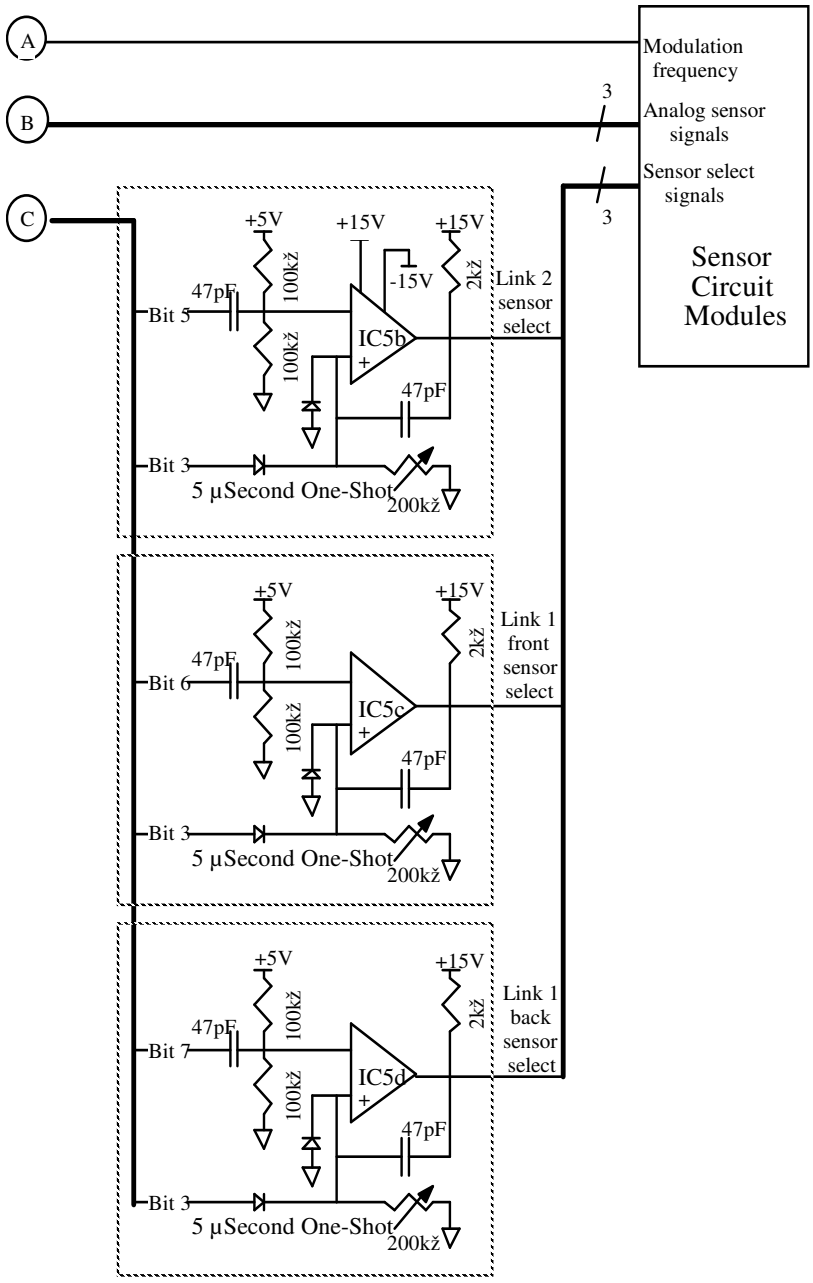


Figure 2-9. Sensor interface circuit, part 2.

The analog to digital converter (ADC) converts the analog signal at its input “In” to a digital number available at the “Dout” output, triggered by a high to low transition on its “Trig” input. During the conversion, which takes about 5 µsec, the “Busy” output remains low. This signal controls the sample and hold (IC8), and the “IValid” signal on

IC3. This latter data path allows IC9 to trigger IC3 directly as soon as the conversion is finished, and eliminates the need for the transputer to constantly poll the interface. Note that IC9 is a 12 bit ADC, the eight least significant bits of the digitized data are sent as soon as the conversion is finished. The most significant four bits are multiplexed on the eight bit data bus via a buffer internal to IC9. This data is placed on the output data bus "Dout" when the "Trig" input is high.

The analog sensor signal that is digitized by the ADC is switched by the analog multiplexer IC7. Bits 1 through 3 of the command byte select which output of the three sensor circuit modules is digitized. The binary number represented by the previously mentioned bits corresponds to the number of the channel that is selected, with bit 1 representing the least significant bit of the channel number. For example, if bit 1=0, bit 2=1, and bit 3=0, then channel "Ch2" is selected.

The triggering of the ADC is initiated by bit 4 of the command byte. In principle, this signal can be connected directly to the "Trig" input of IC9, but the trigger signal is preconditioned by IC6 to reduce the noise level of the digitized signal. The origin of this noise is as follows. The sensor interface circuit is located a few feet from the sensor skin, at the base of the robot. Some of the signal from the 135KHz oscillator formed by IC5a is capacitively coupled to the analog sensor signal lines. In addition, the 135KHz signal, and another signal at 67.5 KHz is used for the modulation and demodulation of signals on the sensor circuit module. The result is that there exists a small amount of the modulation frequencies' signal on the analog sensor signal lines. This noise could have been removed by using a high-pass filter, but it is rejected in a different way. First, IC6b divides the frequency of the 135KHz signal by two to 67.5KHz, this signal appears at the chip's "Q" output. The trigger for the ADC is synchronized by IC6a to a rising edge of the 67.5Khz signal. Thus when bit 4 of the command bus is asserted by the transputer, the ADC is not triggered until the next rising edge of the 67.5 KHz signal. By triggering the ADC at the same point with regard to the phase of the 67.5KHz signal, signals that are integer multiples

of the 67.5KHz signal are filtered from the digitized data.

Referring to Figure 2-10, at point T_a bit 4 of the command byte is brought high, but the ADC is not triggered until at point T_b. When bit 4 returns low at T_c, the trigger signal of IC9 is reset for another sample, which occurs again at points T_d and T_e.

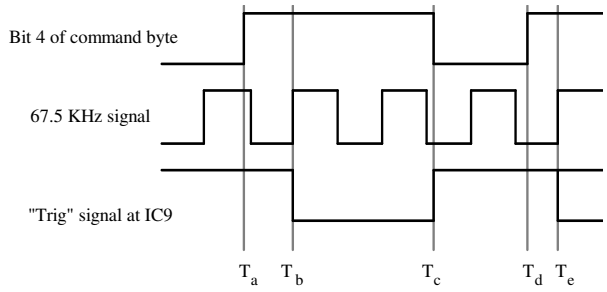


Figure 2-10. Triggering the ADC in phase with the 67.5KHz signal.

A slight drawback of this triggering scheme is that the trigger command will be delayed by up to 14.8 µsec, however this short amount of time can be considered negligible in our application.

The sensors are addressed in a serial fashion by a short pulse on the Sensor Select inputs of the sensor circuit module, and are reset by a long pulse on this same signal line. The short pulse is delivered by the One-Shots formed by IC5b, IC5c, and IC5d. Each sensor section is pulsed by its own One-Shot as indicated on Figure 2-9. The exact pulse length is adjustable by the 200kΩ variable resistor. The long high pulse required for the reset function is provided by bit 3 of the command byte. The Sensor Select signal remains high as long as bit 3 is a "1". By asserting bit 3, the entire skin is reset together, synchronizing the address counters on the three sensor circuit modules.

2.4.3 Sensor circuit modules

A sensor circuit module contains the majority of the electronics necessary to implement the sensing function. There are three such modules in the current version of the

system, each connected to a different part of the skin. The module can be divided into two parts, see Figure 2-11. The first part is the sensor addressing circuit, which decodes which sensor is being addressed currently, the second part is the sensor detection circuit which amplifies and filters the signals from the light detectors.

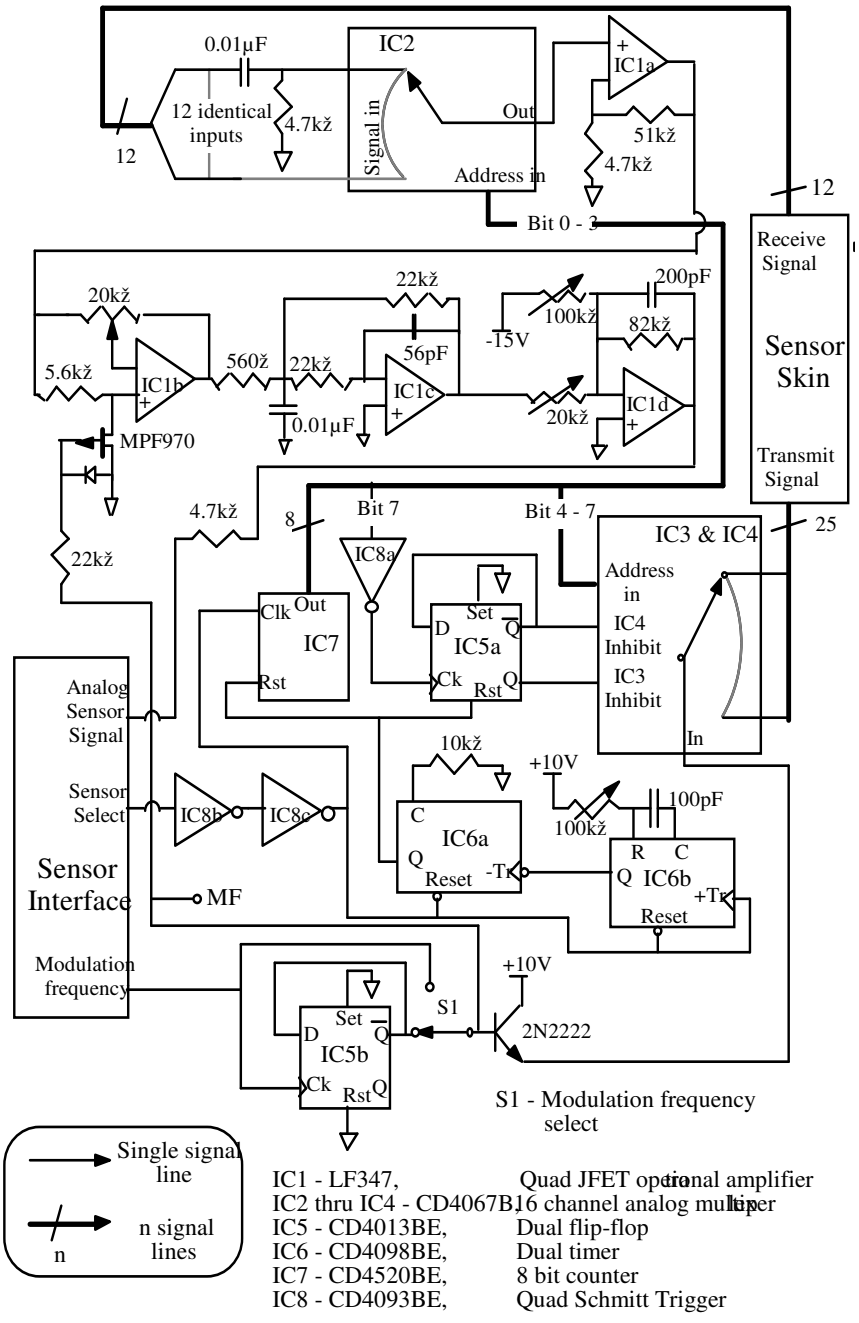


Figure 2-11. Schematic of the Sensor Circuit Module.

To increase the reliability of the sensor system, it is desirable to minimize the amount of interconnecting wires that run across the joints of the robot. There is a conflicting requirement, however - it is also desirable to be able to address each sensor individually, so that the position of obstacles with respect to the arm can be determined. If a parallel addressing scheme is used, similar to how computer memory is accessed, $log_2 n$ lines will be needed, where n is the total number of sensors. By using a serial addressing scheme, the sensors can be addressed using just one signal line.

The addressing scheme is organized as follows. Each sensor circuit module has a counter which keeps track of which sensor is currently being addressed. The counter is incremented on every rising edge (low to high transition) of the serial clock line, which causes a new sensor to be selected. Thus pulses of any length, including short ones, will increment the counter. The counter is reset to zero with a long pulse by using a pulse discriminator on the sensor circuit module. In the implemented system, pulses that are "high" for longer than 10 µsec are considered reset pulses; pulses shorter than 10 µsec increment the counter. This addressing scheme allows just one signal line to address a potentially infinite number of sensors.

An obvious drawback of this scheme is that random addressing is not possible. All sensors with addresses lower than a desired sensor will have to be selected when picking a particular sensor. By keeping the pulses as short as possible, one can quickly address a particular sensor, and minimize the amount of wasted time. Note, however, that in general in such a sensor system there will be a need to poll all sensors periodically. In this case the order of addressing is immaterial, and hence the advantages of the serial addressing scheme outweigh its disadvantages.

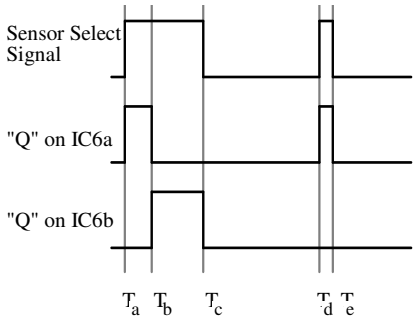


Figure 2-12. Pulse Discriminator wave forms.

Referring to Figure 2-11, the Sensor Select signal from the Sensor Interface is first "cleaned up" by Schmitt triggers IC8b and IC8c, and is then connected to the "Clk" input of the eight bit counter, which keeps track of which sensor is currently being selected. The pulse discriminator, which decides when the counter should be reset is composed of the dual one-shot IC6.

The pulse discriminator operates as follows. While the Sensor Select line is low, the one-shots' outputs "Q" are low, and the eight bit counter is not reset. On the arrival of a pulse on the Sensor Select line at time T_a, one-shot IC6b is triggered by the positive edge, and its output "Q" goes high. If the Sensor Select line stays high longer than 10 µsec, it will allow IC6b to time out, causing its output to drop low at T_b. This triggers IC6a, and its output "Q" goes high, which resets the counter. Thus pulses longer than 10 µsec cause the counter IC7 to be reset. On the other hand, if the Sensor Select signal goes low before IC6b times out, such as at T_d and T_e, Figure 2-12, no reset pulse will be generated, and the counter increments normally.

The light transmitted by the Infra Red Emitting Diode (IRED) is amplitude modulated, and then synchronously detected to increase the immunity to other light sources. It also allows operation on several frequencies, since light transmitted by one sensor can be rejected by another detector which is demodulating at a different frequency. This will be explained in Section 4.3 below.

The output byte of IC7 at its output "Out" controls the analog multiplexers that switch the appropriate optical components to the sensor circuit. The least significant four bits are connected to the analog multiplexer (mux) IC2, which selects among the 12 signals from the preamplifiers on the sensor skin. The signal from the analog switch is first high-pass filtered by IC1a to remove components due to room lighting from the received signal [28]. It is then connected to the synchronous detector IC1b, which demodulates the transmitted signal. Operation of the detector is explained below. After demodulation, the signal is then low-pass filtered by the three pole Butterworth filter composed of IC1c and IC1d. The cut-off frequency of the Butterworth filter is at 10KHz. The output of IC1d is then connected to one of the input channels on the Sensor Interface Board via a 4.7kΩ resistor which provides short circuit protection for IC1d's output stage.

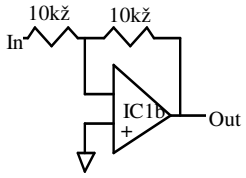
The settling time of the Butterworth filter, which determines overall sensor response time, is approximately 0.25 msec. A higher bandwidth filter would settle in less time, but would also make the circuit more noise prone. Since the two links of the arm are polled in parallel, the limiting factor in the sensor update rate is the polling of the larger of the two sensor skin sections, which is link l_3. In the current implementation, the sensor polling rate is such that the entire skin is polled once every seventeenth of a second.

The switch S1 selects the operating frequency of a particular Sensor Circuit Module. By selecting the output of IC5b, the circuit operates at 67.5 KHz, or one half the frequency of the signal "Modulation frequency", supplied by the Sensor Interface Circuit. The lower frequency (67.5KHz) is used for sensors on link l_2, and the higher frequency (135KHz) is used for sensors on link l_3.

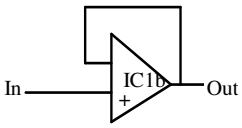
2.4.4 Operation of the demodulator

Instead of using a full-fledged synchronous demodulator, which requires an analog multiplier, a more simple way of demodulating the signal is used. The demodulator IC1b operates by alternately amplifying the signal at the output of IC1a by +1 and -1 in step with

the digital signal at the gate of the MPF970 transistor. This signal is also the same one that is transmitted by the selected IRED, thus the flashing of the IRED is in step with the operation of IC1b. The demodulator operates as follows. Assuming that the 20kΩ variable resistor is at its center, the circuit can be broken into two versions. One version, shown in Figure 2-13a, is the approx. equiv. circuit when the signal "MF" is high; the other, shown in Figure 2-13b is when that signal is low [33]. The gain of the demodulator stage thus alternates between +1 and -1 depending on the logic level of the signal "MF".



(a) MF is logic high, gain is -1



(b) MF is logic low, gain is +1.

Figure 2-13. Approximate circuits for the demodulator depending on the state of signal "MF".

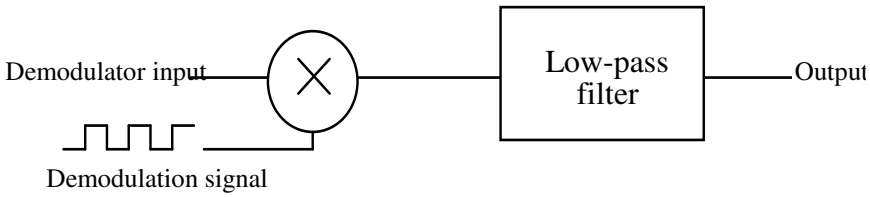


Figure 2-14. Model of the demodulator and low-pass filter.

To analyze the effect of the demodulator on its input signal, we model it as an analog multiplier with one input connected to a square wave of frequency ω_s and its output connected to a low-pass filter.

A true demodulator uses a sine wave as its demodulation signal. By using the described circuit, a square wave becomes the demodulation signal. The effect of the demodulator on its input signal is analyzed in two ways, in the frequency domain and in the time domain.

The operation of the demodulator is analyzed by applying a sine wave of known frequency ω_t to the input of the demodulator. The magnitude of the spectrum of the demodulation signal (square wave) is shown in Figure 2-15.

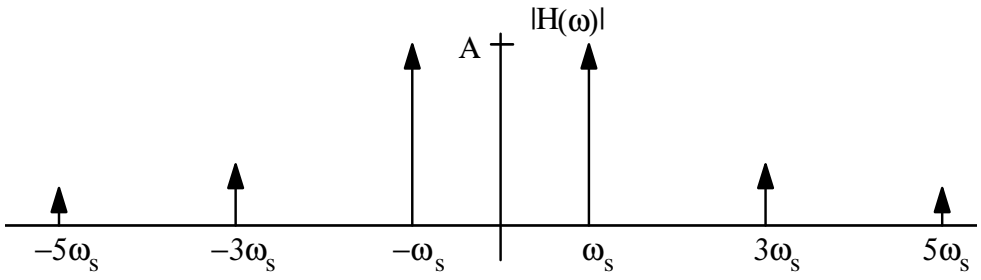


Figure 2-15. Sketch of the magnitude of the spectrum of a square wave at angular frequency ω_s.

The multiplication of the two signals at the input of the multiplier produces signals whose frequencies are the sum and difference of the frequencies of the original signals, and is equivalent to translation of the spectrum of the input signal [32]. Thus the spectrum of the signal at the output of the demodulator is that at the input shifted by $\omega = \pm \omega_s, \pm 3\ \omega_s, \pm 5\ \omega_s$,...., etc. The output of the demodulator is fed to a low-pass filter to remove unwanted components. If a component of the input signal is shifted into the pass band of the low-pass filter, it will be passed unattenuated. This will happen when the input signal has a component at $\omega_t = n\ \omega_s$, where n = 1, 3, 5 .....etc. More formally stated, the frequency response of the switching synchronous detector is the convolution of the spectrum of the square wave and the response of the low-pass filter, Figure 2-16.

To gain a more intuitive understanding of the operation of the demodulator, let us examine its operation in the time domain. In Figure 2-17, the input signals to the analog multiplier are shown on the left, and the output signals of the low-pass filter on the far right. The output signal of the demodulator is filtered by the low-pass filter, 14.

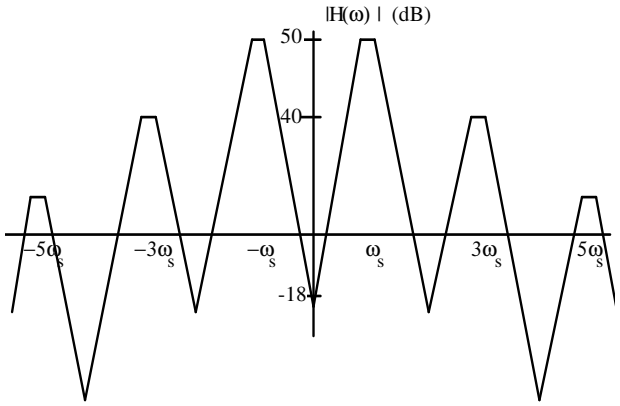


Figure 2-16. Frequency response of the switching synchronous detector; ω_s is the frequency of the demodulating square wave.

Let us consider three examples: assume first that the demodulation signal and Signal 1 are in phase, and the signal swing is between ±1 Volt, Figure 2-17a. The resulting output signal is a constant 1 Volt because the inversion operation of the multiplier is in phase with the polarity of Signal 1.

In the second example, the input signal is Signal 2. The output of the demodulator, shown to the right of Signal 2, is filtered by the low-pass filter, Figure 2-17b. Since this signal has an average (DC) value of 0 Volts, it is blocked by the low-pass filter, and there is no signal at the output of the low-pass filter. Thus signals at twice the frequency of the demodulation frequency are highly rejected by this system. It can similarly be shown that signals at one half the frequency of the demodulation frequency are also highly rejected. This response is the reason why the operating frequency of the sensors of link l_2 and l_3 are at a 2:1 ratio. Light emitted by one link's transmitter does not affect the operation of a receiver on the other link.

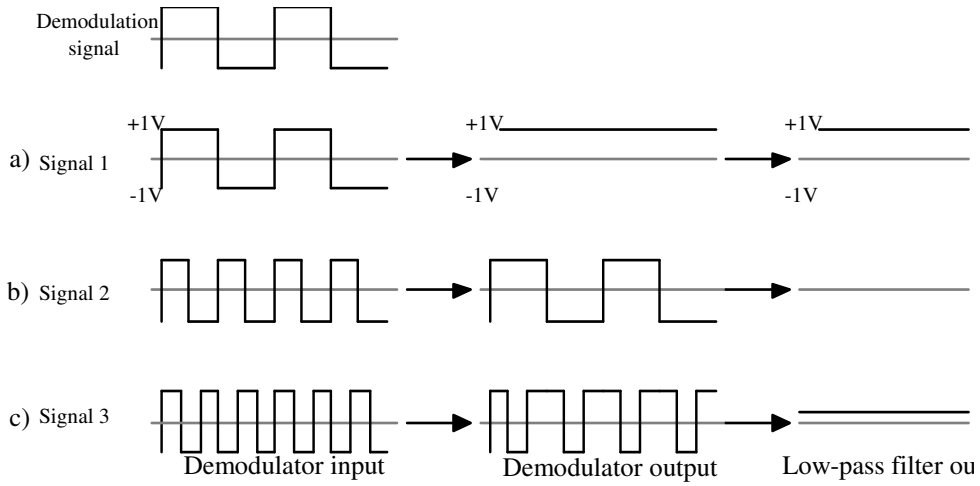


Figure 2-17. Examples of demodulator waveforms.

In the third example, the frequency of Signal 3 at the input of the demodulator is three times the demodulation frequency, and its phase is as shown in Figure 2-17c. In this case, the output of the demodulator has a DC value of 0.3333 Volts. Thus the output of the low-pass filter is 0.3333 Volts. By continuing this example for signals of higher frequencies, it can be shown that signals at odd multiples of the demodulation frequency are passed. More precisely, if:

the voltage swing of the input signal	$= \pm A$ Volts	and
the frequency of the demodulation signal	$= \omega_s$,	and
the frequency of the input signal	$= \omega_t$, where $\omega_t = n\ \omega_s$	
		n = 1,3,5,.....etc.

then the DC value of the output of the low-pass filter $<$ A/n Volts

The inequality in the last expression is due to the uncertainty of the phase between the input signal and the demodulation signal. The results of this time domain analysis agree with those results of the frequency domain analysis. Signals at odd multiples of the demodulation frequencies are passed, while signals at even multiples of the demodulation frequency are rejected.

The drawback of the simpler demodulation circuit used in the sensor system is its sensitivity to signals at the odd multiples of the demodulation signal frequency. It is unlikely, however, that there will be stray optical radiation modulated at such a high frequency in the robot environment. Furthermore, for the incident light to produce a false reading, it must have a strong component in the infrared region since the optical components are molded from a specially colored plastic. In addition, high frequency components are attenuated by the *1/n* factor mentioned in the time domain analysis. Even flashing light sources such as cathode ray tubes operate in a much lower frequency band (<16KHz).

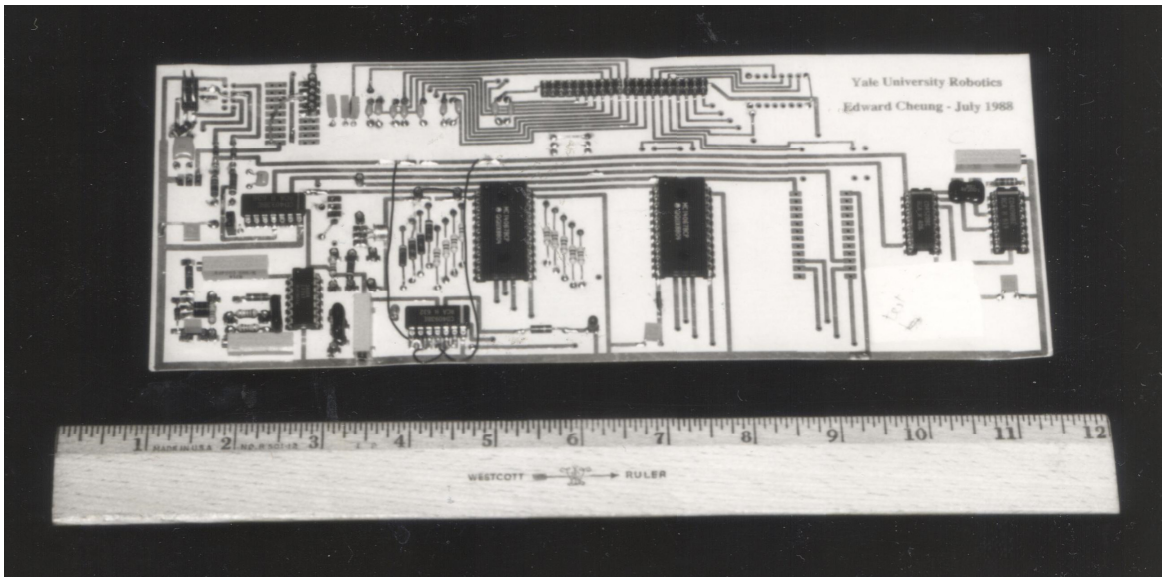


Figure 2-18. Sensor circuit module.

The advantage of our switching synchronous demodulator over a multiplying synchronous demodulator is its simplicity and compactness. By using the switching detector, the need for a pure sine wave and an analog multiplier is eliminated. Since the transmitted signal is a square wave, it can easily be used as the control signal for the switching detector. The detection function can be performed using a single op-amp, and

the need for circuits that can fit in a small space can be important in these applications.

A photograph of the sensor circuit module is shown in Figure 2-18. The electronic components are mounted on a flexible circuit board of the same material as the sensitive skin. The board is then fastened to the arm using Velcro fasteners, and the sensor skin is wrapped over it.

2.4.6 The Sensor Skin

The sensor skin is manufactured of a Kapton based material, only 0.02cm (0.0085") thick. Therefore, the resulting circuit board is sufficiently flexible to allow the board to be fastened onto a curved surface. Both sides of the material are copper cladded, resembling a regular printed circuit board. The material is etched and drilled using standard printed circuit techniques. After processing, the circuit board provides for both structural support and electrical interconnection for the optical components. Availability of this material has drastically reduced the problem of mounting the optical components. It allows the placement, with great precision, of the optical devices with little labor.

There were three major challenges faced when covering the arm with a sensitive skin. The first relates to covering link l_2 of the robot arm. As Figure 2-1 shows, link l_2 of our arm is composed of a trapezoidal closed kinematic chain whose width varies with the change in the joint value Θ_2. Human skin stretches and flexes while maintaining its sensing function, an ability that is difficult to duplicate in electronic devices. To solve this problem, each of the two sections of link l_2 is covered by its own sensor sheet which moves independent of the other. The second challenge was covering the wrist of the arm. Although only the first three degrees of freedom of the arm are directly controlled, to provide collision-free gross motion, we also wish to maintain utility and safety of the wrist. The solution used here was to cover the wrist with a domed section, see Figure 2-21, that has a slot, narrow enough to assure sensing beam overlap at neighboring sensor pairs but sufficiently wide to allow for the wrist motion. The third challenge relates to covering the

joint j_3. To avoid wear and tear of the skin around the joint, the solution chosen was to have the section of sensitive skin covering link l_2 "slide" under the section covering link l_3, see Figure 2-20.

Altogether, three sections of the skin are used to cover the robot arm, two sections for link l_2, and one section for link l_3. To save labor and production costs, a single large "standard" skin section was designed. One copy of the standard section was used to cover link l_3; the two smaller skin sections covering link l_2 are cut out of another copy of the standard section, see Figure 2-21. This design standardization necessitated some small compromises in the overall placement of sensor pairs. Alternatively, the circuitry on the Sensor Circuit Module could probably have been incorporated onto the sensor skin together with the optical components, but it would have necessitated interrupting the regular pattern of sensor pairs.

The overall size of the resulting section covering link l_3 is 61cm x 122cm, while the smaller sections covering link l_2 are 51cm x 51cm. One challenge during the production was the sheer size of the artwork - several companies specializing in printed circuit boards were not able to accommodate such a large pattern. The resolution of the circuit production equipment is also important because lines as narrow as 0.125cm (0.05") have to be reproduced in some skin sections.

Out of the few copies of the standard sheet in the manufacturing batch, three were used for a "dry run": the sensor sheets were cut, drilled, and mounted on the arm without optical components. This allowed final adjustments of the placement of mounting posts and mounting holes and, more importantly, allowed fine-tune planning of the wrist coverage. As mentioned before, the latter includes slits cut into the end of the artwork and joined into a dome shaped enclosure around the wrist. A large opening was then cut into the resulting dome, to allow the wrist motion, see Figure 2-19. This necessitated transplanting several sensor pairs to new locations. The final sensor was constructed from three copies of the standard sheet.

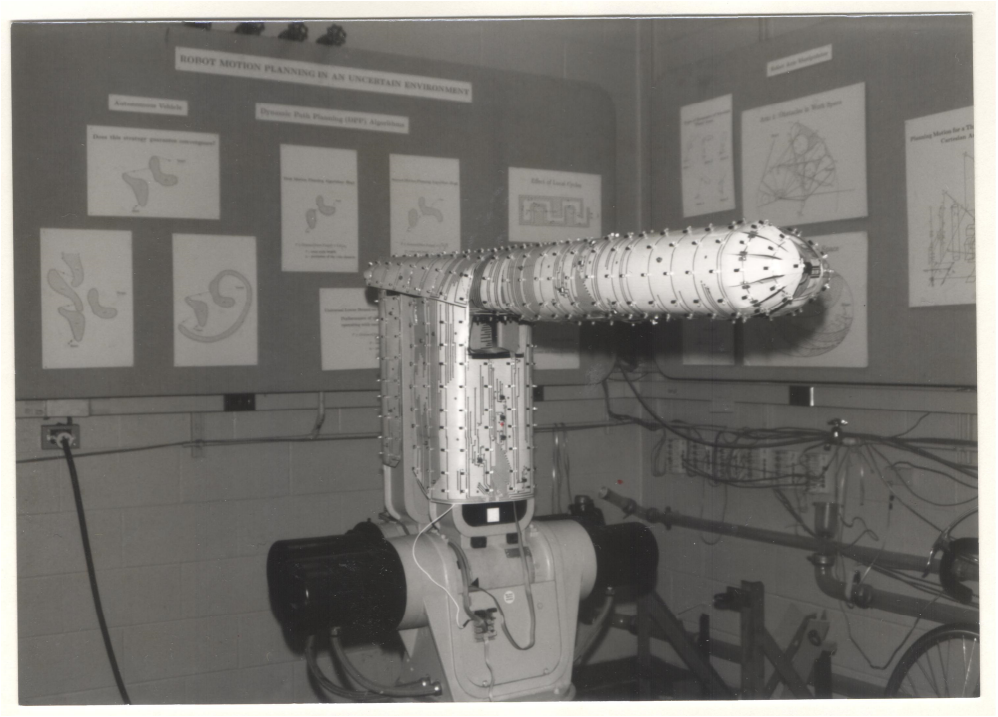


Figure 2-19. Front view of the arm with installed sensitive skin. Note the skin dome covering the robot wrist.

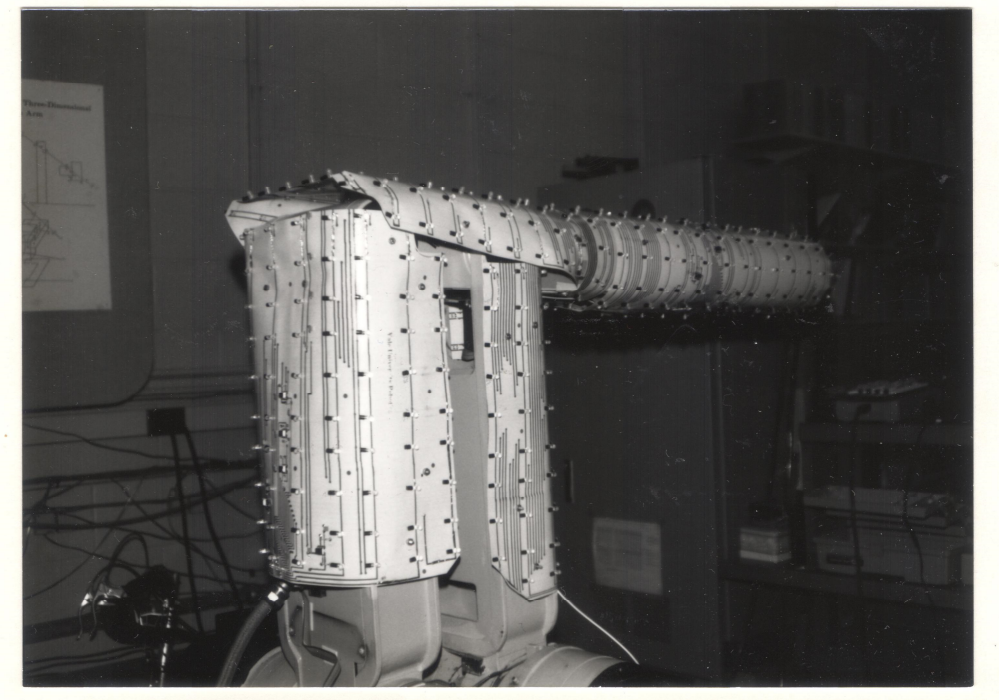


Figure 2-20. Back view of the arm with sensitive skin. Note the sliding section of the skin around the joint j_3.

During the manufacturing of the skin, there was no preselection of optical components for uniform sensor response, nor was this necessary, as tests of the response of the sensitive skin showed. Limitations of the skin's performance was dominated by other factors, which are discussed in Section 2.5.2. The sensitive skin took 6 months to design, manufacture and install.

2.4.7 Sensor skin circuitry

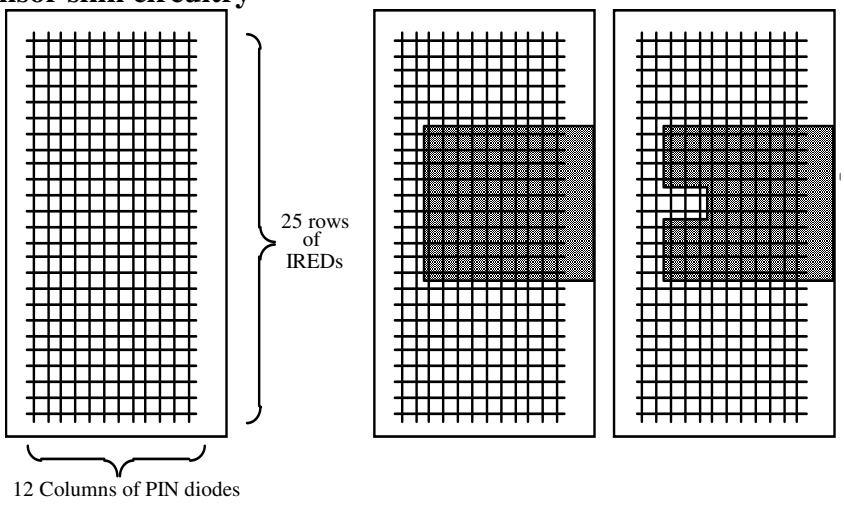


Figure 2-21. Sheets of the sensitive skin. a) The standard sheet as used for link l_3, contains 12 x 25 sensor pairs. b) Piece of the sheet (shaded area) cut for the back part of link l_2; contains 10x10 sensor pairs. c) Piece of the sheet cut for the front part of link l_2; contains (10x10 - 6) sensor pairs.

To reduce the amount of wiring on the arm, the optical components are arranged in an irregularly spaced row - column format. On the (standard) section of sensor skin covering link l_3, there are 25 rows of IREDs, each containing 12 emitting diodes, see Figure 2-21. The PIN diodes are arranged in 12 columns, each containing 25 light detectors. All the PIN diodes in a column are wired in parallel, and then placed in a reverse bias configuration. As a result of this wiring configuration, the output of all the PIN diodes are

summed, and the signal from each of the 12 columns is preamplified by an op-amp operating in transresistance mode. The output of the op-amp is then connected via a connector to the Sensor Circuit Module, Figure 2-22. The IREDs common to a row are wired in series. The resistor R_i limits the pulsing current to a 25 mA peak. This arrangement causes all the IREDs in a row to pulse in unison.

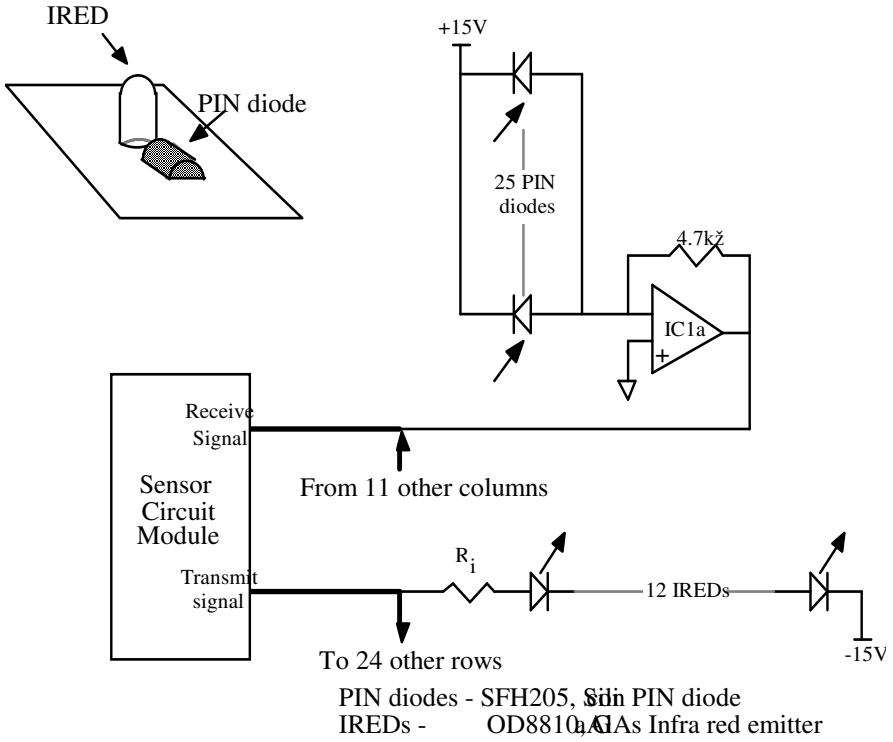


Figure 2-22. Circuit on the sensor skin.

Two major criteria were considered during the selection of optical components: electrical and optical properties. Cost was less of a factor because in general the prices of these components do not vary widely. As far as electrical properties are concerned, the most efficient and powerful devices available were selected. Selecting the optical properties required more thought, however. In order to reduce the effect of obstacle size on the amount of reflected light (and thus sensor reading), it is desirable to reduce the beam

width of the transmitted signal as much as possible. This comes from the fact that at distances at which the sensor operates the emitted light of the IRED illuminates only a spot that is about 5cm in diameter on the obstacle, causing the rest of the obstacle to be invisible to the sensor. A compromise must be used, however: if the beam widths of the emitters are too narrow, small obstacles may be missed if they can fit between two sensors' beams. Ultimately, the limited choices available between the beam widths led to selecting an IRED with a half-power angle of 10^{o} [45].

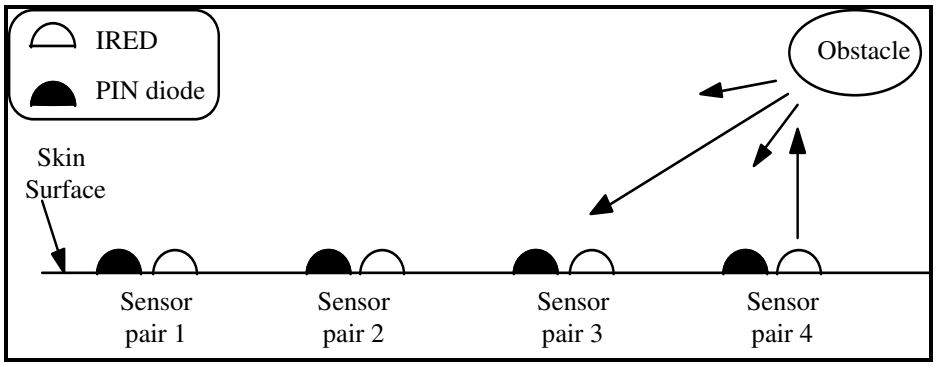


Figure 2-23. Light from the emitter in one sensor pair can be received by the detector in another sensor pair.

As far as the PIN diodes are concerned, to reduce the effect of room lighting, the material of its enclosure should double as an ambient light filter. The last element to select is the kind of lens provided with the PIN diode, which determines its directional sensitivity. It turns out that by a proper choice of the lens the sensor insensitivity to the surface reflectance property of the obstacle can be improved. The wavelength of infrared light is in the range of 700-900 nanometers, scattering light equally in all directions, and tending obstacles to appear matte [6]. Many obstacles, however, do reflect light specularly to some degree, causing light to be reflected away from the transmitter. This problem can be partially solved using the fact that the outputs of all the PIN diodes in one column are summed. Thus, the light beam emitted from one sensor pair can be picked up by a PIN

diode at another location along the arm. This effectively results in a receiver that is physically distributed over the arm. Light reflected to another sensor pair - for example from Sensor pair 4 to Sensor pair 3, in Figure 2-23 - will be hitting the PIN diode at a large angle Φ. For this light to be received, the PIN diode should have a concave lens, or no lens at all. Since the former type is not available, PIN diodes with no lenses were selected.

2.4.8 Mounting the skin on the arm

The sensor skin is mounted on the robot arm using several fasteners distributed over the surface of the arm. A cross-sectional view of link l_3 and the sensitive skin is shown in Figure 2-24. Each fastener presents a bolt attached at an angle of 10^o to the surface of an acrylic foot. The foot is a 2.5cm x 2.5cm piece of acrylic glued to the surface of the arm. The 10^o angle accommodates the curvature of the sensor skin at the mounting point. Two nuts threaded onto the bolt clamp the sensor skin down, and allow continuous adjustment of the cross-sectional shape of the skin. The two skin sections covering link l_2 are fastened using a similar technique.

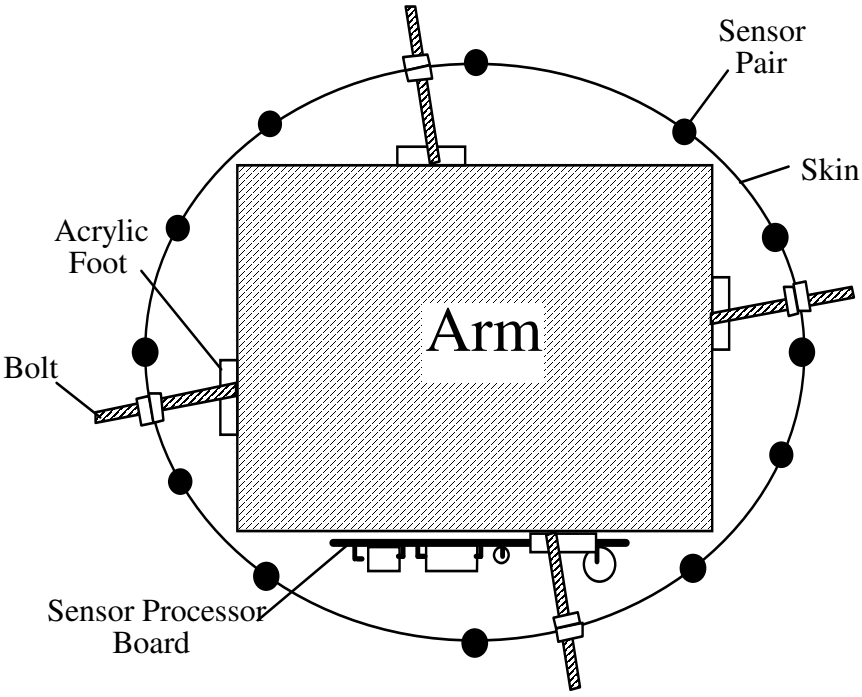


Figure 2-24. Cross-sectional view of the arm with sensitive skin. Note the gradual curve in the skin and the variable sensor spacing - the distance between sensors is smaller at the areas of greater curvature of the skin.

2.5 Performance of the sensor circuit

2.5.1 Individual sensors

The sensor response was assessed experimentally using a randomly selected sensor pair and a 7.5cm x 7.5cm piece of white paper as the test obstacle. The response is measured as a function of the distance D from the skin to the test obstacle, and the angle Φ between the perpendicular to the test obstacle and the sensing axis, see Figure 2-25.

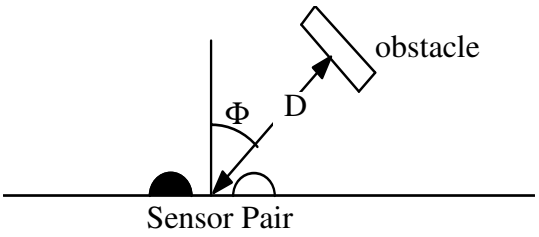


Figure 2-25 Test setup for measurement of sensor response.

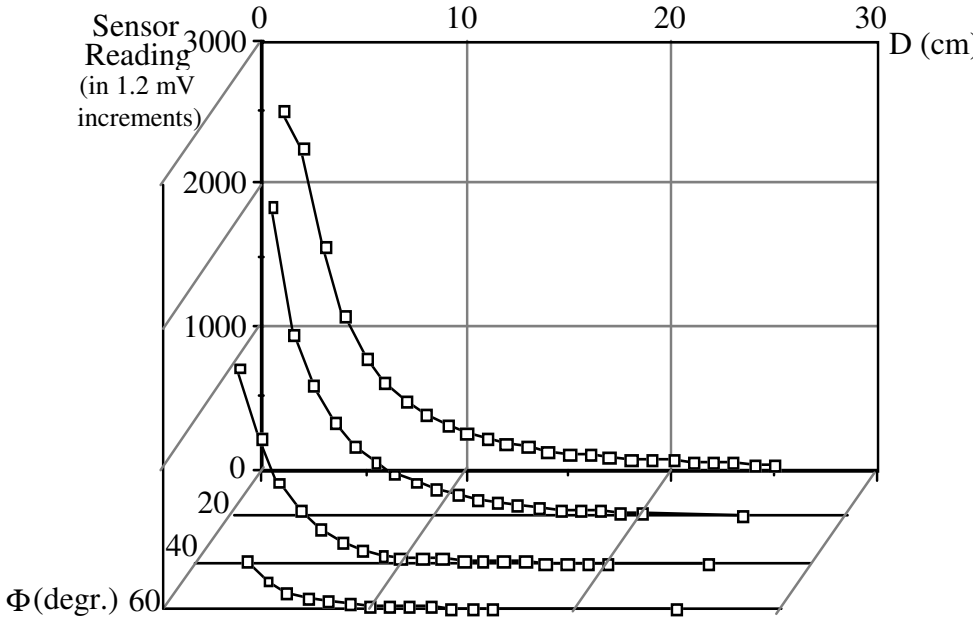


Figure 2-26. Sensor output for various values of the distance D to the obstacle, and for four values of the angle Φ between the obstacle and the sensing axis.

The output of the sensor, where each unit represents 1.2 mVolts (thus 1000 = 1.2 Volt), is shown in Figure 2-26 for four values of the angle Φ, Φ= 0º, 20º, 40º, and 60º. Twenty samples were taken for each position of the obstacle, and the results were then averaged.

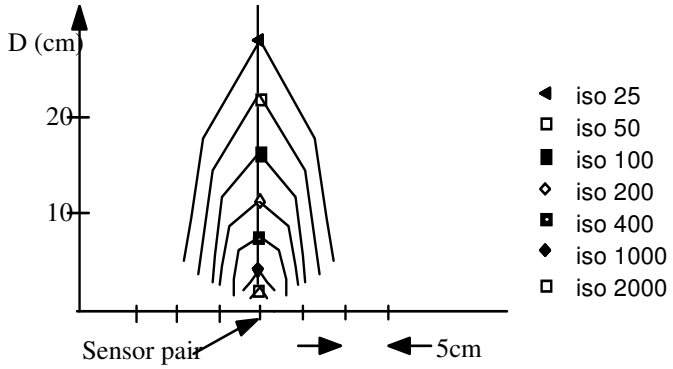


Figure 2-27. Directional response of one sensor pair.
Iso X refers to the line of constant sensor output X.

In Figure 2-27, the sensor data is plotted to show the sensitivity cone for one sensor pair. Lines of constant sensor reading are shown; e.g., "Iso 25" refers to the line of a constant sensor reading of 25. The tick marks on the horizontal axis indicate the locations of sensor pairs. Note that if the skin is placed flat, adjacent sensitivity cones overlap when an obstacle similar in color and size to the test obstacle is being detected. Where the skin is curved to fit the contour of the arm, the sensor spacing is reduced to ensure overlap of the sensitivity cones, see Figure 2-23.

Because of the sufficient density of sensor pairs along the sensitive skin (less than or equal to 5cm between the neighboring sensor pairs), the neighboring sensor beams overlap, and so no obstacle that is at least as detectable in terms of its size and texture as the test obstacle can approach the arm undetected. In fact, if the test obstacle were the smallest possible obstacle in the work space of the arm, and the skin were placed flat, the minimum sensor spacing could be increased from 5cm to 11cm. However, if the sensor

spacing is increased, or if the sensor skin is bent over a sharp corner, the resulting sensitive area around the arm will have a very irregular or “bumpy” shape. This may lead to rough motion while following the contour of sharp obstacles, since the motion planning system servos around a constant sensor reading. Therefore, sharp corners in the skin material should be avoided and corners should be rounded gradually, see Figure 2-23.

One of the main sources of noise in the sensor circuit is the analog multiplexer that switches the PIN diodes in and out of the circuit. Because they contain analog as well as digital circuitry, some of the digital signal is capacitively coupled into the analog signal, causing glitches in the signal at the multiplexer's output. These glitches are amplified together with the weak analog signal, and are not completely removed by the filtering circuits after the multiplexer. The result is that the three pole Butterworth filter at the sensor's output needs more settling time, slowing down the overall sensor. This source of noise is the largest factor limiting performance of the sensor. Currently, the sensor is scanned once every one seventeenth of a second.

In our earlier 2D feasibility study conducted at Philips Laboratories, a different sensor was built that used phototransistors as the light detector. The schematic of this 2D sensor is given in Appendix A. The advantage of this device is that because of its higher built in gain (about a factor of 100 over a PIN diode), its signal amplification circuit needs less gain, and is therefore less susceptible to noise pickup. Its disadvantage is that its low bandwidth (a PIN diode has a bandwidth about 100 times greater) requires a lower modulation frequency, and therefore a lower bandwidth filtering circuit which has a slower settling time. The constructed skin sensor for the 3D implementation allowed a settling time per sensor that is one fourth the time of the 2D sensor, as is summarized below.

3D sensor settling time :	0.22 msec/sensor.
2D sensor settling time :	1.0 msec/sensor.

2.5.2 Skin performance as a whole

It was hoped during the design of the skin that the overall quiescent (no obstacle) sensor reading would be uniform over the entire sensor. This would allow the quiescent output to be electronically nulled, and the gain to be maximized. Unfortunately, this is not the case, and the quiescent level varies widely among the sensors. To avoid saturating the sensor electronics, the electronic gain is kept low, and the overall gain is achieved by a software multiplication. In addition, each sensor is nulled in software during an initial startup phase.

Due to the yellowish color of the sensor skin material, some light of one link's sensor can reflect off the surface of the other link. For some sensors near j_3, see Figure 2-1, especially for those sensors on link l_1 under the skirt covering j_3, there is a dependence of the sensor's output on the value of Θ_3. This effect is canceled by taking quiescent sensor readings at regular intervals of Θ_3 during the zeroing procedure. Data between samples is obtained by linear interpolation of adjacent samples.

The initial sensor zeroing procedure takes control of the robot arm, and moves it through the range of Θ_3 to take samples of the entire sensor skin. The sensor data is sampled for twenty intervals of Θ_3 for each of the 475 sensors. The required storage capacity for this data is about 38 kbytes. This amount of memory can be reduced since only approximately 25% of the sensors exhibit any Θ_3 dependence, but this was not done because storage capacity is not a limitation.

After some initial use of the skin, it was noticed that the sensors need an initial warm-up period. The quiescent sensor output readings drift after startup, and stabilizes after about a half hour. The source of this drift is suspected to be the optical components on the sensor skin.

2.6 Conclusion

This chapter presents the experimental apparatus needed for the motion planning experiments. The assembled computer system consists of three transputers as the main processing units, an IBM-AT as the user interface, and a MicroVax as the data logger and graphical display system.

The original General Electric P5 robot arm controller was modified to allow interface to the transputer control system. To simplify the setup of the robot arm, all of the original electronics in the robot controller were used. The interface scheme transforms the robot arm into an open loop positioning device. Using a suitably written software driver, arm joint velocities can be commanded.

The developed skin consists of an array of sensor pairs, mounted on a Kapton based flexible circuit board, which is then wrapped onto the robot arm. The circuit board provides for both structural support and electrical interconnection for the electronic components. A total amount of about 475 sensors are integrated onto the skin, each with a range of about 15 cm. The entire skin is scanned once every seventeenth of a second.

Empirical investigations show that the constructed sensor performs the task of a sensitive skin, despite the susceptibility of noise due to the weak signal provided by the light receivers. Since the quiescent (no obstacle) sensor output level is non-uniform across the surface of the skin, and also since the sensors of one link can detect the surface of the other link, every sensor is zeroed in software during an initialization phase.

Chapter 3
Sensor Data Processing : Step Planning

3.1 Introduction

The overall Motion Planning system is hierarchical, consisting of the Motion Planning algorithm that determines the overall strategy, and the local Step Planner that is used for sensor data interpretation and for planning the smallest steps that constitute the arm's global motion. The objective of Step Planning is to generate incremental motions that constitute either the arm motion toward the target position in free space along a prespecified trajectory (e.g., a straight line for the end effector), or its maneuvering around an obstacle, while keeping in close proximity to it. The former mode takes place when the skin senses no obstacles, whereas the latter mode takes place when at least one skin sensor senses an obstacle. In principle, the use of the Step Planning method is not limited to a proximity skin sensor; it can also be used with tactile sensor arrays, vision based collision detection systems, or collision detection through a CAD based system where the location of all the obstacles have been entered into a database. In these instances, the Step Planner can then be used to locally provide the high level path planning software with the incremental safe moves at the current position of the robot arm.

Since controlling motion in free space is quite straightforward, we will concentrate now on Step Planning in the vicinity of obstacles. The motion of the robot arm in physical space, or work space, is equivalent to moving a point automaton in the corresponding *configuration space*, whose axes correspond to the arm's degrees of freedom. We present the Step Planning process in terms of the configuration space of those joints that are being controlled. Hereafter, the term automaton refers to the point in the configuration space representing the position of the arm. When the sensor system informs the Motion Planner that the arm encountered an obstacle, the Planner prescribes the arm (the automaton) to move along the intersection curve between the obstacle and a certain plane, or along the

intersection curve of two obstacles. Since the intersection curve itself is unknown before hand, it is the task of the Sensor System to sense it and then of the Step Planner to maneuver the arm along the curve.

The sensors, when they detect an obstacle, provide the robot with information about the location of obstacles in the work space. The Step Planning algorithm converts this information into "objects" such as planes and lines in the arm's configuration space, which determine directions and commands for the next step to be given to the robot drive systems.

In our system, the location of each sensor on the skin is determined relative to the arm body, with a look-up table. In contrast to tactile or haptic systems [1,2,26,47,48,49, 62], our arm does not touch the obstacle, but moves instead along the obstacle's surface, while keeping at some distance from it. Using the sensor data and the Step Planning algorithm described next, surface contour following occurs in a collision free fashion. For each sensor that contacts an obstacle, the conditions necessary to slide about that sensor are determined. Sliding is defined as moving the automaton while the robot arm maintains contact with the obstacle. To slide, the local normal of the tangent plane at the contact point between the robot and the obstacle in configuration space is found (this tangent plane is a tangent line in 2D). Motion in this tangent plane can be considered the safest motion possible locally. The procedure for the generation of the local normal is described in Section 3.2 and 3.3, for the 2D and 3D cases respectively. Special cases of the local normal generation are discussed in Section 3.4.

If more than one sensor are simultaneously in contact with obstacles, more than one tangent plane appear, and a choice must be made as to which of these planes is the safest. New local normals are repeatedly calculated after each step of the robot arm. The local normal selection procedure is described in Section 3.5.

3.2 Local tangent generation for the 2D case

In the 2D case, the motion of the robot arm can be represented by an automaton

that operates in the two-dimensional configuration space (Θ_1, Θ_2) where Θ_i are the angular joint variables. Correspondingly, any obstacle in the work space of the arm has its image in the configuration space. Suppose that the arm is in contact with an obstacle. This corresponds to the automaton being positioned on the boundary of that obstacle's image in the configuration space. The automaton's general strategy will be to move along the obstacle boundary until some condition for leaving the obstacle dictated by the motion planner is satisfied. Locally, a step along the obstacle boundary corresponds to a step along a tangent line that can be defined at the point of contact between the robot and the obstacle. Now the automaton should move along this tangent line so as to guarantee that no collision takes place. If the size of the step is chosen so that after its completion the arm is still positioned within the area that is currently known (based on the sensor data) to be free of obstacles, the motion is guaranteed to be collision-free. This tangent line, an essential factor in surface following, can be found from the location of the obstacle with respect to the arm.

The procedure for local tangent generation described below assumes robot links of zero thickness (Figure 3-1). However, the Step Planning algorithm can also be used for real robots with arms of non-zero thickness, as will be shown in Section 3.4.3. Errors in the local tangent calculation due to, for example, inaccurate parametrization of the location of a sensor can be reduced because the sensor system provides an analog proximity reading. This proximity reading is used to achieve better contour following of the obstacle, similar to the rotation of the calculated local tangent used in [9,12], which allowed the robot arm to successfully follow the contour of obstacles. By calculating the difference between the desired sensor output level and the actual sensor output level as the obstacle is being followed, the local tangent can be adjusted to track the contour of the obstacle, and to compensate for the errors in the calculation process. This feedback procedure is explained in Section 3.5.

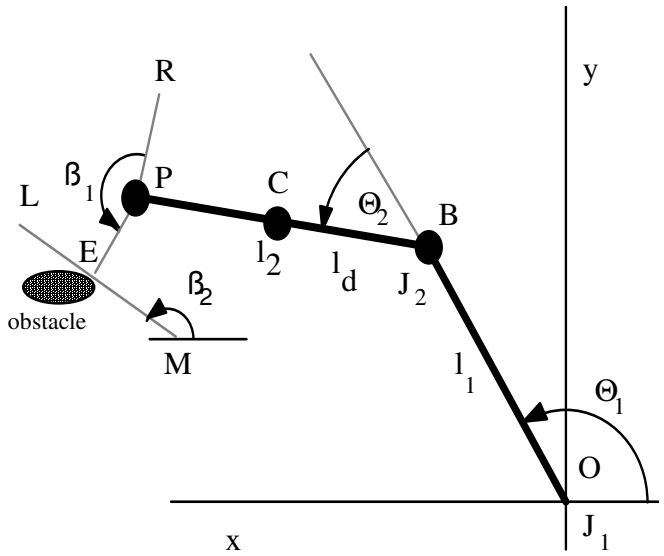


Figure 3-1. Sketch of the 2D arm with revolute joints.

Consider a simple two-link planar robot arm with two revolute joints, Θ_1 and Θ_2, Figure 3-1. Link 1 and link 2 of the arm are represented by the line segments OB and BP, of lengths l_1 and l_2, respectively; l_d is the distance between the point C and B; J_1 and J_2 are the arm joints. Point P represents the wrist; point B, which coincides with joint J_2, represents the arm elbow; point O is fixed and represents the origin of the reference system. Angle β_1 is the angle between the line PE and PR, and angle β_2 is the angle between the line LM and the x-axis.

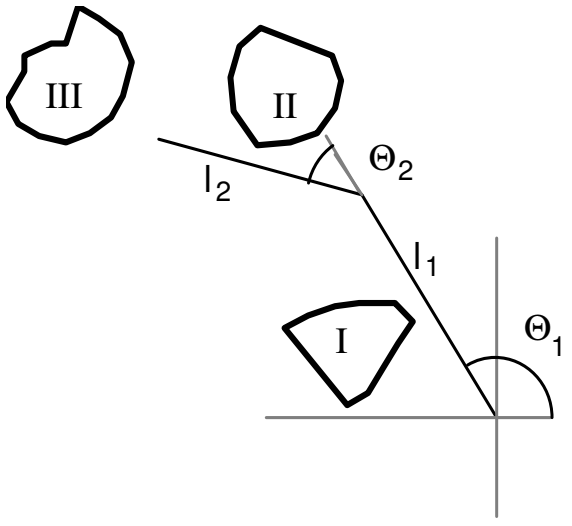


Figure 3-2. Obstacle types for the 2D arm.

For the purpose of calculating the local tangents, obstacles are divided into three types, Type I, Type II, and Type III, see Figure 3-2. The full derivations for the local tangent expressions are given in Appendix B; the final expressions are given below. In the expressions, $\frac{d\Theta_2}{d\Theta_1}$ refers to the slope of the local tangent.

Type I are those obstacles that obstruct link l_1. Since link l_2 is irrelevant in such cases, then $d\Theta_1 = 0$ and $d\Theta_2 \neq 0$. The local tangent in this case is therefore parallel to the Θ_2-axis.

Type II are those obstacles that obstruct link l_2. Assume that the sensor detecting the obstacle is at point C in Figure 3-1. We wish to slide the arm along the obstacle at this point. The local tangent to the obstacle boundary in this case is:

$$\frac{d\Theta_2}{d\Theta_1} = -\left(\frac{l_1}{l_d}\cos\Theta_2 + 1\right) \tag{3.1}$$

Type III are those obstacles that obstruct the wrist or elbow of the arm. The local tangent to the obstacle boundary in this case is:

$$\frac{d\Theta_2}{d\Theta_1} = -\left(\frac{\frac{l_1}{l_2} + \cos\Theta_2 + \sin\Theta_2 \tan(\Theta_2 + ß_1)}{\cos\Theta_2 + \sin\Theta_2 \tan(\Theta_2 + ß_1)}\right) \tag{3.2}$$

3.3 Local normal generation for the 3D case

In the 3D case, instead of a local tangent line, a local normal to the tangent plane at the contact point of the obstacle in the configuration space is found. Similar to the 2D case, motion along the tangent plane minimizes the likelihood of collision with the obstacle. Here, the motion of the robot can be represented by an automaton that operates in the three-

dimensional configuration space $(\Theta_1, \Theta_2, \Theta_3)$ where Θ_i are the angular joint variables.

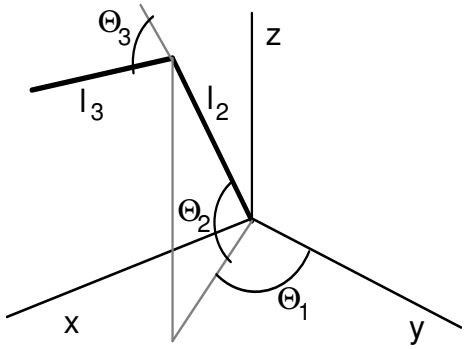


Figure 3-3. Sketch of the 3D robot arm with revolute joints.

Depending on their location with respect to the arm links, obstacles can be grouped in four categories easily recognized by the sensor system: Type I, Type II, Type III, and Type IV. Since in the considered arm the first two joints coincide and so link l_1 is absent, Type I obstacles never occur; the other types are defined below. Hereafter, the following substitutions will be made to enable a more compact notation:

$$c_i = \cos\theta_i\ ,\ s_i = \sin\theta_i\ ,\ c_{(i-j)} = \cos(\theta_i - \theta_j);$$
$$s_{(i-j)} = \sin(\theta_i - \theta_j)\ ,\ i = 1,2,3;$$
$$c_\alpha = \cos\alpha\ ,\ s_\alpha = \sin\alpha,\ c_\beta = \cos\beta\ ,\ s_\beta = \sin\beta.$$
$$d_\Theta P \text{ means P differentiated with respect to } \Theta.$$

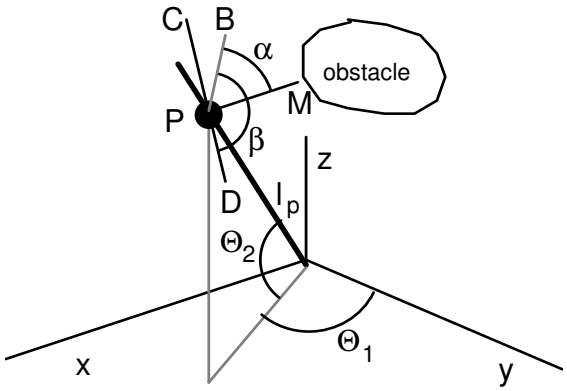


Figure 3-4. Type II obstacle.

Type II are those obstacles that obstruct link l_2, Figure 3-4. Since only link l_2 is

obstructed, we temporarily ignore link l_3. Point P is the location of a sensor on the arm that is sensing an obstacle, and PM represents the sensing axis of that particular sensor, where PM is perpendicular to link l_2. To slide along the Type II obstacle, the arm must incrementally move the point of contact in a direction parallel to the line CD, which is perpendicular to PM. Line segment PB is also perpendicular to l_2 and lies in the plane that contains the z-axis and link l_2. Let ß denote the angle between lines PB and PD, and l_p - distance between the origin and point P along the link l_2. The local normal to obstacles of this type can be found using the following procedure.

Using the equations of forward kinematics [46], we find the vector of cartesian coordinates of P as a function of the joint angles:

$$P = \begin{pmatrix} P_x \\ P_y \\ P_z \end{pmatrix} = l_p \begin{pmatrix} c_2\, s_1 \\ c_2\, c_1 \\ s_2 \end{pmatrix} \tag{3.3}$$

The local normal can be found by using the Jacobian [46], which relates incremental motions of a point on the robot arm, to incremental motions of the arm's joints. Differentiating P with respect to θ, denoted by $d_\theta P$, we can find the Jacobian:

$$J_1 = d_\theta P = l_p \begin{bmatrix} c_2\, c_1 & -s_2\, s_1 \\ -c_2\, s_1 & -s_2\, c_1 \\ 0 & c_2 \end{bmatrix} \tag{3.4}$$

J_1 is a two column matrix because θ_3 does not enter in (3.3). Using (3.4), find the motion of P in terms of the motion of joints θ_1 and θ_2:

$$\begin{pmatrix} dP_x \\ dP_y \\ dP_z \end{pmatrix} = J_1 \begin{pmatrix} d\theta_1 \\ d\theta_2 \end{pmatrix} \tag{3.5}$$

We wish to move point P of the arm in a direction parallel to the line CD, Figure 3-4, perpendicular to PM and l_2. This corresponds to these two vectors being parallel to

each other:

$$\begin{pmatrix} dP_x \\ dP_y \\ dP_z \end{pmatrix} \text{ parallel to } \overrightarrow{CD} \tag{3.6}$$

The directional vector of CD is:

$$\overrightarrow{CD} = \begin{pmatrix} -\left(s_\beta\, c_1 + s_2\, c_\beta\, s_1\right) \\ s_\beta\, s_1 - s_2\, c_\beta\, c_1 \\ c_2\, c_\beta \end{pmatrix} \tag{3.7}$$

By substituting (3.7) into (3.5) and solving for $d\theta_1$, $d\theta_2$, $d\theta_3$, find the derivative of the joint vector which defines the motion needed to move point P along CD . Thus the direction of motion is defined by the vector:

$$\begin{pmatrix} d\theta_1 \\ d\theta_2 \\ d\theta_3 \end{pmatrix} = w \begin{pmatrix} -s_\beta \\ c_\beta\, c_2 \\ \lambda \end{pmatrix} \tag{3.8}$$

where λ and w are constants

The quantity λ in (3.8) can be arbitrary because in the case at hand link l_3 is not obstructed, and so it is free to move. Therefore, θ_3 is free to swing in any direction, and $d\theta_3$ can have any value. Instead of the angle β, the sensor system will actually provide another angle, α, which describes the line PM and relates to β as

$$\alpha = \beta - \pi/2 \tag{3.9}$$

Substituting (3.9) into (3.8), obtain:

$$\begin{pmatrix} d\theta_1 \\ d\theta_2 \\ d\theta_3 \end{pmatrix} = \begin{pmatrix} c\alpha \\ s\alpha\, c_2 \\ \lambda \end{pmatrix} \tag{3.10}$$

Finally, the normal to the collection of vectors described by (3.10) is:

$$\vec{n} = \begin{pmatrix} s\alpha\, c_2 \\ -\, c\alpha \\ 0 \end{pmatrix} \tag{3.11}$$

Thus, (3.11) is the local normal of the tangent plane at the point of contact with a Type II obstacle.

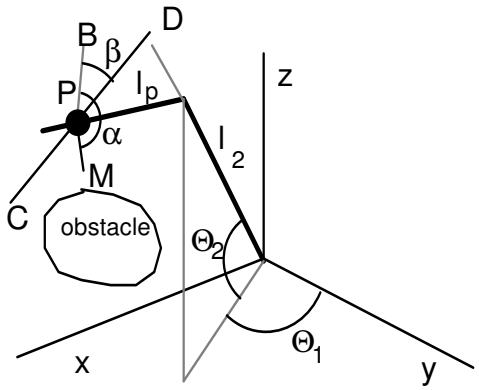


Figure 3-5. Type III obstacle.

Type III are those obstacles that obstruct link l_3, as shown in Figure 3-5. The arm can be moved safely in work space by moving the point P in a direction parallel to the plane that contains l_3 and the line CD. Line segment PB is perpendicular to l_3 and lies in the plane that contains the z-axis and link l_3; points P and M are as defined above; line PM is perpendicular l_3; line CD is perpendicular to the line PM and link l_3; α is the angle between PB and PM; and β is the angle between PB and CD. Using forward kinematics, find the coordinates of point P:

$$P = \begin{pmatrix} (l_2\, c_2 + l_p\, c_{(2-3)})\, s_1 \\ (l_2\, c_2 + l_p\, c_{(2-3)})\, c_1 \\ (l_2\, s_2 + l_p\, s_{(2-3)}) \end{pmatrix} \tag{3.12}$$

Differentiating P with respect to θ, obtain the Jacobian of the arm

$$J_2 = d_\theta P = \begin{bmatrix} c_1\,(l_2\, c_2 + l_p\, c_{(2-3)}) & -s_1\,(l_2\, c_2 + l_p\, s_{(2-3)}) & s_1\, l_p\, s_{(2-3)} \\ -s_1\,(l_2\, c_2 + l_p\, c_{(2-3)}) & -c_1\,(l_2\, c_2 + l_p\, c_{(2-3)}) & c_1\, l_p\, s_{(2-3)} \\ 0 & (l_2\, c_2 + l_p\, c_{(2-3)}) & -\, l_p\, c_{(2-3)} \end{bmatrix} \tag{3.13}$$

where $s_{(2-3)} = \sin(\theta_2 - \theta_3)$, $c_{(2-3)} = \cos(\theta_2 - \theta_3)$

The line segment PM is the normal of the tangent plane at the contact point P in work space:

$$\overrightarrow{PM} = \begin{pmatrix} -(c_1\, s_\alpha + s_1\, s_{(2-3)}\, c_\alpha) \\ s_1\, s_\alpha - c_1\, s_{(2-3)}\, c_\alpha \\ c_{(2-3)}\, c_\alpha \end{pmatrix} \tag{3.14}$$

We will need two non-collinear vectors perpendicular to line PM. In principle, any pair of such vectors can be used; the ones chosen in our system is

$$\overrightarrow{t_1} = \begin{pmatrix} 0 \\ -\, c_{(2-3)}\, c_\alpha \\ s_1\, s_\alpha - c_1\, s_{(2-3)}\, c_\alpha \end{pmatrix}$$
$$\overrightarrow{t_2} = \begin{pmatrix} c_{(2-3)}\, c_\alpha \\ 0 \\ c_1\, s_\alpha - s_1\, s_{(2-3)}\, c_\alpha \end{pmatrix} \tag{3.15}$$

Find the direction in configuration space that will cause P to move in the direction of M:

$$J_2^{-1}\, \overrightarrow{PM} = \begin{pmatrix} s_\alpha\, l_p \\ 0 \\ c_\alpha\,(l_p\, c_{(2-3)} + l_2\, c_2) \end{pmatrix} \tag{3.16}$$

The motion of point P anywhere in the plane spanned by t_1 and t_2 is guaranteed to

be locally collision free. We can now use J_2 to find the directions of motion in configuration space that would keep the arm in this plane. The local normal to the tangent plane at the point P in configuration space, vector $\vec{n}$, is normal to $\vec{p_1} = J_2^{-1}\,\vec{t_1}$ and to $\vec{p_2} = J_2^{-1}\,\vec{t_2}$. This vector is given as:

$$\vec{n} = \begin{pmatrix} s_\alpha \left(\frac{l_2}{l_p} c_2 + c_{(2-3)}\right) \\ - c_\alpha \left(\frac{l_2}{l_p} c_3 + 1\right) \\ c_\alpha \end{pmatrix} \tag{3.17}$$

Thus, (3.17) presents the local normal of the contact plane in configuration space. One may ask why we couldn't have found (3.17) simply by transforming the vector in (3.14) to configuration space. Indeed we could if the normal to the tangent plane in workspace transformed to the normal of the tangent plane in configuration space. This is not true in general, and the reason for that is that the mapping from workspace to configuration space is nonconformal. In other words, angles between lines and planes are not preserved by the mapping, and as a result the arm does not slide along the obstacle if it moves perpendicularly to the transformed vector in (3.14).

However, the plane in the configuration space spanned by the vectors p_1 and p_2 is indeed the tangent plane in that space. Thus, the normal vector in the configuration space can be found directly, using vectors p_1 and p_2 - which produces vector $\vec{n}$ in (3.17). Any motion in the configuration space perpendicular to $\vec{n}$ is therefore safe.

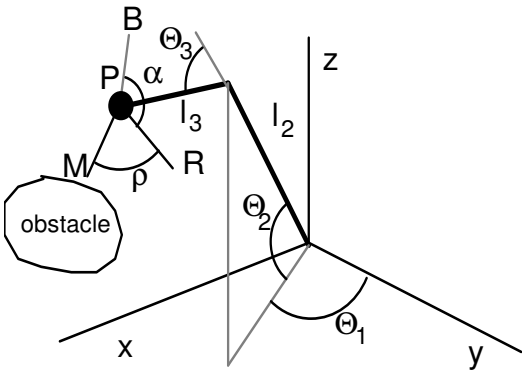


Figure 3-6. Type IV obstacle.

Type IV are those obstacles that obstruct the arm wrist or elbow, see Figure 3-6. Here, line segment PB is perpendicular to link l_3 and lies in the plane that contains the z-axis and link l_2; the plane containing points B, P, and R is perpendicular to link l_3. Notation: PR is the projection of PM onto the plane containing B, P, and R; the angle between BP and PR is α, and the angle between the plane BPR and PM is ρ. Finding the local normal here is similar to the case of a Type III obstacle, except now $l_p = l_3$ is substituted in the expressions for forward kinematics and Jacobian calculation. First, find the directional vector of PM:

$$\overrightarrow{PM} = \begin{pmatrix} -c_1\ s_\alpha\ c_\rho + s_1\left(c_{(2-3)}\ s_\rho - s_{(2-3)}\ c_\alpha\ c_\rho\right) \\ s_1\ s_\alpha\ c_\rho + c_1\left(c_{(2-3)}\ s_\rho - s_{(2-3)}\ c_\alpha\ c_\rho\right) \\ s_{(2-3)}\ s_\rho + c_{(2-3)}\ c_\alpha\ c_\rho \end{pmatrix} \tag{3.18}$$

Find two non-collinear vectors, $\overrightarrow{t_1}$ and $\overrightarrow{t_2}$, each perpendicular to PM:

$$\overrightarrow{t_1} = \begin{pmatrix} 0 \\ -s_{(2-3)}\ s_\rho - c_\alpha\ c_{(2-3)}\ c_\rho \\ s_1\ s_\alpha\ c_\rho + c_1\left(c_{(2-3)}\ s_\rho - c_\alpha\ s_{(2-3)}\ c_\rho\right) \end{pmatrix}$$

$$\overrightarrow{t_2} = \begin{pmatrix} -s_{(2-3)}\ s_\rho - c_\alpha\ c_{(2-3)}\ c_\rho \\ 0 \\ -c_1\ s_\alpha\ c_\rho + s_1\left(c_{(2-3)}\ s_\rho - c_\alpha\ s_{(2-3)}\ c_\rho\right) \end{pmatrix} \tag{3.19}$$

Finally, for a Type IV obstacle the local normal to the tangent plane in the configuration space is:

$$\vec{n} = \begin{pmatrix} s_\alpha \left(\frac{l_2}{l_3} c_2 + c_{(2-3)} \right) \\ -c_\alpha \left(- \frac{l_2 \, s_3 \, s_\rho}{l_3 \, c_\alpha \, c_\rho} + \frac{l_2}{l_3} c_3 + 1 \right) \\ c_\alpha \end{pmatrix} \tag{3.20}$$

3.4 Handling Special Cases

3.4.1 Robot Arms with protruding elbows

For certain articulated arms, such as the PUMA 562, the second link extends out on either side of the joint J_2, causing the elbow to be susceptible to collision, Figure 3-7. To find the local normal for obstacles obstructing the elbow, the same method described above is used, except Θ_3 is now replaced by $\Theta_3 + \pi$.

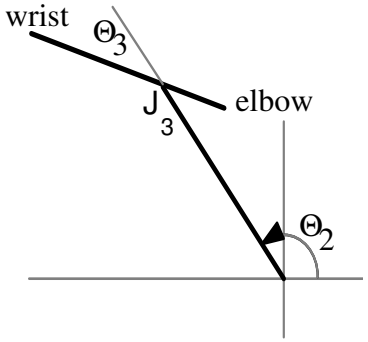


Figure 3-7. Example of an extended elbow on link l_3.

3.4.2 Singularity of the Jacobian. Local normal degeneracies

Interestingly, the problem of singularities that so often plagues various robot motion control techniques, does not present a difficulty in our case. This is because, although the Jacobian is being used in the derivation of the normal vectors, its determinant

can be factored out because only the direction of the normal vector, not its norm, is being used.

Local normal degeneracies

Another type of special case takes place when all the components of the normal vector $\vec{n}$ turn to zero; this corresponds to $\alpha = (\pi/2) + n\,\pi$, where n=0,1,2,.. in (3.11), (3,17), and (3.20) respectively. To consider this case, it is sufficient to assume that only one sensor is sensing an obstacle. To find the local for $\alpha = \pi / 2$, substitute this value of α into any of the expressions (3.11), (3.17), or (3.20); the resulting local normal is :

$$\vec{n} = \begin{pmatrix} u \\ 0 \\ 0 \end{pmatrix} \tag{3.21}$$

where u is a constant. Assume for the moment that $u \neq 0$; the case u=0 is considered below. This local normal implies that safe motion of the arm relative to the obstacles in this situation must involve only incremental change of Θ_2 and Θ_3 (see Figure 3-5). When $\alpha = \pm \pi / 2$ the plane that contains l_2 and l_3 is perpendicular to the line PM, and the arm can move in that plane without collision with the obstacle.

Another special situation develops when the point P is on the z-axis: then, the following relationship between Θ_2 and Θ_3 holds:

$$c_2 = - \frac{l_p}{l_1}\, c_{(2-3)} \tag{3.22}$$

In this case, the local normal is of the form

$$\vec{n} = \begin{pmatrix} 0 \\ u_1 \\ u_2 \end{pmatrix} \tag{3.23}$$

where u_1 and u_2 are constants

The expressions for local tangent are always well defined except in the case when both of the above situations occur simultaneously, thus when:

$$c_2 = -\frac{l_p}{l_1} c_{(2-3)} \quad \text{and} \quad \alpha = \pm \pi / 2$$

in which case the local normal becomes:

$$\vec{n} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} \tag{3.24}$$

Obtaining the null vector suggests that any incremental motion of the arm can be considered collision free. As already described above, an incremental change of Θ_2 and Θ_3 can be considered safe because the plane that contains l_2 and l_3 is perpendicular to PM. As for Θ_1, because point P lies on the z-axis, its motion is not affected by any incremental change of Θ_1. Thus, an incremental change of Θ_1 is also safe. By repeating the sensing and Step Planning phase iteratively at every step, proper collision free motion can be achieved.

3.4.3 Robot arm links of varying shape and size

Although the local normal procedure was derived for an arm with zero thickness, this algorithm can also be used for real robot arms with links of non-zero thickness. Recall that in the 3D case the local normal describes a tangent plane in the configuration space that causes the arm link to move perpendicular to the sensing axis in the work space. Since the distance between the obstacle and the link of zero thickness does not enter into the equations for the local normals, the generated local normal is valid regardless of the distance from the obstacle to the link. In the case of a real robot arm with non-zero

thickness, its thickness does not affect the local normal procedure. Therefore the local normal method can be used with links of any thickness. Note that the proximity reading has its role during Step Planning where the local normal is modified accordingly to servo the arm to a desired sensor level. Below it will be shown that almost any shape of the link can be accommodated also.

For links with circular cross sections, the sensing axis coincides with the perpendicular from the sensor to the link axis, see Figure 3.8. The dotted line in the figure is the reference from which the angle α of the sensing axis BC is measured.

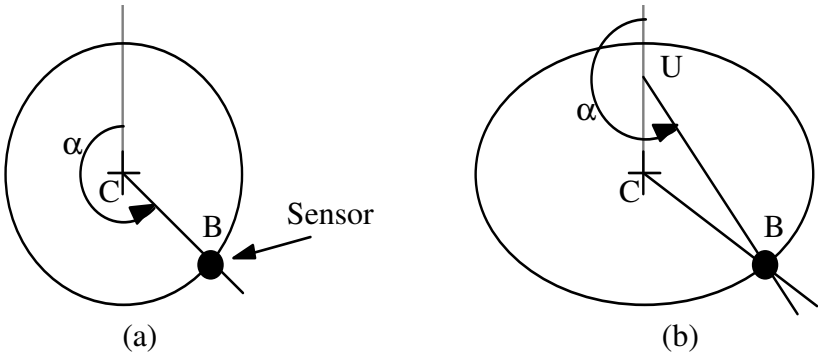


Figure 3-8. Cross-section of arm links. For links with a non-circular cross-section, the parameter α is measured as shown in Figure 3-8b.

For non-circular link cross-sections, such as in Figure 3-8b, the sensing axis BU no longer coincides with BC. By measuring the angle α as shown in Figure 3-8b, the local normal method can be used to find the local normal of the sensor at point B. Since the direction of the sensing axis of the sensor, such as sensor B in Figure 3-8, does not depend on the shape of the cross-section of the link, this method can be extended to links of any cross-sectional shape and size.

3.4.4 Robot arm joint limits

The joint limits of real robot arms can easily be incorporated into the local normal

approach. When the robot is near its joint limits, local normals are added that are perpendicular to the obstruction caused by the obstacle in the configuration space. For example, if the arm is simultaneously at the maximum joint limits of Θ_1, Θ_2, and Θ_3, the corresponding local normals are respectively: $\begin{pmatrix} -1 \\ 0 \\ 0 \end{pmatrix}$, $\begin{pmatrix} 0 \\ -1 \\ 0 \end{pmatrix}$, and $\begin{pmatrix} 0 \\ 0 \\ -1 \end{pmatrix}$.

3.5 Different modes of the local normal selection procedure

Once the collection of local normals corresponding to all points of contact between the arm and obstacles is found, a procedure to select the proper normal to guide the motion of the arm is needed. At each step along the arm's path, and while in contact with obstacles, the motion of the arm is guided by sliding it along a chosen normal plane, specified by a local normal generated as described in Section 3.2 or 3.3. In the work space, this corresponds to the arm sliding along a contact point near the sensor that generated the selected local normal. By choosing the proper local normal at each step, the arm can move in an environment amidst obstacles while avoiding collisions.

For the purpose of motion planning, depending on the level of automatic control desired by the user, the normal selection procedure can fall in one of three major categories: (i) the low level control, called the Repeller mode; (ii) the intermediate level, called Teleoperation mode; (iii) the higher level control, called Automatic Motion Planning mode. Each of these is explained in more detail below. Other uses of the local normal technique not related to motion planning are discussed in Section 3.5.3.

Repeller mode.

In the low level of control, the robot arm can be commanded to move in the direction of the weighted sum of all the local normals. This causes the arm to move away from obstacles that are closest at the moment. This procedure can be used to avoid collisions, to "push" the arm into a desired position, and can also be used to test the sensor hardware, local normal generation routines, robot interface, and robot arm motors. The

Repeller also proved to be an effective way of quickly demonstrating the operation of the proximity sensor skin, and became the most often used “demo”.

Teleoperation mode.

Recently, there has been research in the area of teleoperation and telerobotics with robot arm manipulators [4,25,41,50]. The arm is commanded to move in real-time by a human operator who is viewing the progress of the operation directly or via TV cameras, sometimes assisted with force feedback. Traditionally, both the task of moving to the goal position and avoiding obstacles that can obstruct the arm along the path are performed by the human operator. With the help of the sensitive skin the second task can be performed more automatically. With the Teleoperation mode, motion commands can be passed directly from the human operator to the robot arm, unless obstacles are obstructing the arm. In the latter case, motion of the arm is modified by sliding the arm along the surface of the obstacle such that the distance between the current position and the commanded position in the configuration space is minimized. How to perform the sliding along obstacles is explained below in Section 3.5.2.

The data from the sensitive skin can be useful in teleoperation systems not only when the obstacles are occluded or invisible from the operator, but also when there is a significant time delay between the specification of a command and its execution such as is common when the operator and robot are separated by a large distance. Due to the time delay, the operator may not be able to give the appropriate commands to avoid an approaching object even when the entire scene is clearly in view. In this case some form of local obstacle avoidance, such as the Repeller or Teleoperation mode, is then necessary.

The implementation of this motion control procedure is just slightly more complicated than the Repeller mode above. Input commands to the first three degrees of freedom are read from the potentiometers of a miniature robot arm whose kinematics is similar to the P5 robot arm. This miniature arm functions as the master in a master-slave

relationship to the P5 robot arm. The slave arm moves as commanded by the master arm if there are no obstacles nearby, or slides, under the automatic control, along obstacles that are in the way by locally minimizing the distance between the desired (i.e. designated by the master arm position) and actual position of the arm in configuration space. Overall performance is dependent on the sensor gain: when the gain is set too high the robot arm, when halted, tends to oscillate near the surface of the obstacle.

Automatic Motion Planning mode.

On the higher level of automatic control, a fully automatic motion planning takes place. In this case, the user supplies only the desired target coordinates, and the robot control procedure finds a path to the target if one exists. The motion planning algorithm, fully described in Chapter 4, produces motion that takes place either in free space, or along obstacle surfaces. The procedure to accomplish the latter task is presented below - first, for the simpler 2D case, and then for the 3D case.

3.5.1 Local tangent selection for sliding: 2D case

Every point contact between the arm body and an obstacle has an associated sensor pair that does the corresponding obstacle detection. For every sensor pair that detects an obstacle, a local tangent is calculated based on the method described above in Section 3.4. If more than one sensor detects an obstacle, more than one tangent is generated.

The robot arm shown in Figure 3-1 can be represented by a point in the configuration space (Θ_1, Θ_2). Because the values $(\Theta_i + 2\pi)$, i=1,2, correspond to the same position of each link, the configuration space represents a common torus. Or, if the range of a joint is limited it represents a subset of the torus. Correspondingly, any obstacle in the work space has its image in the configuration space. Then, the path planning problem becomes that of moving an automaton on the surface of the torus. Although the actual algorithm takes into account the topology of the torus (for more detail, see [36]), its main

idea can be presented in terms of the planar configuration space, Figure 3-9.

The arm starts by moving from the start position S to the target position T, along the line segment ST. If an obstacle is encountered, the arm turns in a prespecified local direction (e.g. left, or clockwise as in Figure 3-9) and follows the contour of the obstacle until the line segment ST is again met.

In case of a clockwise (cw) local direction, a counterclockwise (ccw) rotation of the calculated local tangent will cause the robot arm to move away from the obstacle, Figure 3-10. The amount of guaranteed collision-free rotation within one step is determined by the distance at which the obstacle is detected and by the arm geometry. In order for this strategy to work, the distance cannot be zero, which dictates the use of proximity sensors. As for the actual distance between the arm and an obstacle during the contour following, it is unimportant as long as, on the one hand, the arm does not touch the obstacle and, on the other hand, it stays in "contact" with it.

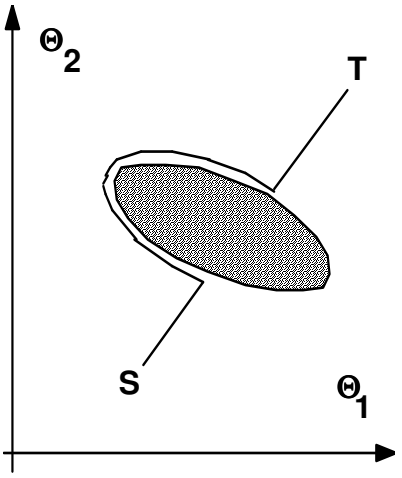


Figure 3-9. Configuration space of robot arm of Figure 3-1.

If the arm is too close to the obstacle, the local normal can be rotated ccw to increase the distance to the obstacle. The reverse is done if the sensed obstacle is still a long distance from the arm. No adjustment is made to the normal to be followed if the output of the sensor is at a preset value. This assures that the distance between the arm and the obstacle

is within the sensing range.

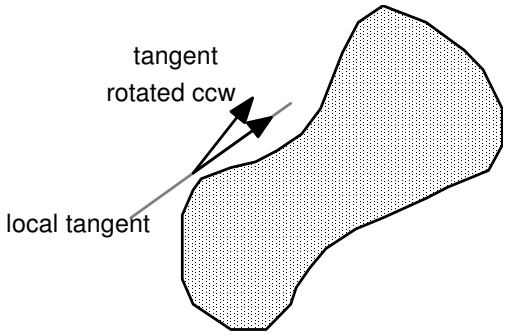


Figure 3-10. Configuration Space: ccw rotation of local normal causes the robot to move away from the obstacle.

On the lower level of control during contour following, an important control parameter is the proportional feedback gain, K_p. This gain is expressed in units of degrees per error in volts; here, the error is defined as the difference between the desired reference level, V_r, and the sensor output voltage, V_s. Then, the rotation of the local normal in degrees, Rot, is found as $Rot = K_p (V_r - V_s)$; the sign of Rot determines the direction of rotation. If, for example, K_p is set to 10 degrees/volt, and the sensor detecting the obstacle has an output voltage V_s= 2.5 volts, then the local normal will be rotated ccw 5 degrees. The resulting value of the local normal is equal to the sum of the value Rot and the angle between the original local normal and the positive Θ_1 axis. By using proportional feedback of the sensor level error ($V_r - V_s$), the arm is made to track the contour of the obstacle at the desired sensor reading. Increasing the gain K_p will result in better tracking performance, although too high a proportional gain results in an oscillatory motion of the arm while following the obstacle contour.

A more difficult situation in contour following develops when more than one sensor pair senses one or more obstacles simultaneously. Then, a number of local normals are calculated, of which one is selected for executing the next step. Two methods were developed for selecting among the local tangents: the first is called the geometric method,

and the second - the sensor output based method.

Geometric tangent selection method.

We illustrate this method with the help of two examples, one utilizing only one obstacle, and the other utilizing several obstacles. The method chooses a local normal to follow as the next small step by a process of elimination, so as to produce collision-free motion.

1. Interaction with a single obstacle.

Suppose the robot arm is moving from point P' towards point P, in the direction of increasing Θ_1, Figure 3-11a. At point P the arm is obstructed by obstacle 2, which presents an obstacle of Type II. The local tangent at P is EB, Figure 3-11b. In the vicinity of point P, a further increase in Θ_1 results in crossing the local tangent, which is likely to lead to a collision with the obstacle, Figure 3-11a. If the local direction is "left", the next move along EB towards point B will cause the arm to slide along obstacle 2 to the position indicated by the dotted line in Figure 3-11a. Continual recalculation and motion along the resultant tangent constitutes the process of contour following.

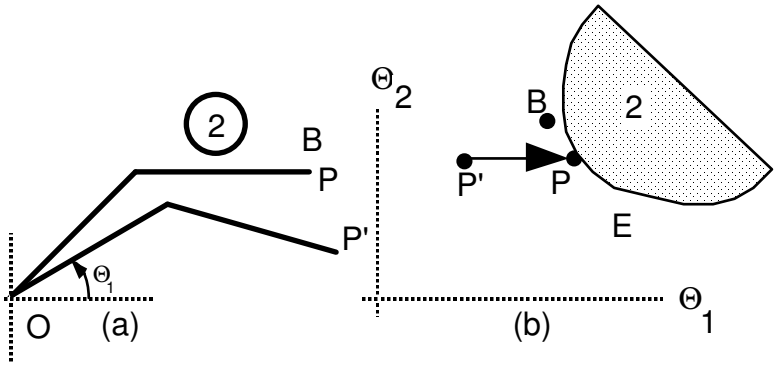


Figure 3-11. Arm interaction with a single obstacle:
a) work space, b) configuration space.

2. Interaction with multiple obstacles.

Suppose the robot arm is moving from point P' to P, in the direction of increasing

Θ_1, Figure 3-12. At point P, the arm is obstructed by three obstacles that are sensed by the sensor system, Figure 3-12a. A local tangent is then calculated for each obstacle. Obstacles 1, 2, and 3, of Type I, II, and III, produce tangents CD, BE, and AF, respectively. For the local direction "left", the robot can move towards one of the points A, B or C. Moving to point C necessitates crossing the lines BE and AF, with a large likelihood of penetrating the obstacles 2 and 3 associated with these two local tangents, an unacceptable situation. Similarly, moving towards point B could cause collision with obstacle 3, because the local tangent AF associated with it is crossed. Moving towards point A would not cause the crossing of any local tangents, which means that no collision will take place, and is therefore the chosen move. As a result of this move, the arm loses contact with obstacles 1 and 2, but remains in contact with obstacle 3. Analogously, a local direction "right" would necessitate a move towards point D for collision-free contour following.

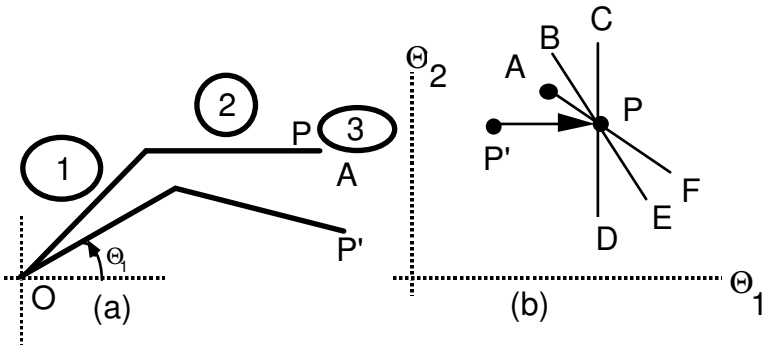


Figure 3-12. Arm interaction with multiple obstacles:
a) work space, b) configuration space.

Sensor output based tangent selection method.

This method does not require the computation of a local tangent for each sensor that senses an obstacle. Instead we simply select the sensor with the maximum (closest) reading, and calculate its local tangent. This local tangent is then rotated as before to maintain a desired sensor reading during the motion around the obstacle. Although this method sounds too simple, it works successfully, and is validated below.

The sensor output based tangent selection procedure relies on the fact that the robot

arm, when it comes in contact with an obstacle, in general slides along one contact point with the obstacle. The one exception to this is when the arm makes the transition from one contact point on the arm to another.

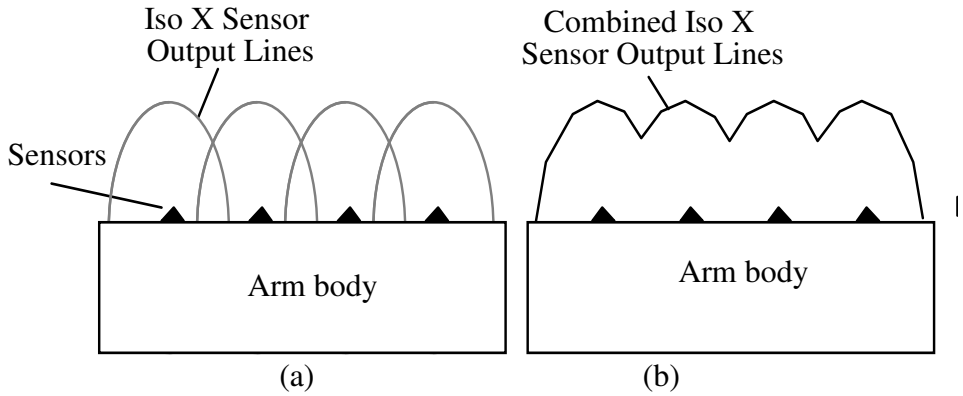


Figure 3-13. Iso X Sensor Output lines of the Skin.

Figure 3-13a shows a portion of the arm with the sensor output isolines (the lines of constant sensor output) of the associated sensors. Denote Iso X to be the isoline of level X. If we consider the entire sensor skin as a whole, with a single scalar output voltage, which is the maximum value of all the individual sensors, we obtain a sensor whose Iso X sensor output lines appear as shown in Figure 3-13b.

Imagine that the Combined Iso sensor line represents the surface of an imaginary tactile robot arm that slides along the work space obstacle. Then, its configuration space obstacles are the same as with the original robot arm, but "grown" by some amount, Figure 3-14. The size of the growth depends on the kinematic configuration of the arm, and also on the sensing range. The key behind the tangent selection procedure is that in work space, the arm, in general, slides along one bump caused by the Iso X sensor lines at a time (unless it is making the transition between the Iso X of one sensor to another). This is also true in configuration space where the arm follows the contour of the obstacle of the imaginary robot arm. The bump along which it slides is associated with the sensor with the maximum reading. Thus we calculate the local tangent of the sensor with the maximum reading and

adjust for sensor feedback by rotating its local tangent while moving along the obstacle.

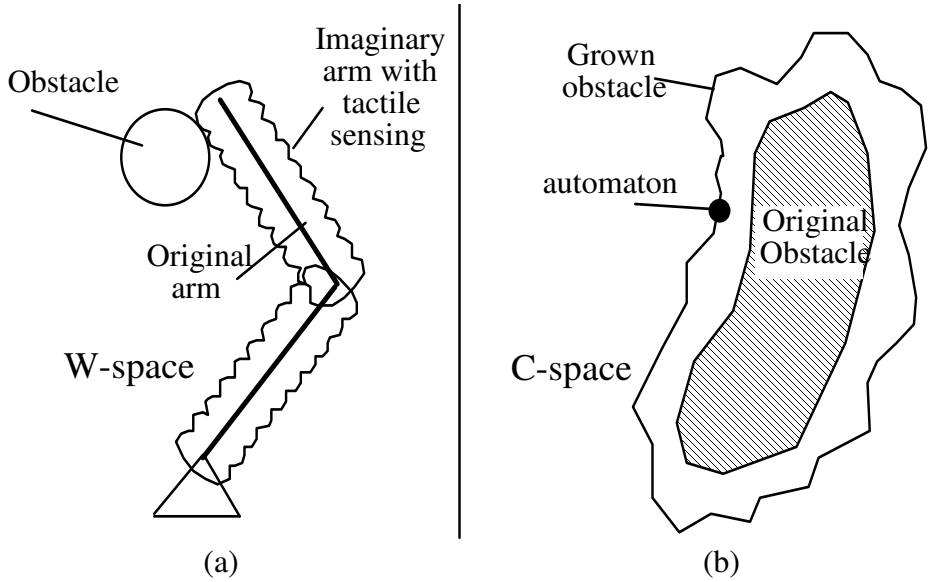


Figure 3-14. Effect of sensor range on the configuration space. Obstacles are grown by an amount that depends on the position of the arm and the sensing range.

Note that safety concerns are addressed since the sensor that has the maximum reading is selected, thus the obstacle closest to the arm is used in the Step Planning procedure. If the obstacle is too close, using sensor feedback the distance is increased as the arm moves.

Performance comparison between two selection methods

One difference between the performance of the two Step Planning strategies is shown in Figure 3-15. The motion of the arm using the geometric method is shown in Figure 3-15a, and motion with the sensor output based method is shown in Figure 3-15b. While following obstacles, and switching between two local tangents, the geometric method, since it takes into account all current local tangents, tends to look ahead and round corners. The second, non-geometric method, relies only on the current sensor output level. Since the combined Iso X line tends to have a discontinuous tangent where two beams

overlap, the motion with the sensor based method tends to be less smooth.

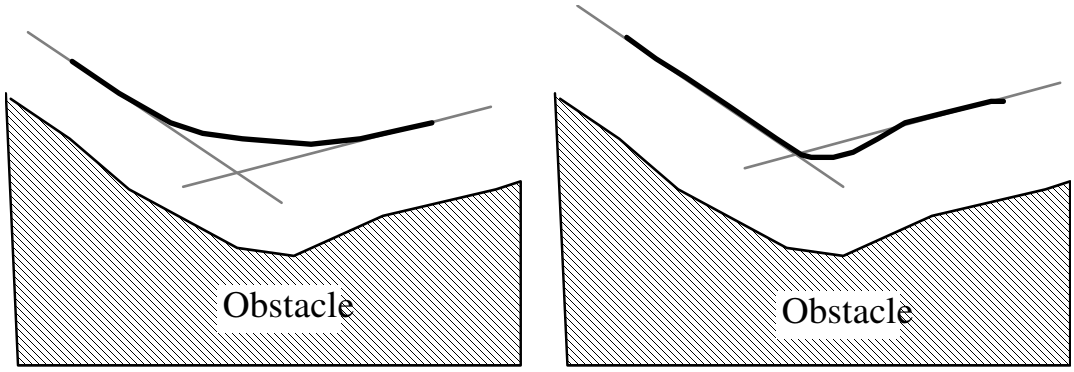


Figure 3-15. Performance comparison of the two tangent selection methods: arm path in the configuration space. (a) Geometric tangent selection, (b) sensor output based tangent selection

The advantage of the second tangent selection procedure is that it is computationally more efficient and easier to implement. If n is the number of sensors simultaneously sensing an obstacle, the geometric method requires more than n times more floating point calculations, since it requires calculation of a local tangent and one arctangent for each sensor. The second method requires the computation of only one local tangent.

The disadvantage of the second method is that corners in the path are not rounded, as is illustrated in Figure 3-15.

To summarize, the sensor output based tangent selection method operates as follows: it

1) Selects the sensor with maximum reading,
2) Calculates the sensor's local tangent, and then
3) Rotates the local tangent (ccw or cw) to account for sensor feedback.

Handling Limit Cycles

Initial trials using either of the two local tangent selection methods presented above showed that the arm sometimes can find itself in a limit cycle, where it oscillates between

two tangents. This tended to occur when the motion speed was high, and the arm was at a concavity of an obstacle. Careful examination of the experimental data showed that, while in the limit cycle, the sensor level error (V_r - V_s, see Section 3.5.1) was very large. The source of the limit cycle was that this error caused a very large amount of rotation of the calculated local tangents, which in turn caused the arm to "double back" on its path.

The initial solution was to decrease the proportional feedback gain K_p, but this deteriorated tracking performance, without guaranteeing a remedy to the limit cycle problem. A better way of limiting the amount of tangent rotation is illustrated in Figure 3-16. Here, A and B are local tangents, and D is the line that bisects them. Suppose the arm is initially located at point P, and follows the local tangent A to point P'. At point P', the arm generates both local tangents A and B, and has moved so close to the obstacle that the local tangent B needs to be rotated ccw by 90^o. Without limiting the amount of rotation, this could cause the arm to double back along A, and result in a limit cycle. By limiting the rotation of the local tangent B until it is parallel to the line D, the bisector of local tangents A and B, we can guarantee that the arm moves away from the obstacle corresponding to local tangent B, while not getting closer to the obstacle associated with the local tangent A. Using this rotated tangent the arm moves to the point P", where the procedure of tangent generation, rotation and selection repeats.

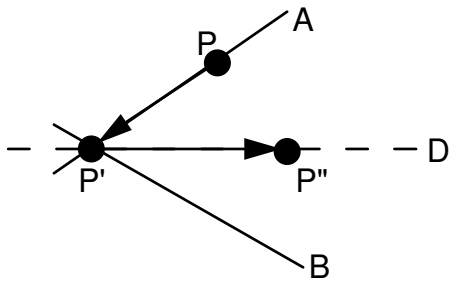


Figure 3-16. Maximum rotation of the local tangent to prevent limit cycles.

3.5.2 Local normal selection for sliding: 3D case

In the 3D case, for every sensor that detects an obstacle, a local normal is generated. The local normal of a particular sensor determines the conditions necessary to slide the arm along the obstacle near that sensor. If more than one sensor detect an obstacle, one or two (depending on the task) should be selected out of the collection of normals, and used to guide the arm's motion. The motion planning algorithm described in Chapter 4 calls for motion along the intersection of a plane and an obstacle, or motion along the intersection of two obstacles. The local normal selection procedure for each of these cases is described below.

If the task consists of following the intersection of a plane and an obstacle, see Section 4.7.2, the tangent planes obtained in the normal generation procedure can be intersected with the plane. The resulting tangent lines can then be treated (once a local direction is decided) just as the 2D case, selecting a tangent line with the geometric method, or with the sensor based method.

If the task consists of following the intersection of a Type II and Type III obstacle in configuration space, see Section 4.7.3, things become more complicated. Recall that a Type II obstacle is one that obstructs link l_2, and a Type III obstacle is an obstacle that obstructs link l_3. Since both links are each following an obstacle in work space, two local normals now need to be selected, one that corresponds to a sensor on link l_2, and the other corresponding to a sensor on link l_3. Selection of these two local normals is by the sensor output based local normal selection method. A local normal selection method based on geometric considerations was not found for this situation. Similar to the 2D case, sensor output is the sole criteria in the sensor output based normal selection procedure. The sensor with the maximum reading is selected for each link; then a local normal is calculated for each of the two resulting sensors, and used to find the next move step. The direction vector of the intersection of the two tangent planes can be found by taking the cross product of the two normal vectors, $c = n \times p$, see Figure 3-17.

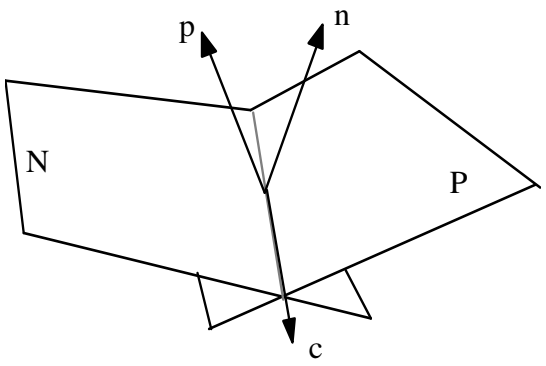


Figure 3-17. Intersection of two tangent planes; vector n is normal to plane N, p is normal to P, and c = n x p.

In the 2D case, feedback of the sensor level error is used to enable tracking of the contour of the obstacle at a desired sensor reading. Sensor level error is defined as (V_r - V_s), where V_r is the desired sensor output, and V_s is the actual sensor output. The local tangent of the obstacle is rotated by an amount proportional to this error. In the 3D case, instead of rotating a vector, sensor feedback is incorporated in another way. Instead of intersecting two planes to find the intersection curve between two obstacles, two cones are intersected instead. The axis of the cone coincides with the local normal of the tangent plane of the obstacle, while the angle between the sides of the cone and the tangent plane is proportional to the sensor error, Figure 3-18.

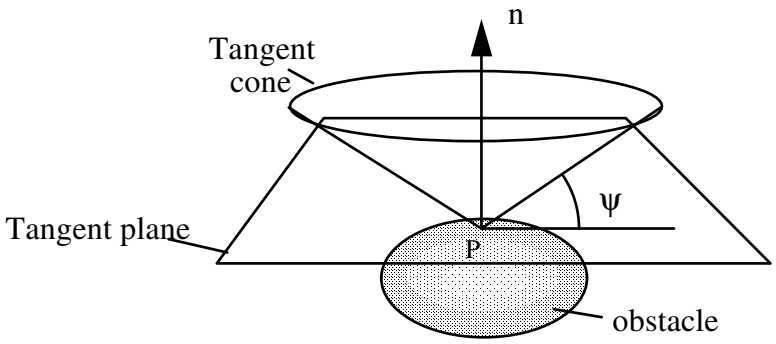


Figure 3-18. Tangent cone derived from the local normal vector n and the sensor reading; $\psi = K_p (V_r - V_s)$, where K_p - proportionality constant, V_r - reference sensor output, and V_s - actual sensor output.

The reason for choosing a tangent cone over the tangent plane in the determination of the intersection curve is that it represents in a single geometric object both the direction of the normal vector, and the sensor reading. If $V_r = V_s$, the resulting cone reverts to a plane. If the arm moves on the surface of the cone, it will move approximately tangential to the obstacle, while increasing or decreasing its distance to the obstacle, depending on the sensor reading. By intersecting the tangent cone of the two sensors selected with the sensor output based method, we can move along the intersection curve between a Type II and Type III obstacle, while using the feedback of both sensor errors for accurate tracking of the intersection curve.

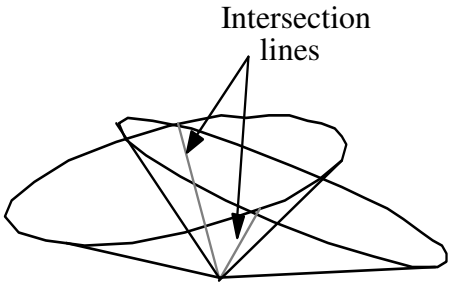


Figure 3-19. Two intersection lines of a pair of tangent cones.

The two intersection lines $\overrightarrow{s_{1,2}}$ of the two tangent cones with axis $\vec{n}$ and $\vec{p}$ are given by

$$\vec{s}_{1,2} = \alpha\, \vec{n} + \beta\, \vec{p} \pm \gamma \frac{(\vec{n} \times \vec{p})}{|(\vec{n} \times \vec{p})|} \tag{3.25}$$

where

$$\alpha = \frac{\psi_n}{\psi_n - \psi_p(\vec{n} \cdot \vec{p})}$$

$$\beta = \frac{\psi_p}{\psi_p - \psi_n(\vec{n} \cdot \vec{p})}$$

$$\gamma = \sqrt{1 - (\alpha^2 + \beta^2)}$$

$\psi_p = K_d (V_r - V_p)$; $\psi_n = K_d (V_r - V_n)$

K_d - proportional gain; V_r - Reference sensor output

V_p - Sensor reading of p; V_n - Sensor reading of n

The above expression for $\overrightarrow{s_{1,2}}$ contains a "±", which results in the two intersection lines for the two tangent cones. If one imagines the arm traveling on the intersection curve between the N-plane and P-plane, Figure 3-17, a "+" in the above expression causes the arm to keep the P-plane on its left as it moves on the intersection curve of the N-plane and P-plane. The reverse is true if a "-" is used.

To summarize, the local tangent selection procedure to follow the intersection (in configuration space) between a Type II and Type III obstacles is as follows:

1) Select the sensor with the maximum reading for each link, and
2) Calculate the local tangents for the two above sensors, then
3) Find the intersection curve using the cone intersection method.

The above procedure has been implemented in the motion planning system, and has been shown to successfully guide the arm along the intersection curve between a Type II and Type III obstacle.

3.5.3 Other uses for the local normal procedure

Uses of the sensitive skin and the associated local normals of obstacles in the configuration space are not restricted to motion planning alone. The system can also be used for mapping or recognizing obstacles in the work space of the arm. This approach is similar to that in [3,26,47,62], where a sensor is fixed to the robot arm for the purposes of haptic exploration, and dynamic shape estimation. This use of the sensor data was not tested in our system.

If obstacles in the work space are stationary, it can be argued that not all sensors need to be sampled at all times. Only those sensors that are located on parts of the arm susceptible to collision need be checked. For instance, if the arm and all the obstacles are stationary, no sensors need to be checked at all. Once the arm starts to move, only those

sensors that are "facing" the direction of motion (in the work space) need be checked. In principle, the relevant sensors can be found by using an extensive lookup table, but a simpler (and much less memory intensive) check can be performed instead, as follows:

> A sensor "faces" the direction of motion of the arm if the dot product of the direction of motion and the corresponding local normal is negative.

Since the limiting factor on the sensor update rate in the implemented system is the settling time per sensor, sensor update rate can be increased by skipping those sensors that do not face the direction of motion. The disadvantage of this method is that the local normals of all the skin sensors will need to be calculated at every iteration, not only those that correspond to a sensor that exceeds a preset threshold. Although not implemented in our system, this use of the local normal is expected to double the sensor scanning rate, since on average only half the sensors face the direction of motion at any one time. The above technique can also be used if the obstacles in the robot environment are not all stationary, but moving sufficiently slowly.

3.6 Conclusion

In this chapter, the procedure for sensor data interpretation and processing is presented. The described Step Planning procedure converts proximity information in the work space of the robot arm into local normal or tangent data in the 2D or 3D configuration space, and finds the collection of incrementally safe moves for the arm.

To calculate the local normals in the configuration space of the arm, obstacles are first divided into several types, depending which link of the arm they obstruct. The conditions necessary to slide the arm along an obstacle at the location of the sensor are found. This corresponds to the tangent line/plane at the contact point in the configuration space in the 2D/3D case.

For every sensor that detects an obstacle, a local normal/tangent is found. Two methods are presented to select the local tangent to cause sliding of the arm in the 2D case, a geometric and a sensor output based method. The geometric method requires more floating point calculations, but leads to smoother motion than the non-geometric (sensor output based) method. The two tangent selection methods extend readily to the 3D case if the intersection between a fixed plane and an obstacle is to be followed. No valid geometric method was found for some operations needed in 3D motion planning, but the sensor output based method extends readily to the 3D case.

For motion planning, the local normals can be used in one of three major ways, depending on the level of automatic control desired. For the lower level of control, the robot moves according to the weighted sum of the local normals, which causes the arm to move away from obstacles. This procedure can be used to avoid collisions, to "push" the arm into a desired position, and can also be used to test the sensor hardware, local normal generation routines, robot interface, and robot arm motors. In the second mode, for the intermediate level of control, the arm moves according to the input provided by the user, except when interacting with obstacles, in which case the arm slides along the obstacle surface. This mode can be especially useful in teleoperation systems. The third mode presents the highest level of automatic control: in it, the user supplies only the desired target position of the arm, the rest is done by a fully automatic motion planning algorithm (see Chapter 4).

The developed sensor system and associated local normal technique can also be used in the areas not directly related to motion planning such as for the purposes of obstacle characterization and location, and for efficient control of sensor scanning on the skin.

We emphasize that what is crucial for the described planning systems is the right input information, and not the specific sensing media that provides this information. For example, the proximity information used as input to the Step Planning procedure can be from a proximity sensing skin such as in our system, or from a tactile sensor array, vision

system, or CAD based collision detection system.

Chapter 4

Automatic Motion Planning Algorithm for a Whole-Sensitive Arm

4.1 Introduction

Once the experimental apparatus and the step planning algorithm are in place, the last element to be added to complete the motion planning system is an algorithm for transforming the preprocessed sensor data into global planning decisions that guide the arm to its target position. A version of such an algorithm is presented in this chapter.

Work on algorithms for robot arm systems has been an active area of research in recent years [51,52,66]. It has been shown, specifically, that algorithms with proven convergence can be designed for motion planning of simple planar [36,39] and three-dimensional robot arm manipulators operating among unknown obstacles of arbitrary shape [10,14,37,38,58]. In these algorithms, the problem of motion planning for an arm is reduced to that of an automaton operating on the surface of a 2D manifold. Resulting path planning procedures are quite elegant and computationally inexpensive, and require the arm to "slide" along or follow the contours of obstacles encountered along its way. Although at present convergence is guaranteed only for an environment with stationary obstacles, the fact that no preliminary information about the obstacles is needed opens a possibility to apply this approach to time-varying environments.

The Motion Planning algorithm described in this chapter is designed for the robot arm manipulator in the motion planning system, an open kinematic chain consisting of five revolute joints. The first three degrees of freedom are used for motion control, Figure 4-1, the remaining ones, which adjust the orientation of the arm wrist are not controlled directly. The objective of the arm is to move from the starting (S) to the target configuration (T). If T is not reachable at all from S, the algorithm is to conclude so in finite time.

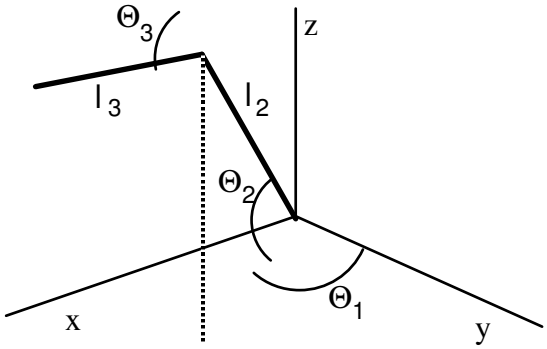


Figure 4-1. Sequence of arm joints and links.

Moving the arm in its *work space* is equivalent to moving a point in the three dimensional *configuration space* whose axes correspond to the three controlled degrees of freedom of the arm. As before, the term *automaton* refers to the point in the configuration space that represents the position of the arm.

If the automaton were operating in a planar (2D) configuration space, a number of maze searching algorithms could be used to solve the motion planning problem [35,36,39]. These algorithms rely on the fact that there are only two directions for the automaton to move around an obstacle - clockwise or counterclockwise.

The difficulty with extending such algorithms to the three-dimensional case is that, in general, there is an infinite number of ways for the arm to maneuver around a 3D obstacle. Previous work on motion planning for 3D robot arms with incomplete information about the environment [38,58] exploit the natural constraints imposed by the kinematics of the arm in order to narrow down the available choices during its navigation. These constraints arise because for these class of arms the last joint (furthest from the base) is prismatic. This allows one to infer some global information about obstacles interacting with the arm based on local contact information, and leads to the narrowing of choices during navigation. Unfortunately for the arm used in this work, no such natural constraints are known. Since no global information can be inferred from the obstacles obstructing link

l_3, the algorithm needs a complete search of the obstacle in the worst case.

The general logic of the Motion Planning algorithm is as follows. The algorithm is divided into three phases, each performing a successively more complex search for the target. The least complex search, performed during Phase 1, can be designed a number of ways; for the sake of simplicity, an existing planar path planning algorithm Bug2 [38] was used. Experience had already been gained using this algorithm during the 2D feasibility study [9,12], which showed that a planar robot arm equipped with an array of proximity sensors mounted on the surface of the arm could be made to operate in an unknown environment.

In Phase 1, the motion of the automaton is restricted to a plane, called the M-plane, which contains points S and T and satisfies some additional conditions, see Section 4.4. The automaton moves on the M-plane towards T, or moves on the intersection of obstacles and the M-plane to maneuver around them.

If T is on a part of the M-plane disconnected by obstacles from S, Phase 1 will not be able to find a path to it, which causes the search to be taken over by either Phase 2 or Phase 3, depending on the results obtained during Phase 1. The purposes of the search during the other phases is to find unexplored regions of the M-plane. If this is successful, Phase 1 is again used to explore the newly found section.

The most complex and thorough search of the C-space occurs during Phase 3, where the complete search is performed. This search is conducted along the intersections of the obstacles in C-space with appropriately spaced planes.

The complete search performed during Phase 3 can be time consuming, and should therefore only be used until all other approaches have failed. Phase 2 presents another way that unexplored sections of the M-plane can be reached. The Motion Planning algorithm recognizes two types of obstacles, Type II, which obstruct link l_2, and Type III, which obstruct link l_3, see Figure 4-1; since link l_1 is of zero length in our arm, no Type I obstacles appear. The intersection curve between two obstacles of different types is also easily

recognized by the sensor system. This is because in this case the arm is in simultaneous contact with a Type II and a Type III obstacle. Note that from the arm's standpoint, it never interacts with more than two obstacles, one of Type II and the other of Type III. This is because the arm has no way of distinguishing two obstacles of the same type.

In general, the intersection curve between two obstacles of different types forms a closed curve with no branches (see Section 4.7.3). If the automaton leaves the M-plane to follow the intersection curve, it will eventually return to the M-plane. To find out whether or not the newly encountered region of the M-plane has already been explored, the procedure checks if this intersection curve was encountered during Phase 1.

The Motion Planning algorithm starts by invoking Phase 1. If this phase fails to find a path to T, execution proceeds to one of the other two phases. If one or more intersection curves between a Type II and a Type III obstacle are found during Phase 1, Phase 2 is used to follow the intersection curves, and to search for unexplored sections of the M-plane. Otherwise, execution proceeds directly from Phase 1 to Phase 3, where the obstacle is explored by intersecting it with parallel planes.

The above three phases cover all possible situations that may occur concerning the obstacles and the location of the start and target positions. The automaton is guaranteed to find a path to T, since it does a complete search in the worst case. In general the entire path usually looks quite reasonable. In those difficult cases where a complete search is necessary, the algorithm's performance is still quite good compared to that of a human operator in similar situations [40].

The following sections in this chapter are organized as follows. First, since the planar algorithm is important to the operation during Phase 1, it is discussed separately in Section 4.2. Then, obstacle types are defined in Section 4.3, and the properties of each of the obstacle types are discussed in Section 4.4. Since real robot arms typically have joint limits, their incorporation into the motion planning system is necessary, which is presented in Section 4.5. The choice of the M-plane affects the type of motion the arm executes while

maneuvering around obstacles. In Section 4.6, the manner in which the M-plane is selected is justified. Then, Section 4.7 presents details of each phase of the Motion Planning algorithm, and the chapter is concluded with Section 4.8.

4.2 The Bug2 algorithm.

The 3D algorithm described below makes use of a procedure for moving an automaton in the plane, called *Bug2* [35]. Briefly stated, under this algorithm the automaton first moves along the straight line, called *M-line* (for "main line"), which connects S and T positions. If an obstacle is encountered, a *Hit Point* is defined at the encounter point, the automaton turns in the prespecified *local direction*, left or right, and begins following the contour of the obstacle. This continues until the M-line is again met at a distance closer to the target than the lastly defined Hit Point; here, distance is defined as the Euclidean distance along the M-line. This encounter point is then defined as a *Leave Point* and the automaton continues its motion towards the target. The algorithm will reach the target if a path exists, or conclude that no such path exists if this is the case. This algorithm was successfully used as the overall Motion Planning algorithm in the 2D experimental system described in [9,12,44].

4.3 Obstacle types

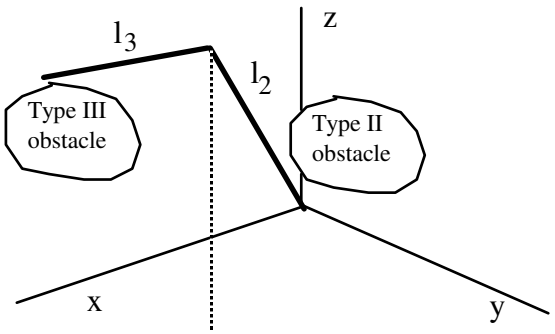


Figure 4-2. Obstacle types.

Depending on their location with respect to the arm links, obstacles can be grouped

into three major categories easily recognized by the sensor system[2] : *Type I*, *Type II*, and *Type III*. Referring to Figure 4-2, obstacles of Type II obstruct link l_2, and obstacles of Type III obstruct link l_3 . Since in the considered arm the first two joints coincide and so link l_1 is of zero length, Type I obstacles never occur. Because the location of every sensor is known to the sensor controlling computer, the type of the obstacles obstructing the arm can be easily identified through the use of a look-up table. Note that in principle the same obstacle can be of different types depending on the link it interacts with. Also, the arm has no way of distinguishing between obstacles of the same type, and so two or more such obstacles will be interpreted as one obstacle. In other words, from the arm's standpoint, it never interacts with more than two obstacles -- one of Type II and the other of Type III.

4.4 Properties of C-space obstacles

Referring to Figure 4-2, observe that a Type II obstacle will, observe that by definition, a Type II obstacle never obstructs link l_3, thus allowing free motion along the Θ_3 axis, see Figure 4-1. Note that if an obstacle could interact with both link l_2 and l_3, it would consist of a union of a Type II and Type III obstacle. We wish to analyze the interaction of each of these portions of the obstacle separately, first for a Type II obstacle.

[2] In Chapter 3, four obstacle types were used. In the Motion Planning algorithm, both Type III and Type IV obstacles are considered Type III since an obstacle obstructing the arm endpoint (Type IV obstacle) is considered equivalent to an obstacle obstructing link l_3 (Type III).

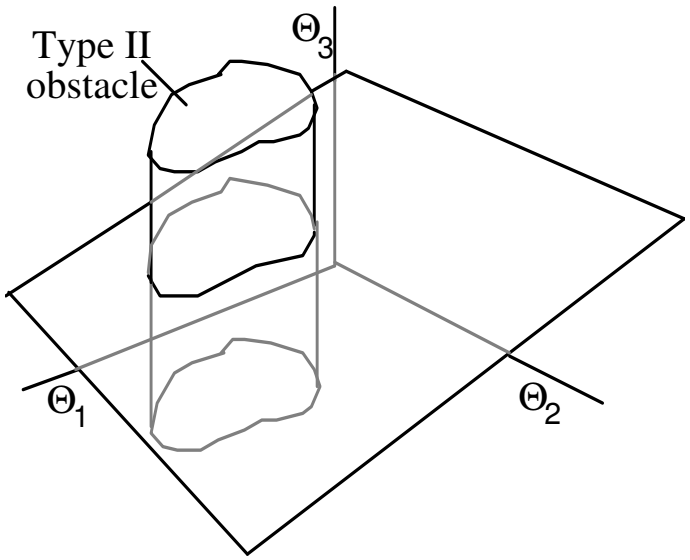


Figure 4-3. A Type II obstacle forms a generalized cylinder in C-space.

In terms of the robot three-dimensional C-space, if the automaton is obstructed by a Type II obstacle, it will still be able to move freely in the Θ_3 direction. This means that Type II obstacles in the C-space have a shape of a *generalized cylinder* whose axis is parallel to the Θ_3 axis, refer to Figure 4-3. A generalized cylinder has the property that the cross section in the plane perpendicular to its axis is a simple closed curve that does not change along the axis. Therefore, when the arm is obstructed by a Type II obstacle, the motion of the automaton in the Θ_3 direction is of no use as far as obstacle avoidance is concerned.

Unfortunately, the topology of a Type III obstacle exhibits no such distinct characteristics, and so in the worst case this specific arm may be forced to carry out a complete search of the obstacle surface in order to find a path to the target position.

4.5 Effect of joint range limits

Because in real arm manipulators the range of motion of each joint is typically limited, it is desirable to incorporate the joint limits into the obstacle model. It is easy to show [58] that in terms of their effect on motion planning and obstacle avoidance the joint

limits produce in the C-space planar "obstacles" fully equivalent to images of physical obstacles. For example, the upper limit of joint Θ_3 produces a horizontal plane parallel to the plane of axes Θ_1 and Θ_2. This plane will behave like an obstacle limiting the C-space from above. Based on this observation, the joint limits can be simply treated by the Motion Planning algorithm as artificial planar obstacles of the respective types -- that is, of Type II in case of joint j_1 and joint j_2, and Type III in case of joint j_3.

When the arm encounters a joint limit, the contour of the corresponding obstacle in the C-space is followed just as if it were a real obstacle. This resulting exploration of the joint limits is not very productive however, since the arm already has complete knowledge of its own joint limits. It is more interesting and productive to follow instead the intersection of the obstacles formed by the joint limits, and the real obstacle currently being followed, before the joint limit was hit. Please note that this added heuristic does not affect the algorithmic performance of Phase 1. It has been added to improve the motion of the arm. More details of this motion will be explained below in Section 4.7.1, and examples of this motion can be found in the experimental results, Section 5.3.

Such motion would require the automaton to occasionally leave the M-plane, which raises the question of preserving the convergence properties of the Bug2 algorithm. Since the automaton moves on the surface of a box, whose one side is formed by the M-plane, and the other sides by the joint limit obstacles, convergence of Bug2 is still preserved. The proof is similar to the one in [39], where it was shown that a rectangle can be topologically mapped onto the surface of a torus by pasting the appropriate sides of the rectangle together. This showed that motion planning on the surface of a torus is equivalent to motion planning on a plane. In our case, the box can also be constructed out of a planar surface by pasting the correct sides together. The pattern needed for the box in our case is shown in Figure 4-4. Motion planning on our box is thus equivalent to motion planning on a plane, thus the convergence of Bug2 is preserved.

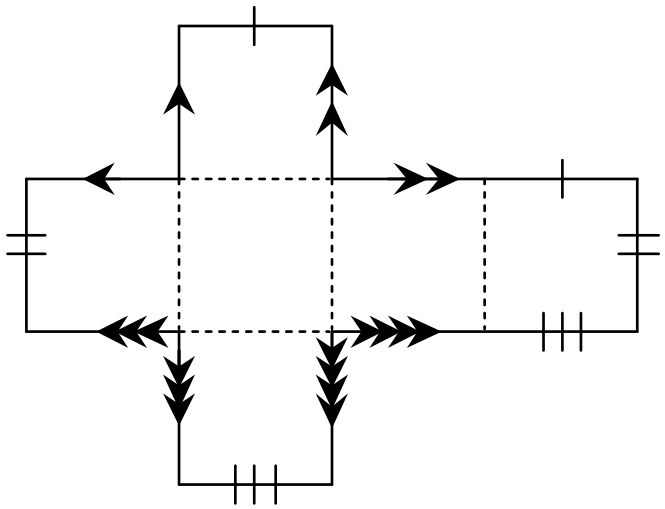


Figure 4-4. Constructing a box from a planar surface. Sides with the same number of dashes or arrows should be pasted together.

4.6 Basic obstacle avoidance strategy

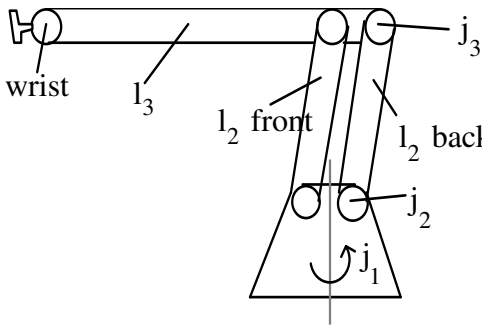


Figure 4-5. Sketch of the robot arm; j_1, j_2, j_3 are arm joints; l_2, l_3 - arm links; link l_1 is of zero length.

A drawing of the robot arm, Figure 4-5, shows that link l_2 is of trapezoidal construction. Because of this, Θ_3 can not be made zero, and the arm thus permanently stays in the "up elbow" configuration. This property can be incorporated into an obstacle avoidance strategy, by realizing that if the arm hits an obstacle on its way to the target, it may be possible to avoid it by increasing both Θ_2 and Θ_3, which causes the arm to "fold up" to a more compact configuration, see Figure 4-6. The use of this heuristic does not affect the convergence of the algorithm, but is added to improve the performance of the

arm while maneuvering around obstacles.

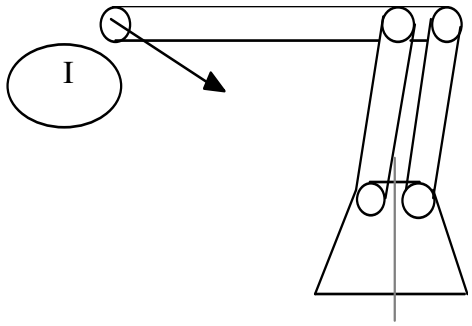


Figure 4-6. Direction of motion of the arm endpoint to fold the arm up.

The simplest way to accomplish the folding of the arm is by maintaining a constant value for $\frac{d\Theta_3}{d\Theta_2}$ throughout the motion. An easy way to accomplish this is to move the automaton on a plane in the three-dimensional C-space. The normal of this plane is chosen by utilizing the local normal generation procedure described in Section 3.3. We assume that the arm is at a position in the middle of its joint range, and its endpoint is in contact with an obstacle, such that sliding along this obstacle, I, Figure 4-6 causes it to fold up. The local normal to this imaginary obstacle thus generates the required motion of the arm. We evaluate expression (3.20) in Chapter 3, and define the following local normal for our particular arm:

$$s = \begin{pmatrix} 0 \\ 0.3951 \\ -1 \end{pmatrix}$$

Although *s* was found for a particular placement of the obstacle and the robot arm, it can still be used over the entire joint range of because of the limited travel of the Θ_2 and Θ_3 axis. Thus moving perpendicular to *s* causes the arm to fold up or extend depending on whether or not Θ_2 and Θ_3 are increasing or decreasing. Increasing both joint angles causes the arm to fold up, decreasing both has the opposite effect.

4.7 Path planning algorithm

4.7.1 General

The basic idea of a planar algorithm is used -- clockwise or counterclockwise maneuvering around obstacles -- for 3D motion planning, by constructing the 3D trajectory of the automaton in 3D C-space as a combination of path segments along 2D surfaces. The motion of the automaton is confined in free space - that is, when no interaction with obstacles takes place - to the *M-line* which is the straight line connecting start and target points S and T. Then, if the motion along the M-line becomes impossible because of interfering obstacles, the automaton will attempt to maneuver around the obstacles while confining its motion to an appropriately chosen plane. The choice of the plane has to satisfy two requirements: it has to guarantee convergence and result in reasonable (from the standpoint of the length of generated paths) trajectories.

This plane, called *M-plane*, for "main-plane" should obviously contain the M-line. In principle, such a plane can be chosen in a variety of ways. A consideration that guided our choice was the desire to minimize, on the average, motion in the direction of *s*, found above in Section 4.6. Moving perpendicular to *s* causes the arm to fold up or extend, which is useful to avoid the obstacle, for reasons mentioned in Section 4.6. The plane in C-space that satisfies these requirements is defined by the M-line and the line t that is the perpendicular to both *s* and the M-line. This plane always exists and is unique, except in the degenerate cases when the M-line intersects or coincides with *s*. In the former case, we can still find the direction of the line t by taking the cross product of *s* and M-line; in the latter case, the choice of t is arbitrary as long as t does not coincide with the M-line.

Similar to the Bug2 algorithm, a *local direction* (left or right) is defined for the M-plane. The local direction is chosen such that the arm folds up when an obstacle is encountered, or in other words, such that the turn is in the direction of the positive Θ_3 axis so that the arm folds up when an obstacle is encountered. This local direction is always

defined unless the Θ_3 axis is perpendicular to the M-plane, in which case the local direction can be chosen arbitrarily, since the arm does not move in the Θ_3 direction while on the M-plane.

While moving on the M-plane using the Bug2 algorithm, if a joint limit is encountered, its corresponding obstacle plane is followed just as if it were a real obstacle. However, from the practical standpoint, exploring and following the contour of a joint limit is considered unproductive and unnecessary since the arm already has complete knowledge of the obstacle planes formed by its joint limits. To improve the performance of the algorithm, the arm follows instead a detour, which is the intersection curve formed by the obstacle representing the joint limits, and the obstacle being followed just before the joint limit was encountered. This intersection curve is followed until the M-plane is once again met, at which point the arm resumes following the intersection curve of the M-plane and the real obstacle. The convergence of the Bug2 algorithm is not affected as indicated in Section 4.5. An example of this motion is given below.

Consider the motion of the automaton in the example shown in Figure 4-7. Starting from S, the automaton moves towards T until it hits the obstacle at point *a*. It then defines a Hit point, turns in the local direction (i.e., the direction of the positive Θ_3 axis), and follows the intersection curve of the obstacle and the M-plane. At point *b*, the arm encounters the minimum Θ_1 joint limit. Instead of following the intersection curve of the joint limit planes and the M-plane in a counter-clock wise fashion (shown as the thin dotted line), the automaton follows the intersection curve of the joint limit and the real obstacle to point *c*. Once the M-plane is once again encountered, at curve *cd*, the intersection curve of the M-plane and the obstacle is followed as before, until the M-line is once again encountered at point *d*. The automaton then defines a Leave Point, leaves the obstacle, and moves towards the Target.

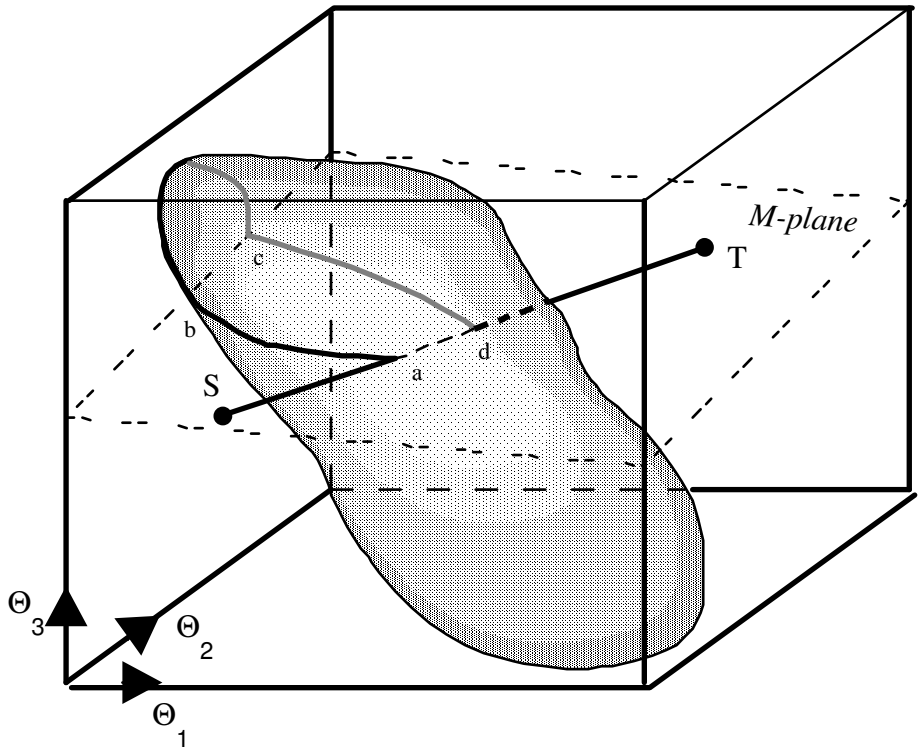


Figure 4-7. Example of following the intersection of a joint limit and a real obstacle (from point *b* to point *c*).

Note that while the automaton is following the joint limits it is in contact with only two kinds of obstacles - the joint limit planes and the real obstacles; their intersection curve in C-space is a closed curve with no branches. For there to be a branch, three or more different kinds of obstacles must touch at a common point. This is not possible since just two kinds exist. Therefore, once this intersection curve is followed, the automaton will eventually return to the M-plane.

When no path to the target can be found in the M-plane, the automaton will be forced to move off the M-plane until it either reaches the target, or finds a previously unexplored region of the M-plane. One consequence of having only two types of obstacles (Type II and Type III) is that their intersection curves form closed curves with no branches, similar to the intersection curve between obstacles and the joint limit planes. This fortunate property guarantees that the automaton will easily recognize some characteristic points (e.g., intersections of the said intersections with the M-plane) and will never fall into an

infinite loop. Such intersection curves thus all lie in (2D) surfaces of C-space obstacles and are in general non-planar; they form potential path segments to connect regions of the M-plane that cannot be reached one from another within the M-plane. If, while moving along the M-plane, the point automation encounters an intersection curve between a Type II and Type III obstacle, it can follow this intersection curve in order to move to another region of the M-plane.

The path planning algorithm consists of three phases; these are explained in more detail in the following sections. Briefly, in *Phase 1*, the automaton moves within the M-plane, or on the intersection curve of an obstacle and a joint limit plane. If Phase 1 terminates and the algorithm cannot reach T, *Phase 2* or *Phase 3* is used to continue the search. The purpose of Phases 2 and 3 is to guide the automaton off the M-plane. During Phase 2, an intersection curve between two obstacles found during Phase 1 is followed in the search of T or an unexplored region of the M-plane. If such a region is found, Phase 1 is used again and the process repeats. Otherwise, Phase 3 is invoked to carry out a complete search of the surface of the current obstacle. This process can result in reaching T or in finding another unexplored region of the M-plane, and the process repeats. Typically, Phase 3 will be needed only if the automaton needs to pass through an opening that has no clues leading to it, such as the opening in a bottle-like obstacle in C-space. Depending on the task and environment, the order of the actual phase execution may change. Furthermore, a given phase may or may not appear during the motion. Very often Phase 1 is sufficient, and the resulting path will look quite reasonable. Only in difficult situations will a complete search be necessary, and in these cases even a human subject will have a difficult time navigating around the obstacles, with the performance not better than of our algorithms [40].

While following the intersection of obstacles and the M-plane, whenever the automaton makes a transition from a Type II obstacle to a Type III obstacle, or reverse, the location where this occurs is noted. These locations, together with points S, T, and Hit and

Leave points (see Section 4.2), form *nodes* of the continuously growing *Connectivity graph*. The *edges* of the graph are segments of the M-line, segments of the intersection curves between the M-plane and obstacles, between obstacles and joint limit planes, and between obstacles of different types.

Each of the nodes S and T in the Connectivity graph has one adjacent edge; each node formed by a Hit or Leave point has three adjacent nodes -- a segment of the M-line and two adjoining segments of the obstacle/M-plane intersection curve; and each of the remaining nodes has four adjacent edges formed by the obstacle/M-plane intersection curve, on the one hand, and the intersection curve between two obstacles or an obstacle and a joint limit plane, on the other hand. From the practical standpoint, since the size of the Connectivity graph is proportional to the number of obstacles in the workspace, it is usually relatively small -- in a complex setup, for example, it would be rather difficult to generate more than 10-15 nodes.

Considering motion planning as the exploration of the Connectivity graph, if the nodes generated during Phase 1 fail to complete the task, Phase 2 further explores the unknown edges in the graph. Unless the target T is spotted, the whole graph may have to be explored during the search. If both Phase 1 and Phase 2 fail to find a path to the Target, a complete graph search is used to find a path. This is accomplished by Phase 3, which operates by exploring the intersection curves between the obstacles in the C-space and appropriately spaced planes. If a previously unexplored region of the free space is found, the algorithm repeats to search this new portion of the C-space.

During Phase 3, the automaton defines a *Mark point* on each of the intersection curves between the obstacles in the C-space and the appropriately spaced planes mentioned above. These points are entered into the *Mark point graph*, which serves as a reference path to allow the automaton to return to a previously explored location on the obstacle, and it also serves as the means to ensure that all parts of the obstacle are explored, see Section 4.7.4.3. This graph is used only during Phase 3.

During Phase 1 and Phase 2, only the Connectivity graph needs to be stored. In Phase 3, the storage needs are increased because the path of the automaton is sampled and stored at a regular interval, and because the Mark point graph is also stored. In a relatively complex environment, one can expect a few hundred points for the path sampling and 40-50 nodes for the Mark point graph. Below, each phase of the algorithm is presented in more detail, followed by the summary of the Motion Planning algorithm.

4.7.2 Phase 1 of path planning

During Phase 1, the automaton's path includes three types of motion: from S towards T along the M-line, and along the intersection curves of two kinds - between an obstacle and the M-plane, or between a joint limit plane and an obstacle. First, the automaton completes processing of the M-plane region that is being currently explored. Only after it becomes clear that the task cannot be completed within this region, the automaton will attempt to find another section of the M-plane by invoking another phase. Algorithmically, the procedure for planning a Phase 1 motion is analogous to the Bug2 algorithm mentioned before, except the automaton does not follow the joint limit planes when a joint limit is encountered. In the latter case, the intersection curve of that joint limit and the current obstacle is followed until the M-plane is once again encountered, as shown in the example in Figure 4-7.

If the lastly defined Hit Point is encountered again, before the next Leave Point is defined, the Phase 1 procedure concludes that no path to T can be found within the current M-plane region (for proof, see [35]). This means that the obstacle the automaton is following may be dividing the M-plane into disconnected regions, and so the only option left is to look for another connected region of M-plane. This can be done by proceeding to another phase of the procedure, Phase 2 or Phase 3.

4.7.3 Phase 2 of path planning

Phase 2 concentrates on generating those motions outside the M-plane that are short of a complete search. As mentioned above, the intersection curve between two obstacles, one of Type II and the other of Type III, can easily be distinguished by the arm's sensors. In general, this curve presents a closed curve with no branches, see below for exceptions. If followed by the automaton, the intersection curve will lead to another section of the M-plane. To find out whether or not the newly encountered region of the M-plane has already been explored, the procedure simply checks whether the intersection point between the seam and the M-plane region is already in the graph.

The intersection curve between a Type II and Type III obstacle has in general no branches. An exception is for example when four obstacles touch at one point, see Figure 4-8. The intersection curve of the Type II and Type III obstacles, shown as the thick line, has a branch in the middle of the combined obstacle, where the wedges touch each other. This case is a rare occurrence, since all the obstacles involved must touch at a single point.

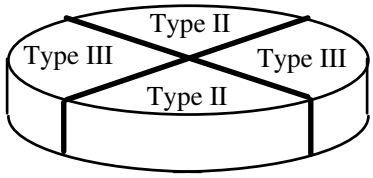


Figure 4-8. A rare example of a branch in the intersection curve between a Type II and Type III obstacle (shown with the thick line). The branch is in the center of the pie, where the four obstacles touch each other.

Every node in the Connectivity graph that does not lie on the M-line is connected to two edges that lie in the M-plane, and two edges that belong to the intersection curve between a Type II and Type III obstacle. The two former edges are known, since they are part of the path that the automaton traveled during Phase 1 when the node was found. The two latter edges are explored during Phase 2. Nodes in the Connectivity graph that are on the M-line are connected to two edges that lie in the M-plane, and a section of the M-line.

At the start of Phase 2 the automaton is at the lastly defined Hit Point. If the current node has unexplored edges they are explored first. Otherwise, the search proceeds to any known node with an unexplored edge that is also on the same connected region of the M-plane as the automaton. Failing this, the search expands to any other known node in the graph that has an unexplored edge. If there are no nodes in the Connectivity graph with unexplored edges, Phase 2 terminates. Once a node is found with an unexplored edge, a path to that node using the already explored edges is found using a standard depth-first graph search technique [55].

Once at the node, the automaton leaves the M-plane to follow the unexplored edge formed by the intersection curve between the Type II and Type III obstacle until it returns to the M-plane. By definition, this return point occurs at a node in the Connectivity graph. If this node is already known, meaning that this section of the M-plane had already been explored using Phase 1, the automaton searches for another node with an unexplored edge. Otherwise, Phase 1 is used to explore this new region of the M-plane. Phase 2 terminates as soon as T is found, or after all the known nodes on the Connectivity graph have been explored.

4.7.4 Phase 3 of path planning

4.7.4.1 General

Phase 3 is responsible for the complete search of certain sections of C-space. This may be needed for instance when the automaton has to pass through a hole of a Type III obstacle shaped liked a bottle. If no intersection curves or other clues lead to the mouth of the bottle, a complete search of the surface of the bottle must be carried out in order to find its opening. Unlike an exhaustive search, which may theoretically require infinite time, the complete search in our case is finite and relatively short due to an effective use of data from the sensitive skin. Due to the finite sensing range of individual sensors of the sensitive skin, at a given position of the automaton in C-space there is a finite region around it that

can be "sensed" for obstacles. Though the exact shape and size of this region is a complex function of the arm position, the sensor range, and the obstacle albedo, a minimum sphere inscribed into this region can simply be assumed and its radius R_s calculated. As the automaton moves along the surface of an obstacle, the sensor range allows it to explore the adjacent swath across the surface of the obstacle, enabling a finite number of passes over the surface of the obstacle to explore it all. This is implemented by intersecting the obstacle with appropriately spaced planes. The automaton thus explores the surface of the obstacle by moving along the intersections between such planes and the obstacles.

In Phase 3, the automaton moves along the surface of the obstacle until an unexplored section of M-plane is found. Then Phase 1 is again invoked to find out if T is reachable. Since the M-plane divides the obstacle into two halves, Figure 4-9, in general both halves must be explored during Phase 3.

Phase 3 starts at the last Hit Point defined during Phase 1. The reason for this is as follows. The lastly defined Hit Point lies on the intersection curve between the M-plane and an obstacle. If a path to M-plane has not been found during the previous stages, this means that the said intersection curve must be forming a complete ring around T. Thus the obstacle completely encloses T, and it is its surface that has to be explored during Phase 3, Figure 4-9.

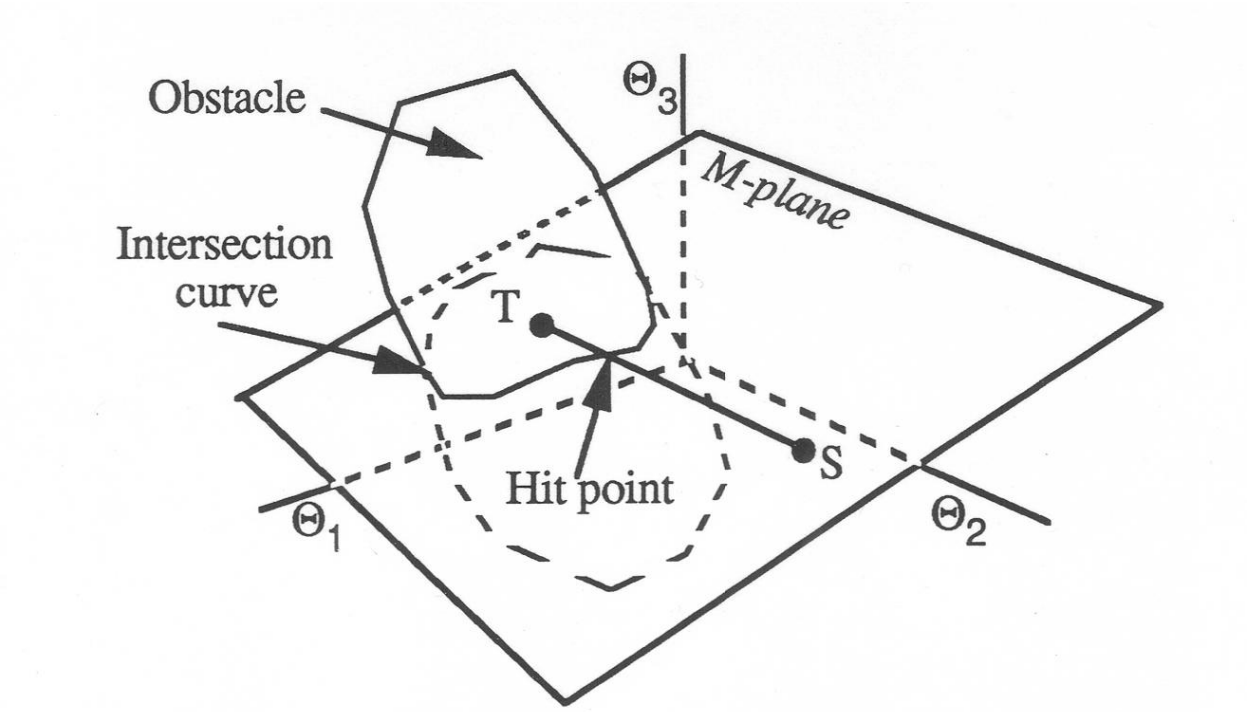


Figure 4-9. Example of an intersection curve between
the obstacle and the M-plane that completely surrounds T.

Below various aspects of the Phase 3 exploration strategy are explained followed by a summary of the overall progression of the phase. Afterwards, the entire Motion Planning algorithm, including all its phases, is summarized in Section 4.7.5.

4.7.4.2 Basic obstacle exploration strategy in Phase 3

An obstacle is explored by intersecting it with planes that have a spacing of R_s. These planes are designated M_i, i=0, 1, 2,..., with the original M-plane explored during Phase 1 being M_0. While the automaton is exploring the M_i-plane, its position is sampled and stored at a regular interval. In a typical scene with obstacles, this produces on the order of 150 points per plane. Data points collected on this plane as well as those collected during the exploration of the M_{i-1}-plane are retained for the branch check explained below.

When the automaton first starts moving on M_i (at the start of Phase 3, i=0), it remembers the current location as a *Mark point*, and enters this as a *Mark point node* in the

continuously growing *Mark point graph*. Nodes generated in this manner will have in general two adjacent edges leading to the previous and the (unknown) next Mark point respectively. Nodes with three or more edges are defined at a branch in the obstacle, see Section 4.7.4.3. An example of an edge in the Mark point graph is the path of the automaton between the points *a* and *e* in Figure 4-10.

When the automaton hits the Mark point on the M_i-plane again, it has explored this intersection with the obstacle, and needs to move to the next plane, M_{i+1}. It does so by intersecting the obstacle with planes, called the I-planes, that are perpendicular to the M-planes and that also have a spacing of R_s. I-planes guide the motion of the automaton between two successive M-planes; they are designated I_j, j=0, 1, 2,... When moving on the intersection of an I-plane and the obstacle, the automaton can either hit the next M-plane, M_{i+1}, or depending on the obstacles shape, hit the current M-plane, M_i. In the former case, the automaton explores the intersection of the obstacle and M_{i+1}-plane as it did before on the M_i-plane. In the latter case, it moves towards the next I-plane, and repeats the attempt to move to the M_{i+1}-plane. The next two examples illustrate this process.

Example 1. M_{i+1}-plane is reachable from M_i-plane.

An example of this case is shown in Figure 4-10. Assume that the automaton has followed the intersection curve (not shown) of the M_i-plane and the obstacle, and has returned to the Mark point defined previously at point *a*. It now attempts to move to the M_{i+1}-plane via the I-planes. Note that the placement of the I-planes is unimportant, as long as their spacing is R_s. After encountering the plane I_0 at point *b*, the automaton follows the intersection of I_0 and the obstacle in an attempt to reach the M_{i+1}-plane. It does not succeed and returns to the M_i -plane instead at point *c*. It concludes that it is not useful to use the I_0-plane to move to the M_{i+1}-plane, and then moves on the intersection of the M_i-plane and the obstacle towards the I_1-plane, and encounters I_1 at point *d*. While moving on the intersection of I_1 and the obstacle, it does succeed in hitting the M_{i+1} -plane at point

e. At this location the automaton defines a Mark point, and explores the M_{i+1} -plane as it did before with the M_i -plane. Note that the Mark point nodes associated with the M_i- and M_{i+1}-planes have been entered into the Mark point graph, together with an edge that connects the two nodes. This edge corresponds to the path of the automaton between points *a* and *e*.

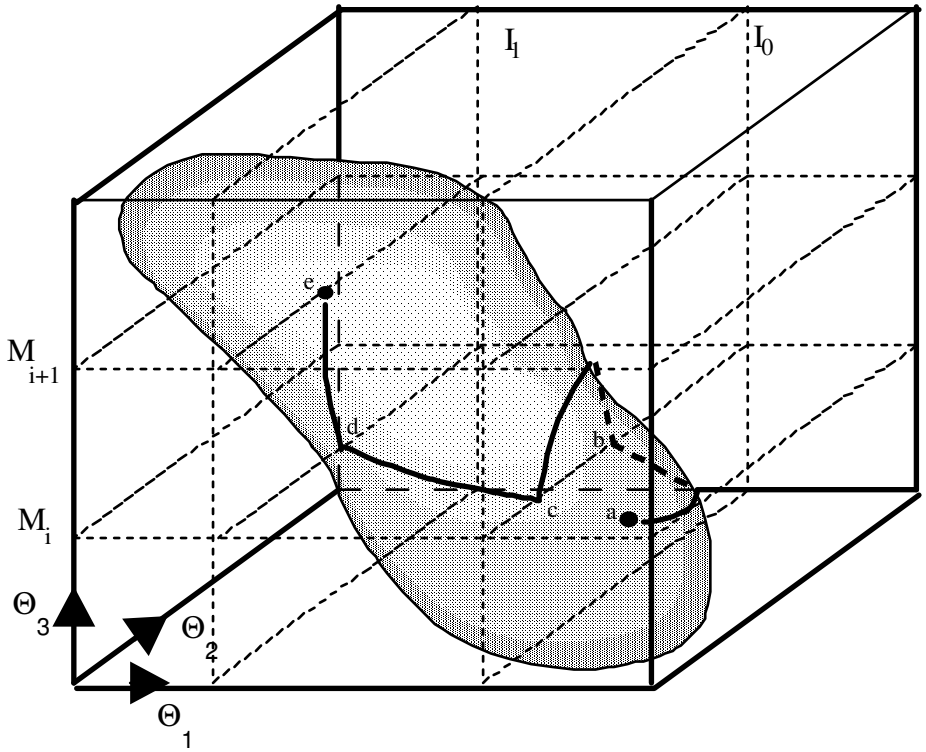


Figure 4-10. Example: the automaton moves from the M_i-plane to the M_{i+1}-plane. The whole path, *abcde*, is on the surface of the obstacle. The automaton starts at point *a*, moves to *b* along the the M_i-plane, continues in I_o-plane until *c*, then again in M_i-plane until *d*, and then in I_i-plane until it reaches M_{i+1}-plane at point *e*.

Example 2. M_{i+1}-plane is not reachable from the M_i-plane.

In this case, the M_{i+1}-plane has no intersection with the obstacle, refer to Figure 4-11. Starting at the Mark point at point *a*, the automaton first moves along the intersection between the obstacle and the M_i-plane until the I_0-plane is hit at point *b*. It then follows

the intersection of the obstacle and I_0, but does not succeed in hitting the M_{i+1}-plane. After encountering the M_i-plane again at point *c*, the automaton heads for the I_1-plane, which is encountered at point *d*. Moving on the intersection curve of the obstacle and the I_1-plane, it once again encounters the M_i-plane at point *e*, and then moves towards the I_2-plane. The automaton arrives at point *d*, and concludes that I_2-plane has no intersection with the obstacle, and that it has therefore explored the part of the obstacle above M_i-plane. Note that in this case the Mark point node from the M_i-plane has only one adjacent edge, which connects it to the Mark point node from the M_{i-1}-plane.

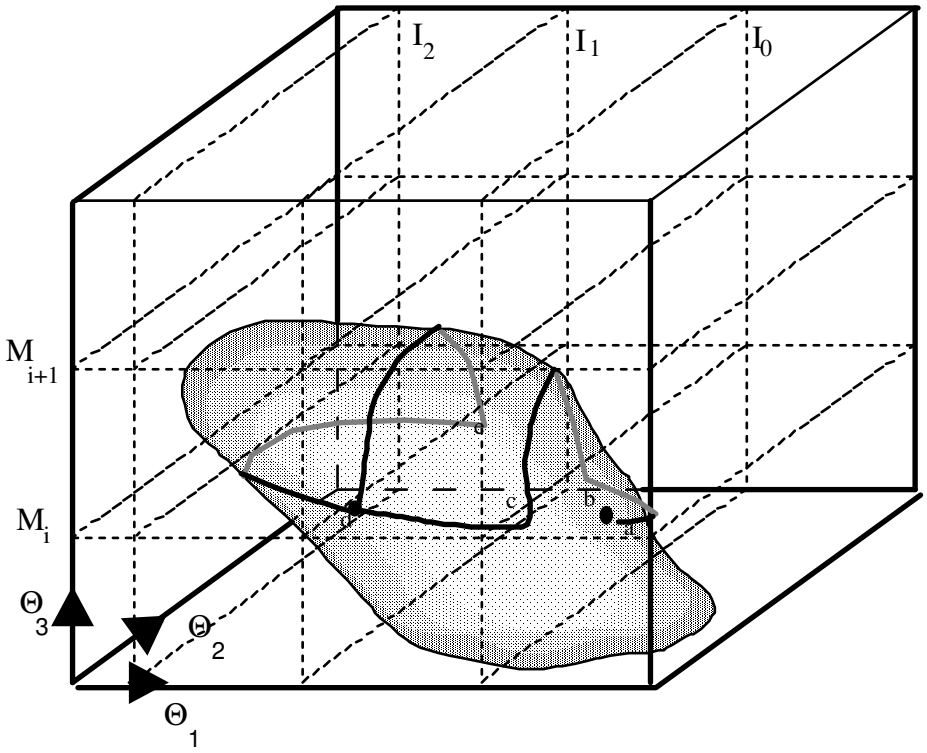


Figure 4-11. Example: the automaton's failed attempt to move from the M_i-plane to the M_{i+1}-plane, because it is not reachable along the obstacle surface.

Recall that the purpose of the search during Phase 3 is to find unexplored regions of the M_i-plane while moving to the M_{i+1}-plane. In an example shown in Figure 4-12 the automaton defines at point S a Mark point and moves in the direction of the arrow to

explore the intersection curve of the obstacle and the M_{i-1}-plane. After this plane has been finished, the automaton moves on the intersection curve of the obstacle and the I_0-plane, until the M_i-plane is encountered at point *a*. The exploration repeats on M_i-plane. After that, the automaton attempts to move to the M_{i+1}-plane using the I_0-plane, but instead follows the intersection curve of the obstacle and the I_0-plane into the hole of the obstacle to find an unexplored region of the M_i-plane at point *b*. To find out if T is reachable from this point, Phase 1 is used to move straight to T. In this case, T is reached, and execution ends. If T were not reached, the automaton would have returned to point *b*, to resume exploring the inside of the obstacle to find a path to T, and to finish the complete search of the obstacle.

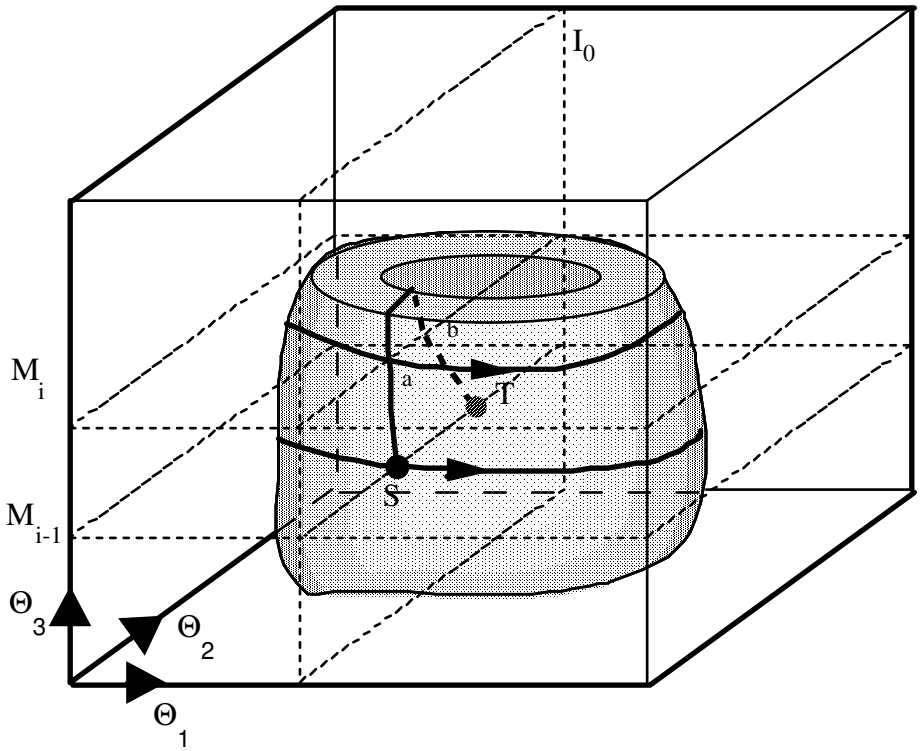


Figure 4-12. While attempting to move to the M_{i+1}-plane, the automaton encounters an unexplored section of the M_i -plane at point *b*.

4.7.4.3 Exploration of branches in the obstacle

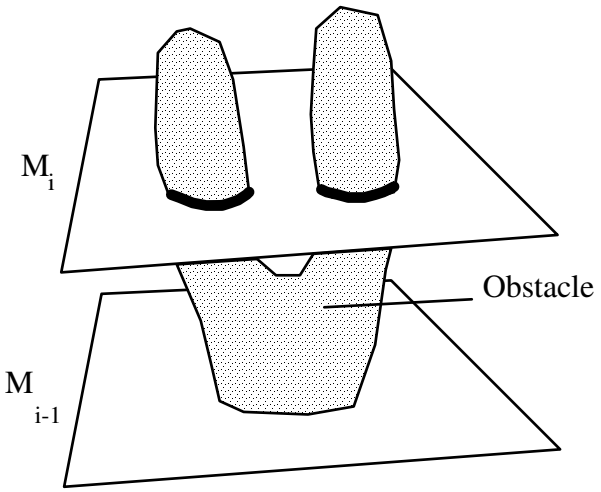


Figure 4-13. Here, the intersection (shown in the thick line) of the obstacle and the M_i -plane forms two disconnected regions.

For concave obstacles, it is possible for sections of the intersection of an obstacle and the M_i -plane to be disconnected due to a branch in the obstacle. An example of this is shown in Figure 4-13; the intersection of the M_i -plane and the obstacle, shown as the thick line, consists of two disconnected sections.

In this case, the automaton may have to eventually explore all branches. The corresponding condition is detected by performing a check, called the branch check, every time exploration of the M_i-plane has been completed. During the motion of the automaton its position is sampled and stored at some interval. After the M_i-plane has been explored, the branch check compares the paths on the M_i-plane and the M_{i-1}-plane. Since the automaton covers a swath while exploring the obstacle's surface, we need to detect a sufficiently large gap between the two intersection curves to conclude that there is a branch in the obstacle, Figure 4-14.

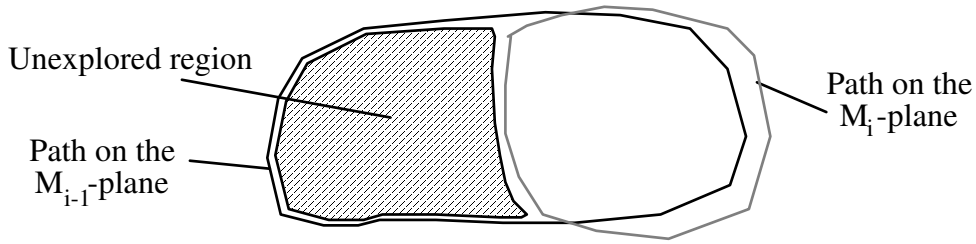


Figure 4-14. Paths of the automaton projected onto the M_i-plane in the example of Figure 4-13. The unexplored region is due to the branch in the obstacle.

The branch check is performed by calculating the distance between the two curves at every sampled point on the M_i-plane. A branch in the obstacle is detected if at any point the distance between the two intersection curves is greater than $2R_s$ (a value decided upon experimentally), Figure 4-14. Because there is a branch in the obstacle, there will also be a branch in the Mark point graph. Recall that a regular Mark point has two edges connected to it, see Section 4.7.4.2. In the case of a branch, the Mark point node from the M_{i-1}-plane has three adjacent edges, one that connects to the Mark point node from the M_{i-2}-plane, and two that connect to the two Mark point nodes from the M_i-plane. In the case of a branch in the Mark point graph, unknown edges can be explored in a number of ways, for example, by a depth-first or a breadth-first search [55]. Note that for the purposes of this check, only the paths on the M_i and the M_{i-1} plane need to be retained in memory; in other words, the path on the M_{i-2} can be discarded as soon as the M_i-plane is encountered.

Once the automaton concludes that there is a branch in the obstacle, the area is explored in a manner similar to the previous strategy for moving from the M_i- to the M_{i+1}-plane, which is to divide the area to be explored into slices using I-planes with a spacing of R_s. The area to be explored is the gap between the intersection curves on the M_i and M_{i-1} -plane, shown as the dashed region in Figure 4-14. While moving on the obstacle, if a previously unexplored section of the M_i or M_{i-1} -plane is encountered, obstacle exploration is resumed, by defining a new Mark point and exploring the intersection of the obstacle and the M_i or M_{i-1} -plane.

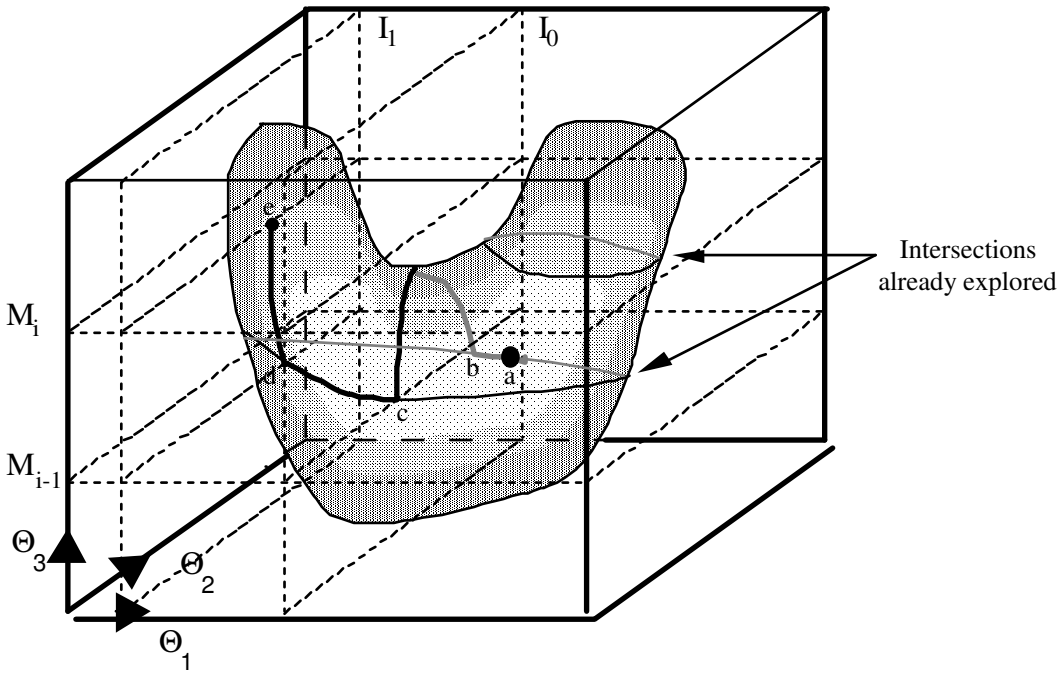


Figure 4-15. Exploration of a branch in the obstacle.

This is illustrated in an example shown in Figure 4-15, using the same obstacle as Figure 4-13. The motion of the automaton starts at point *a*: the automaton moves in the M_{i-1}-plane to the I_0-plane (point *b*) and follows the intersection of I_0 and the obstacle. It fails to hit the M_i-plane and returns to the M_{i-1}-plane instead at point *c*. It then moves on the intersection of the obstacle and the M_{i-1}-plane until the I_1-plane is hit at point *d*. After following the intersection of I_1 and the obstacle, it encounters the unexplored intersection curve of the M_i-plane and the obstacle, point *c*.

The Motion Planning algorithm explores the unknown edges in the Mark point graph in a breadth-first fashion. For the example in Figure 4-15, the automaton finishes the M_i-plane by exploring the newly found, at point *c*, intersection of the obstacle and the M_i-plane. It then continues by moving upward and exploring the left part of the obstacle by intersecting it with the M_{i+1}-, M_{i+2}-plane etc. Once this left part has been explored, the automaton returns to the M_i-plane to explore the unknown edge of the associated Mark

point node, and finishes exploring the part of the obstacle on the right.

4.7.4.4 Returning to a point on the surface of the obstacle

As the automaton moves over and explores the obstacle's surface, there may be a need to return to a previously explored location. This can be, for example, a branch in the obstacle. To avoid storing every past point on the surface of the obstacle, the procedure uses the Mark points to reference the previous path.

An example of moving from one Mark point to another is given using Figure 4-10, the Mark point at point *a* is connected to the Mark point at point *c* via the M_i- and I_1-plane. If the automaton wishes to move from point *a* to point *e*, it moves along M_i until the I_1-plane is met (at point *d*), it then moves on the I_1-plane until it reaches point *e*. To move in the opposite direction, from point *e* to point *a*, the automaton reverses the above steps.

If the automaton is required to move a larger distance over the surface of the obstacle (past several Mark point nodes), a path thru the Mark point graph is searched using standard graph search techniques. The automaton then moves over the obstacle by moving from one Mark point to another.

4.7.4.5 Overall progression of Phase 3

At the start of Phase 3, the M_i-planes and the I_j-planes are defined; the spacing between subsequent M_i-planes and the I_j-planes respectively is R_s. The M_i-planes are parallel to the M-plane defined in Section 4.6, with the M-plane coinciding with the M_0-plane, while the I_j-planes are perpendicular to the M_i-planes. The automaton begins by defining a Mark point and puts this into the Mark point graph. It then moves on the intersection of the M_0-plane and the obstacle until the current Mark point is again encountered. The automaton then concludes that the M_0-plane has been explored, and attempts to move to the M_1-plane by using the I_0-plane as the guide, as described in Section 4.7.4.2. If the M_1-plane is reached, the automaton continues to explore the obstacle by

defining another Mark point, and inserting this into the Mark point graph. The automaton then explores the intersection of the obstacle and the M_1-plane. If the automaton does not succeed in moving to the M_1-plane via the I_0-plane, it tries to use the I_1-plane next, proceeding as described in Example 2 of Section 4.7.4.2.

Each Mark point has in general two unknown edges when it is defined, one edge is connected to the Mark point defined previously, and the other is left unknown until its connectivity to the next Mark point is determined. A Mark point has more than two edges if it is defined at a branch in the obstacle, as described in Section 4.7.4.3. Thus in the example above, the connectivity between the Mark point on the M_0-plane and the one on the M_1-plane is entered into the Mark point graph when the latter Mark point is generated. Subsequent Mark points and their connectivity are also entered into the Mark point graph. An example between an edge between two Mark points is the interval between points *a* and *e* in Figure 4-10.

The exploration of a part of the obstacle continues until the automaton does not succeed in moving from the M_i-plane to the M_{i+1}-plane while moving on the surface of the obstacle. An example of this case is given in Example 2 in Section 4.7.4.2. A Mark point node with an unknown edge, generated at a branch in the obstacle, is then selected from the Mark point graph, and a path to it is found using standard graph search techniques. The automaton moves to the selected Mark point, the procedure is described in Section 4.7.4.4, and explores the unknown branch in the obstacle. This continues until no nodes in the Mark point graph with unknown edges remain. If this occurs, and T has not been found, it can be concluded that the complete search is finished and no path to T exists.

4.7.5 Motion planning algorithm: Summary

Initiation: Define the M-line, M-plane, Connectivity graph, Mark point graph, and the counters *i*,*j*,*h*,*m*, and the variable *t*. Denote O to be the current obstacle. Set t=0. Go to Phase 1.

Phase 1 (Motion along the M-plane).

Step 1. Move toward T, checking for these conditions:

1. Arrive at T. The procedure terminates.
2. Meet an obstacle. Define a Hit Point; turn in the local direction; go to Step 2.

Step 2. A. Follow the obstacle boundary in M-plane, checking for one of these conditions:

1. Arrive at T. The procedure terminates.
2. Encounter the M-line. If the distance between the current position and T is shorter than that between the lastly defined Hit Point and T, define a Leave Point and go to Step 1; otherwise, continue - go to A.
3. Encounter an intersection curve between a Type II and a Type III obstacle. Mark this location as a node in the Connectivity graph; continue - go to A.
4. Encounter the lastly defined Hit Point before the next Leave Point is defined. Phase 1 cannot find a path to T on this section of the M-plane. If t=0 go to Phase 2, else go to to Step 8 of Phase 3.
5. Meet a joint limit. Go to Step 3.

Step 3. Move on the intersection of the joint limit plane and the obstacle until the M-plane is again encountered; then go to Step 2.

Phase 2 (Motion along the intersection curve between a Type II and a Type III obstacle).

Step 1 Identify a node in the Connectivity graph with a yet unexplored edge in the current M-plane region or, if none are found, in another M-plane region. If none are found, Phase 2 cannot be used to find a path to T, go to Phase 3. Otherwise, move to the node and go to Step 2.

Step 2 Move along the unexplored edge until M-plane is encountered. If this point is already in the Connectivity graph, go to Step 1. Otherwise, enter the point as a node into the Connectivity graph; go to Phase 1.

Phase 3 (Complete search along the obstacle surface).

Step 1 Define M_i-planes, parallel to M-plane and spaced at R_s, and I_j-planes, perpendicular to M-plane and spaced at R_s. Set i=0, j=0, h=1, t=1. Move to the lastly defined Hit Point on M_i-plane. Go to Step 2.

Step 2 Define a Mark point, and place it into the Mark point graph. Move along the intersection curve between O and M_i-plane until the Mark point is encountered again. Unless i=0, compare the path on M_{i-1}-plane to the one on M_i-plane; if there is a branch go to Step 6, else go to Step 3.

Step 3 Define the I-planes, perpendicular to the M-planes and spaced at R_s. Set *m*=number of I-planes that intersect O. Set *j*=0. Go to Step 4.

Step 4 Move on the intersection of the M_i -plane and O until the I_j -plane is encountered, then move on the intersection of the I_j -plane and O, checking for the following:

1. M_{i+h} -plane encountered: that is, next M-plane found; go to Step 2.
2. Unexplored section of M_i -plane encountered. Remember current location as S'. Call Phase 1 to move to T.
3. Previously explored section of M_i -plane encountered: set j=j+1. If j < m, repeat Step 4, else go to Step 5.

Step 5 Select a Mark point node in the Mark point graph with an unknown edge; if none found, T is unreachable: the procedure terminates; else search in the Mark point graph for a path to the Mark point node with the unknown edge. Move to the selected Mark point, and go to Step 2.

Step 6 Explore the branch in the obstacle by defining the I-planes perpendicular to the M-planes and spaced at R_s. Set *n*=number of I-planes that intersect the area to be explored. Set *j*=0. Go to Step 7.

Step 7 Move on the intersection of the M_{i-1}-plane and O until the I_j-plane encountered, then move on the intersection of the I_j-plane and O, checking for the following:

1. Unexplored section of M_i -plane is encountered: go to Step 2.
2. Previously explored section of M_{i-1} -plane is encountered: set j=j+1. If j < n, repeat Step 7, else go to Step 5.

Step 8 Check for the following conditions after using Phase 1:

1. T is reached. End execution.
2. Otherwise, set h = -h, return to S' with Phase 1, go to Step 2.

4.8 Conclusion

In this chapter the Motion Planning algorithm for a three degree of freedom articulated robot arm is presented. The algorithm consists of three phases. In Phase 1, the automaton moves within a chosen fixed plane. If Phase 1 terminates and the algorithm

cannot conclude whether the target T is reachable, Phase 2 or Phase 3 is used to continue the search. The purpose of Phases 2 and 3 is to guide the automaton off the M-plane. During Phase 2, an intersection curve between two obstacles found during the Phase 1 is followed in the search of T or an unexplored region of the M-plane. If such a region is found, Phase 1 is used again and the process repeats. Otherwise, Phase 3 is invoked to carry out a complete search of the surface of the current obstacle. This process can result in reaching T or in finding another unexplored region of the M-plane, and the process repeats. Depending on the task and environment, the order of the actual phase execution may change. Furthermore, a given phase may or may not appear during the motion.

As can be seen in Chapter 5, the algorithm generates quite reasonable paths in a fully unknown environment. A complete search is often not needed, but is invoked in those unfortunate cases when the automaton needs to pass through an opening that has no clues leading to it, such as the opening in a bottle-like obstacle in C-space. Such a search, even if required, is shown to be of a reasonable length due to non-zero sensing range of the sensitive skin.

For a rather crowded environment with 4-5 curved concave and convex obstacles, which is not likely to occur, and assuming an unfortunate case where the robot will have to do a complete search, the size of the Connectivity graph will be on the order of 10 to 15 nodes; the Mark point graph will contain about 40-50 nodes; and the sampling table for Mi- and M_{i-1}-plane exploration will contain 600-800 points. On the total, the robot will need enough memory to store about 1000 points, each point being a 3D vector. This estimate is quite conservative and thus quite reasonable. As to the time constraints, note that the real-time control is only a function of local interaction with obstacles, and so it does not depend on the complexity of the environment. In other words, a capability to produce fast smooth motion translates into requirements to sensor range and reaction time, raw sensor data processing capabilities, and local normal calculation time, rather than into any constraints on the motion planning algorithm.

Chapter 5

Algorithm Implementation

5.1 Introduction

In this chapter, the software implementation of the developed robot motion planning and control system is discussed. The computer system consists of three Transputers from the INMOS corporation, an IBM-AT, and a MicroVax workstation; refer to Chapter 2 for details on the computer and sensor hardware.

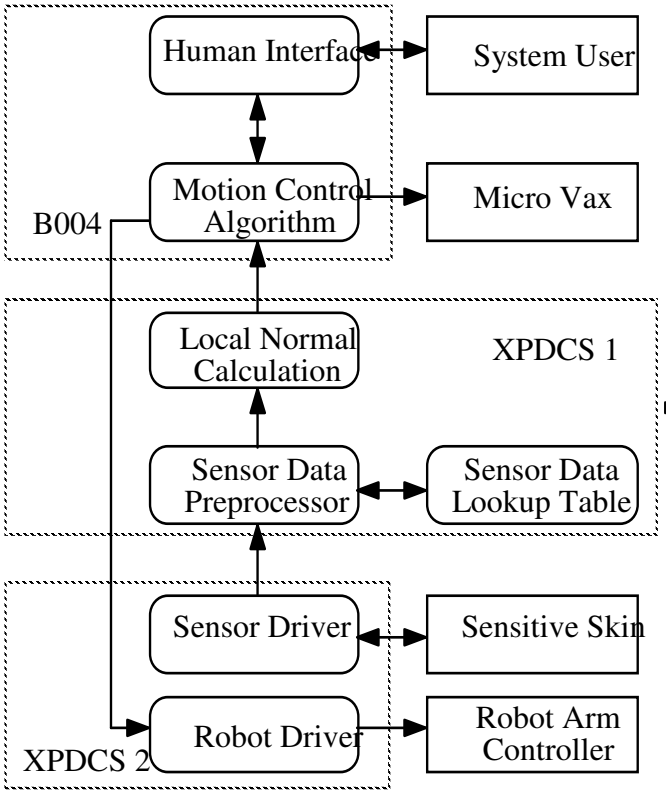


Figure 5-1. Major components in the implemented robot motion control system.

Figure 5-1 shows the major operational blocks and their overall relationship in the implemented motion control system. The software and hardware blocks are indicated by

boxes with round and square corners respectively. Each dotted line box corresponds to one Transputer. XPDCS 2 handles the low level processing by polling the Sensitive Skin and updating position commands to the Robot Arm Controller. XPDCS 1 handles the intermediate processing by implementing the local normal formulas presented in Chapter 3. Programs that run on B004 handle the user interface and the motion planning algorithm discussed in Chapter 4. All programs that run on the Transputers are written in OCCAM, a parallel processing language, which are then compiled in an IBM-AT that acts as the program development platform. In addition, the MicroVax workstation provides graphical display and documentation of experiments. This package is written in C-language. The screen of the workstation displays the real-time motion of the arm in the W-space and in the C-space. For the degrees of freedom of the arm that are monitored, the data is projected onto an imaginary viewing screen that is perpendicular to the desired viewing direction. Using the mouse and menu based user interface, the viewing direction and the viewing magnification can be changed. In addition to real-time display of the experiment, the package can save the data collected during an experiment for later inspection, and can also generate hard copies of the path of the robot arm using a laser printer (see examples below). In the next section, a more detailed description of each of the software blocks is given.

5.2 System Software

Using the robot interface architecture described in Section 2.3, the P5 robot arm has been transformed into a positioning device. The Robot Driver block, Figure 5-1, is needed to command velocities, or to control motions requiring a large travel by gradually incrementing or decrementing every position command sent to the robot controller. The robot controller receives a new position command every 15 msec from the Robot Driver.

The entire sensor skin is polled once every 62 msec by the Sensor Driver, and the raw readings are placed in an array that resides on XPDCS 1. Programs that reside on this transputer board run asynchronously with the Sensor Driver. One of these programs is the

Sensor Data Preprocessor which subtracts the raw sensor reading from the quiescent sensor level in the Sensor Data Lookup Table, obtained during the initialization stage. The lookup table also contains the parameters used in the local normal generation procedure which characterize the location of a particular sensor on the arm. The processed sensor data and the parameters from the lookup table are then sent to the Local Normal Calculation blocks, which implements the formulas described in Chapter 3. For every sensor whose output exceeds a preset threshold, its local normal is calculated and placed into a list. Once all the sensors in the skin have been read once, this list is sent to the B004. If, for example, only one local normal is generated, it would arrive at B004 once every 39 msec; for 20 normals in the list, the local normal list is updated once every 42 msec.

The Motion Control Algorithm, which runs asynchronously with the other processors on the B004, always uses the latest list of local normals from XPDCS 1. The local normals are used in different ways depending on the motion control procedure selected by the user using the menu driven interface. The three motion control strategies mentioned in Section 3.5, as well as an assortment of sensor reading and robot control utilities, can be chosen interactively from the menu, before the actual motion starts. Experiments with various parts of the motion planning algorithm are described below.

Since the various data processing routines run asynchronously to each other, the time delay between the appearance of an obstacle and the delivery of its local normal is dependent on the relative phase of each of the asynchronous routines processing the data. In the worst case this delay can be over 100 msec, while in the best case it can be below 1 msec. For example, in the worst case, suppose an obstacle appears at t=0, just after the sensor closest to it on the arm has been read. The obstacle will be detected on the next read of that sensor, at t=62 msec. In addition, assuming the worst case for the synchronization of the local normal tangent calculation, the delivery of the local tangent to the B004 will take place at t = 62+39 = 101 msec.

The motion planning program that resides on B004 executes at cycle times of about

5 msec, and seems to remain the same regardless of whether it is running Phase 1,2, or 3. Thus a new position command is given to the motors once every 5 msec. As was mentioned above, the motor driver sends position commands to the robot controller once every 15 msec; there is also a built in delay in the robot controller. Adding all these delays together shows that it may take up to 150 msec between the time an obstacle first appears and the moment the corresponding command is given to the robot motors. This rather long delay limits the maximum speed and performance of the arm while moving around obstacles.

On the other hand, if all the processes "line up" favorably, it may take only a few milliseconds for the data to ripple through the system. One can expect the average system performance to lie between these two extremes. Note, however, that sensing and sensor interpretation (local normal generation) can quite easily be done in parallel. Even with relatively slow and inexpensive processors, large improvements in system performance can be expected, since sequential processing is the largest source of latency in the system.

5.3 Conclusion

This chapter describes the implementation of the motion and step planning algorithms described in Chapter 3 and 4. The algorithms run on three Transputers; the first, called XPDCS 1, is responsible for hardware interface and low level processing of the sensor and robot data. It handles the polling of the sensors and the updating of the robot arm commands. The second, XPDCS 2, handles the intermediate processing of the sensor data, and implementing the expressions of the local normal presented in Chapter 3. The local normals describe the conditions necessary to slide along the contact point of the obstacle in C-space. Finally, the third Transputer, the B004, runs the high level motion planning algorithm described in Chapter 4. Programs running on the Transputers are written in a parallel processing language, and compiled using an IBM-AT. A MicroVax workstation serves as the graphical display system.

A detailed analysis of the execution times of the various computer programs

running in parallel on the Transputers is also presented. It is shown that the time delay between the appearance of an obstacle, and the moment a command is given to the motors can vary between 5-150 msec. This time delay depends on the synchronization of the various data processing programs. The overall time delay can be reduced by increasing the number of processor, and should result in an improvement in system performance.

Chapter 6

Experimental Results

6.1 Introduction

As described in Chapter 4, the motion planning algorithm is divided into three phases. To highlight the motion during a particular phase, each experiment below is designed so as to utilize two out of three phases of the algorithm. The purpose of these experiments was to demonstrate real-time feasibility of the developed approach and to assess the quality of the arm motion, especially during its maneuvering around obstacles. Recall that the arm has no beforehand knowledge about the obstacles in its workspace; all motion planning is done in real-time, based on on-line sensing.

Two experiments are described: the Experiment 1 involves Phase 1 and Phase 2 of the algorithm, and Experiment 2 involves Phase 1 and Phase 3. Each of the experiments was documented via a sequence of photographs and a plot of the arm path in its 3D C-space. Being 2D projections of 3D curves, these plots are usually difficult to interpret. Instead of clarifying things with the exact reconstruction of C-space obstacles - which is hard to do because obstacles are of irregular shapes - we choose to precede Experiment 1 with two examples, Example A and Example B. Each example explains the operation of a particular phase of the motion planning algorithm, using imaginary obstacles that qualitatively reflect the shape of obstacles used in Experiment 1. Example A and Example B highlight the operation of Phase 1 and Phase 2 respectively during Experiment 1. The path of the arm in the C-space in each of these examples will hopefully help the reader understand how the actual path is generated during Experiment 1.

In the experiments, the work space contains obstacles of irregular shape, see Figures 6-2 and 6-3. No information about the obstacles was given to the arm beforehand, and no information collected in one experiment was used in the other experiment. The

generated paths were documented by photographing some of the arm positions during the motion, and also via the C-space graphics presentation of the paths on the MicroVax workstation.

6.2 Experimental Setup

A sketch of the arm is shown in Figure 6-1; Figure 6-2 shows a photograph and a sketch of the overhead view of the experimental setup. The obstacles in W-space of the arm, A, B, C, D, are named as shown in Figure 6-2b. Hereafter, their corresponding images in the C-space named A', B', C', D' respectively, see the sketch in Figure 6-7. Figure 6-3 shows various views of the experimental setup taken at positions marked on Figure 6-2.

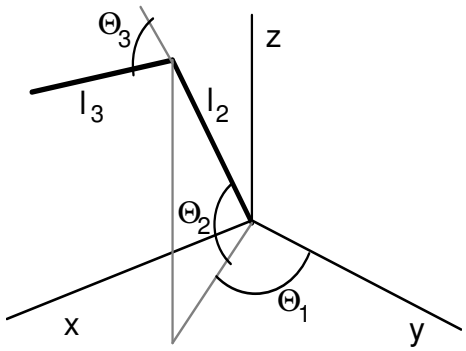


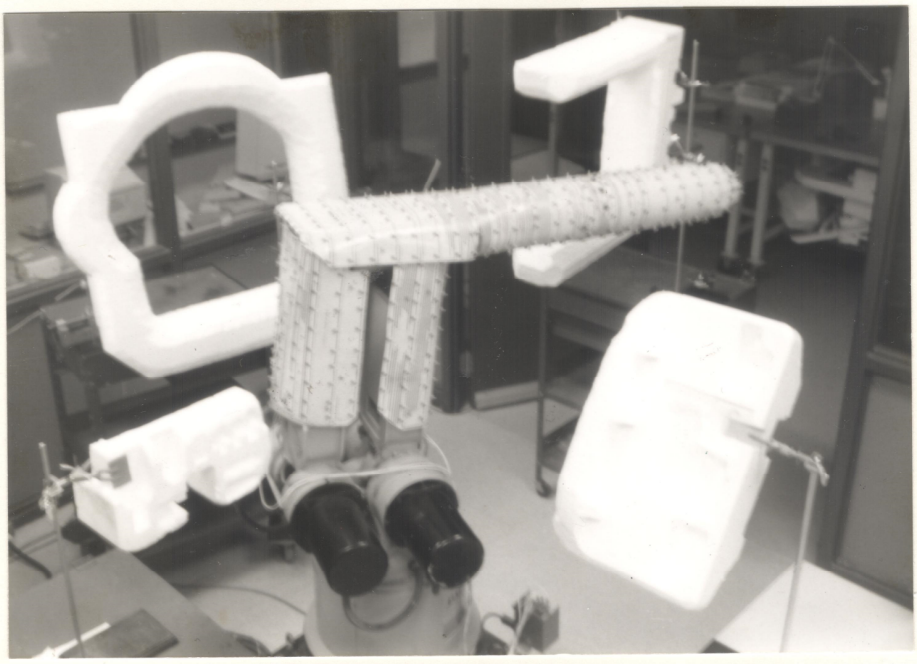
Figure 6-1. Sketch of the robot arm

Link l_2 of the arm is 60cm long, its thickness varies from 25cm to 35cm depending on the arm position; link l_3 is 78cm long and 16cm thick. Obstacle A is a block shaped object, measuring about 20cm x 38cm x 10cm. Obstacle B is also block shaped and measures approximately 35cm x 55cm x 13cm. The horse-shoe shaped obstacle C has an opening between its fingers of 40cm, and each finger is 48cm long. Finally, the ring shaped obstacle D has an opening of about 55cm radius, its ring is about 15cm thick. As one can see in Figure 6-2a, the obstacles have planar as well as curvilinear surfaces.

A special note about obstacles A and C. Obstacle C is located such that due to its

joint limits the arm cannot pass over the higher finger, nor can it pass under the lower finger of the horse-shoe. Also, due to obstacle A, the arm cannot completely move "back" towards the base of the arm enough to completely clear of obstacle C. The only alternative is to pass through the fingers of the horse-shoe C. The orientation of obstacles A and C with respect to the arm cause the formation of a wall that separates the C-space of the arm with is a small opening in the wall through which the arm can pass, Figure 6-5. The handling of these two obstacles by the arm is described in more detail in Example B, Section 6.3.

a)



b)

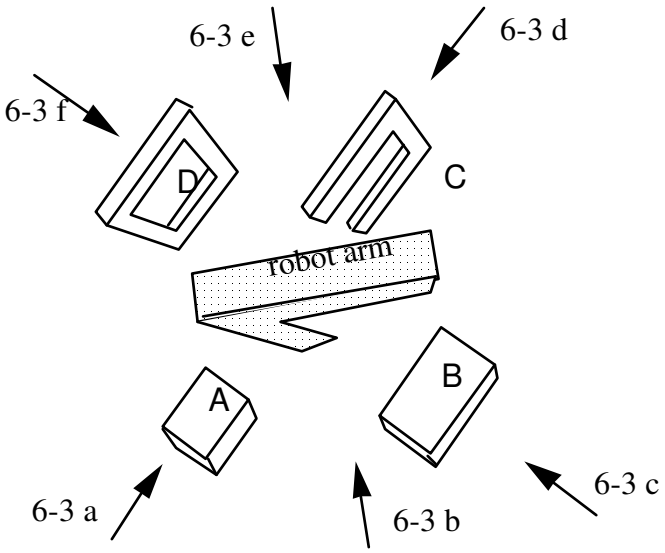


Figure 6-2. a) Photograph and b) a sketch of an overhead view of the experimental setup. The arrows in the sketch mark the view directions from which the photographs of Figure 6-3 were taken. The arm position is fixed throughout as in (a).

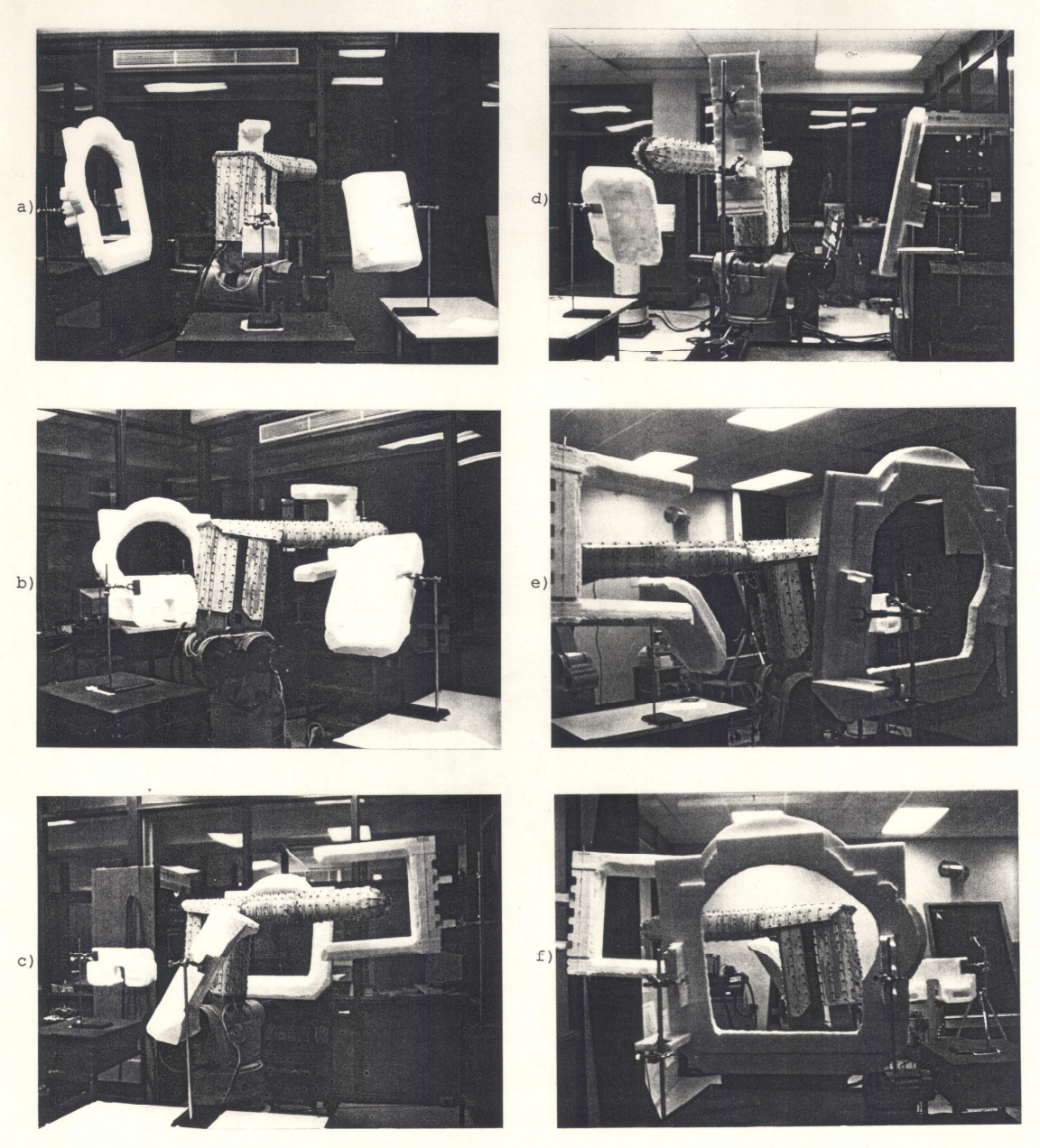


Figure 6-3. Views of the arm from the directions shown in Figure 6-2; the arm position is fixed as in Figure 6-2a.

6.3 Experiment 1.

Example A.

This example is a slightly idealized demonstration of one part of Experiment 1. It shows the step by step operation of Phase 1, follow Figure 6-4. The rectangular box that limits the figure indicates the arm joint limits. Thus the arm motion is restricted to the inside of the box. The obstacle shown touches the minimum and maximum limits of joint Θ_3, as well as the minimum and maximum limits of joint Θ_2. There is, however, an opening near the obstacle in the bottom right corner which connects the left and right free semi-spaces of the C-space. The C-space image of obstacle B in the actual experimental setup, Figure 6-2, is similar to the obstacle used in this example.

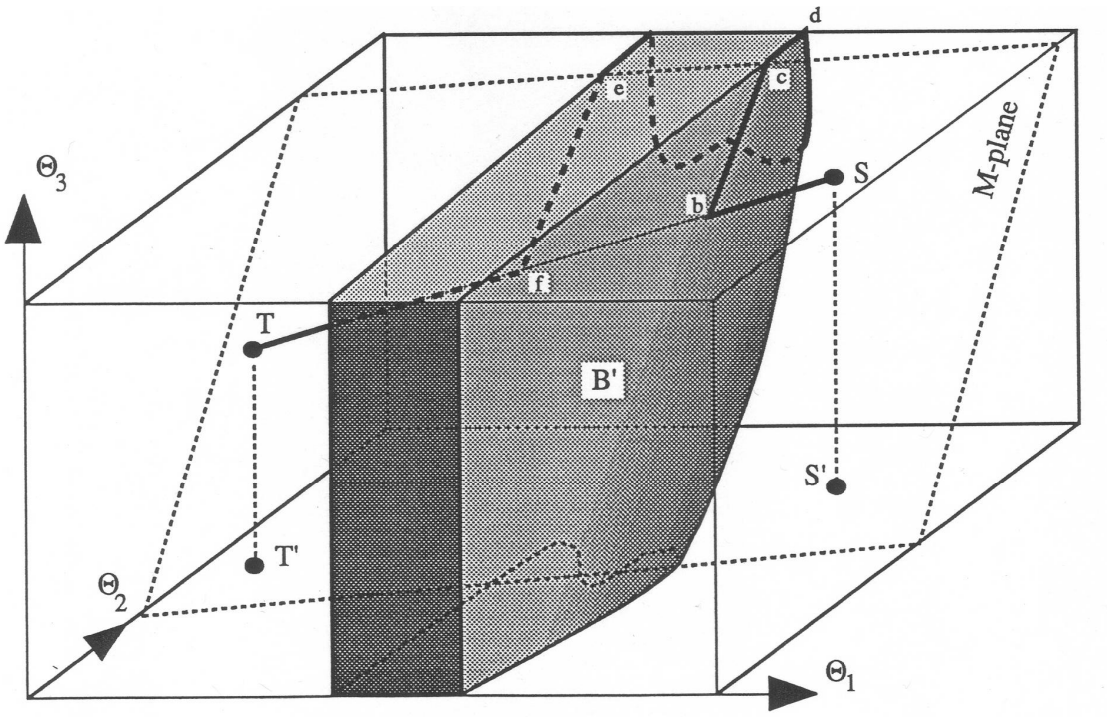


Figure 6-4. C-space of the arm with one obstacle that touches the joint limits. The path of the arm is shown in a thick line.

The algorithm starts with Phase 1; starting from S the arm moves towards T, but hits the obstacle at point *b*, where it defines a Hit Point according to the algorithm. It then turns in the local direction (towards the positive Θ_3 direction) and follows the intersection

of the M-plane and the obstacle (see Section 4.6 on how the M-plane is chosen). In W-space, this corresponds to the arm endpoint moving towards the base of the robot. At point *c*, the arm hits the positive Θ_3 joint limit, and follows the intersection of the corresponding limit plane and the obstacle until it hits the M-plane again. On the way, the arm hits the maximum Θ_2 joint limit at point *d* and continues to follow the intersection of that joint limit and the obstacle. This continues until the M-plane is again met at point *e*. The arm then leaves the maximum Θ_3 joint limit and moves on the M-plane until it encounters the M-line at *f*, where it leaves the obstacle and heads straight for T.

Example B.

In this example, the operation of Phase 2 is demonstrated; follow Figure 6-5. The two obstacles, C' and A', are roughly equivalent to obstacles C and A in Figure 6-2, are of Type III and Type II respectively, and intersect each other such that they form a wall dividing the C-space into two halves. There is, however, an opening in this wall, and the path through this hole is found during Phase 2. The route that leads to the hole is the intersection curve between the two obstacles (refer to Section 4.7.3 for details on the algorithm). The arm hits this intersection curve during Phase 1, while moving on the M-plane. Since such a seam is always a closed curve with no branches, if the arm follows this curve, it will eventually return to the M-plane. After Phase 1 fails, the arm initiates Phase 2, returns to the seam and follows it until the M-plane is again encountered, point *s*. Then Phase 1 starts again and the arm arrives at T along the line ST.

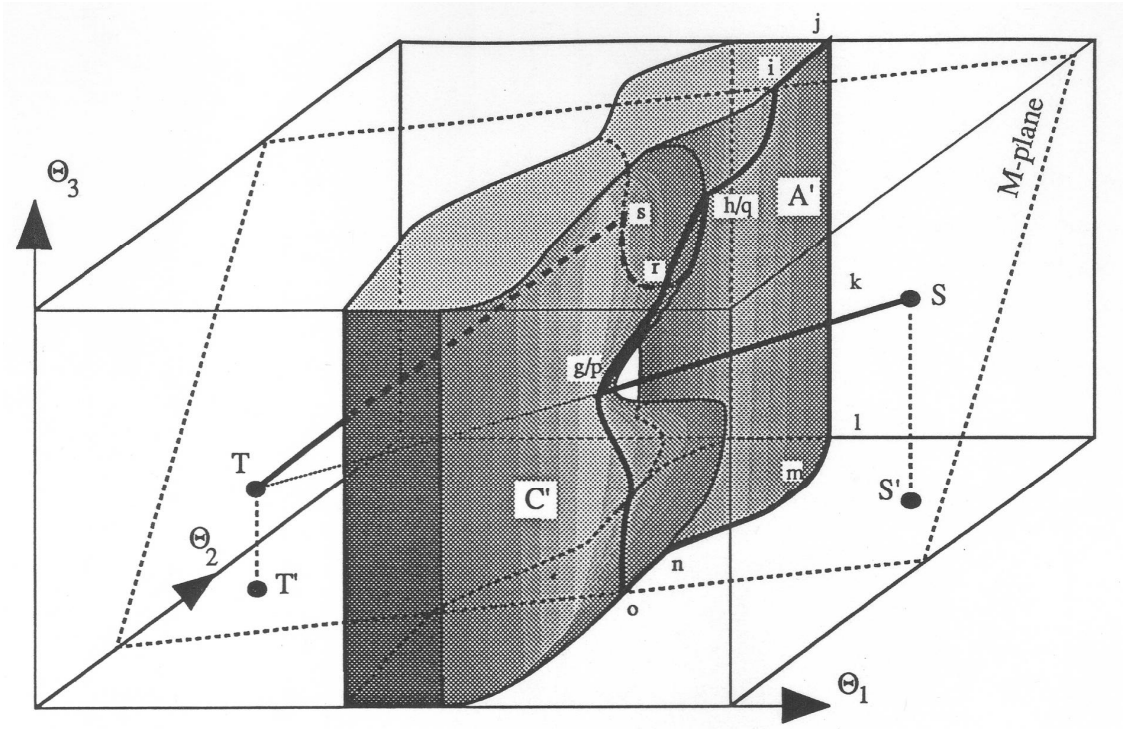


Figure 6-5. C-space of the arm with two intersecting obstacles. The obstacles form a wall with a hole in it. The path of the arm is shown in a thick line.

In more detail, the motion proceeds as follows starting with Phase 1, the arm moves from S towards T, hits obstacle C' at point *g*, defines a Hit point, and turns in the local direction to follow the intersection curve of the M-plane and the obstacle. At point *p*, the intersection between a Type III and a Type II obstacle is detected, a node in the Connectivity graph is defined, and its location is marked. Then, motion continues in the M-plane; after hitting the joint limit at point *i*, the arm follows the intersection of the joint limit plane and the obstacle, points *j*, *k*, *l*, *m*, *n*, until it reaches point *o*, where it resumes moving on the M-plane. This continues until point *h* is hit (same as point *g*), where the Hit point was defined. Note that the path segment *g*-*h* will be traversed twice. Phase 1 thus fails to find a path to the target, and since a graph node had been defined before, Phase 2 is called to finish the task. The arm moves to point *q* (same as point *h*) where the seam between obstacles A' and C' is found again, and follows the seam through the hole past point *r* until it intersects the M-plane at point *s*. Since this section of the M-plane is

disconnected from the one that contains S, the arm can now attempt to move straight to T without fear of getting into a local cycle. Phase 1 is therefore called once again at point *s*, and the arm moves straight to T.

Note that it was possible to use Phase 2 in the above example because a node was defined beforehand during Phase 1. If no such nodes were defined, a complete search would be needed to find the hole in the obstacle. In this case, Phase 3 is needed to explore the obstacle. This case is demonstrated in Experiment 2.

Actual motion during Experiment 1.

In this experiment the arm is required to move past the block shaped obstacle B, get through the horse-shoe shaped obstacle C, and move around and into the ring shaped obstacle D, Figure 6-2. The sequence of photographs in Figure 6-6 present some of the arm positions along the actual path. Figure 6-7 presents a sketch of the obstacles in the C-space, and Figure 6-8 presents the projection of the arm's motion in C-space documented at the MicroVax workstation. To allow one to track the progress of the experiment both in W-space and in C-space, the arm positions marked by letters *a*,*b*,*c*,... etc. on these figures refer to the same points respectively. Recall that the motion planning algorithm has no á priori knowledge of the objects, and no calculation of the shape and location of the obstacles in C-space takes place.

The arm motion proceeds as follows (refer to Figures 6-6, 6-87, and 6-8). The arm starts moving from S towards T, but hits obstacle B' at point *b*, and starts following the intersection of the M-plane and obstacle B'. This motion along the surface of B' takes the arm back to the M-line, point *f*, where it resumes its motion towards T. At point *g*, the arm hits obstacle C', where a Hit point is defined, and the arm then follows the intersection of the M-plane and obstacle C'. At point *h*, obstacle A' is encountered, and a change from a Type III to a Type II obstacle is detected. A node is defined, and its position stored. At

point *i*, the arm hits the maximum Θ_3 joint limit and starts following the intersection curve of the joint limit planes and the obstacle, points *j*,*k*,*l*, *m*, n, until it reaches again M-plane, at point *o*. Then, motion in M-plane resumes until the Hit point *p* (the same point as *g*), where Phase 1 ends, and Phase 2 is used to continue the task. The arm moves - for the second time - to point *q* (same as *h*), where it encounters the seam between the Type II and III obstacles A', C'; it then follows this seam until the M-plane is again met at point *s*. Here, a new M-plane is defined, M'-plane, with the same target point and a new start point, S' and Phase 1 is initiated again to move towards T. While moving to T, the arm hits obstacle D' at point *t*, moves along the intersection of D' and M'-plane to point *u*, along the intersection of D' and a joint limit plane back to M'-plane, and then again along the intersection of D' and M'-plane to point *w*, where it meets M-line and heads straight for T. The whole motion in this experiment, including the real-time sensor processing and motion planning, took 2 minutes and 32 seconds to complete.

Figure 6-6. Various arm positions during Experiment 1; see Figures 6-7 and 6-8 for the corresponding points in C-space.

Figure 6-6. (continued).

Figure 6-6. (continued).

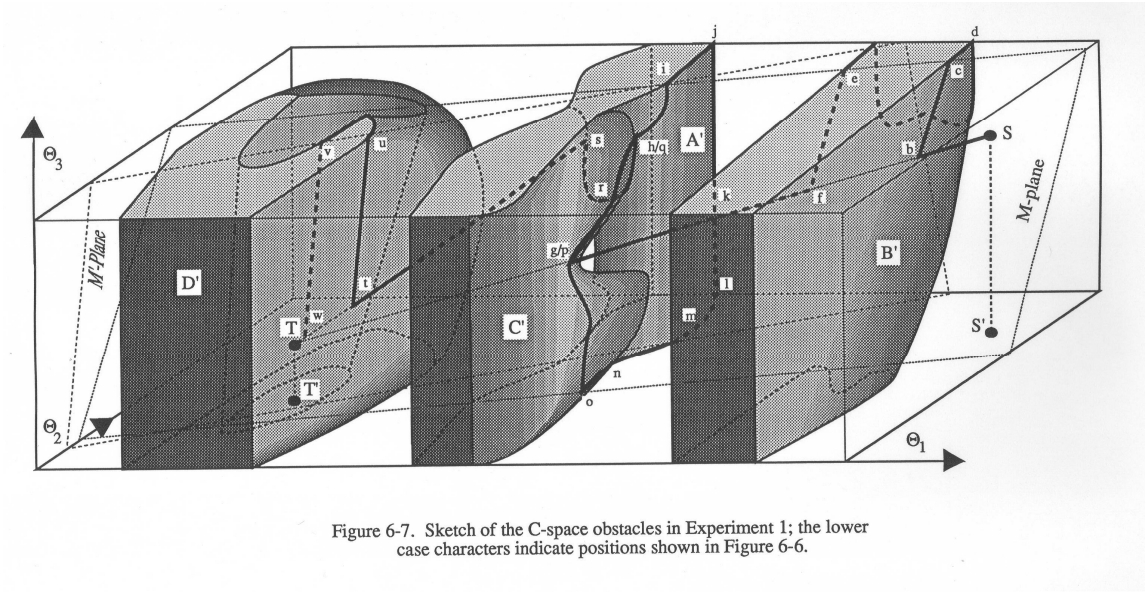


Figure 6-7. Sketch of the C-space obstacles in Experiment 1; the lower case characters indicate positions shown in Figure 6-6.

Figure 6-7. Sketch of the C-space obstacles in Experiment 1; the lower case characters indicate positions shown in Figure 6-6.

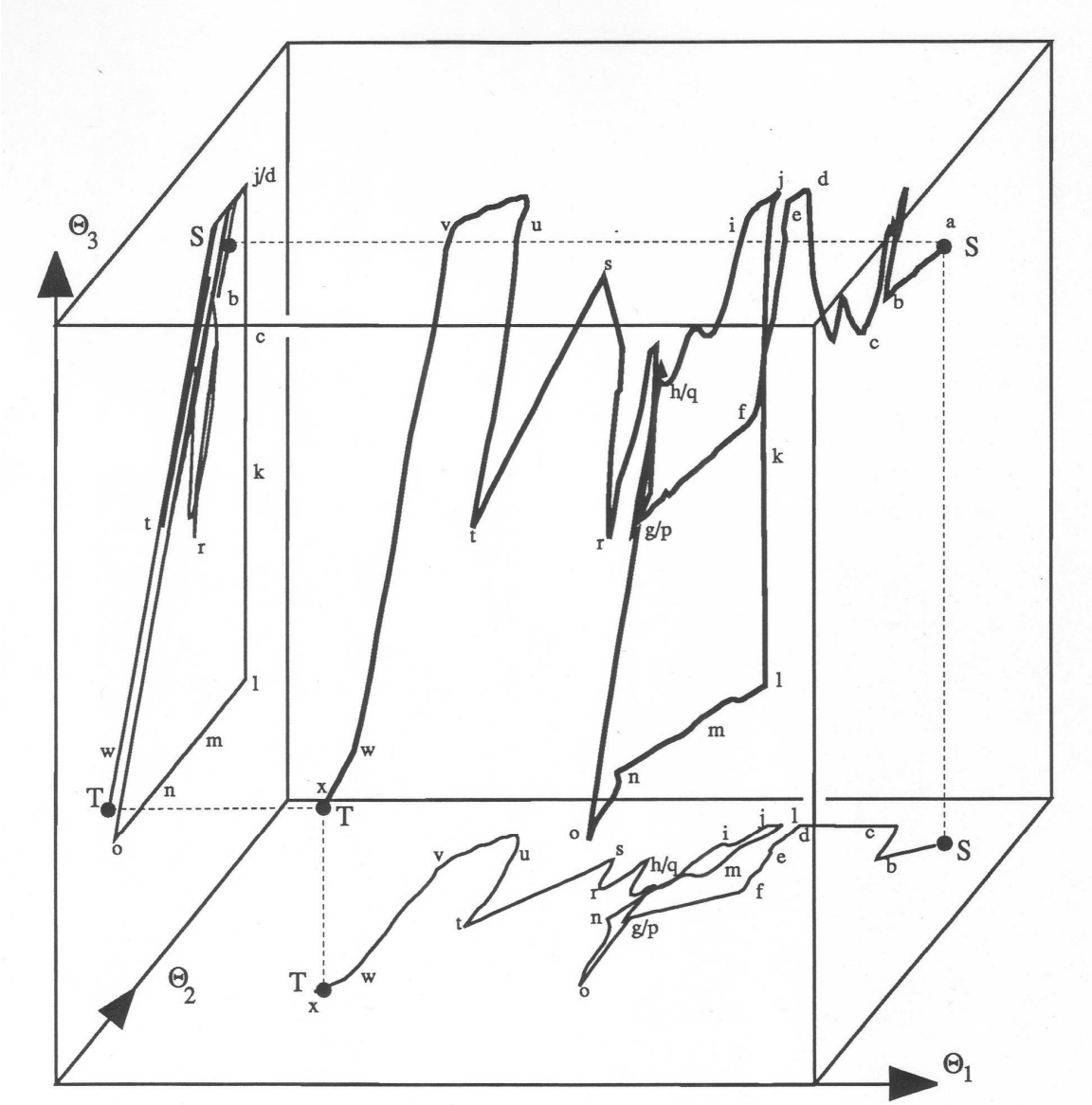


Figure 6-8. Path of the arm in the 3D C-space, Experiment 1.

6.4 Experiment 2.

In the previous experiment it was possible to use Phase 2 to find the hole in the obstacle caused by the intersection of obstacles A' and C', Figure 6-7, because a node was defined beforehand in Phase 1, when the intersection curve was hit. If no nodes were defined during the first use of Phase 1, a complete search would instead be needed to find this hole. Such a situation develops in Experiment 2, with the result that Phase 3 is needed to accomplish the task. For simplicity, obstacles B' and D' have been removed from the

W-space (a simple application of Phase 1 can be used to maneuver around these obstacles).

Also for simplicity, the documentation of this experiment starts with the arm on the surface of the obstacle after the initial exploration of the obstacle using Phase 1. At the end of Phase 1, the arm hits the previously defined Hit point, and concludes that no path to T exists on this section of the M-plane. Since no nodes were defined during Phase 1, Phase 2 is skipped, and Phase 3 is used next. For the sake of brevity, we omit this initial pass of Phase 1 (refer to Experiment 1 for an example of this initial pass with Phase 1).

Figure 6-9 presents photographs of some of the arm positions during this experiment. Figure 6-10 presents the corresponding C-space projections of the path as displayed on the MicroVax screen, and Figure 6-11 is the corresponding C-space sketch. As before in Experiment 1, the letters *a*, *b*, *c*,.... *p* in the figures correspond to the same points; however, the points *q*, *r*, *s* in Figures 6-10 and 6-11 do not correspond to any photographs in Figure 6-9.

As explained in Section 4.7.3, Phase 3 of the algorithm explores the surface of the obstacle by intersecting it with appropriately spaced planes parallel to the M-plane. Each such M-plane is numbered as Mi, i=0, 1, 2, 3, ..., with M0 being the original M-plane.

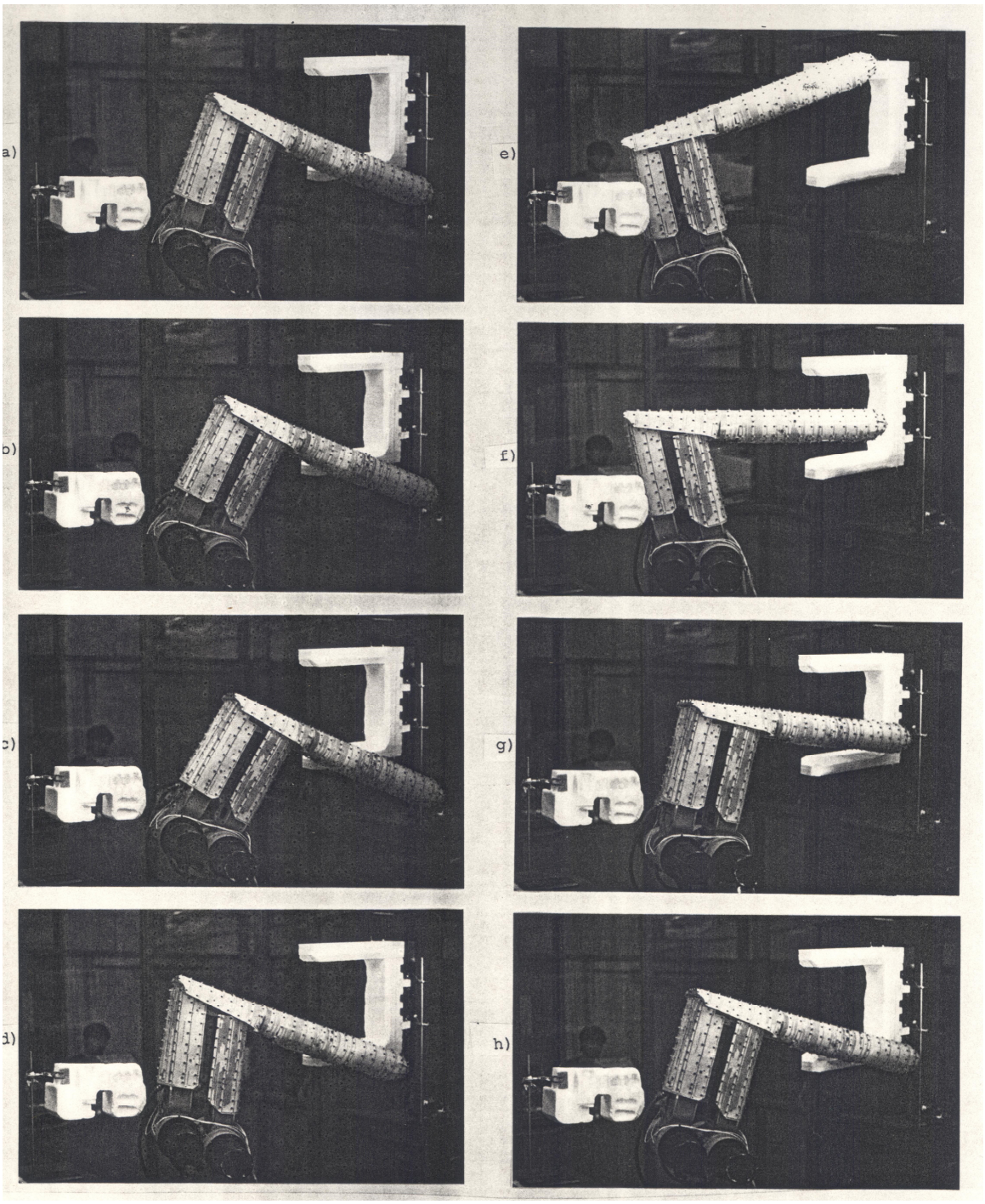


Figure 6-9. Various arm positions during Experiment 2. See Figures 6-10 and 6-11 for the corresponding points in C-space.

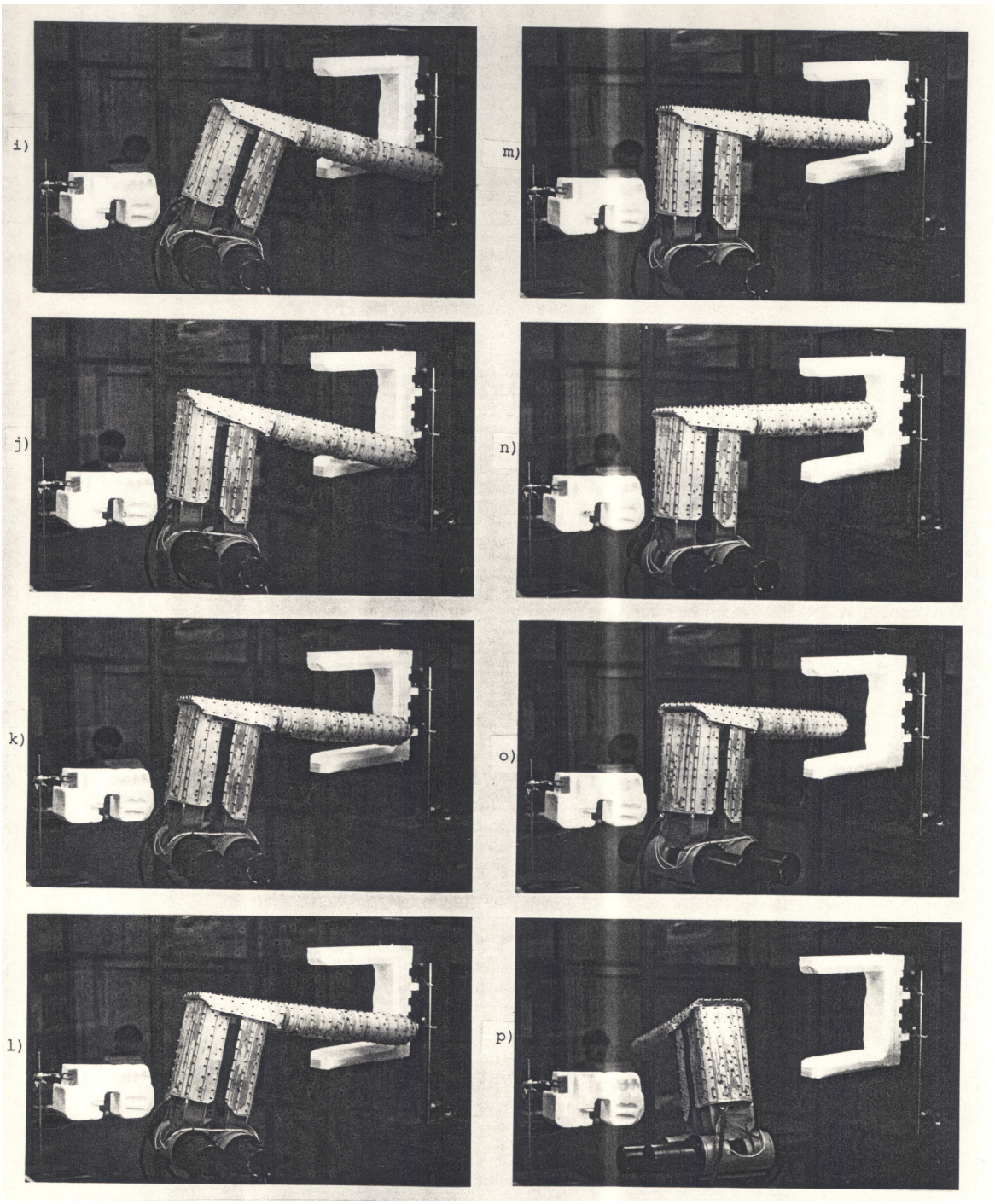


Figure 6-9 (continued).

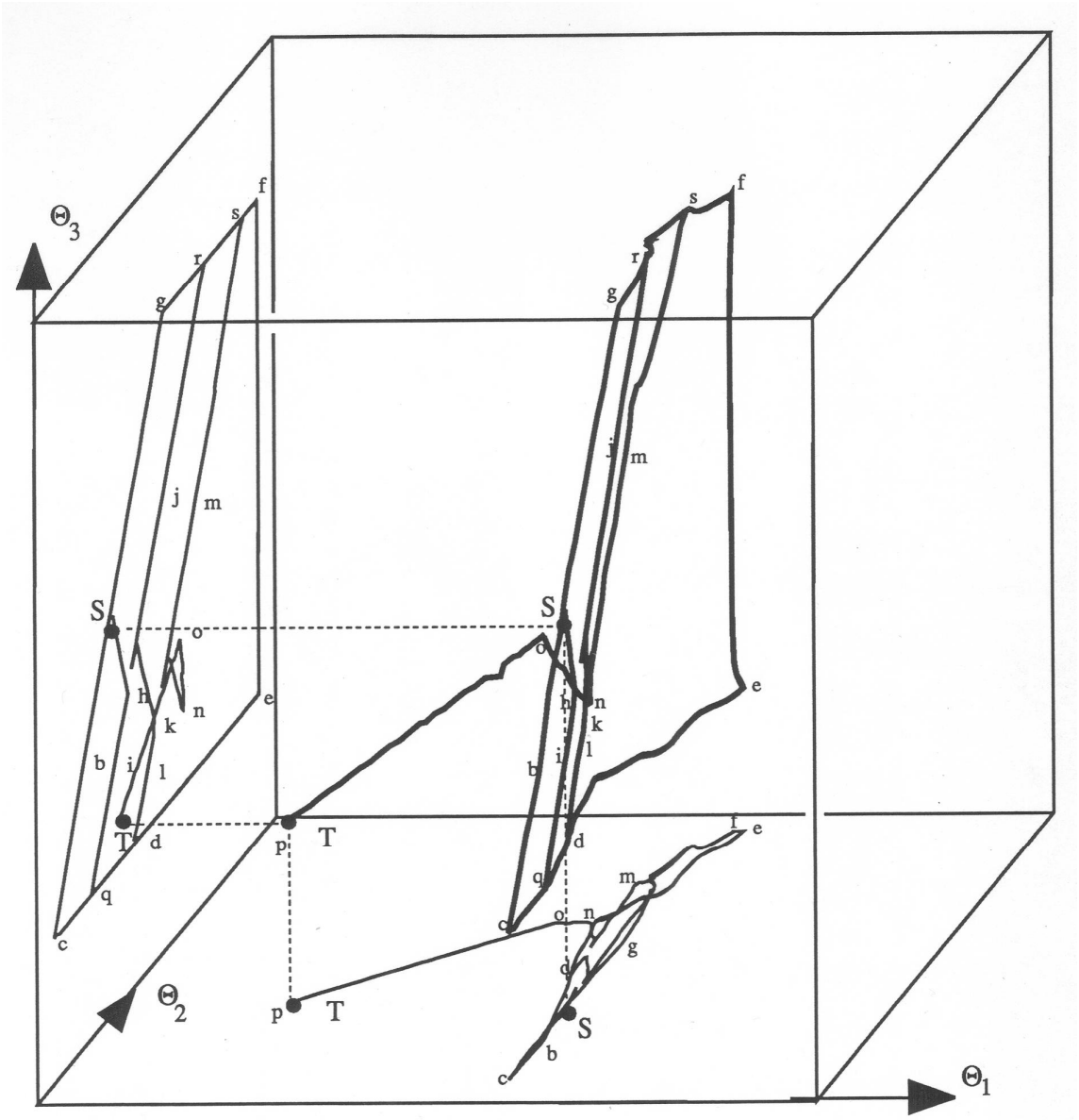


Figure 6-10. Path of the arm in the 3D C-space, Experiment 2.

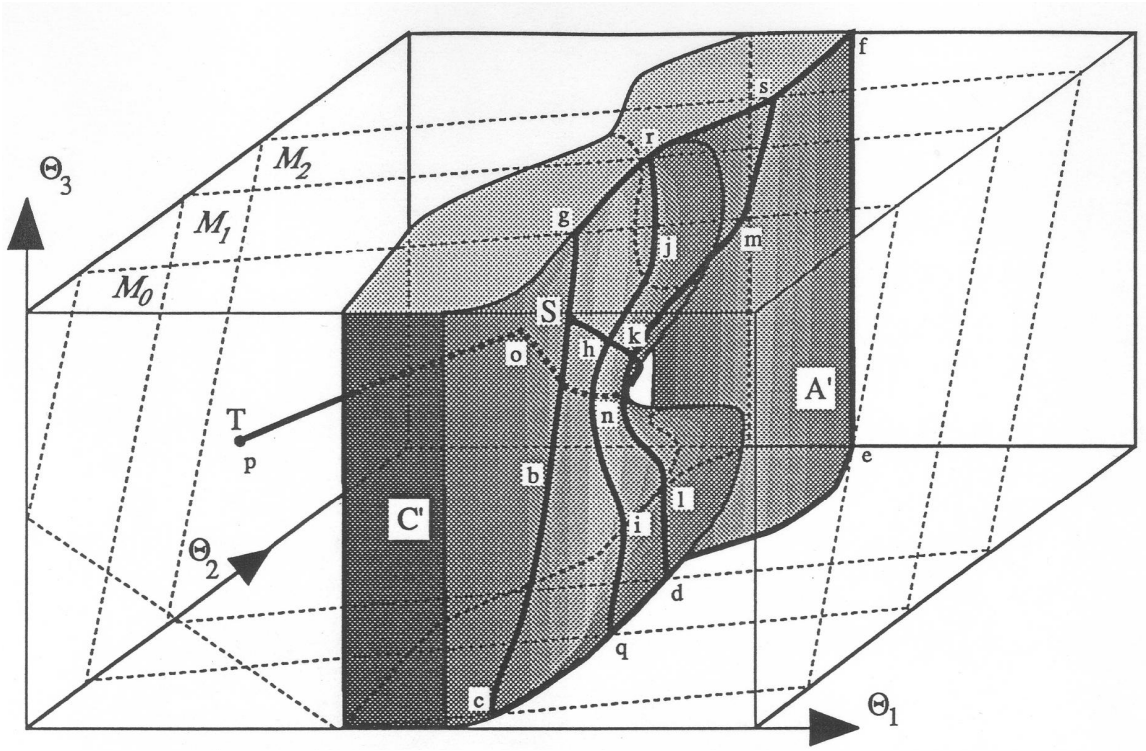


Figure 6-11. A sketch of the C-space obstacles, Experiment 2.

In more detail, the trajectory is as follows (refer to Figures 6-9, 6-10, and 6-11). The arm starts at S and moves along the intersection of the obstacle and the M_0-plane. This continues past point *b* until the arm hits the minimum Θ_3 joint limit at point *c*, whereupon it follows the intersection of the joint limit planes and obstacles C' and A', points *q*, *d*, *e*, *f*, *s*, *r*, until at point *g* it meets M_0-plane again. The arm then continues to move on the M_0-plane towards point S. Here it concludes that the section of the M_0-plane just explored should be abandoned, and attempts to move to the next plane, M_1-plane. This is done by moving on the intersection curve of the obstacle C' and a plane perpendicular to M_0-plane (see Section 4.7.4.2 for details). At point *h*, the arm finds M_1-plane and follows the intersection of this plane and the obstacle. Similar to the case on M_0-plane, the arm reaches the joint limit at point *q*, repeats a part of the path past points *d*, *e*, *f*, *s* until it hits M_1-plane again at point *r*. The arm then continues on the intersection of the obstacle and M_1-plane until it hits point *h* once again. The process then repeats for M_2-plane, starting at point *k*

and passing through points *l*, *d*, *e*, *f*, *s, m*. When the arm returns to point *k* on M_2-plane, it attempts to find M_3-plane, but instead ends up going through the hole between the obstacles B' and C', Figure 6-11. In the course of this motion, it hits M_2-plane again, now at a location that has not been previously explored (if this were a previously explored section it would also have been explored when points *m*, *k*, *l* were visited). The arm thus concludes that a new unexplored section of C-space has been found, and tries to use Phase 1 to head straight for T. In this case, there are no further obstructions, so the arm arrives at T. This experiment took 4 minutes and 17 seconds to complete.

6.5 Robot arm performance in the experiments - Discussion.

On the total, the generated motion has been quite acceptable. No collisions with the obstacles took place; the minimum distance between the arm and the obstacle varied within the range 2 to 5 inches. The arm was able to maintain a good velocity typical for such industrial manipulators; the linear velocity of the arm endpoint has been in the range of 3 to 14cm/sec, with the average 9cm/sec. In spite of tight time constraints, polling of all 500 sensors of the skin is done within the normal sample rate of the arm. Even when challenged by a crowded environment with multiple obstacles, the real-time processing was adequate: at some positions the planning system had to base its decisions on up to 20 local normals -- that is, up to 20 sensors on the skin simultaneously sensed obstacles.

The experiments were designed such as to present simple as well as difficult collision avoidance tasks. The arm easily maneuvers around a simple block shaped obstacle B, see Figure 6-2. Moving around this obstacle is representative of tasks involving compact shapes such as boxes, columns, walls, etc. The local strategy is to avoid the obstacle by folding the arm up by moving the arm endpoint towards the base of the robot, and trying to move around the obstacle in a more compact configuration.

The horse-shoe shaped obstacle C, Figure 6-2, presents a more difficult case. As mentioned above, because of its joint limits the arm cannot pass over or under the obstacle.

Furthermore, due to obstacle A, the arm can only pass thru the fingers of obstacle C. Other examples of tasks of this kind are moving past an obstructing wall, with a slot cut into it, or maneuvering around a particularly cluttered part of the robot's work-cell in a factory environment. For example, there may be obstacles on a table forming the bottom part of the horse-shoe, and steel beams overhead that form its top part. Similarly, maneuvering around and into a ring shaped obstacle such as D, Figure 6-2 is representative of tasks where the arm needs to insert/extract itself into/from an opening. In all of these cases, the performance of the motion planning system has been quite good.

6.6 Conclusion

This chapter describes the results of experiments with the motion planning system. Two experiments designed so as to test different modes and capabilities of the system are presented. The first experiment makes use of Phases 1 and 2 of the algorithm, whereas the second uses Phase 1 and 3. Explaining the logic behind a rather complex 3D motion that takes place in the experiments present a challenge of its own. To clarify the details, sketches of the obstacles in the C-space, photographs of key points along the experiment, and plots of the motion of the arm in the C-space are included. Being 2D projections of 3D curves, these plots are usually difficult to interpret. For the same reason, two examples preceding Experiment 1 are given; they describe the operation of Phase 1 and Phase 2 respectively using imaginary obstacles that qualitatively reflect the shape of the real obstacles used in the experiment.

The overall generated motion was quite acceptable. No contact with the obstacles was made during the experiments, and the task was accomplished with a velocity typical for industrial or space manipulators.

Chapter 7

Conclusion

This work presents an approach to sensor-based motion planning for robot arm manipulators operating in a complex environment where every point of the arm body is subject to potential collision. This capability is essential if robot arms are to be used autonomously in unhealthy, hazardous, or hostile environments such as the interior of a nuclear reactor core, in outer space, in underground mining, in undersea exploration, or in unstructured factory work cells. To realize such a system, three major areas were investigated: (i) construction of a sensitive skin and interface for a robot arm, (ii) assembly of adequate signal and data processing apparatus, (iii) formulation of the low level sensor data processing strategy and design of an appropriate overall motion planning strategy.

The main contributions of this thesis are: (1) the design and construction of a novel sensitive skin sensor, (2) development of a sensor data processing algorithm to convert the sensor data into incremental motions of the robot arm, (3) the design of an automatic motion planning algorithm to guide the robot arm around obstacles, (4) assembly and testing of those components in a complete motion planning system. The prototype system that resulted presents a completely new and unique achievement which for the first time promises to grant robots the capability of autonomous operation in unknown complex environments. A brief summary of these contributions follows.

(1) Sensitive skin. Although in principle various sensor types could be used in our system, choosing a sensing media that would produce a more or less universal system presented a special challenge. After considering the options for sensing media, such as capacitance, inductance, and light, optical radiation was selected for sensing, using the amount of light reflected from objects for proximity measurement. This simple way of extracting the proximity data was chosen to best handle the complexities associated with

integrating several hundred proximity sensors onto a circuit board. The sensitive skin is built out of an array of 475 optical proximity pairs. The sensor pairs are mounted on a Kapton based flexible circuit board, which provides for both electrical interconnection as well as structural support for the sensor pairs. The emitted light is first amplitude modulated and then synchronously detected to increase immunity to ambient light. Each link's sensors emit light modulated at a different frequency, allowing the two links to be polled in parallel. The sensor skin is read seventeen times a second, which is compatible with the sampling rates of typical industrial arm manipulators, and has a range of approximately 15 cm.

In addition to overcoming problems associated with interconnecting many sensor pairs, the sensitive skin must also be able to handle the flexing of the robot arm at the joints, the covering of the gap in link l_2 due to its trapezoidal construction, and the covering of the wrist. These problems were overcome by constructing the skin out of several sections, each covering a particular part of the arm, and by covering the wrist with a dome-like structure.

An existing robot arm was modified for use in the motion planning experiments by attaching a host computer interface to the original robot controller. This allowed the continued use of the original robot controller, simplifying the interface. Using this interface, the robot is converted into a positioning device. Velocities can be commanded using appropriate software. The interface can be electrically disconnected to allow for isolation of the fault, should a malfunction arise.

(2) Sensor data processing. The assembled signal and data processing system consists of three transputer boards built by the Inmos Corporation, which function as the main sources of on-line computing power. In addition, an IBM-AT is used as the program development platform and user interface, and a MicroVax workstation as the graphical output device and data logger. Programs for the transputers are written in OCCAM, a parallel processing language, and then compiled on the IBM-AT and downloaded into the

transputer network.

(3) Step Planning. The problem of processing the data from many sensors is handled by the step planning algorithm, which transforms the sensor data into local tangent normals in the configuration space. Once obtained, these normals are then used to slide the arm along, or move it perpendicularly away from the obstacle depending on the chosen robot control strategy. This last factor is in turn dependent on the desired level of automatic control.

(4) Global motion planning. This can be done in one of three modes. For the lower level of automatic control, the robot arm can be made to simply move away from obstacles, remaining still if there are no obstacles nearby. This "Repeller" mode can be used to avoid obstacles, manually position the arm into a desired configuration, and can also be used to test the sensor system, the sensor data processing strategy, and the robot hardware.

For an intermediate level of automatic control, the user's motion commands are passed directly to the robot arm unless there is an obstacle in the path of the arm. In the latter case, the arm is commanded to slide along the surface of the obstacle using the local normal of the obstacle. This Teleoperation mode is useful in systems where traditionally the human operator is saddled with the tasks of reaching the target configuration while at the same time avoiding obstacles. Using this motion planning system, the task of avoiding obstacles is being handled automatically.

For the highest level of automatic control, the user supplies only the desired target position of the arm, and the motion planning algorithm does the rest. The motion planning algorithm guides the motion of the arm around obstacles, and concludes that a path to the target can not be found if this is the case.

(5) Implementation and experimental results. Two experiments demonstrating the operation of the motion planning algorithm are presented. They involve obstacles whose number, shape, placement, and size are unknown to the motion planning system. During the experiments, no contact with the obstacles took place, and the motion was completed

in a reasonable time. The experiments show that the system can indeed operate safely in a rather complex crowded environment.

Besides robot arm manipulators, the sensitive skin sensor system can be added to a piece of machinery in almost any application requiring generation of mechanical motion. Even assuming careful programming by the user, unexpected obstacles in the work space of a robot can always occur. If the robot is equipped with a sensitive skin, it can avoid such obstacles, saving undesirable down-time of the equipment, unacceptable human injury, and costly damage to the robot arm or environment.

There are other uses for the sensitive skin and motion planning system that were not investigated in this work. These include obstacle characterization and location, and directed control of the sensor scanning sequence in order to increase the sensor polling rate.

The motion planning methodology applied to the articulated robot arm in our system can also be used with other two or three degree of freedom arms with different kinematic configurations. Following the major requirement of the sensor skin discussed in Section 2.1, all parts of the robot arm that can potentially interact with the obstacles in its environment should be covered by a sensitive skin. On the algorithmic level, the local normal generation procedure can be similarly derived for arms of other kinematic configurations using a process analogous to that described in Chapter 3. The Step Planning method should also extend readily to robot arms of other kinematic configurations. In the area of Automatic Motion Planning, algorithms are already known that guarantee convergence to the desired target position without a complete search for three dimensional robot arms of various kinematics [58].

The sensitive skin and step planning algorithm can also be applied to robot arms that use more than three degrees of freedom for positioning the arm end point. These arms are called *kinematically redundant* because a particular point in the work space of the robot can be reached via an infinite number of arm configurations [29]. Here, the concept of the

local normal vector at the contact point between the robot arm and the obstacle in C-space is still applicable. Similar to the three dimensional case presented in this thesis, the higher-dimensional local normal describes the conditions necessary to locally slide the arm along the contact point. Developing specific techniques for deciding how the sliding along obstacles and motion in free space should take place, or in other words, how to accomplish automatic motion planning for redundant arms with sensitive skin, is an issue for future research.

Appendix A

The 2D Sensor Schematic

The schematic of the 2D sensor module is shown in Figure A-1. Only two degrees of freedom of the arm are considered here. Incident light at a given sensor pair is converted into a current by the phototransistor OP1, that is selected by an analog multiplexer, IC1. This signal is then high pass filtered and amplified by IC2, an operational amplifier, which is then connected to IC3, the PLL (see Section 2.2). The VCO signal of the PLL is available on the exterior of IC3's package. This signal is connected to IC5, a one-shot timer that generates a clean pulse for transmission. The duration of this pulse is 5 μsec, and is repeated at a rate of about 10kHz. The output of IC5 is boosted by the high current amplifiers T1 and T2, and is then switched to the appropriate IRLED by IC6, an analog multiplexer. The OP2 IRLED is pulsed 'on' at currents of 1 Ampere. Both multiplexers receive the same 4 bit address from the computer controlling the sensor system, resulting in the IRLED and the corresponding phototransistor being addressed as a pair.

In addition to the PLL, IC3 also contains the second multiplier that multiplies the VCO signal with the amplified phototransistor current from IC2. The output of the multiplier, available on the exterior of IC3's package, is amplified and level shifted by IC4, an operational amplifier. This signal, marked 'analog out (Vout)' on Figure A-1, is proportional to the intensity of reflected infra red light from the IRLED, and serves as the proximity signal. A typical sensor response is shown in Figure A-2 : the test object presents of a 5x5cm piece of paper. IC3 also has additional circuitry that thresholds the output of the second multiplier. This digital signal is marked 'digital out' on Figure A-1, and provides for an on-off type indication of the presence of an obstacle.

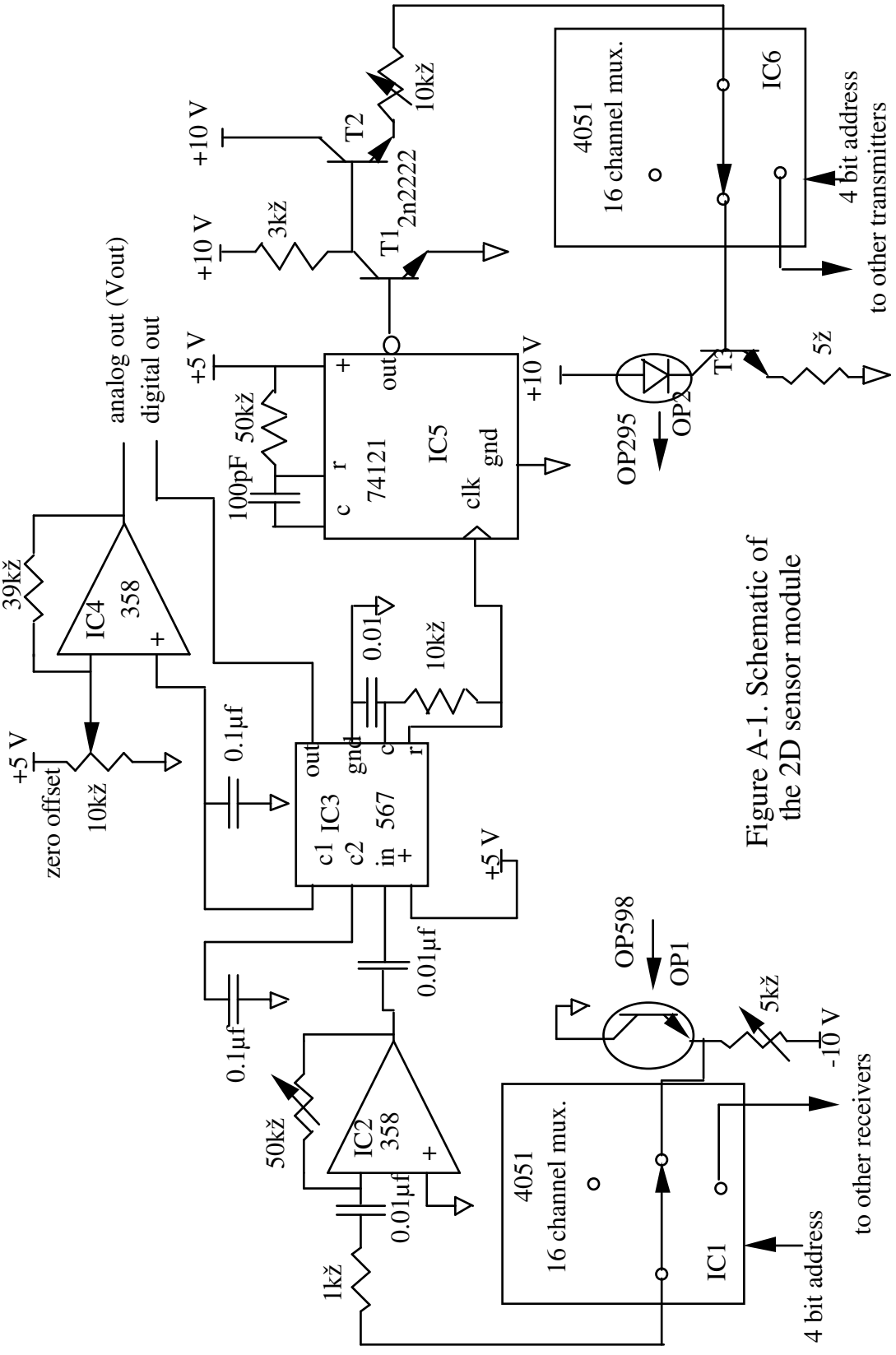


Figure A-1. Schematic of the 2D sensor module

The repetition rate of the one-shot timer, IC5 , is varied from one sensor module to another, allowing interference-free operation under the situation when two different modules are directly facing each other. This situation can occur when the angle between the two links is small. Then, the light emitted from one module, although shining directly into the detector of some other module, will not affect the response of the other module.

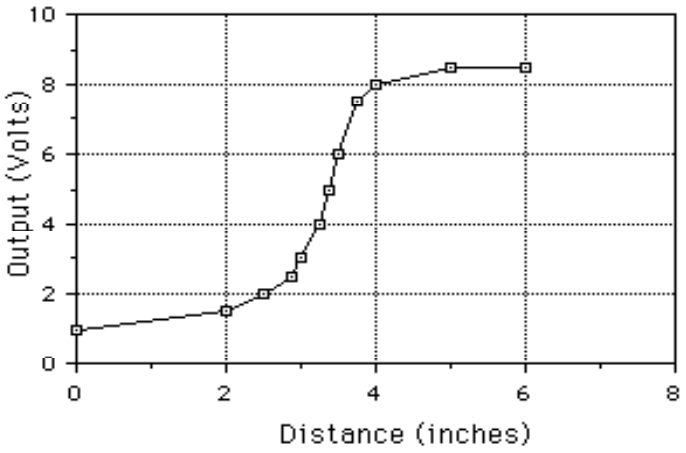


Figure A-2.Typical response of the 2D sensor.

Appendix B

Calculation of the Local Tangent in the 2D Case

The following derivations are valid for a 2 degrees of freedom revolute - revolute arm, Figure B-1, but can be similarly derived for other kinematic configurations. Whenever an obstacle is encountered, the arm attempts to slide along its surface. This sliding is accomplished by a coordinated motion of joints J_1 and J_2, based on the value of the local tangent. To find the latter, the slope of the local tangent $\frac{d\Theta_2}{d\Theta_1}$ is computed at every point along the contour, using the procedure described below.

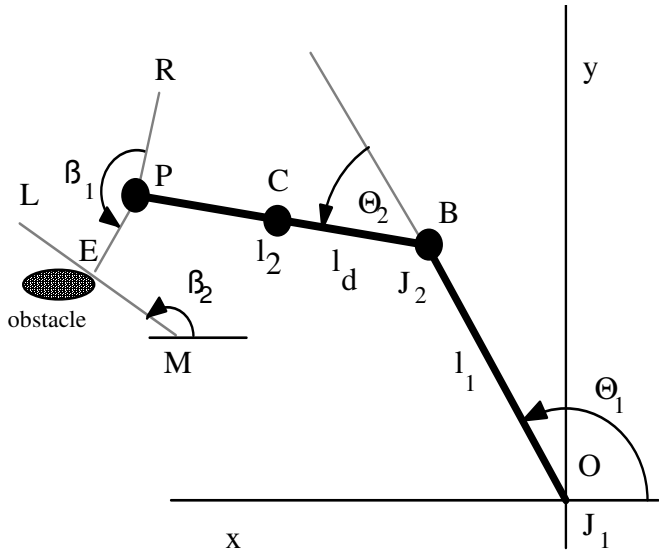


Figure B-1. Sketch of the 2D arm with revolute joints.

Consider a simple two-link planar robot arm with two revolute joints, Θ_1 and Θ_2, Figure B-1. Link 1 and link 2 of the arm are represented by the line segments OB and BP, of lengths l_1 and l_2, respectively; l_d is the distance between the point C and B; J_1 and J_2 are the arm joints. Point P represents the wrist; point B, which coincides with joint J_2,

represents the arm elbow; point O is fixed and represents the origin of the reference system. Angle β_1 is the angle between the line PE and PR, and angle β_2 is the angle between the line LM and the x-axis.

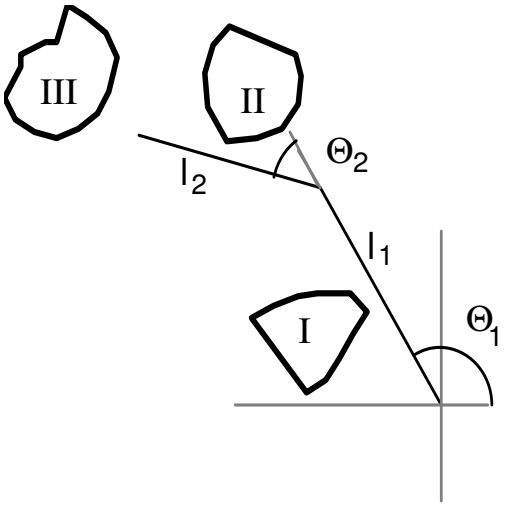


Figure B-2. Obstacle types.

Depending on their location relative to the arm, obstacles that may occur in the work space are divided into three categories, easily recognizable by the sensor system: Type I, Type II, and Type III, Figure B-2. Note that the notion of the obstacle type is relative: the same physical obstacle can correspond to different types at different moments, depending on what part of the arm interacts with it. Now we consider each of the types.

Type I are those obstacles that obstruct link l_1. Since link l_2 is irrelevant in such cases, then $d\Theta_1 = 0$ and $d\Theta_2 \neq 0$. Therefore, the local tangent to the obstacle boundary in this case is parallel to the Θ_2-axis.

Type II are those obstacles that obstruct link l_2. Assume that link l_2 of the arm is obstructed at point C by a Type II obstacle, at the distance l_d from the joint J_2, Figure B-1. Then, the estimates of $d\Theta_1$ and $d\Theta_2$ at C can be found as follows. Write the expressions

for the x and y coordinates of C, c_x and c_y, respectively:

$$\begin{aligned} c_x &= l_1 \cos(\Theta_1) + l_d \cos(\Theta_1 + \Theta_2) \\ c_y &= l_1 \sin(\Theta_1) + l_d \sin(\Theta_1 + \Theta_2) \end{aligned} \tag{B.1}$$

Take total derivatives:

$$\begin{aligned} dc_x &= -l_1 \sin(\Theta_1) d\Theta_1 + dl_d \cos(\Theta_1+\Theta_2) - l_d \sin(\Theta_1+\Theta_2) d\Theta_1 - l_d \sin(\Theta_1+\Theta_2) d\Theta_2 \\ dc_y &= l_1 \cos(\Theta_1) d\Theta_1 + dl_d \sin(\Theta_1+\Theta_2) + l_d \cos(\Theta_1+\Theta_2) d\Theta_1 + l_d \cos(\Theta_1+\Theta_2) d\Theta_2 \end{aligned} \tag{B.2}$$

Since C is a stationary point, $dc_x = dc_y = 0$. Find dl_d from both equations of (B.2):

$$dl_d = \frac{[-l_1 \cos\Theta_1 - l_d \cos(\Theta_1+\Theta_2)] d\Theta_1 - l_d \cos(\Theta_1+\Theta_2) d\Theta_2}{\sin(\Theta_1+\Theta_2)} \tag{B.3}$$

and

$$dl_d = \frac{[l_1 \sin\Theta_1 + l_d \sin(\Theta_1+\Theta_2)] d\Theta_1 + l_d \sin(\Theta_1+\Theta_2) d\Theta_2}{\cos(\Theta_1+\Theta_2)} \tag{B.4}$$

Equating the right hand sides and eliminating the denominators in (B.3) and (B.4), obtain:

$$\begin{aligned} &[\cos(\Theta_1+\Theta_2)(-l_1\cos\Theta_1 - l_d\cos(\Theta_1+\Theta_2)) \\ &-\sin(\Theta_1+\Theta_2)(l_1\sin\Theta_1 + l_d\sin(\Theta_1+\Theta_2))]\, d\Theta_1 \\ &= l_d \left[\cos^2(\Theta_1+\Theta_2) + \sin^2(\Theta_1+\Theta_2)\right] d\Theta_2 \end{aligned} \tag{B.5}$$

Now, the ratio $d\Theta_2 / d\Theta_1$ is found as

$$\begin{aligned} \frac{d\Theta_2}{d\Theta_1} &= \frac{1}{l_d}\left(-l_1\cos\Theta_1\cos(\Theta_1+\Theta_2) - l_1\sin\Theta_1\sin(\Theta_1+\Theta_2) - l_d\right) \\ &= -\left(\frac{l_1}{l_d}\left\{\cos\Theta_1\left(\cos(\Theta_1+\Theta_2)\right) + \sin\Theta_1\left(\sin(\Theta_1+\Theta_2)\right)\right\} + 1\right) \end{aligned} \tag{B.6}$$

After simplification, the local tangent to a Type II obstacle at point C is given by:

$$\frac{d\Theta_2}{d\Theta_1} = -\left(\frac{l_1}{l_d}\cos\Theta_2 + 1\right) \tag{B.7}$$

Type III are those obstacles that obstruct the arm wrist, point P, Figure B-1. In addition, for certain arms such as the PUMA 562 whose elbow does not correspond to the arm joint J_2, obstacles obstructing the elbow can also be considered of Type III, refer to Section 3.4.1. To find the local tangent in this case, the inverse Jacobian of the arm is used. Denote $dx = dP_x$ and $dy = dP_y$ in case the wrist is obstructed, and $dx = dB_x$ and $dy = dB_y$ in case the elbow is obstructed. Then, the following relationship holds:

$$d\Theta_1 = \frac{l_2\cos(\Theta_1+\Theta_2)\,dx + l_2\sin(\Theta_1+\Theta_2)\,dy}{l_1 l_2\cos\Theta_2}$$
$$d\Theta_2 = \frac{\left[-l_1\cos\Theta_1 - l_2\cos(\Theta_1+\Theta_2)\right]dx - \left[l_1\cos\Theta_1 - l_2\cos(\Theta_1+\Theta_2)\right]dy}{l_1 l_2\cos\Theta_2} \tag{B.8}$$

Sliding of the wrist P along the obstacle corresponds to its moving along the line segment LM, Figure B-1. Define $ß_1$ as the angle between the line perpendicular to link 2 and the line from P to the obstacle, and $ß_2$ as the angle between the line LM and the positive x-axis. Then,

$$\frac{dy}{dx} = \tan ß_2$$

or

$$dy = dx \tan ß_2 \tag{B.9}$$

where $ß_2 = \Theta_1 + \Theta_2 + ß_1 - \pi$. Substituting the expression for dy from (B.9) into (B.8), obtain the ratio $d\Theta_2 / d\Theta_1$:

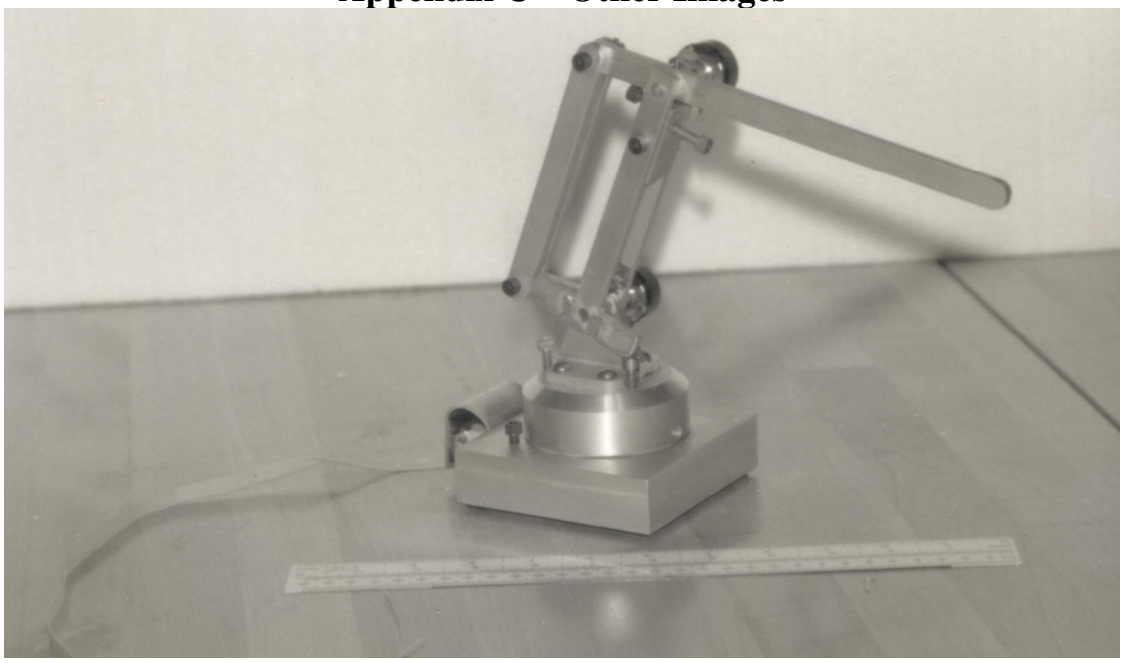
$$\frac{d\Theta_2}{d\Theta_1} = -\left(\frac{l_1 \cos\Theta_1 + l_2\left(\cos\Theta_1\cos\Theta_2 - \sin\Theta_1\sin\Theta_2\right) + \left[l_1 \sin\Theta_1 + l_2\left(\sin\Theta_1\cos\Theta_2 + \cos\Theta_1\sin\Theta_2\right)\right] \tan ß_2}{l_2\left[\cos\Theta_1\cos\Theta_2 - \sin\Theta_1\sin\Theta_2 + \left(\sin\Theta_1\cos\Theta_2 + \cos\Theta_1\sin\Theta_2\right) \tan ß_2 \right]} \right) \tag{B.10}$$

After simplification, the expression for the local tangent in the case of a Type III obstacle appears as:

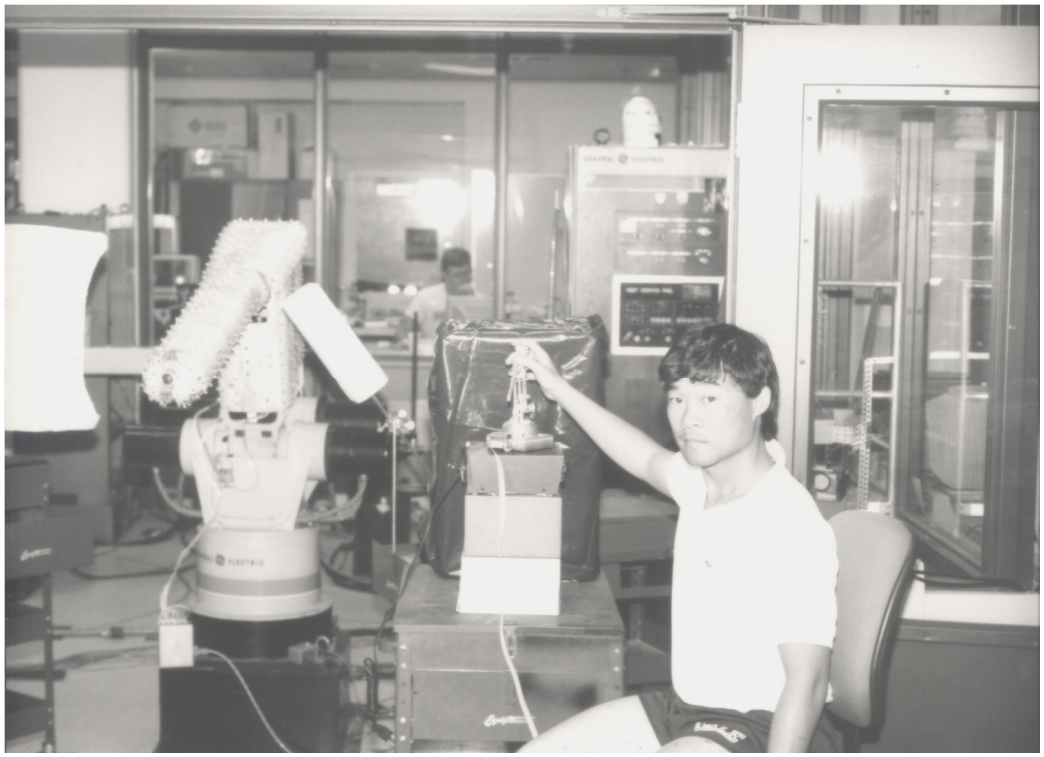
$$\frac{d\Theta_2}{d\Theta_1} = -\left(\frac{\frac{l_1}{l_2} + \cos\ \Theta_2 + \sin\ \Theta_2 \tan\left(\Theta_2 + ß_1\right)}{\cos\Theta_2 + \sin\ \Theta_2 \tan\left(\Theta_2 + ß_1\right)} \right) \tag{B.11}$$

Summarizing, for a Type I obstacle, the local tangent is parallel to the Θ_2-axis, and for Type II and Type III obstacles it is given by the expressions (B.7) and (B.11), respectively.

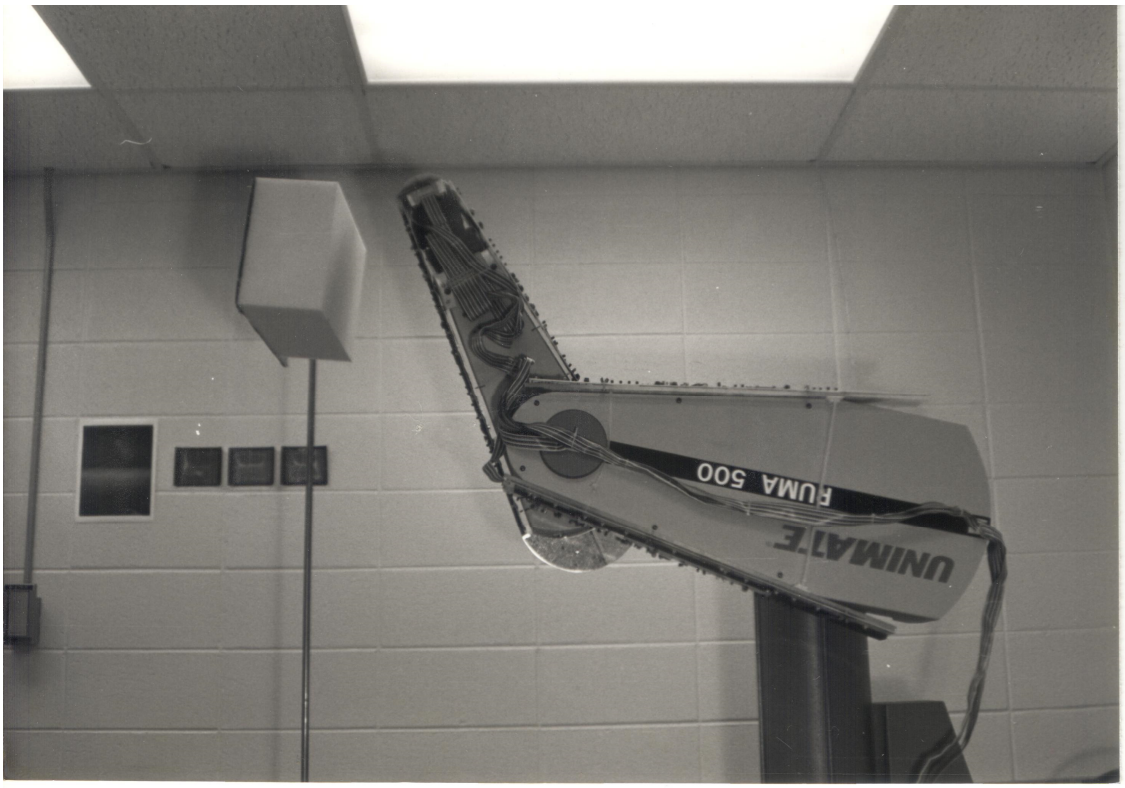
Appendix C – Other Images



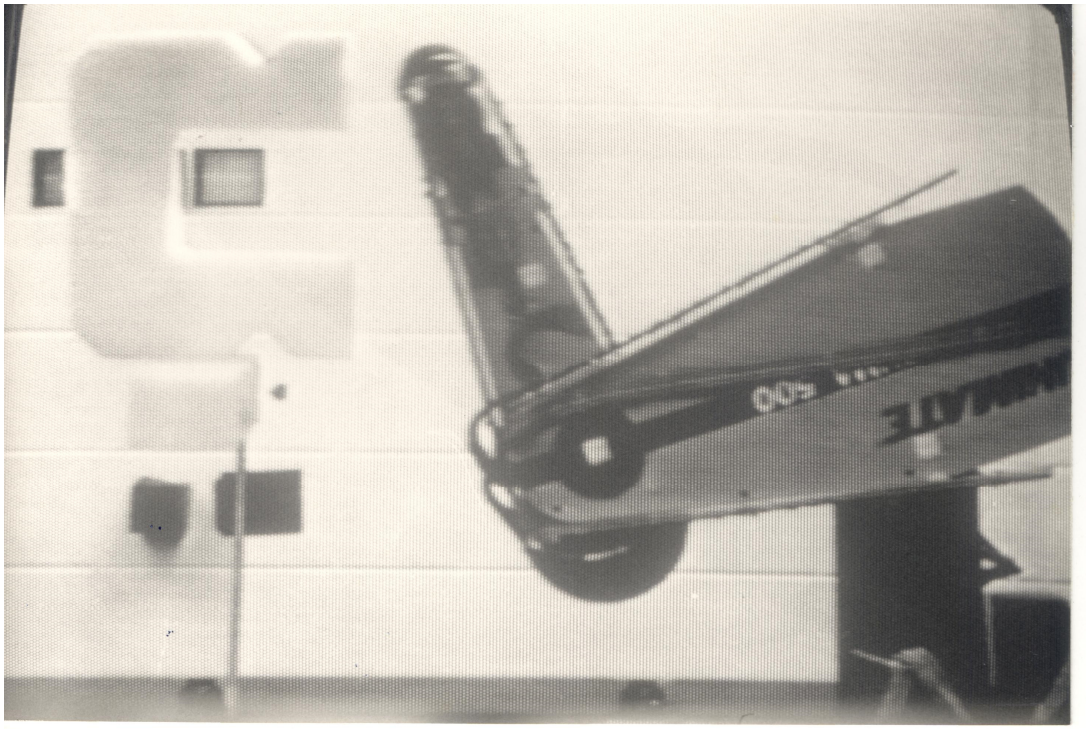
Mini-Master Controller.



Mini-Master in use.

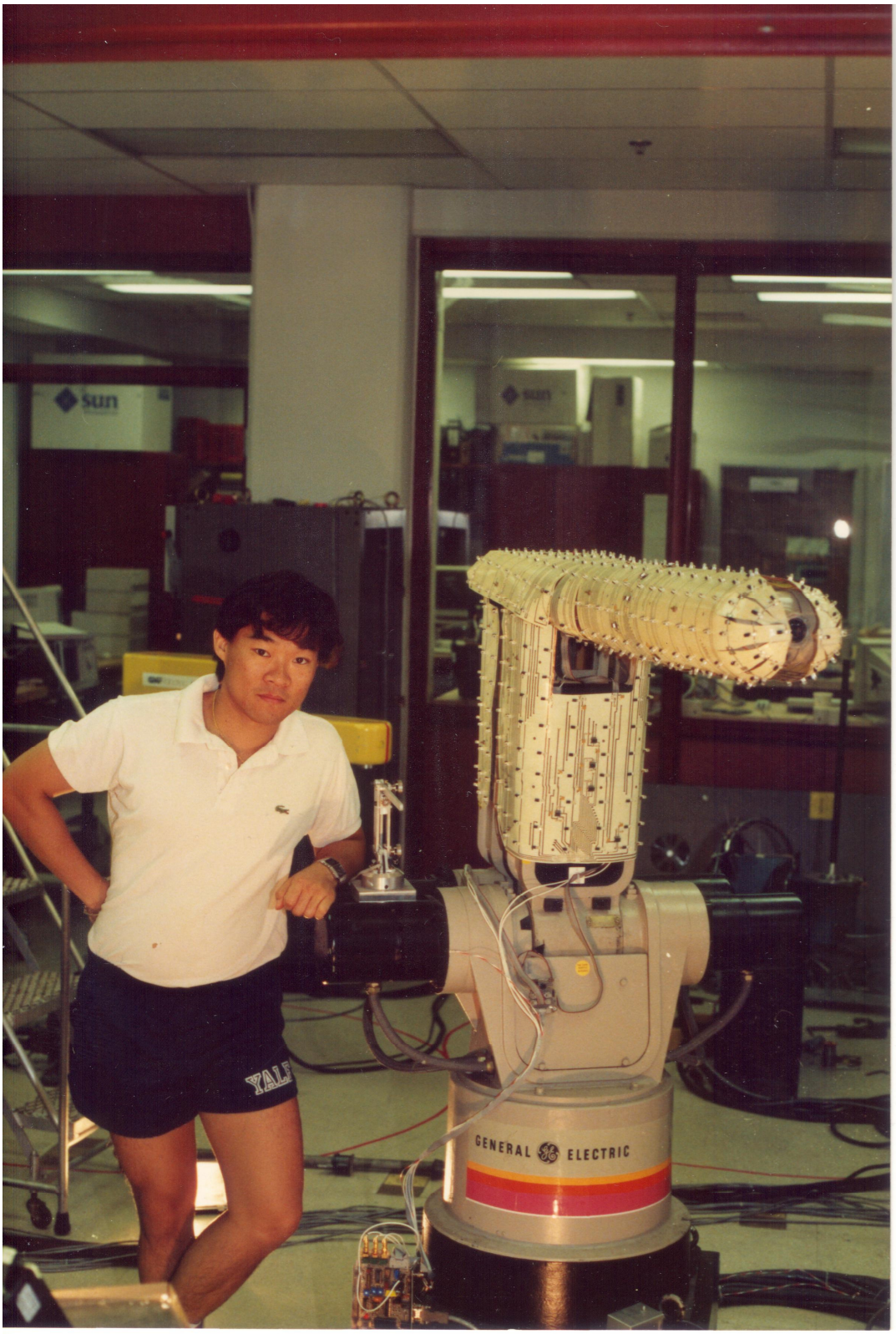


PUMA in the 2-D tests.

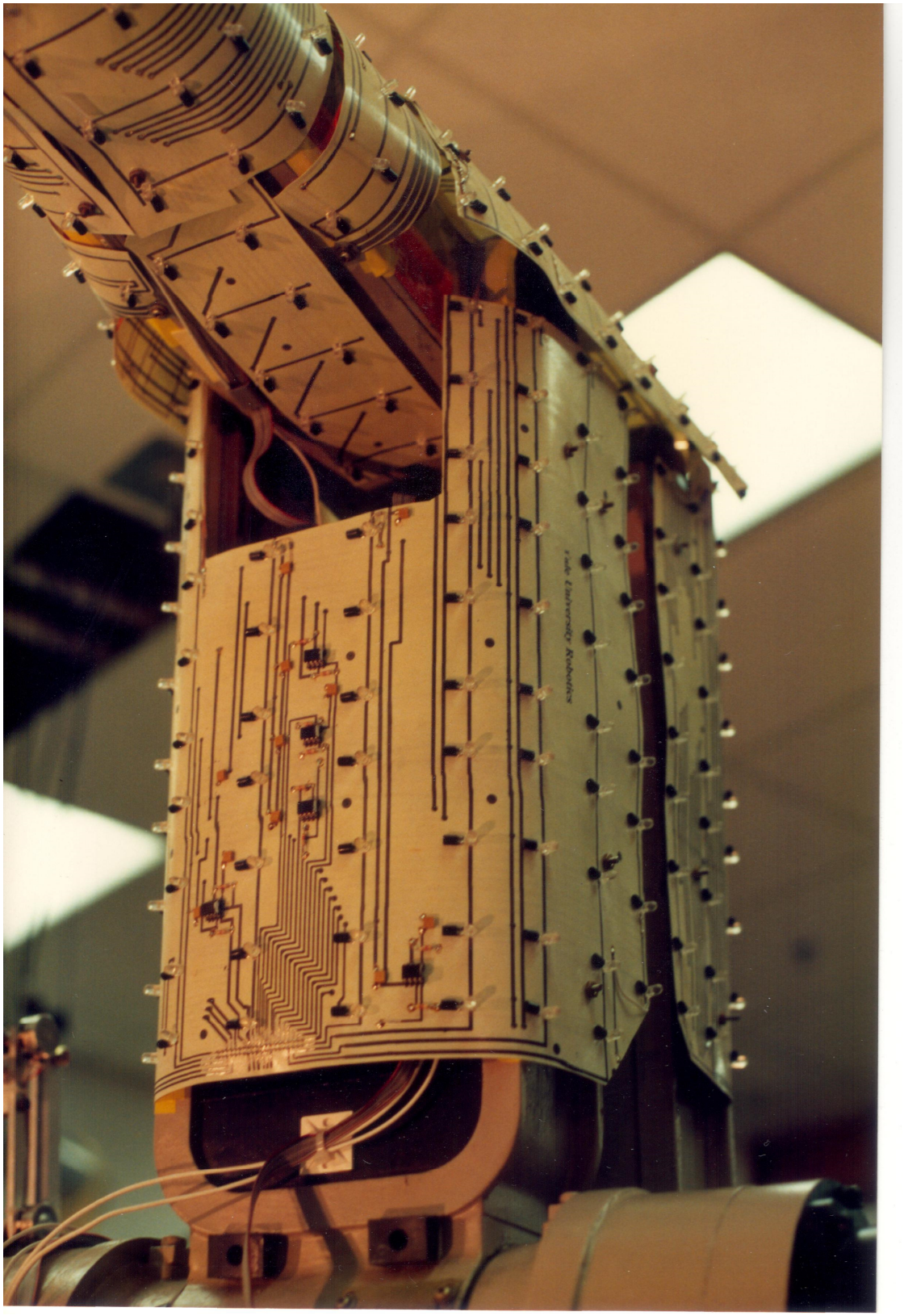


PUMA with C-Shaped object.

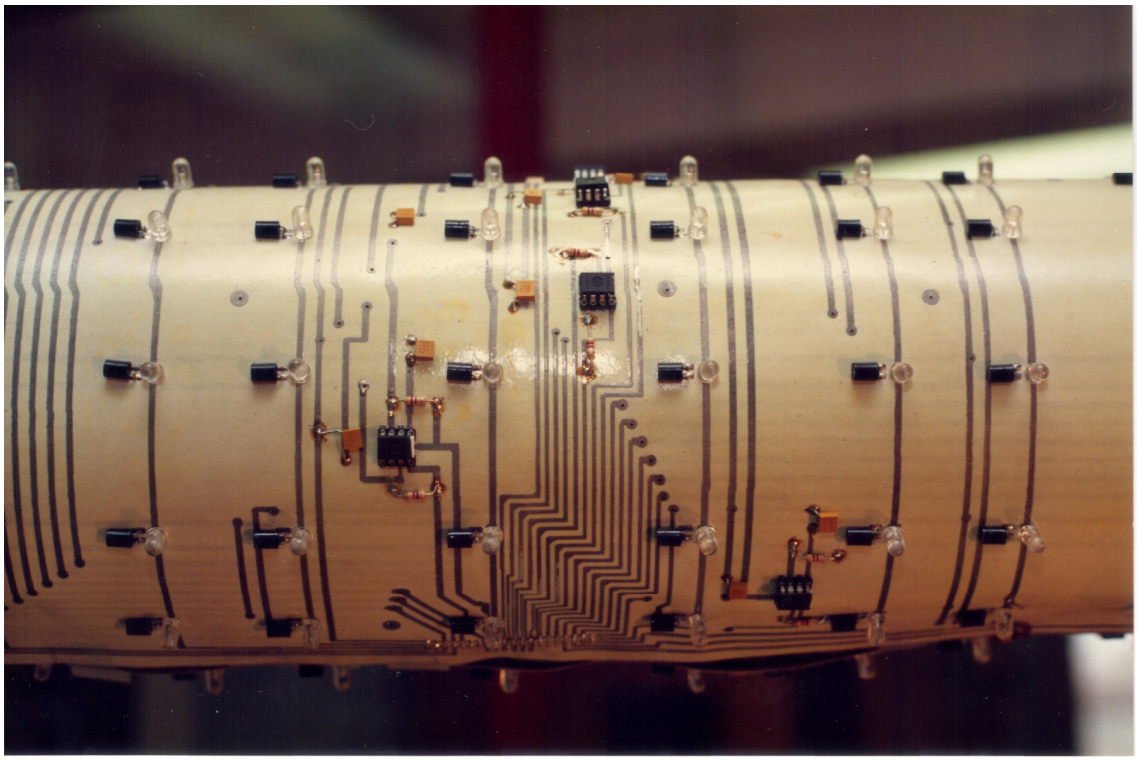
PUMA with 2-D sensors



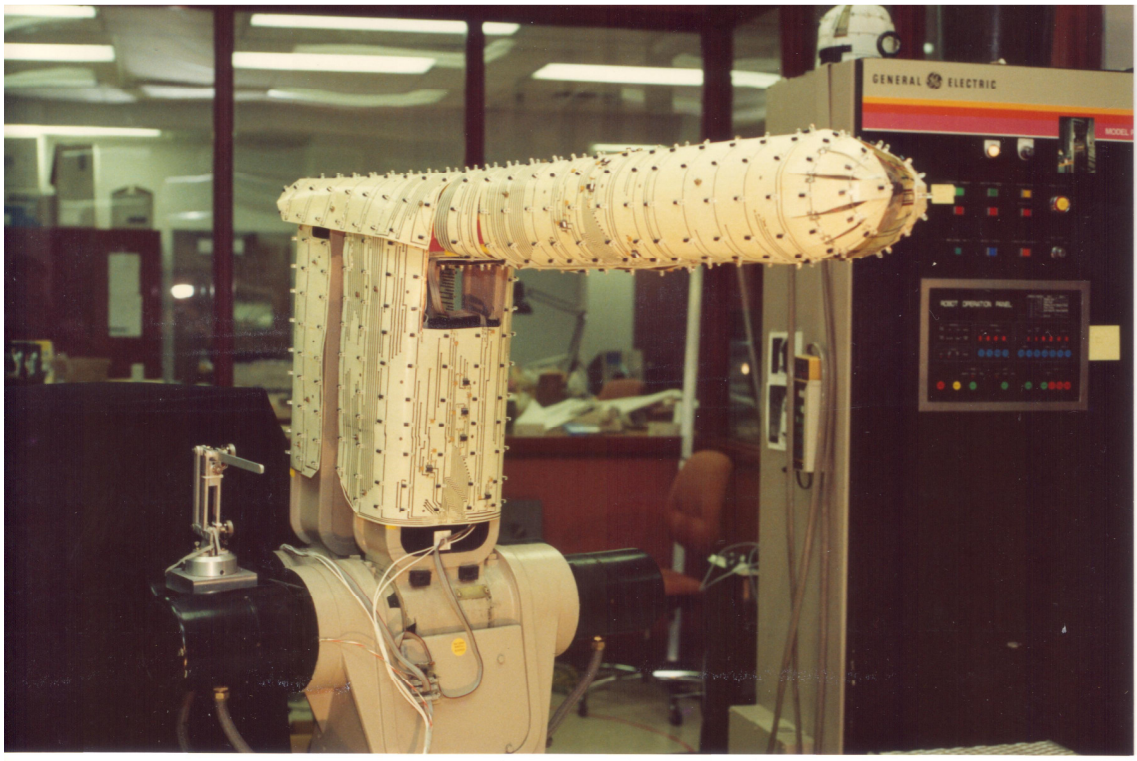
Author with the sensitive skin cladded arm. Circa 1990.



View of the elbow hinge of the arm.



Close-up of the arm sensors.



Overall view of the arm and controller in the background.

Bibliography

1. S. Ahmad, C. Lee, Shape Recovery from Robot Contour-Tracking with Force Feedback, Proc. 1990 IEEE Conference on Robotics and Automation, Cincinnati, OH, May 1990.

2. P. Allen, Mapping Haptic Exploratory Procedures to Multiple Shape Representations, Proc. 1990 IEEE Conference on Robotics and Automation, Cincinnati, OH, May 1990.

3. Catalog S-1, Automatic Timing and Control Co., King of Prussia, PA, 1983.

4. P. Backes, K. Tso, UMI: An Interactive Supervisory and Shared Control System for Telerobotics, Proc. 1990 IEEE Conference on Robotics and Automation, Cincinnati, OH, May 1990.

5. D.J. Balek and R.B. Kelly, Using Gripper Mounted Infrared Proximity Sensors for Robot Feedback Control, Proc. 1985 IEEE Conference on Robotics and Automation, St.Louis, Missouri, March 1985.

6. D. Ballard, C. Brown, Computer Vision, Prentice-Hall, New Jersey, 1982.

7. M. Buehler, L. Whitcomb, F.Levin, D. Koditschek, A Distributed Message Passing Computational and I/O Engine for Real-time Motion Control, Proc. American Control Conference, American Control Society, Pittsburgh, PA, June 1989.

8. Y.C. Chen, M. Vidyasagar, Some Qualitative Results on the Collision-Free Joint Space of Planar n-DOF Linkage. Proc. 1987 IEEE International Conference on Robotics and Automation, Raleigh, North Carolina, April 1987.

9. E. Cheung and V. Lumelsky, Proximity Sensing in Robot Manipulator Motion Planning: System and Implementation Issues, Technical Report No. 8802, Center for Systems Science, Yale University, January 1988.

E. Cheung and V. Lumelsky, Proximity Sensing for Robot Manipulator Motion Planning: System and Implementation Issues, IEEE Transactions on Robotics & Automation, December 1989.

10. E.Cheung and V. Lumelsky, Real-time Path Planning Procedure for a Whole-Sensitive Robot Arm Manipulator, Technical Report No.8913, Center for Systems Science, Yale University, November 1989.
11. E. Cheung and V. Lumelsky, A Sensor Skin System for Motion Planning Control of Robot Arm Manipulators, Technical Report No. 8915, Center for Systems Science, Yale University, December 1989.
12. E. Cheung and V. Lumelsky, Motion Planning for Robot Arm Manipulators with Proximity Sensors, Proc. 1988 IEEE Conference on Robotics and Automation, Philadelphia, PA, April 1988.
13. E. Cheung and V. Lumelsky, Development of Sensitive Skin for a 3d Robot Arm Operating in an Uncertain Environment, Proc. 1989 IEEE Conference on Robotics and Automation, Scottsdale, AZ, May 1989.
14. E. Cheung and V. Lumelsky, Motion Planning for a Whole-Sensitive Robot Arm Manipulator, Proc. 1990 IEEE Conference on Robotics and Automation, Cincinnati, OH, May 1990.
15. I. Cox, Blanche: An Autonomous Robot Vehicle for Structured Environments, Proc. 1988 IEEE Conference on Robotics and Automation, Philadelphia, Pennsylvania, March 1988.
16. H. Durrant-Whyte, B.Y.S. Rao, H. Hu, Toward a Fully Decentralized Architecture for Multi-Sensor Data Fusion, Proc. 1990 IEEE Conference on Robotics and Automation, Cincinnati, OH, May 1990.
17. A. Elfes, Sonar Based Real-World Mapping and Navigation, IEEE Journal of Robotics and Automation, June 1987.
18. B. Espiau, J.Y. Catros, Use of Reflectance Sensors in Robotics Applications, IEEE transactions on Systems, Man, and Cybernetics, December 1980.
19. J. Evans, B. Krishnamurthy, W. Pong, T. Skewis, B. Barrows, S. King, C. Weiman, G. Engelberger, Creating smart robots for hospitals, Proceedings of Robots 13

Conference, Washington, DC, May 1989.

20. A. Flynn, Combining Sonar and Infrared sensors for Mobile Robot Navigation, International Journal of Robotics Research, Vol.7, No.6, December 1988.
21. A. Flynn and R. Brooks, MIT Mobile Robots - what's next? Proc. 1988 IEEE Conference on Robotics and Automation, Philadelphia, Pennsylvania, March 1988.
22. GE Solid State Data Book, High Reliability CMOS ICs, General Electric, Somerville New Jersey, 1988.
23. S. Gordon and W.Townsend, Integration of Tactile-Force and Joint Torque Information in a Whole-Arm Manipulators, Proc. 1989 IEEE Conference on Robotics and Automation, Scottsdale, AZ, May 1989.
24. Range Finder ICs and Related Components, Hamamatsu Technical Notes No. TN-108, Hamamatsu Corporation, Bridgewater, New Jersey, 1987.
25. S. Hayati, T. Lee, K. Tso, P. Backes, A Testbed for a Unified Teleoperated-Autonomous Dual-Arm Robotic System, Proc. 1990 IEEE Conference on Robotics and Automation, Cincinnati, OH, May 1990.
26. H. Hemami, Differential Surface Model for Tactile Perception of Shape and On-Line Tracking of Features, IEEE Journal of Systems, Man, and Cybernetics, March/April 1988.
27. C.A. Hoare, OCCAM 2 Reference Manual, Prentice-Hall, New York 1988.
28. P. Horowitz, W. Hill, The Art of Electronics, Cambridge University Press, Cambridge 1980.
29. J.P. Karlen, J.M. Thompson,J.D. Farrell, H.I. Vold, Reflexive Obstacle Avoidance for Kinematically-Redundant Manipulators, NASA Conference on Space Telerobotics, Pasadena California, January, 1989.
30. D. Kriegman, E.Triendle, T.Binford, A Mobile Robot: Sensing, Planning, and Locomotion, Proc. 1987 IEEE Conference on Robotics and Automation, Raleigh, NC, March 1987.

31. B. Krishnamurthy, B. Barrows, S. King, T. Skewis, W. Pong, C. Weiman, Help-mate: a mobile robot for transport applications, Proceedings of the 1988 SPIE Conference, Boston, MA, November 1988.
32. B. Lathi, Signals, Systems and Communication, John Wiley & Sons, New York 1965
33. Linear Data Book, National Semiconductor Corporation, 1982.
34. The U-System, Location Sensor Systems Inc., Holland, OH, 1983.
35. V. Lumelsky and A. Stepanov, Path Planning Strategies for a Point Mobile Automaton Moving Amidst Unknown Obstacles of Arbitrary Shape, Algorithmica, Springer-Verlag, 2, 1987.
36. V. Lumelsky, Effect of Kinematics on Motion Planning for Planar Robot Arms Moving Amidst Unknown Obstacles, IEEE Journal of Robotics and Automation, June 1987.
37. V. Lumelsky, K. Sun, On the Motion of Simple 3D Arm Manipulators with Uncertainty. Proc. 5-th Yale Workshop on Applications of Adaptive Systems Theory, New Haven, May 1987.
38. V. Lumelsky, Continuous Motion Planning in Unknown Environment for a 3D Cartesian Robot Arm, Proc. 1986 IEEE International Conference on Robotics and Automation, San Francisco, 1986.
39. V. Lumelsky, Dynamic Path Planning for a Planar Articulated Robot Arm Moving Amidst Unknown Obstacles, Automatica, Sept 1987.
40. V. Lumelsky, On Human Performance in Telerobotics. IEEE International Workshop on Intelligent Robots and Systems (IROS '90), Tsuchiura, Japan, July 1990.
41. D. McGovern, Factors affecting control allocation for augmented remote manipulation, Ph. D. Thesis, Mechanical Engineering Department, Stanford University, Stanford, California, 1974.
42. L. Matthies and S.A. Shafer, Error Modeling in Stereo Navigation, IEEE Journal of Robotics and Automation, June 1987.
43. G. Miller, R. Boie, and M. Sibilia, Active Damping of Ultrasonic Transducers for

Robotic Applications, Proc. IEEE International Conference on Robotics, Atlanta, Georgia, March 1984.

44. W. Newman, High-Speed Robot Control in Complex Environments, Ph.D. Thesis, MIT, October, 1987.
45. Opto-Diode Data Book, Opto-Diode Corporation, 1988.
46. R.P. Paul, Robot Manipulators: Mathematics, Programming, and Control, MIT Press, 1983.
47. K. Pribadi, J. Bay, H. Hemami, Exploration and Dynamic Shape Estimation by a Robotic Probe, IEEE Transactions on Systems, Man and Cybernetics, Vol.19, no.4, July/August, 1989.
48. K. Roberts, Robot Active Touch Exploration: Constraints and Strategies, Proc. 1990 IEEE Conference on Robotics and Automation, Cincinnati, OH, 1990.
49. J.S. Schoenwald, A.W. Thiele and D.E. Gjellum, A Novel Fiber Optic Tactile Array Sensor, Proc. 1987 IEEE Conference on Robotics and Automation, Raleigh, North Carolina, April 1987.
50. B.J. Schroer, Telerobotics Issues in Space Applications, Robotics and Autonomous Systems, Vol. 4, No. 3., November 1988.
51. J.T. Schwartz and M.Sharir, On the "Piano Movers" Problem. II. General Techniques for Computing Topological Properties of Real Algebraic Manifolds, Advances in Applied Mathematics, 1983.
52. M. Sharir, Algorithmic Motion Planning in Robotics, Computer, March 1989.
53. T. Skewis, J. Evans, V. Lumelsky, B. Krishnamurthy, B. Barrows, Motion Planning for a Hospital Transport Robot, Yale Technical Report No. 9001, March 1990.
54. Electro Optic Devices, Controls and Systems, Scientific Technology Incorporated, Mountain View, CA, 1984.
55. R. Sedgewick, Algorithms, Addison-Wesley Publishing Co. Reading, MA, 1984.
56. R. Shoureshi, R. Evans, W. Stevenson, Optically Driven Learning Control for Industrial

Manipulators, IEEE Control Systems Magazine, October 1989.

57. W. Stevenson, Optical Sensor for Alignment, Distance Measurement, and Feature Location in Automated Manufacturing Operations, Proc. of the 7th International Congress on Application of Lasers and Electrooptics ICALEO '88, Santa Clara, CA, 1988.
58. K. Sun and V. Lumelsky, Motion Planning with Uncertainty for a 3d Cartesian Robot Arm, 5th International Symposium on Robotics Research, Tokyo, Japan, August 1989.
59. K. Sun, V. Lumelsky, Computer Simulation of Sensor-based Robot Collision Avoidance in an Unknown Environment, Robotica, Vol.5, 1987.
60. R. Taylor, M. Mason, Sensor-based manipulation planning as a game with nature, 4th International Symposium Robotics Research, August 1987.
61. Transputer Development System, Prentice-Hall, New York 1988.
62. T. Tsujimura, T. Yabuta, Object Detection by Tactile Sensing Method Employing Force/Torque Information, IEEE Transactions on Robotics and Automation, Vol. 5, no 4, August, 1989.
63. The TTL Data Book, Texas Instruments, 1981.
64. Ultrasonic Range Finders, Polaroid Corporation, 1982.
65. M.Winston, Opto Whiskers for a Mobile Robot, Robotics Age, Vol. 3 #1, 1981.
66. C.K. Yap, Algorithmic Motion Planning. Advances in Robotics, vol. 1: Algorithmic and Geometric Aspects, (J.T. Schwartz and C. Yap, Eds.), Hillsdale, New Jersey: Erlbaum, 1987.