California AHMCT Research Center
University of California at Davis
California Department of Transportation

ON-LINE CONTROL ON AN INDUSTRIAL ROBOT FOR CRACK SEALING USING PROXIMITY SENSING

Phillip A. Kahrl
Bahram Ravani

AHMCT Research Report
UCD-ARR-92-12-01-01

Interim Report of Contract
IA65P307 (SHRP H-107A)

December 1, 1992

* This work was supported by the Strategic Highway Research Program and the California Department of Transportation (Caltrans) Advanced Highway Maintenance and Construction Technology Center at UC, Davis

Abstract

Each year, the state of California alone spends approximately $10 million on the sealing and filling of pavement cracks in order to retain structural integrity of roadways and extend time between major rehabilitions. The associated costs are approximately $1800 per lane mile with 66% attributed to labor, 22% to equipment and 12% to materials.

Currently, research is underway at the University of California, Davis to design and build a prototype Automated Crack Sealing Machine (ACSM). The subsystems of the ACSM will detect cracks and guide process equipment over the cracks to rout, heat, clean and seal. For transverse cracking, the process equipment will be manipulated with an industrial robot arm mounted on the rear of the ACSM support vehicle.

The purpose of this thesis is to develop and test algorithms for crack-following with a robot manipulator using the relative proximity sensor and to expand upon these algorithms to incorporate data from the machine vision system as well. The result will be a flexible and robust control algorithm that will incorporate all available data for control and will be able to function with failures or errors in the machine vision system and associated subsystems.

This thesis will briefly address the overall problem of crack sealing. The selection of an industrial robot and the selection and operation of the relative proximity sensor will also be addressed. The algorithm for control of the robot with the local proximity sensor relies heavily on control algorithms developed for compliant motion with endpoint force sensing.

TABLE OF CONTENTS

LIST OF APPENDICES

LIST OF FIGURES

ABSTRACT

Each year in California, the State Department of Transportation (CalTrans) spends over $100 million maintaining approximately 33,000 lane-miles of flexible pavement (Asphalt Concrete - AC) and 13,000 lane-miles of rigid pavement (Portland Cement Concrete - PCC). A portion of these maintenance activities involves the sealing and filling of cracks (approximately $10 million per year) which, when properly performed, can help retain the structural integrity of the roadway and considerably extend the mean time between major rehabilitations. A typical operation to seal meandering cracks in AC pavement involves a crew of about eight individuals which can seal between one and two lane miles per day. The associated costs are approximately $1800 per mile with 66% attributed to labor, 22% to equipment and 12% to materials.

Currently, research is underway at the University of California, Davis to design and build a prototype Automated Crack Sealing Machine (ACSM). The purpose of the ACSM will be to address both longitudinal and transverse cracks in the pavement. The subsystems of the ACSM will detect cracks and guide process equipment over the cracks to rout, heat, clean and seal. For transverse cracking, the process equipment will be manipulated with an industrial robot arm mounted on the rear of the ACSM support vehicle.

Cracks will be detected by a machine vision system located at the front of the ACSM support vehicle. The presence of a crack will be verified by an optical relative proximity sensor mounted on the robot end-effector. The relative proximity sensor will also detect the exact positions of cracks more precisely than the machine vision system. Control of the manipulator will utilize data from both the machine vision system and the relative proximity sensor.

The purpose of this report is to develop and test algorithms for crack-following using the relative proximity sensor and to expand upon these algorithms to incorporate

data from both the machine vision system and the proximity sensor. The result will be a flexible and robust control algorithm that will incorporate all available data for control and will be able to function with failures or errors in the machine vision system and associated subsystems.

This report will briefly address the overall problem of crack sealing. The selection of an industrial robot and the selection and operation of the relative proximity sensor will also be addressed. The algorithm for control of the robot with the local proximity sensor relies heavily on control algorithms developed for control with endpoint force sensing.

Once algorithms have been developed for crack following using proximity sensing, control structures will be introduced that use data from the machine vision system as well. The result will be a complete control algorithm for the Robot Positioning System of the ACSM.

CHAPTER 1 - INTRODUCTION

This chapter describes the motivation for and development of an automated crack sealing machine. Much of this chapter has been extracted from (Velinsky, 1991) and (Schulteiss and Velinsky, 1991). Details on the development of the automated crack sealing machine can be found in these references.

1.1 - PROBLEM DESCRIPTION

Worldwide, a tremendous amount of resources are expended annually maintaining highway pavement. In the state of California alone, the Department of Transportation (Caltrans) spends approximately $100 million per year maintaining about 33,000 lane-miles of Asphalt Concrete (AC) pavement and 13,000 lane-miles of Portland Cement Concrete (PCC) pavement. Approximately $10 million of this maintenance budget is used to seal and fill cracks in the pavement. When properly performed, crack sealing and filling can help retain the structural integrity of the roadway and considerably extend the mean time between major rehabilitation.

Sealing and filling of cracks is a labor-intensive and tedious operation. A typical operation for sealing cracks in AC pavement involves a crew of eight persons. This crew can seal approximately 2 lane-miles per day at a cost of about $1800 per lane-mile. 66% of this cost is attributed to labor, 22% to equipment, and 12% to materials. Furthermore, the procedure is not standardized and there is a large distribution in the quality of the resultant seal. Additionally, the workers must be on the road surface adjacent to moving traffic, thus exposing them to a great deal of physical danger.

1.2 - THE NEED FOR AN AUTOMATED CRACK SEALING/FILLING MACHINE

The final goal of the SHRP H-107A project, of which this report project is a part, is to develop a prototype automated crack sealing machine that will sense, prepare, and seal (or fill) cracks and joints in AC and PCC pavement. The goal of this project is to

investigate the development of such a machine considering only the topics of road surface preparation, positioning system (system for positioning the cleaning, heating, routing, and sealing/filling devices) configuration selection, and positioning system concept design selection. The primary objectives of the project are to:

- Increase the cost-effectiveness of the crack sealing and filling operations,
- Increase the quality, consistency, and life of the resultant seals and fills,
- Increase the safety of work crews and highway users
- Increase the use of remote and automatic equipment operation and control to attain the above.

A machine that can satisfy the objectives listed above will have the added benefits of reducing lane and highway closures and thus, will play a significant role in the reduction of traffic congestion, a considerable problem in major urban regions around the world. The cost effectiveness of such a machine will be realized through a combination of increased speed and reduced manpower, in addition to the higher quality seal which will reduce the frequency of major highway rehabilitations.

1.3 - MACHINE SPECIFICATIONS

To have the greatest impact, such a machine should satisfactorily perform the following tasks automatically:

- Sense the occurrence and location of cracks in pavement.
- Prepare the crack and pavement surface for sealing/filling. This task includes the removal of vegetation, loose debris, dirt film, and moisture. In addition, preheating of the road surface may be necessary to ensure maximum sealant adhesion and refacing of reservoirs (routing) may be required.
- Prepare the sealant/filler for application; i.e., heat and mix the material, etc.
- Dispense the sealant/filler over the crack.
- Form the sealant/filler into the desired configuration
- Finish the sealer/filler.

The equipment prototypes may be derived from modifying existing equipment or from the development of new equipment (with preference given to suitable commercially available equipment), and each equipment design may include one or more pieces of equipment.

1.4 - MACHINE ARCHITECTURE

Two component systems will be utilized to address the spectrum of commonly occurring cracks to be sealed. The longitudinal crack sealing machine will seal construction joints at the edge of the roadway which run parallel to the roadway. The second machine is the general crack sealing machine which will address the more complex problem of sealing transverse and random cracks on the road surface.

The automated crack sealing machine (ACSM) architecture has the following major subsystems:

- Vision Sensing System (VSS)
- Local Sensing System (LSS)
- Applicator Peripherals System (APS)
  - Heating/Cleaning/Debris Removal Subsystem
  - Router
  - Sealant Applicator
- Robot Positioning System (RPS)
  - General Machine Positioning System
  - Longitudinal Machine Positioning System
- Vehicle Orientation and Control (VOC)
- Integration and Control Unit (ICU)
  - Systems Integration
  - Robot Path Planning (Off-Line)

The purpose of the VSS in conjunction with the LSS is to locate and describe pavement cracks. The APS includes the Heating/Cleaning/Debris Removal Subsystem,

the Router and the Sealant Applicator. The Heating/Cleaning/Debris Removal Subsystem will include all hardware and software necessary to heat and clean the road surface during crack preparation. The router will shape the crack to optimal sealing geometry. The Sealant Applicator is responsible for sealant dispensing and seal configuration. The Robot Positioning System is responsible for moving the applicator assembly along the crack during sealing. On the general machine, the General Machine Positioning System will consist of a robot manipulator and a controller. On the longitudinal machine, the Longitudinal Positioning System will position the applicator using a programmable controller and hydraulic actuators. The Vehicle Orientation and Control System will monitor changes in vehicle position from when the crack is sensed to when it is actually sealed. Finally, the Integration and Control Unit will oversee the entire crack sealing procedure by monitoring all peripherals to ensure proper operation and controlling communication between subsystems.

1.5 - MACHINE OPERATION

Machine operation will begin with detection of cracks by the VSS which will be mounted on the front of the support vehicle. The VSS scans the road, detects cracks and sends the crack positions to the ICU. The VOC transforms the crack locations to the workspace of the RPS. The LSS will be used to verify and precisely locate the cracks. The RPS will manipulate the equipment of the APS to heat clean and seal the cracks.

The purpose of this report is to develop control algorithms for the General Machine component of the Robot Positioning System. For simplicity, we will refer to the General Machine component of the Robot Positioning System as the RPS for the remainder of this report.

1.6 - CONTROL OF THE RPS

One of the primary tasks for design of the RPS was to implement ways of tracking cracks in the pavement using the sensor information available. Sensor information

includes crack locations in the world coordinate system obtained from the VSS and crack locations with respect to the end-effector of the manipulator using the LSS. The purpose of the LSS is to verify and precisely locate cracks after initial detection by the VSS. Data from the VSS is processed in order to detect pavement cracks by comparing gray shades. Once the data has been collected, a path is planned along a crack and transformed into the reference frame of the robot (Lasky and Ravani, 1993). The manipulator will then follow the crack by moving to points defined along the pre-planned crack. Vision data processing and path planning for crack data from the VSS is primarily a problem of off-line path planning and will not be addressed in this thesis.

The LSS is extremely accurate and is capable of measuring cracks in three dimensions. For this reason, the LSS can be used to verify the presence of pavement cracks and provide data to the robot in order to implement closed-loop control for crack following. Closed loop control with the RPS and the LSS can be used to account for inaccuracies in the VSS or as an override in case of failure of the VSS, the ICU or the VOC. The main limitation of the LSS is that it can only detect cracks along a single line of laser light; therefore, three-dimensional information can only be obtained by moving the LSS along a crack. Closed-loop control of the RPS for crack following using data from the LSS will be the primary focus of this thesis.

1.6.1 - PREVIOUS WORK

A thorough search was conducted in the literature for methods of on-line control for industrial robots. The search included on-line control of robots using vision and force sensing and techniques used in automated welding.

Important considerations in the literature search concerned not only the nature of the task being performed but the type of hardware being used as well. For the ACSM, it is necessary to implement the control algorithms using an industrial controller. The primary sensor to be used is the LSS. The operation and capabilities of the LSS is described in Chapter 2. It can be assumed that *a-priori* information on the task may not

always be available except the starting location on orientation of a crack.

Much work has been done involving on-line control of robots using visual sensing. Most of this work involves 2-D cameras using image processing techniques to identify features or objects in the environment. The work involving vision sensing has little relevance to the crack following problem since the LSS will be mounted on the end-effector and is a relative proximity sensor. The LSS is not capable of identifying features with respect to a fixed reference frame.

Considerable work has also been done in the area of force sensing for on-line control. Endpoint force sensing is a well developed technology and has been applied to a wide range of tasks. Some tasks such as contour following with force sensors are conceptually very similar to the problem of crack following. While force sensing cannot be used for the task of crack following, many of the algorithms used for control with force sensing can be used for control with the LSS. On-line control using force sensing for an industrial robot is discussed in (De Schutter 1988) and (De Schutter 1990). Furthermore, a generalized approach for control using an arbitrary type of sensor is given in (Espiau 1990). This topic will be further developed in Chapter 4.

The problem of following cracks in the pavement is similar to seam tracking which is often used in automated welding. (Bamba 1984) describes an algorithm for seam tracking for arc welding using a sensor similar to the one used with the LSS. The main difference between tracking seams in automated welding and tracking cracks for automated crack sealing is that the welding process is done at a much slower speed than the crack sealing process.

CHAPTER 2 - ROBOTIC CRACK SEALING

This chapter will discuss the application of robotics to the ACSM. The job of the robot positioning system (RPS) is to guide all of the sealing equipment over cracks in the pavement. The RPS is a vital system since performance of the RPS is a limiting factor in overall performance of the ACSM. Additionally, the use of a robot manipulator on a vehicle on the highway represents an unusual application of robotics. Most robotics applications take place in factories which have relatively controlled and structured environments. The use of robotics in more revolutionary applications outside of the factory has often involved custom made manipulators and controllers. For the ACSM, off-the-shelf technologies will be used if possible. Therefore, development of the RPS represents an unusual and previously untested application of an industrial robot. The first section of this chapter will outline the requirements of the robot positioning system (RPS) for the ACSM. Section 2 will outline machine concepts for the RPS and describe how an appropriate manipulator was selected.

2.1 - REQUIREMENTS OF THE RPS

The requirements of the RPS general machine were set forth in SHRP proposal H-107 (Velinsky 1990). The purpose of the RPS is to physically connect the various components of the crack sealing machine. The RPS must move applicator assemblies and sensors along cracks in the pavement. The original specifications for the RPS sought a system that would be capable of sealing cracks along a full lane width (approximately 13 feet) at an average vehicle speed of 2 MPH. The specifications also called for a system capable of accommodating crack preparation methods including routing, heating and cleaning. Additionally, the machine was to be capable of maintaining the position of applicator assemblies in the presence of physical disturbances. The time required for converting the machine from "road travel" configuration to "crack sealing" configuration was also to be minimized. This is an important factor since the machine must reach a full

13 feet of lane-width while not adding to the overall maximum truck width of 8 feet. Finally, the specifications called for a machine composed of as many commercially available elements as possible.

2.2 - SELECTION OF A MANIPULATOR

Two different types of manipulators were strongly considered for the positioning system. The first type of system considered was a Cartesian coordinate based system also commonly referred to as a gantry robot. This type of system has several strong advantages including no singularities in the workspace, high payload capacity, high speed and dynamics that are independent of end-effector position. The main disadvantage of this type of system is the fact that it must be extend beyond a lane width while conducting the sealing operation. The positioning system must also be stowed to allow for down-the-road travel of the crack sealing truck.

The other type of system considered was a robotic arm system. The advantages of this system include the fact that the arm is self-supporting and can extend beyond the reach of the support vehicle. With a robotic arm system, the internal components of the applicators and peripherals will be more accessible due to the lack of an outer frame that would be present with a gantry system. It is also possible to operate two manipulators simultaneously in the same workspace with this type of system.

A robotic arm system was selected for the positioning system. It was decided to use this configuration as opposed to a gantry system in order to allow for simultaneous operations on the same crack and to avoid the difficulties involved with stowing a gantry system (Schulteiss and Velinsky, 1991). The design concept for the positioning system is shown in figure 2.1. It will use a pair of manipulators mounted on a linear slide. The slide will increase the workspace of the manipulator and will eliminate singularities near the edge of the workspace. *Process carts* will be used to support the weight of the applicator assemblies. The process carts will also give the applicator assemblies vertical compliance with the roadway. This simplifies the problem of controlling the manipulator

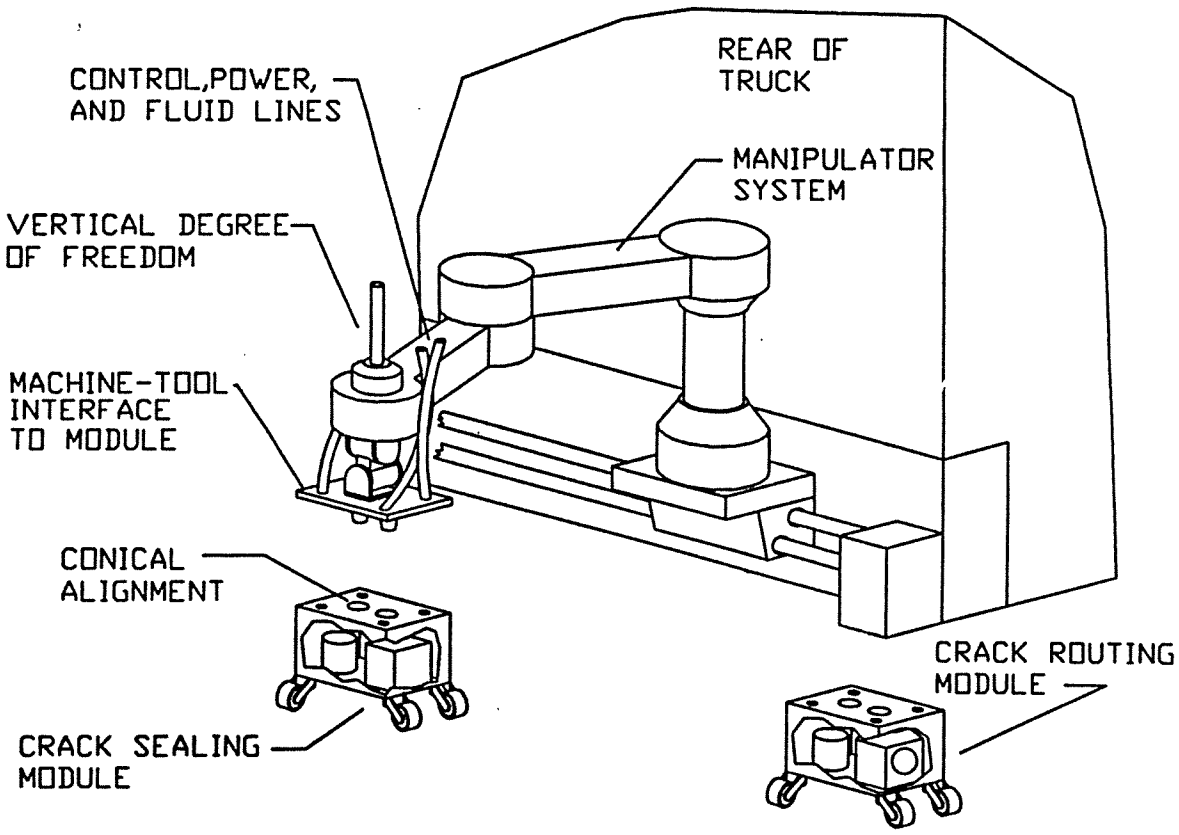


***Figure 2.1** Conceptual design for a robot arm positioning system (Velinsky, 1991)*

by reducing it to a two-dimensional problem. For the prototype ACSM, only one manipulator will be used.

The robot arm must be capable of controlling position and orientation in a plane parallel to the road. The arm must also have good dynamic load carrying capacity and a large workspace. Of all available robot configurations, SCARA configurations showed the greatest promise for the RPS.

2.2.1 - SELECTION OF A SCARA ROBOT AND LINEAR SLIDE

A large number of commercially available robots were considered for the RPS. Custom manipulators and controllers were eliminated due to time and cost constraints. Among the commercially available manipulators the most important criteria for selection

were: workspace, payload, controllability and cost. It was determined that a SCARA configuration that could be used in an inverted configuration would be preferred because a larger workspace would be available. SCARA configuration robots are especially well suited to two-dimensional problems. SCARA manipulators have fewer singularities in a two-dimensional problems than many manipulators that possess more degrees-of-freedom. Singularities pose a major problem for on-line control systems. The inverted configuration will allow the manipulator to reach underneath the slide which effectively doubles the size of the workspace. An analysis of the workspace of the selected manipulator is given in Appendix E.

A linear slide also had to be integrated with the manipulator. The linear slide increases the workspace of the robot and eliminates singularities near the edge of the workspace. It was desirable to find a linear slide system that could be integrated with the robot controller in order to simplify the problem of controlling a manipulator with a redundant degree-of-freedom.

A GMF-A510 manipulator mounted in an inverted configuration on a linear slide was selected for use by the RPS. The controller is a GMF A-510 RH series KAREL controller. The KAREL controller is capable of incorporating the slide position into Cartesian locations for forward kinematics calculations. Additional software was written to account for the redundant degree-of-freedom in the inverse kinematics. The algorithm for integrating the linear slide with the manipulator and the associated code written in KAREL are given in Appendix F. Specifications for the GMF A-510 manipulator, the KAREL RH series controller and the linear slide are given in Appendix H.

Preliminary tests of control algorithms for the RPS were done using an Adept-3 manipulator with an Adept A-series controller. The Adept-3 is a SCARA configuration manipulator with kinematics that are very similar to the A-510. The Adept controller and the GMF KAREL controller are also very similar in function.

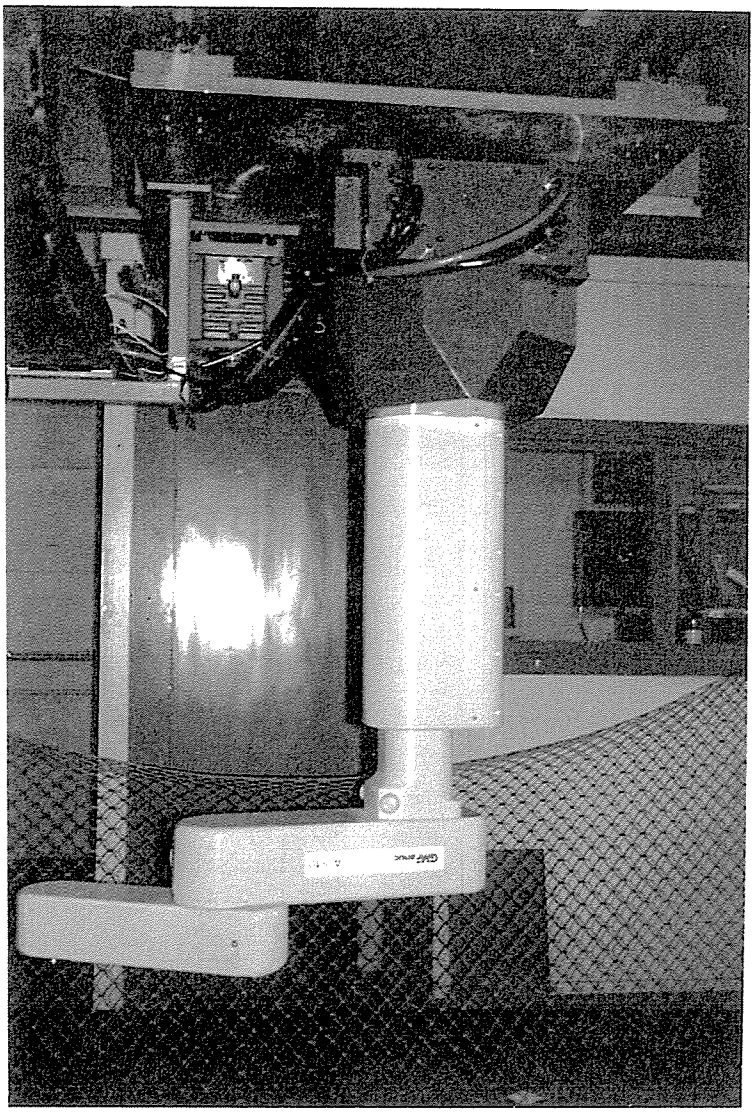


Figure 2.2 GMF-A510 manipulator and linear slide.

2.3 - CONCLUSIONS AND SUMMARY

This chapter has described the design concept for the RPS and how an appropriate manipulator and controller were selected. The manipulator to be used is a SCARA configuration manipulator which has been inverted and mounted on a linear slide. This configuration was chosen from a selection of commercially available manipulators for ease of control, payload and workspace. The manipulator has a redundant degree-of-freedom to eliminate some singularities and to increase the workspace. Additional code

had to be written by the user to account for the redundant degree-of-freedom in the inverse kinematics of the robot.

Sensors commonly used on robot end-effectors include force, vision, proximity and tactile. For this work we are primarily interested in proximity sensors. Proximity sensors return the distance from the sensor to an object in the environment. Proximity sensors can be either absolute or relative. An absolute proximity sensor returns the distance to an object at its nearest point. Ideally, an absolute proximity sensor will return the radius of a sphere centered at the sensor in which the sensed object must lie. A relative proximity sensor returns the distance to an object relative to the orientation of the sensor. The reading returned from a relative proximity sensor may not be the shortest distance to the sensed object. When using a relative proximity sensor it is often desirable to keep the sensor oriented such that it is normal to the object and hence will return the shortest distance to the object.

Proximity sensing is similar to force sensing for many tasks. Force sensors usually return contact forces and torques between the manipulator and the environment. If there is some compliance between the end-effector tooling and the object being manipulated, the relative distance between the end-effector and the object can be estimated. This can be done by dividing the force magnitude by the estimated stiffness between the manipulator and the object. In this way, a force sensor can be used to return proximity to an object. Therefore, some algorithms for control using force sensing can also be accomplished using proximity sensing.

A generalized approach for obtaining task definitions using endpoint sensors is given in (Espiau, 1992). The approach is valid for any type of sensor. The task definition consists of a primary sensor controlled task function and a secondary task function on the subspace of motions that will not affect the sensor readings. Once the task definition is obtained, it provides the error signal that the controller will try to bring to zero during the time of the task.

In addition to closed-loop control, proximity sensors can also be used in error recovery routines. Error recovery is usually implemented if the end-effector loses contact

with the object being manipulated before the task is complete. Error recovery may involve relocating the object in order to complete the task. Proximity sensors are often very useful in error recovery routines. Objects can be located by conducting a 'search' algorithm while monitoring the output of the sensor.

3.2 - OPERATION AND SPECIFICATIONS OF THE LSS

The Local Sensing System is a relative proximity sensor for sensing pavement cracks. The sensor projects structured laser light onto a surface and determines distances to points on the surface using a CCD camera and the principal of triangulation. The location of a crack is extracted from the crack profile. The sensor was selected for use on the ACSM and is manufactured by MVS Modular Vision Systems Inc., De Miniac,

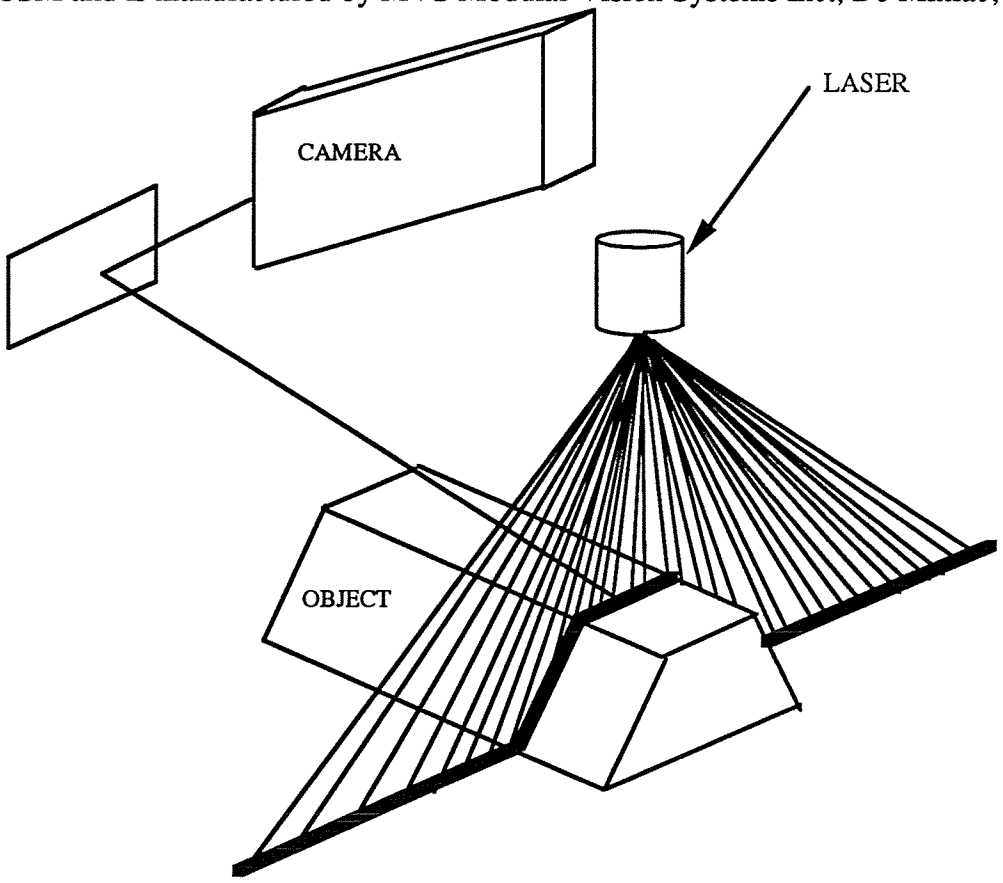


***Figure 3.1** Stuctured Light (Krulewich and Velinsky 1992).*

Montreal, Canada, Model # MVS-30. The sensor was specifically designed for use in robotics application and can operate in harsh environmental conditions. The sensor data is processed with an IBM-486 PC. For increased performance, image profile data is processed via a coprocessor board which plugs into a standard ISA-Bus slot.Each reading of depths along a scan line produces a surface profile (figure 3.2). A program running on the IBM 486-33 PC analyzes each profile to determine if there is a crack on the surface. Cracks are detected by observing gradients between consecutive pixels along the surface profile. If the height difference between two consecutive points is determined to be greater than the average surface roughness along the profile, the sensor program determines that the edge of a crack has been found. Once both edges of a crack have been found, the location of the center of the crack is determined (Krulewich and Velinsky 1992). This location will always be relative to the center of the sensor. *The relative distance to the center of the crack is the error signal that is returned to the RPS.*

The LSS has a an effective field of view of approximately 75 mm. The crack location program is capable of updating the RPS at 33 Hz. The resolution of the image data with a 75 mm field of view is less than 0.4 mm. (Krulewich and Velinsky 1992).

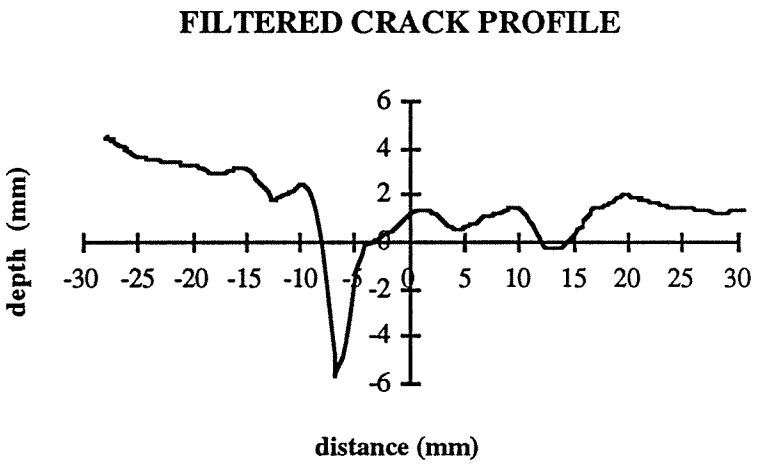


Figure 3.2 *Filtered crack profile. Krulewich and Velinsky (1992)*

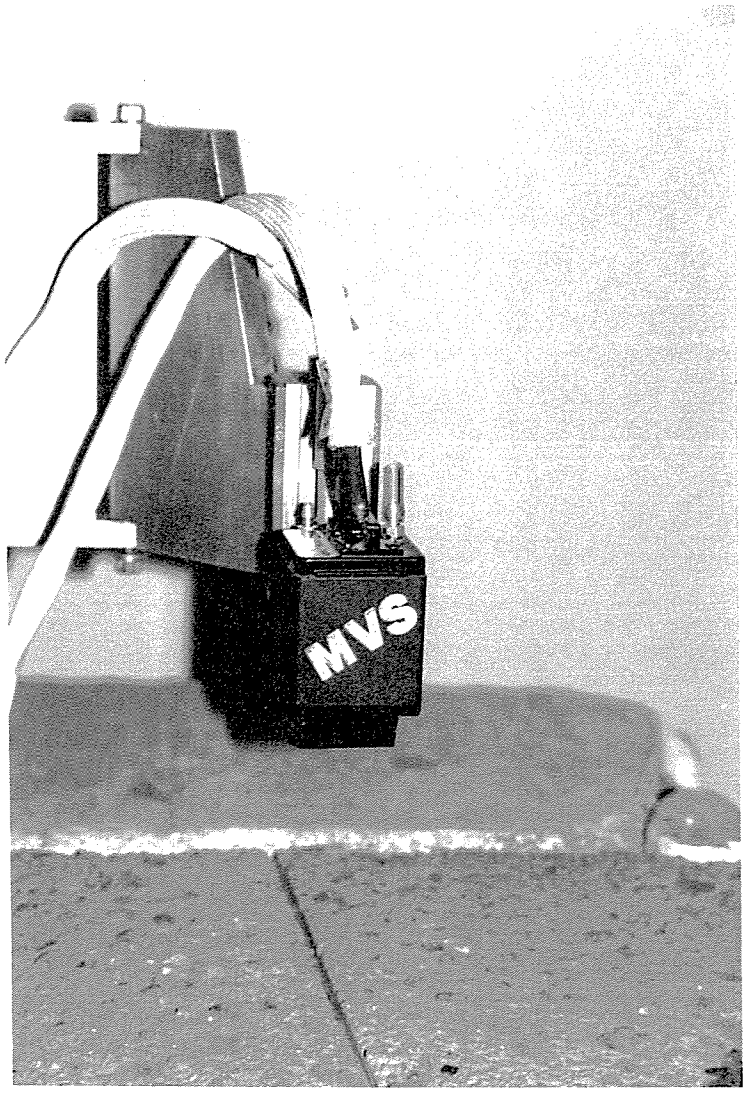


Figure 3.3 *The Local Sensor mounted on a test stand. The cables connect to the laser source, the power supply and the PC.*

3.3 - INTERFACING WITH THE LSS

Once the LSS has calculated the offset distance to the center of the crack, or has determined that there is no crack in its field of view, it must send this information to the robot controller. Since this data transfer is done in a closed control loop, it must be accomplished in a quick and efficient manner. Also, the error data generated by the LSS can be filtered and modified to improve the performance of the closed-loop system with the RPS.

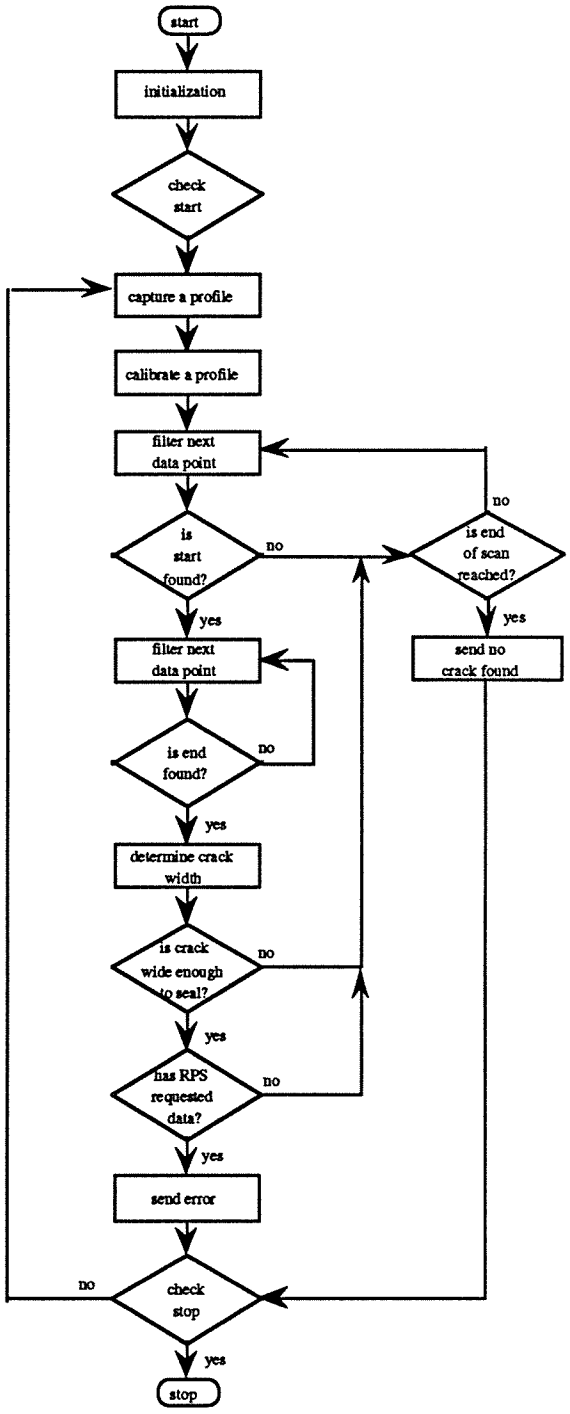


Figure 3.4 *Flowchart for Crack Locating Program (Krulewich and Velinsky, 1992).*

3.3.1 - DATA TRANSMISSION PROTOCOL

Since most robot controllers and the IBM 486-33 PC are equipped with RS-232 serial ports, a serial line was used to transfer data. In order to simplify communications, each error is scaled to be transmitted as a single byte. For example, the largest negative error the sensor can detect (-2.0 in) would be sent as -126 and the largest positive error (2.0 in) would be sent as 126. The baud rate of the serial communications is set at 4800 baud. The RPS will send a single byte to the LSS PC each time it is ready to receive a piece of data. For each piece of data sent from the LSS to the RPS, two bytes must be sent and received. The time to send 2 bytes at 4800 baud is less than 4 milliseconds. Therefore, the actual transmission of data over the serial line does not cause any significant delays in the system.

3.3.2 - LOW PASS FILTERING OF ERROR DATA

A flowchart of the LSS program is shown in figure 3.4. The sensor calculates the offset from each crack profile. Once the error has been calculated, its value is filtered via a second order low-pass Chebyshev filter. The purpose of filtering the error data from the LSS will be described in the following chapter. If the RPS has sent a bit requesting data during the previous cycle, the LSS will send the error to the serial port. Otherwise, another profile will be analyzed and a new offset error calculated.

3.3.3 - SIMULATING SENSOR SATURATION

If the sensor does not find a crack in its field of view, a 'no crack found' condition will be generated. In order to make the control scheme more robust, the LSS code was modified to simulate the condition of sensor saturation. When the sensor is moved such that the crack passes out of its field of view, the LSS program will set the error to be the maximum value that the sensor can return on the side on which the crack was last detected. This value will be run through the low-pass filter along with the other calculated errors. This process is designed to simulate saturation in an analog sensor and sends the robot controller more information than a simple 'no crack found' signal. The

addition of the saturation condition makes the control loop for crack following more robust and less sensitive to cases where the sensor may for some reason not detect a crack.

3.4 - SUMMARY AND CONCLUSIONS

This chapter has described the Local Sensing System (LSS) which will be used to detect cracks in the pavement and close a control loop around a robot controller. The LSS will help account for errors that occur when locating cracks using the VSS .The requirements of the LSS were presented. These requirements were determined by the nature of the crack sealing operation as well as environmental conditions. Many commercially available sensing technologies were examined and a laser scanning sensor using structured light was selected for the LSS. The sensor uses diffusely reflected light from a laser to determine distances along a scan line based on the principle of triangulation. Cracks are detected from analysis of surface profile data from the LSS. The offset distance between the center of the sensor and the center of a crack is sent to the RPS via a serial line. Sensor data is filtered via a low-pass filter and code to simulate sensor saturation has been added in order to improve performance for closed-loop control with the RPS.

CHAPTER 4 - MOTION CONTROL ALGORITHM

The purpose of this chapter is to describe the development of an algorithm for on-line position control of an industrial robot based on endpoint proximity sensing. This algorithm will be used to follow cracks in the pavement with a robot using input from the Local Sensing System (LSS). Section 2 of this chapter will discuss work that has already been done in similar areas and its relevance to the problem at hand. Section 3 will develop an algorithm for hybrid control using relative proximity sensing similar to algorithms for hybrid control using force sensing. Section 4 will describe a computer simulation of the control algorithm that is developed in section 3. Section 5 will describe how to implement the algorithm with an industrial controller. Implementation of a compliant motion algorithm using force control will be covered in section 6. Section 7 will describe the compliant motion algorithm using the LSS as a relative proximity sensor. Finally, section 8 will describe a control and communications architecture that will allow the compliant motion algorithm to flexibly interface with other control systems in order to provide a robust manipulator control system for the automated crack sealing machine.

4.1 PROBLEM STATEMENT

This chapter will address the problem of crack following using two separate algorithms. The first algorithm will assume that there is no a-priori information available about the path to be followed. This algorithm will use feedback from the LSS to control both the position and orientation of the end-effector. This algorithm will only require the approximate starting location and direction of the crack to be followed in order to begin closed-loop control.

4.1.1 PROBLEM STATEMENT FOR FEEDBACK CONTROL FOR POSITION AND ORIENTATION

The problem to be addressed is to follow a planar path using the LSS mounted on a

manipulator end-effector. No *a-priori* information is given about the path except its starting position and orientation. The control problem can be broken down into three parts:

1) Keeping the end-effector centered over the sensed path.
2) Maintaining a specified end-effector velocity tangential to the path.
3) Maintaining end-effector alignment in a direction tangential to the sensed path

4.1.2 PROBLEM STATEMENT FOR USING CLOSED-LOOP CONTROL FOR POSITION AND OPEN-LOOP CONTROL FOR ORIENTATION

If crack direction information is available from the VSS, the problem to be addressed will be to use the LSS to keep the end-effector centered over the crack and move the end-effector tangential to the crack. The crack direction will be determined by data from the VSS. The problem statement will then consist of the first two parts of the problem statement given in section 4.1.1.

Design requirements for the ACSM dictate that the above algorithm be implemented in real-time using an industrial controller and the LSS as off-the-shelf components. Time and cost limitations on the ACSM project eliminate the building of custom manipulators and controllers. Additionally, an algorithm that can be implemented on an industrial controller is much more useful because it can be easily ported to many other robots presently in production. An algorithm that would work on an industrial controller will simplify commercialization of the ACSM.

4.2 - PREVIOUS WORK

Much work has been done in the area of compliant motion using force sensing. Such control schemes are often referred to as "hybrid control", "stiffness control" "compliance control" or "force impedance control". The objective of these control schemes is to regulate the contact forces between a manipulator and the external environment. They combine force sensor and position information to control a manipulator's motion. Although compliant motion is usually done using force feedback,

for specifying compliant motion tasks. The formalism is based on the hybrid control functional specification described by Mason (1981). The second part of the paper develops an algorithm for compliant motion using external control loops closed around a robot positioning system. The algorithm developed is similar to the work of Salisbury except that the use of external control loops makes implementation possible with most industrial controller. This paper refers to compliant motion with force sensing and does not directly address compliant motion with other types of sensing such as proximity sensing. The work of this paper is a primary foundation of work done in this report.

Bamba et. al. (1984) "A Visual Seam Tracking System for Arc-Welding Robots"

This paper addresses the problem of tracking a seam for arc welding using an optical laser sensor mounted on the end-effector of a robot. The sensor uses a laser diode and a PIN diode linear sensor chip to detect seams based on the principal of triangulation. The sensor returns the position of the seam in terms of a radius and an angle in its circular scan of the laser. The measurement is made relative to the sensor and hence the end-effector frame of the robot. The circular scan area of the sensor makes it possible for the sensor to observe the seam ahead of the end-effector. Also the end-effector speed for arc welding is relatively slow.

The problem addressed in this paper is similar to the crack following problem with the exceptions that the LSS uses a linear scan and cannot detect the crack ahead of the end-effector and the end-effector speed for crack following is significantly higher.

Espiau, Merlet and Samson (1990) "Force Feedback Control and Non-Contact Sensing: A Unified Approach"

This paper proposes a global approach to the problem of proximity and force-based control applications in robotics. The concept of and interaction screw is introduced which models how a sensor interacts with the environment. The concept of the

for specifying compliant motion tasks. The formalism is based on the hybrid control functional specification described by Mason (1981). The second part of the paper develops an algorithm for compliant motion using external control loops closed around a robot positioning system. The algorithm developed is similar to the work of Salisbury except that the use of external control loops makes implementation possible with most industrial controller. This paper refers to compliant motion with force sensing and does not directly address compliant motion with other types of sensing such as proximity sensing. The work of this paper is a primary foundation of work done in this thesis.

Bamba et. al. (1984) "A Visual Seam Tracking System for Arc-Welding Robots"

This paper addresses the problem of tracking a seam for arc welding using an optical laser sensor mounted on the end-effector of a robot. The sensor uses a laser diode and a PIN diode linear sensor chip to detect seams based on the principal of triangulation. The sensor returns the position of the seam in terms of a radius and an angle in its circular scan of the laser. The measurement is made relative to the sensor and hence the end-effector frame of the robot. The circular scan area of the sensor makes it possible for the sensor to observe the seam ahead of the end-effector. Also the end-effector speed for arc welding is relatively slow.

The problem addressed in this paper is similar to the crack following problem with the exceptions that the LSS uses a linear scan and cannot detect the crack ahead of the end-effector and the end-effector speed for crack following is significantly higher.

Espiau, Merlet and Samson (1990) "Force Feedback Control and Non-Contact Sensing: A Unified Approach"

This paper proposes a global approach to the problem of proximity and force-based control applications in robotics. The concept of and interaction screw is introduced which models how a sensor interacts with the environment. The concept of the

interaction screw allows force and proximity sensing problems to be treated in the same manner. The paper goes on to discuss the development of control laws for compliant motion using the algorithms and models that have been introduced.

4.2.2 - DESCRIPTION OF HYBRID CONTROL

One of the most common applications for a stiffness control algorithm is a manipulator trying to fit a peg into a close fitting hole. Without force control, the position of the hole must be known exactly for the insertion. If there is an error in the insertion angle, the peg may start to bind. Without force feedback, the manipulator will try to force the peg into the hole resulting in a jammed or broken peg. Using force feedback, the manipulator can adjust the position of the peg to reduce the binding force.

Another application of stiffness control involves following a contoured surface with an end-effector while maintaining a constant contact force by using feedback from an endpoint force sensor. Figure 4.1 shows a task configuration for the two-dimensional contour tracking problem. The surface is a unknown and arbitrary but it is continuous. The object frame is defined at the contact point. One axis is tangential to the surface at the contact point and the other axis is orthogonal to the contact surface. The task frame defines which directions are used for force control and which directions are used for position or velocity control. The contour tracking problem is similar to the crack-following in many ways. Further development of this problem and its similarity to crack following with the LSS will be covered in subsequent sections.

For the problem at hand, there is no actual contact between the manipulator and the environment. However, by reading the output of the LSS, the controller can determine the offset between a crack in a surface and the end-effector. This reading or offset error is similar to the force reading returned by an endpoint force sensor. Therefore, even though there is no actual contact with the environment , stiffness control algorithms with some modifications can be used to control a manipulator using feedback from the LSS mounted on the end-effector.

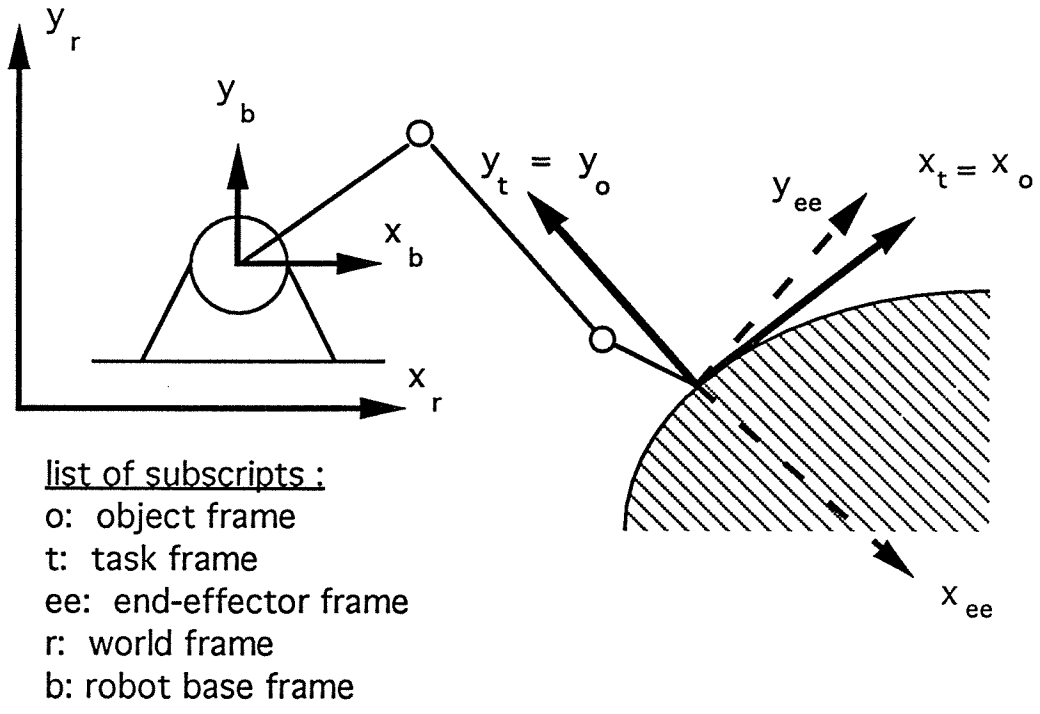


Figure 4.1 *Coordinate Frames for Compliant Motion.*

4.2.3 - NESTED CONTROL LOOPS USING AN INDUSTRIAL CONTROLLER

One problem of implementing a force control scheme with a manipulator is that it the complex and non-linear dynamics of the manipulator have to be taken into account. However, an industrial controller can handle the complex manipulator dynamics and implement position control. It is possible to close a control loop around the position controller and implement force or hybrid control. In this case, the problem is broken down into two more manageable sub problems. The industrial position controller handles the manipulator dynamics while the secondary control loop closed around the controller handles interaction with the environment. (DeSchutter 1990). Figure 4.2 shows a block diagram of the system using an external control loop closed around a position controller

to implement force control. This representation assumes that sensor dynamics are negligible and that the contact forces present are not large enough to effect the dynamics of the position controller. The contact force is measured and is multiplied by the stiffness gain K_0 before being fed back to the position controller. K_0 is the estimated contact stiffness between the end-effector and the environment at the contact point. An external control loop using the LSS does not need to use the stiffness gain K_0 since the LSS measures the actual displacement distance and not contact force. Therefore, the use of K_0 to convert a measured contact force to an estimated distance is one difference between using a force sensor and the LSS when using the control scheme shown in figure 4.2.

In order to use the task frame described above with external control loops, we have to make one important assumption. We have to assume that the dynamics of the manipulator and position controller system which form the inner control loop are independent of the position of the end-effector and the direction of motion. This assumption is true for most industrial robots with decentralized controllers as long as the bandwidths and damping ratio of all independent joint control systems are nearly identical. (DeSchutter 1990).

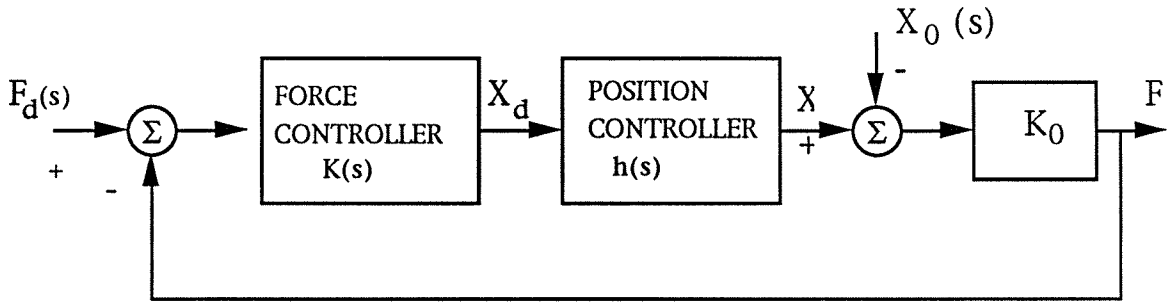


Figure 4.2 *Force control loop closed around a position controller.*

4.2.4 DEFINING THE TASK FRAME FOR FORCE CONTROL

In order to implement hybrid control, we must define a *task frame* which will determine how the force and position control laws are applied. By properly defining the task frame, we can use separate, decoupled control loops to handle force control and position control. By using separate, decoupled control loops, we can avoid the

complexities of control. DeSchutter(1988) defines a task frame for hybrid control which defines orthogonal force and position control directions for the task frame. Figure 4.3 shows the task frame for the case of the contour tracking problem using force control. By using the task frame shown in figure 4.3, we can use the one-dimensional force control scheme in section 4.2.3 to control the contact forces in the force control direction and a separate control loop to control position or velocity in the position control direction.

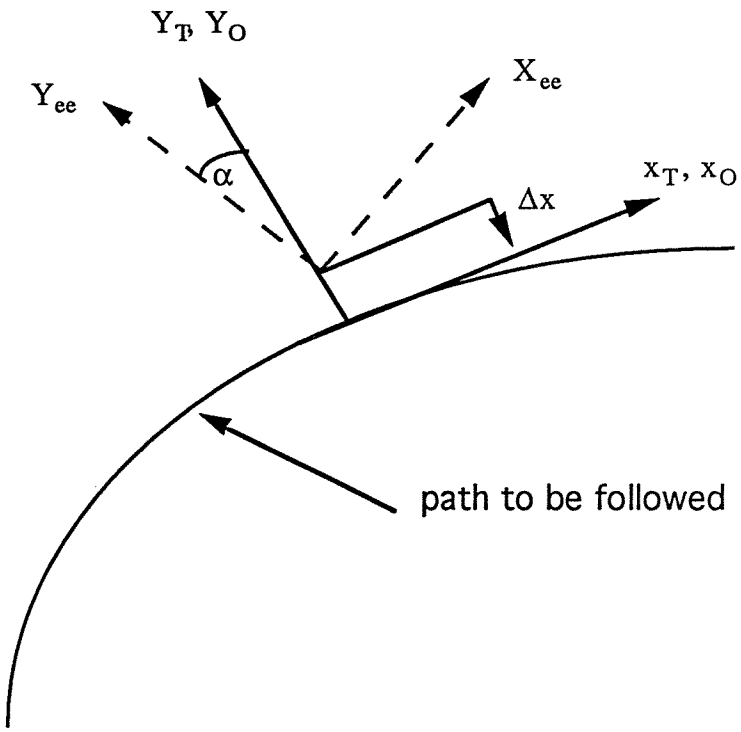


Figure 4.3 *The object frame is represented by X_0 and Y_0 and coincides with the task frame X_t and Y_t. Y_t will be referred to as the normal position control direction and X_t represents the tangential position direction. Δx represents a normal position error. This error can be detected by a proximity sensor using distance or a force sensor with a flexible tool attached. The orientation error is represented by α.*

4.2.5 ESTIMATING THE TASK FRAME FOR FORCE SENSING

The task frame shown in figure 4.3 changes as the end-effector moves along the contour. Therefore, the task frame is constantly changing with respect to the world

coordinate system. In order to implement on-line hybrid control, it is necessary to repeatedly calculate the position and orientation of the task frame as the end-effector moves along the contour. This task is relatively easy to accomplish if a six degree-of-freedom force sensor is used. If we assume negligible friction between the end-effector and the object, we can take the arc tangent of the forces measured along the axes of the world coordinate system and determine a normal to the surface. This normal vector can then be used to determine the orientation of the task frame or the tracking direction.

It is now possible to control the orientation of the end-effector with respect to the task frame at any time. This can be done by estimating the tracking direction and calculating the error between the tracking direction and the actual end-effector orientation. By applying a control law to this error and feeding it back to the position controller, we can create another control loop that will control the orientation of the end-effector with respect to the task frame. Therefore, it is possible to control contact forces, velocity and orientation of an end-effector following a contoured surface by using three separate and decoupled control loops closed around a position controller using a force sensor.

4.3 CONTROL WITH RELATIVE PROXIMITY SENSING

Now that we have reviewed previous work in control with force sensing and proximity sensing, we will develop an algorithm for compliant motion using relative proximity sensing. This section will develop two separate control algorithms. The first control algorithm (sections 4.3.1-4.3.3) will control both the position and orientation of the end-effector using feedback from the LSS. The second control algorithm (sections 4.3.4-4.3.6) will use open-loop control for orientation of the end-effector using data from the VSS and feedback control from the LSS to control position of the end-effector.

4.3.1 DEFINING THE TASK FRAME FOR RELATIVE PROXIMITY SENSING USING FRENET SERRET VECTORS

Estimating the object frame for contour following with endpoint force sensing is not

a difficult process. The vector normal to the surface (the force control direction) can be measured directly by taking the inverse tangents of the forces measured along the axes of the world coordinate system. The velocity-controlled direction can then be set normal to the force-controlled direction in the desired direction of travel. With relative proximity sensing, the direction of the surface normal cannot be measured directly. Therefore, it is necessary to estimate the object frame when using relative proximity sensing. In order to estimate the object frame for proximity sensing, we must first formally define the frame.

We will use the principles of local curve theory to help define the task frame for relative proximity sensing. Consider a parametric curve in 3-D space. The curve can be expressed either as a function of time $\alpha(t)$ or as a function of arc length $\alpha(s)$. At each point along $\alpha(s)$ there exists a *tangent vector* $\mathbf{T}(s)$ which is defined as $\mathbf{T}(s)=d\alpha/ds=\alpha'$. Hence if α lies in a plane, we can express $\alpha(s)$ in the form $\alpha(s)=(x(s),y(s),0)$ and the *tangent vector field* can be expressed as $\mathbf{T}(s)=(x'(s).y'(s),0)$. We can also define the *principal normal vector field* $\mathbf{N}(s)$ such that $\mathbf{N}(s)=\mathbf{T}'(s)/\kappa(s)$ where $\kappa(s)=|\mathbf{T}'(s)|$. Finally, we can define the binormal vector field to $\alpha(s)$ as $\mathbf{B}(s)=\mathbf{T}(s)\mathrm{X}\mathbf{N}(s)$. These definitions for the *tangent vector field*, the *principal normal vector field* and the binormal vector field are from the *Frenet Serret apparatus* for a spatial curve.

The vector fields of the Frenet Serret apparatus can be used as a basis for the definition of the object frame for hybrid control using relative proximity sensing. For our purposes, it is necessary to redefine the principal normal and binormal vector fields. The principal normal vector field is always defined such that it always points towards the center of curvature of α. For the object frame in hybrid control it is desirable to define the normal vector such that its direction is constant relative to the direction of travel of the end-effector. Therefore, we will define the path to be followed as α and the tangential vector field as $\mathbf{T} = \alpha'$ as in the Frenet Serret apparatus. The path will be assumed to lie in a plane such that $\alpha = x\,\hat{\mathbf{i}} + y\,\hat{\mathbf{j}}$ where and are the unit vectors of the axes

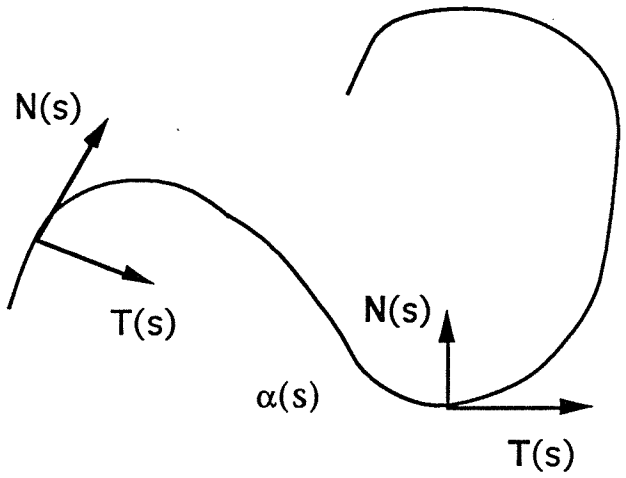


Figure 4.4 *A spatial curve* α(s) *with the normal vectors* **N**(s) *and tangent vectors* **T**(s) *shown.*

of the world coordinate system. We will now define the binormal vector to be constant such that $\mathbf{B}' = \hat{\mathbf{k}}$ and the principal normal vector will be defined as $\mathbf{N}' = \mathbf{B}'\mathrm{X}\mathbf{T}$. With these definitions, the direction of the normal vector will be only dependent on the direction of the tangent vector. The task frame will now be defined as follows. Let the path to be followed lie on a plane and be defined as α. Let $\hat{\mathbf{t}}$ and $\hat{\mathbf{n}}$ be the unit vectors of **T** and **N'** respectively. The task frame will then consist of the axes defined by $\hat{\mathbf{t}}$ and $\hat{\mathbf{n}}$ where $\hat{\mathbf{t}}$ denotes the tangential direction and $\hat{\mathbf{n}}$ denotes the normal direction

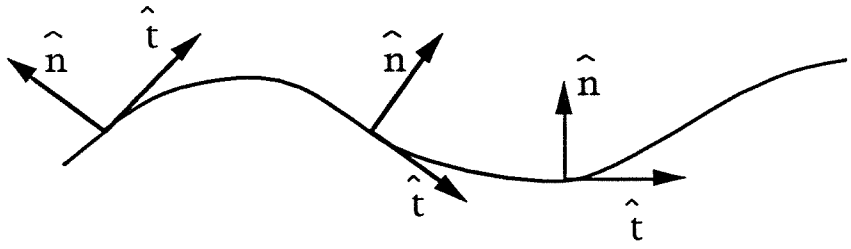


Figure 4.5 *Normal and tangential unit vectors for defining the task frame for hybrid control with relative proximity sensing.*

4.3.2 ESTIMATING THE TASK FRAME FOR RELATIVE PROXIMITY SENSING

So far, this paper has discussed hybrid control using a force sensor and the LSS in the same manner. However, there are important differences between these different types of sensing. Force sensing provides both a force magnitude and the direction of the force. Figure 4.4 illustrates sensing using the LSS. The displacement error (Δx) measured by the sensor is dependent upon α, which is the error between the end-effector orientation and the tracking direction. Therefore, the LSS only returns accurate information when the end-effector is aligned with the tracking direction. Since the LSS is not capable of detecting the direction of the path as a force sensor is, the path direction must be estimated by looking at previous end-effector locations and the associated LSS readings. Since it is impossible to estimate the tracking direction with complete accuracy and the accuracy of the displacement reading is dependent on aligning the end-effector with the tracking direction, a coupling effect exists between the orientation control loop and the normal direction control loop. The cause of the coupling affect is shown in Figure 4.6. Error in estimating the task frame causes motion in the tangential direction which affects offset in the normal direction. This error can cause overshoot and oscillations in end-effector motion which in turn make it more difficult to estimate crack direction. This coupling effect is due to the nature of the sensing. Both the normal direction loop and the orientation control loop must be heavily damped when using the LSS, since instability in one loop will tend to propagate to the other loop.

In order to use the task frames described above with external control loops, we have to make one important assumption. We have to assume that the dynamics of the manipulator and position controller system which form the inner control loop are independent of the position of the end-effector and the direction of motion. This assumption is true for most industrial robots with decentralized controllers as long as the bandwidths and damping ratio of all independent joint control systems are nearly identical. (DeSchutter 1990).

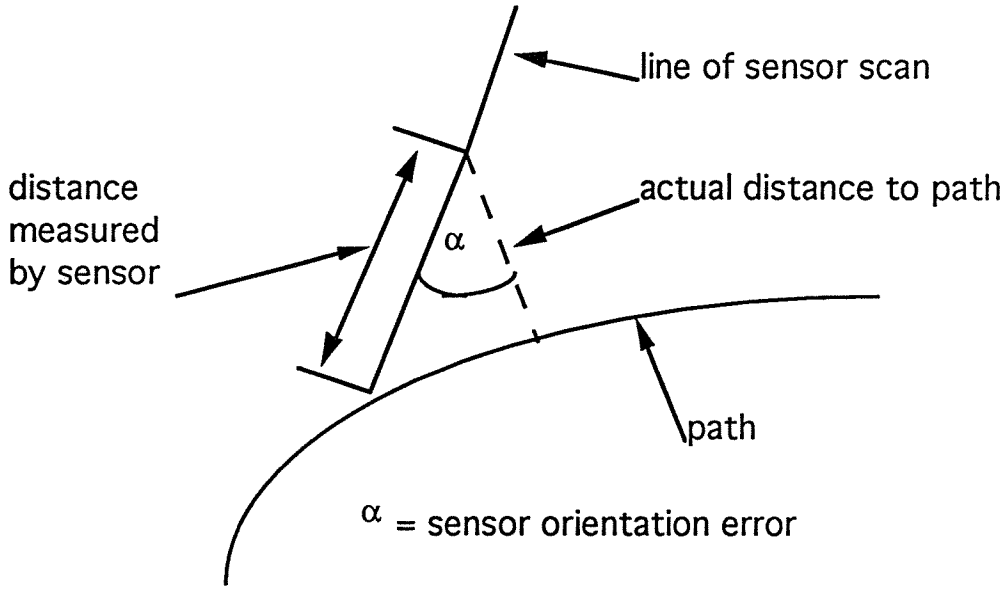


Figure 4.6 *Distance error caused by an orientation error with a relative proximity sensor.*

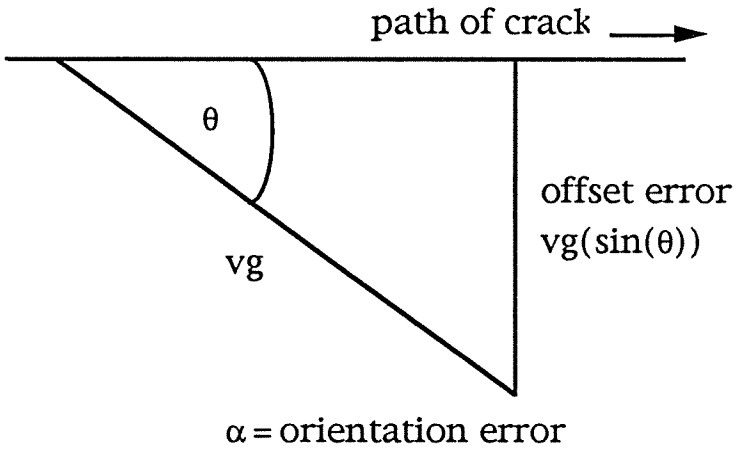


Figure 4.7 *Offset error caused by motion in the tangential direction when an orientation error is present.*

4.3.3 OBTAINING CRACK DIRECTION USING GLOBAL VISION DATA

This section will discuss how data path planning is done for the RPS based on data from the VSS. The subsystems involved include the Vision Sensing System (VSS), the Vehicle Orientation an Control System (VOC) and the Path Planning module. Processing of vision data and the development of the path planning module is not a part of the work

Vehicle Orientation an Control System (VOC) and the Path Planning module. Processing of vision data and the development of the path planning module is not a part of the work of this report and is included for completeness. Details on processing of vision data can be found in Kirschke and Velinsky (1992) and the path planning module is discussed in Lasky and Ravani (1993).

The VSS acquires images of the road using a line scan camera. Scans are taken as the vehicle moves forward. Each scan is 1/16 inch deep and 12 feet wide. The data from the scans are buffered up to build an image consisting of 2"X2" tiles, each of which consists of a 32X32 grid of 1/16" pixels. The VSS software then builds a histogram of gray shades within each tile and computes a statistical moment for each histogram. The tiles are then compared within independent 5X5 tile areas. The comparison algorithm looks for tiles which have a greater amount of contrast relative to neighboring tiles. An example of a comparison grid is shown in Figure 4.7. In this case the algorithm is checking for a crack running in the Northeast direction. A crack will be said to exist if all of the moments of the tiles labeled 'D' and the center tile are greater than the moments of all of the tiles labeled 'C'. Thus, the algorithm will determine if each tile contains a crack. If the tile is determined to contain a crack it will also have an associated direction number from the 5X5 comparison grid. The direction numbers will be in increments of 22.5°.

C	C	C		D
C	C		D	
C		O		C
	D		C	C
D		C	C	C

Figure 4.8 *VSS Comparison* (C) *and Direction* (D) *tiles for a Northeast running crack.*

The job of the path planning module is to convert the output of the vision software into useful paths for the robot manipulator. The raw vision data is insufficient, as it is just an array of potential crack locations, without any sense of relationship between tiles. The path planning algorithm will process this data by filtering out noise, filling in blank

The path planning algorithm operates with a set of points P corresponding to the centers of tiles where a crack exists. Each point P_k also has an associated direction number. The algorithm begins filtering the data by removing isolated image points. The algorithm then looks for connections between isolated segments of points. It does this by finding the endpoints of each section and then 'growing' the endpoints to connect segments that are close together. The 'grown' data is then thinned to unit tile thickness using a simple heuristic algorithm (Lasky and Ravani, 1993).

The algorithm then defines a set of 'visited' points based on the workspace of the manipulator. The set of 'visited' points will consist of consecutive points defining a single path from one end of the manipulator workspace to the other.

Once the set of visited points has been established, a continuous path is created by splining the discrete points together with third order Bézier curves. At each path point, an approximate tangent is known based on the direction number set by the vision system or created during the grow/thin process. This information, along with the order in which the points arranged, can be used to compute the unit tangent vector $\vec{t}_k$ at each point. With this information, the Bézier control points $g_k = [g_{k_0} g_{k_1} g_{k_2} g_{k_3}]$ for each path segment k can be computed as follows:

$$g_{k_0} = P_k \tag{4-1a}$$

$$g_{k_1} = P_k + \vec{t}_k \tag{4-1b}$$

$$g_{k_2} = P_{k+1} - \vec{t}_{k+1} \tag{4-1c}$$

$$g_{k_3} = P_{k+1} \tag{4-1d}$$

Now that Bézier curves have been defined between consecutive points, the entire path can be defined as a collection of intervals which constitute a Bézier spline curve. In order to obtain crack direction information for the robot manipulator, it is necessary to find tangents to the spline curves as a function of the world coordinate 'x'. For the crack sealing operation, we can assume that the curve is monotonic in 'x' and therefore can be expressed as a function of 'x'.

Let the spline curve be denoted as α(u) where u is the parameter for the entire curve. We will now define the local parameter t for each interval as follows:

$$t = \frac{u - u_k}{u_{k+1} - u_k} = \frac{u - u_k}{\Delta u_k} \tag{4-2}$$

Note that t =(0,1) for each interval (u_k, u_{k+1}).

The derivative at any point in the interval is as follows:

$$\frac{d\alpha}{du} = \frac{1}{\Delta u_k}\frac{d\alpha_k(t)}{dt} \tag{4-3}$$

The derivative $\frac{d\alpha_k(t)}{dt}$ can be calculated from the control points and the Bernstein polynomial for each segment as follows:

$$\frac{d}{dt}g^n(t) = n\sum_{j=0}^{n-1}\Delta g_{k_j}B_j^{n-1}(t) \tag{4-4}$$

where:

$$\Delta g_{k_j} = g_{k_{j+1}} - g_{k_j} \tag{4-5}$$

and g_{k_j} denotes the j[th] control point in the interval k. The polynomial $B_i^n(t)$ is defined explicitly by:

$$B_i^n(t) = \binom{n}{i}t^i(1-t)^{n-i} \tag{4-6}$$

This derivative will be in terms of t rather than 'x'. We can solve for t at a desired x and use the result to obtain a derivative in terms of 'x'; however, this process is somewhat tedious. In this case, finding t will involve solving a cubic equation and then taking the solution that lies within the interval (0,1).

The above method will return the exact tangent vector to the spline curve at any given point; however, for our purposes an approximate tangent vector as a function of 'x' will be sufficient. Therefore, a simpler method can be devised to extract the approximate tangent vector without the need to solve a cubic equation for t.

Consider a single interval on the spline curve. We have already defined the local

parameter t that varies for 0 to 1 as u varies from u_k to u_{k+1}. We will now define local parameters for arc length and chord length. The local arc length s will be defined as the length of the curve between P_k and α(t):

$$s = \int_0^t \left| \frac{d\alpha}{dt} \right| dt \tag{4-7}$$

The total arc length between points P_k and P_{k+1} will be defined as s_k where:

$$s_k = \int_0^1 \left| \frac{d\alpha}{dt} \right| dt \tag{4-8}$$

Finally, we will define c as the local chord length along the interval which will be the distance along a chord drawn between P_k and P_{k+1} The total chord length will be denoted as c_k where c_k is the magnitude of a vector between P_k andP_{k+1} or:

$$c_k = \left| \bar{P}_{k+1} - \bar{P}_k \right| \tag{4-9}$$

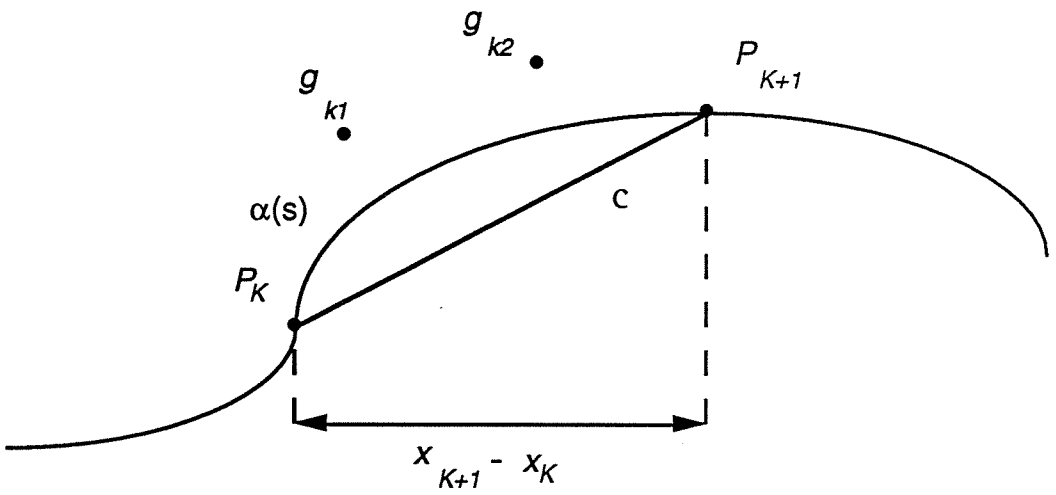


Figure 4.9 *Beziér curve between points P_k and P_{k+1}. The chord for the interval is denoted by 'c'.*

We will now make an approximation and map the arc length along the interval into the chord length along the interval such that:

$$\frac{s}{s_k} \approx \frac{c}{c_k} \tag{4-10}$$

For the crack sealing operation, we will assume that the curve $\alpha(s)$ can be expressed as a function of the Cartesian variable 'x'. and, since the endpoints of each interval are known, it is easy to calculate c for a given value of 'x'. on any interval. Now we have a function that will map the Cartesian variable 'x'. to the local arc length s.

The next step is to reparametrize the curve from $\alpha(t)$ to $\alpha(s)$. We can now express the derivatives of the curve as:

$$\frac{d\alpha}{dt} = \frac{ds}{dt}\frac{d\alpha}{ds} \tag{4-11}$$

we will assume that the speed along the curve $\frac{ds}{dt}$ is constant.

Therefore, the derivatives of the curve with respect to t and s are proportional:

$$\frac{d\alpha}{dt} = K\frac{d\alpha}{ds} \tag{4-12}$$

If we recall the definition of arc length, this relationship implies that we can map t at each interval into arc length as follows:

$$t = \frac{s}{s_k} \tag{4-13}$$

We now use the mapping approximation between c and s:

$$\frac{s}{s_k} \approx \frac{c}{c_k} \tag{4-14}$$

c is mapped to the Cartesian value x. as follows:

$$\frac{c}{c_k} = \frac{x - x_k}{x_{k+1} - x_k} \tag{4-15}$$

This yields an approximation for t along each interval as:

$$t \approx \frac{x - x_k}{x_{k+1} - x_k} \tag{4-16}$$

We can now find the derivative in any interval by using the approximation for t and calculating the derivative with equation (4-4). The direction of the derivative vector will be the tangent to the path at that point. This will allow us to calculates tangent to the curve as a function of x.

The path planning module will identify a crack in the workspace of the manipulator and send the starting location and direction of the crack along with an array containing

values of x. in the Cartesian frame along with the associated crack direction values. The controller will then use the crack direction values to define the task frame and orient the end-effector. The position of the end-effector will be controlled by trying to zero the offset error from the LSS while moving tangential to the crack directions received from path planning.

This control algorithm will be useful if the VSS can accurately determine the shape of a crack, but the exact location of the crack is not known due to errors in the VSS or the VOC. Knowing the crack direction a-priori eliminates the difficulties of trying to estimate crack direction from previous crack locations from the LSS.

4.3.4 CONTROL ALGORITHM

Recall the block diagram for a force control loop closed around an industrial position controller shown in Figure 4.2. A similar diagram for closing a control loop using the LSS is given in Figure 4.10. The controller manipulator system is shown with its separate parts consisting of the inverse kinematics module, the joint controller and the manipulator. Together they take a desired Cartesian input $\underline{x}_D$ and the end-effector moves to the desired position producing the output of the actual end-effector position $\underline{x}_E$. The end-effector position and the local sensor output can be combined to produce the crack location with respect to the task frame. The end-effector frame is used as the task frame since the object frame is not exactly known. The crack location in the task frame is multiplied by the gain vector to generate a new location in task space. The new location is then converted to a location with respect to the world coordinate system and sent to the position controller. The position controller calculates the desired joint angles and servos the joint motors to obtain the next end-effector location.

The block diagram of Figure 4.10 describes the control algorithm using feedback control for both position and orientation. The feedback error is a three element vector of the Cartesian location and an orientation. This same block diagram can be used for the algorithm with open-loop control for orientation if $\underline{d}_T$ becomes a two element vector of

Cartesian position only and the orientation from the VSS is fed directly to the position controller to become part of the desired Cartesian location $\underline{X}_D$.

4.4 COMPUTER SIMULATION OF HYBRID CONTROL

Pending delivery of a robot for use for the ACSM, it was decided to run a computer simulation of the hybrid control algorithm. The purpose of the simulation was twofold: (1) verify the validity of the algorithm for proximity sensing since no documentation had been found on the implementation of hybrid control using proximity sensing and (2) to gain insight into tuning the control laws of the external control loops in a safe simulation environment before implementation with hardware.

4.4.1 MODELING

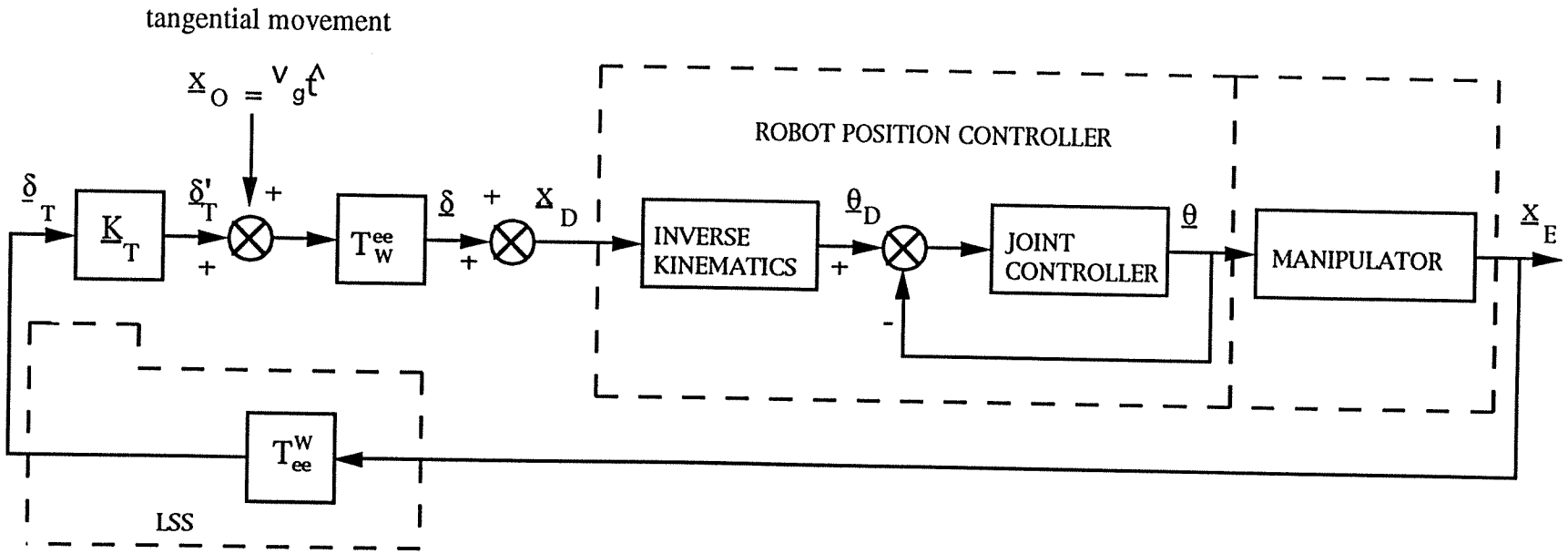
In order to implement a numerical simulation, it is necessary to model the plant, the sensor and the control laws being used. In this case the plant will be the position controller around which the external control loops will be closed. The simulation will assume a simple plant model and use a robust control law to account for uncertainties in the model. The plant model used was a second order system which was approximated from the dynamics of a single joint controller for the Stanford-JPL arm. Approximating plant dynamics from the dynamics of a single joint was suggested by (DeSchutter 1990). In this case, we used the joint with the largest natural frequency. Like most industrial controllers, this system is heavily damped to minimize overshoot. We will assume that the plant model for a position input will be independent of the position of the end-effector or the direction of the motion. For the simulation, sensor dynamics will be neglected although the type of sensing used will be discussed in following sections. Specifics of the modeling for the plant in the simulation can be found in Appendix A.

4.4.2 SENSING

The simulation was designed to simulate hybrid control using three different types of sensing as described below:

FORCE SENSING

Figure 4.10 Block Diagram



This will simulate the a sensor that can accurately determine the distance between the sensor and the desired path and the direction of the desired path at any time.

PROXIMITY SENSING

The simulation will be run with a sensor that can detect the shortest distance between the end-effector and the desired path but not the direction of the path.

RELATIVE PROXIMITY SENSING

The simulation will be run with a sensor that emulates the LSS. The sensor will return the distance between the end-effector and the desired path relative to the current orientation of the end-effector. The sensor cannot directly measure path direction.

4.4.3 HYBRID CONTROL ALGORITHM

PROBLEM STATEMENT

The problem to be addressed by the simulation is the same as in the problem statement of section 4.1 which is to follow a path on a two-dimensional surface using one of the types of endpoint sensing described. No a-priori information is given about the path except its starting position and orientation. Based on the task frames defined in the previous section, the control problem can be broken down into three separate parts:

1) Keeping the end-effector centered over the sensed path.
2) Maintaining a specified end-effector velocity tangential to the path
3) Maintaining end-effector alignment parallel to the tracking direction.

Part 1 of the problem will involve using a control loop in the normal position direction of the task frame using feedback from the sensor. Part 3 will measure the path direction and adjust the end-effector orientation appropriately. Part 2 will not use a feedback loop because the desired velocity is known and can be specified through the position controller.

A flowchart of the hybrid control algorithm is given in figure 4.11. The control algorithm will use velocity vector inputs to the position controller that will be calculated from sensor readings and the desired tangential velocity. Initially, the controller will be

given approximate information on the starting location and orientation of the path. The manipulator will then move tangential to the path and take a sensor reading to obtain its displacement from the center of the path. The sensor input will be run through a digital PID control law to obtain a gain for the velocity to be specified in the normal direction. The manipulator will then move the end-effector normal to the path with the specified velocity. The end-effector will then be rotated so that it is aligned with the tracking direction. A velocity tangential to the path will be specified, the end-effector will move and a new sensor reading will be taken. The loop will continue until the controller receives a signal that the task is complete.

4.4.4 PROGRAMMING THE SIMULATION

The hybrid control simulation was done using MATLAB. MATLAB can be used to obtain time responses of dynamic system models and can be structured into programs similar to many higher-order programming languages. The simulation was constructed to model the manipulator and position controller as described in section 4.4.1. Additionally, the simulation allows the user to specify any one of the three types of sensing discussed in section 4.4.2. The desired path for the manipulator to follow as well as the given starting position and orientation of the end-effector can also be specified. The rate at which the plant controller receives sensor data (the update rate) can also be set.

The transfer function modeling the plant was used to determine the position and velocity of the end-effector in response to a velocity input after set amount of time equal to the sensor update period. This position and velocity are then used as initial conditions for the plant when responding to the next velocity input signal. A separate function calculates velocity gains for the normal position direction using a digitally implemented PID control law. The gains for the PID control law were determined by applying loop shaping to obtain a robust controller for a continuous-time model of the plant. The value of the output of the plant at the end of each time step are stored so that the position of the

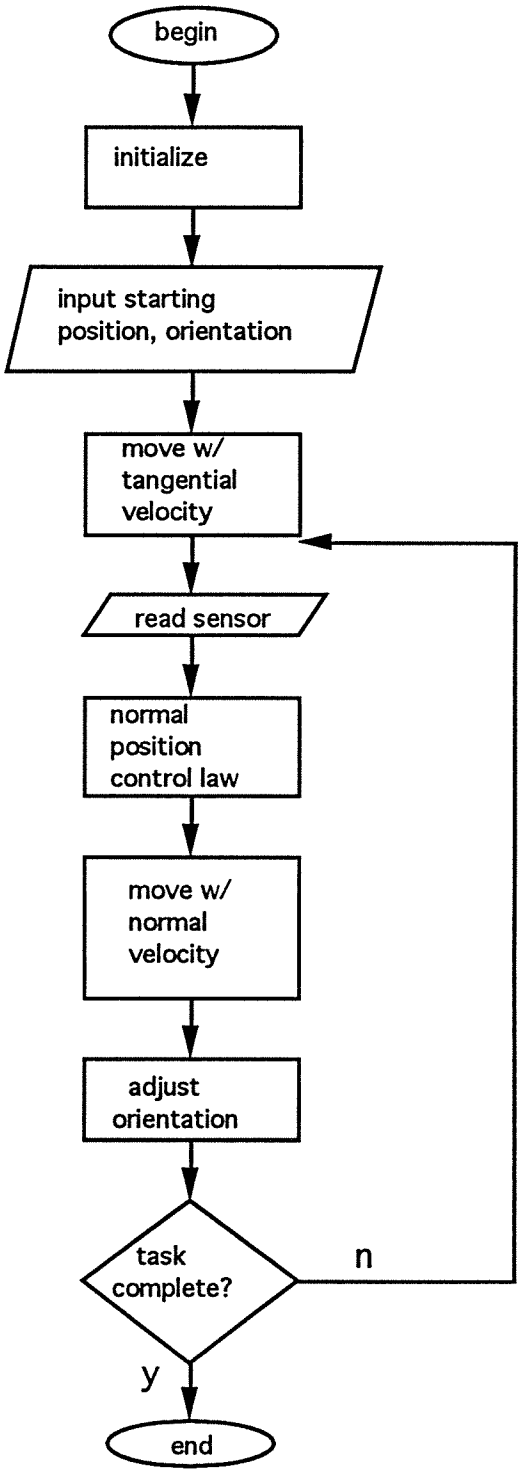


Figure 4.11 *Algorithm for simulated hybrid control.*

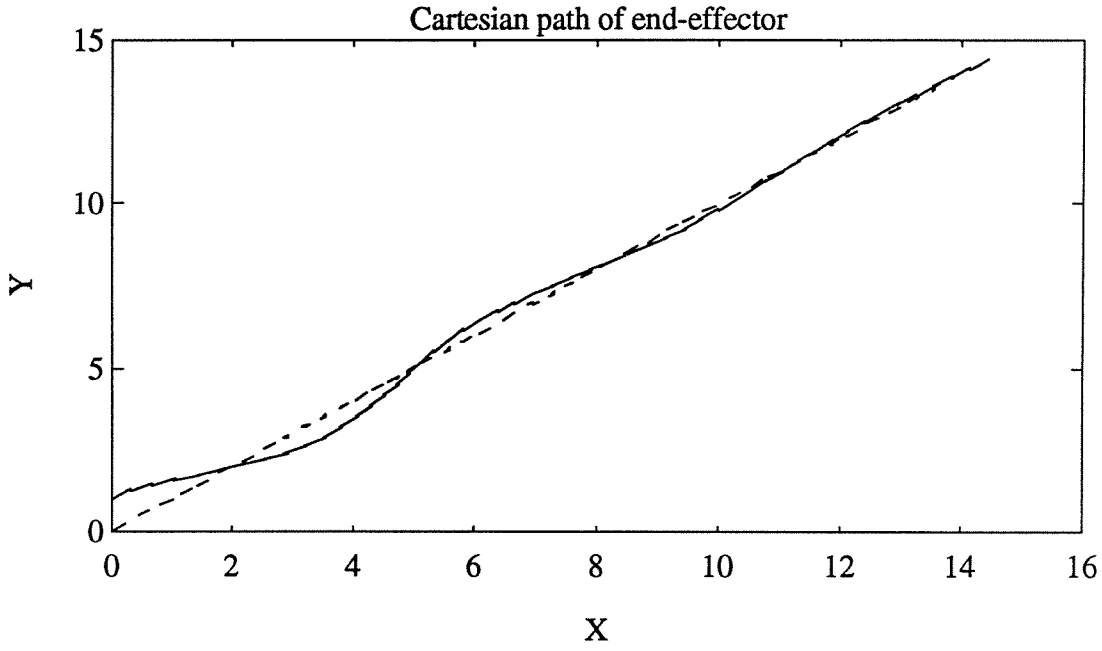


Figure 4.12(a) *Simulated trajectory of the end-effector using relative proximity sensing for hybrid control. The path used uses the function y=x. The end-effector was started at the location (0,1). The dashed line represents the sensed path and the solid line represents the end-effector trajectory.*

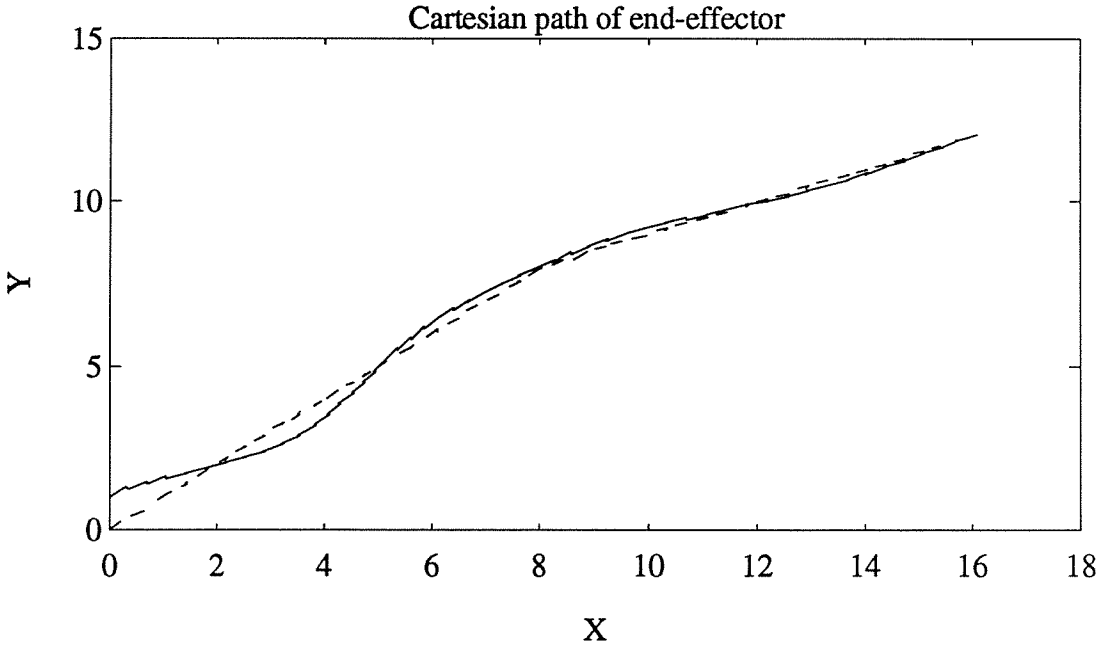


Figure 4.12b) *Simulation results same as above except using the function* y=x *for* x<=4 *and* y=0.5*(x-4) *for* x>4 *for the sensed path..*

time. Documented code for the simulation is given in Appendix A. Results of a simulation are shown in Figure 4.12.

4.4.5 CONTROL WITH RELATIVE PROXIMITY SENSING

Compliant motion control using relative proximity sensing has problems additional to those encountered for compliant motion control using force sensing. Since orientation errors of the end-effector cause errors in proximity measurement, special measures must be taken to control end-effector orientation. Small errors in orientation will cause proximity errors that can be accounted for in the design of a robust controller. Large errors in orientation, however, can cause proximity measurement errors large enough to make the system unstable. The largest errors occur when:

1) The end-effector oscillates to resume tracking following a disturbance.
2) The end-effector orientation is programmed to follow the tracking direction without accurately knowing the tracking direction.

This problem can be resolved taking the dynamics of the end-effector into account and designing a compensator which will act as a low-pass filter to keep the orientation from responding to oscillations in the end-effector position such as those caused by disturbances. The necessary factors in compensator design are:

1) Dynamics of end-effector orientation plant.
2) Oscillation frequencies of end-effector position.
3) Response time to changes in path orientation.

Any compensator design will involve compromises in disturbance and error rejection, response time and stability.

4.5 HYBRID CONTROL WITH AN INDUSTRIAL ROBOT

There are many problems associated with implementing any closed-loop control scheme on an off-the-shelf industrial robot controller. However, the availability of industrial controllers and the effort and costs associated with developing a custom controller make industrial controllers appealing for most applications. Also, most industrial controllers and robot programming languages are similar so that it is often an easy task to port a control scheme that works on one controller to most other controllers

on the market. For this reason, a control scheme using any industrial controller is likely to be usable on most robots presently in use.

4.5.1 CAPABILITIES AND LIMITATIONS OF INDUSTRIAL CONTROLLERS

Two separate controllers were used to implement the hybrid control algorithm developed in this thesis. These controllers are the Adept A-series controller manufactured by Adept Technology in San Jose, California and the GM-Fanuc RH controller manufactured by GM-Fanuc Robotics of Auburn Hills, Michigan. These controllers and other similar controllers represent nearly half of all operational robot controllers in the United States. The Adept and GMF controllers are very similar in capabilities and programming structure.

The primary task that an industrial controller can accomplish is moving its manipulator between two points in Cartesian space. Most controllers can perform this task with a variety of motion profiles and using almost any defined reference frame. Most controllers also have the ability to move along a series of defined points that form a path. While moving along a path, the manipulator can cause the joint decelerations for moving to one point to be smoothly joined to the joint accelerations for moving to the next point. This creates a smooth continuous movement through a series of points. The manipulator may not necessarily pass through each point exactly when following a path.

The primary limitation of industrial point-to-point controllers is the inability to change the destination of the end-effector once a move command has been given. Due to this fact, it is impossible to continuously update the position of the end-effector. If the position can't be updated continuously or at least at known time intervals, then most control theory will not apply to external control loops around the position controller. For this reason, there are limitations to the performance of a system with external control loops closed around an industrial controller that are greater than would be predicted by the dynamics of the controller-manipulator system by itself.

4.6 COMPLIANT MOTION USING FORCE SENSING WITH THE

ADEPT-3 ROBOT

The hybrid control algorithm was implemented on the Adept-3 robot using force sensing. This was done prior to integration with the LSS to test the hybrid control algorithm. A 6-axis force sensor was integrated with the Adept controller. This sensor allowed the control algorithm to be implemented and tested with a minimal amount of time devoted to the task of sensor integration.

A flowchart of the hybrid control algorithm is given in figure 4.13. For each move of the manipulator, the force sensor is read and the force magnitude and direction is calculated from the force reading. The force magnitude is run through a PID control law and multiplied by the estimated stiffness of the end-effector and the environment at the contact point. At the same time, the force direction is compared to the end-effector orientation. The difference between the end-effector orientation and the force direction is also run through a PID control law to obtain an updated orientation. The new point and orientation is used in a move command, the sensor is read and the cycle repeats until the task is complete.

The results of the tests with force control loops control loops closed around the Adept-3 manipulator confirmed the validity of the hybrid control algorithm. No attempt was made to optimize the control loops for the force control loop since a higher priority was set on implementing hybrid control with the LSS for use with the ACSM. Code for implementation of the force control routine and results are in Appendix B.

4.7 COMPLIANT MOTION USING THE LSS

This section will discuss the implementation of hybrid control using feedback from the LSS with two separate industrial SCARA configuration manipulators. The hybrid control algorithm was first tested on the Adept-3 robot. The algorithm was modified slightly and ported to the GMF A-510 robot. Section 4.7.1 will discuss the control algorithm used and any differences between the algorithm used for the simulation or

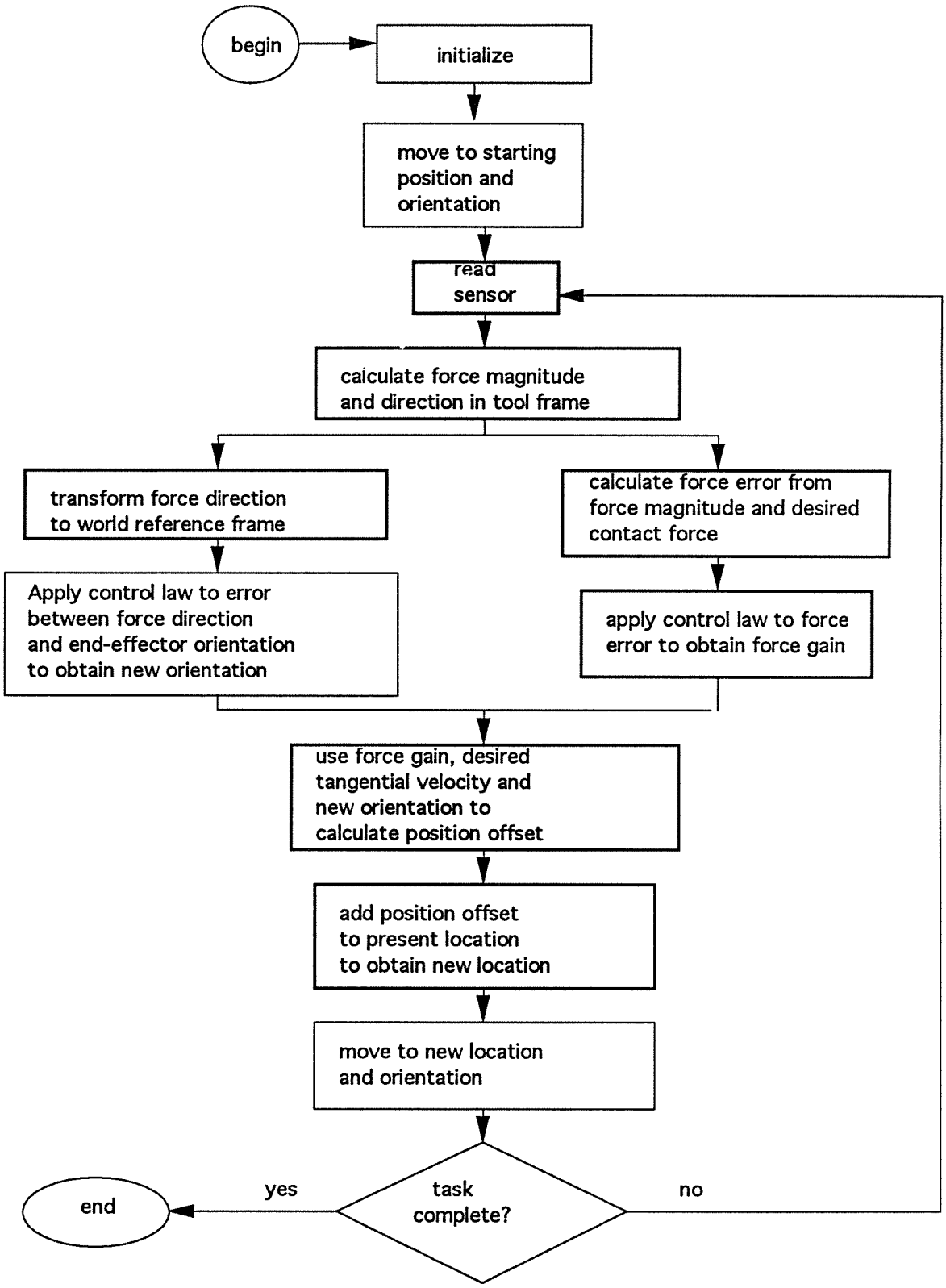


Figure 4.13 *Algorithm for hybrid control using force sensing.*

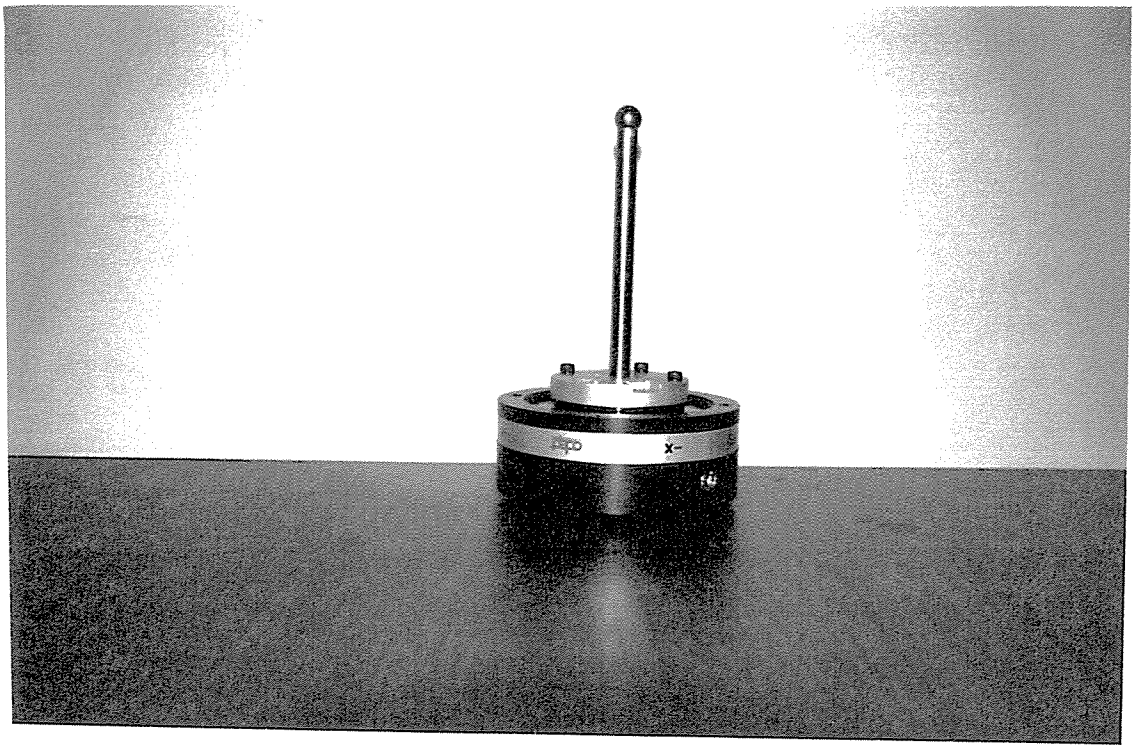


Figure 4.14 *Adept Technologies 6-axis integrated force sensor with probe attached for contour following.*

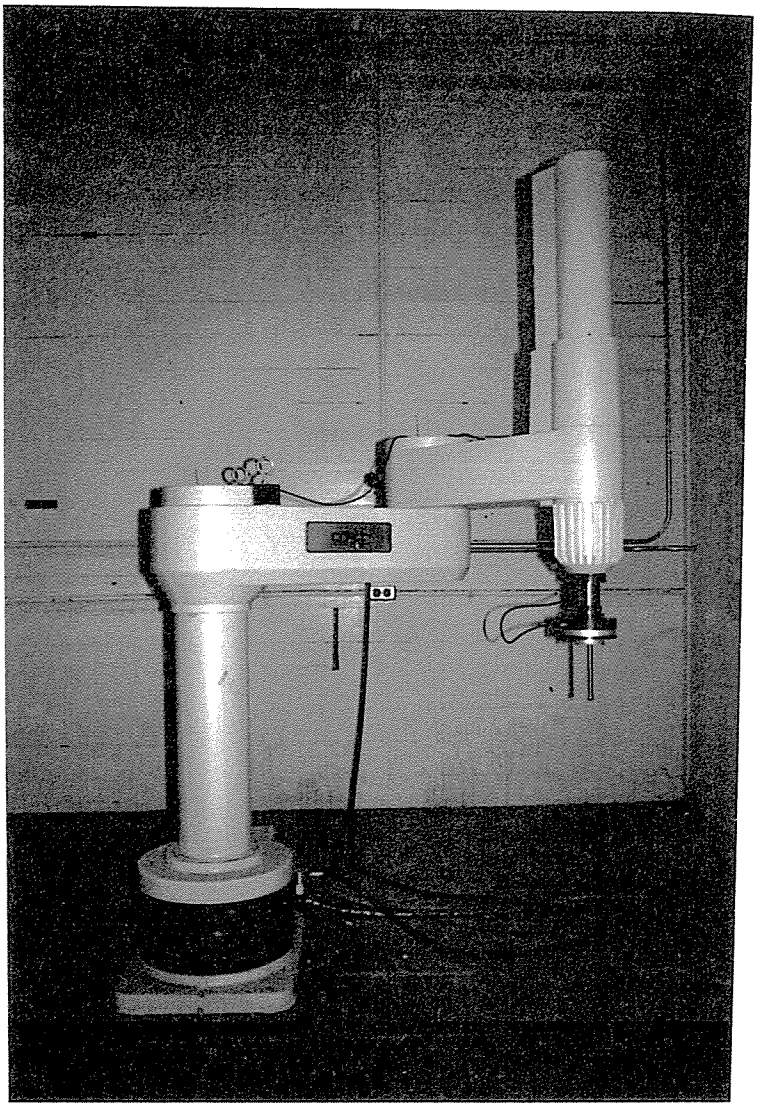


Figure 4.15 *The Adept-3 manipulator with endpoint force sensor attached.*

Figure 4.16 *Crack following with the Adept-3 and the LSS.*

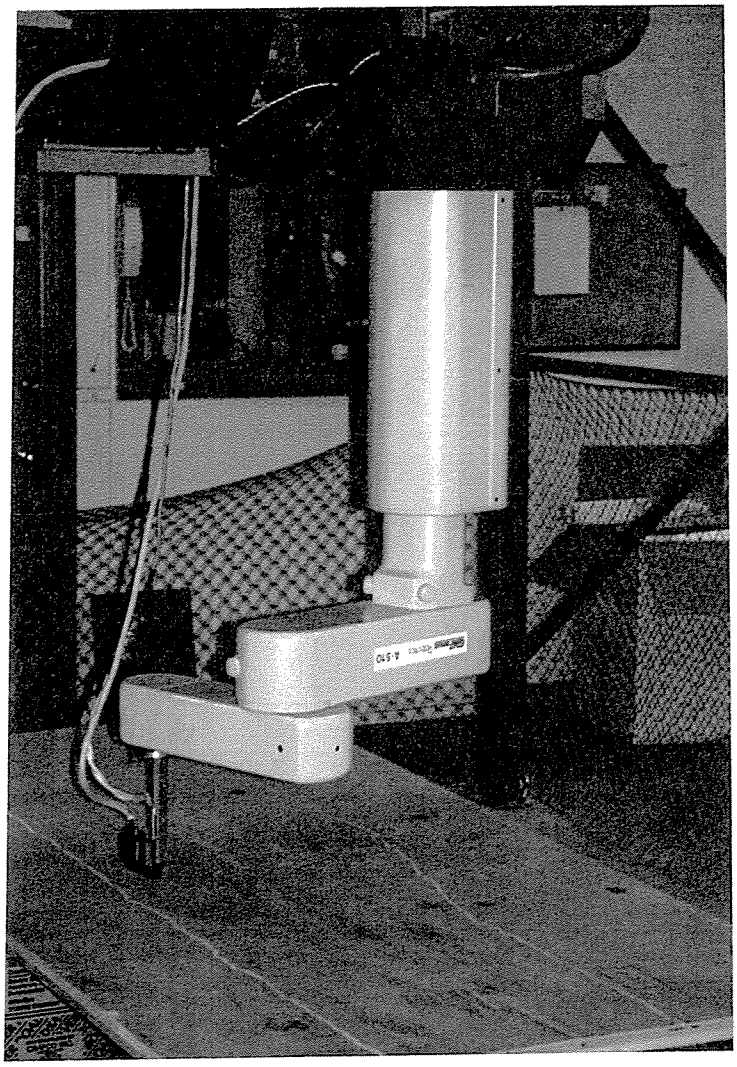


Figure 4.17 *The GMF-A510 manipulator, inverted, mounted on the linear slide with the LSS attached.*

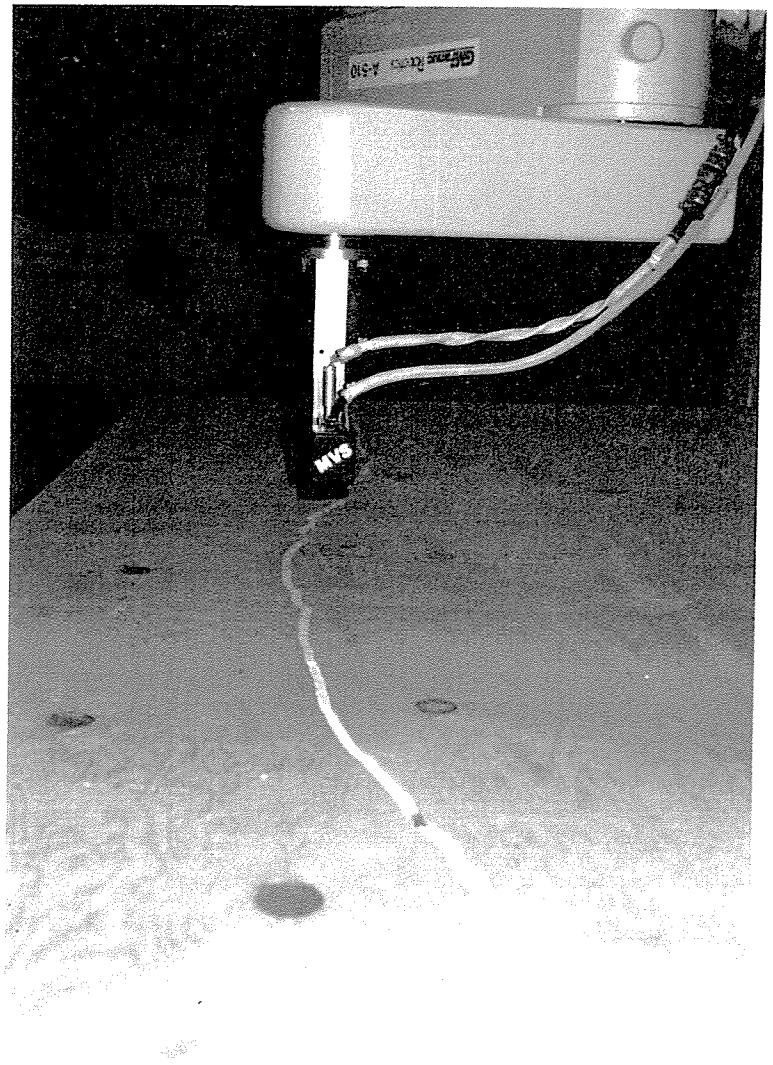


Figure 4.18 Crack following with the GMF-A510 and the LSS.

force sensing. Interfacing the robots with the LSS and the control architecture will be discussed in section 4.7.2. Section 4.7.3 will discuss tuning the controller and section 4.7.4 will discuss filtering of the LSS output to obtain improved performance.

4.7.1 - CONTROL ALGORITHM

A flowchart of the algorithm used to implement hybrid control with the LSS is given in figure 4.19. After all variables have been initialized the end-effector must be moved to the start of the crack and aligned with the direction of the crack at that point. A

sensor reading is then taken and the sensor error is run through a PID control law. At the same time the direction of the crack is estimated by looking at previous end-effector positions and the associated sensor readings. This provides estimated positions of the crack at the previous two points. The estimated orientation is calculated by taking a secant between these two points. The difference between the actual end-effector orientation and the estimated crack direction is also run through a PID control law to obtain a gain that is added to the current end-effector orientation to obtain the new orientation input. A new offset point is calculated relative to the current position using the normal direction gain, the new orientation and the tangential velocity gain (which is set as constant). The offset point is then added to the current position to transform it to the world coordinate system. The new point is then used as a motion input, a sensor reading is taken and the loop continues until the controller receives a signal that the task is complete.

Both the Adept and GMF controllers will process commands while a move command is taking place. Therefore, it would seem that using the above algorithm, the controller will generate points at the same rate at which the controller could cycle through the loop. However, the motion constraints of the controller will only allow the controller to have at most two move commands whose destination points have not yet been reached. If it receives a move command for a third point, it will halt program execution until one of the previous points has been reached. Once the controller is executing the loop, the cyclic rate will be the same as the time necessary for the end-effector to move between two consecutive generated points. This time varied between 50 ms and 500 ms during tests with the algorithm using the Adept-3 manipulator. Figure 4.20 illustrates the differences between the update rate of the controller and the update rate of the LSS.

4.7.2 CONTROL ARCHITECTURE AND COMMUNICATION

Interfacing the Adept or GMF controller with the LSS is essential to closing the normal direction control loop for hybrid control. The error data from the LSS is

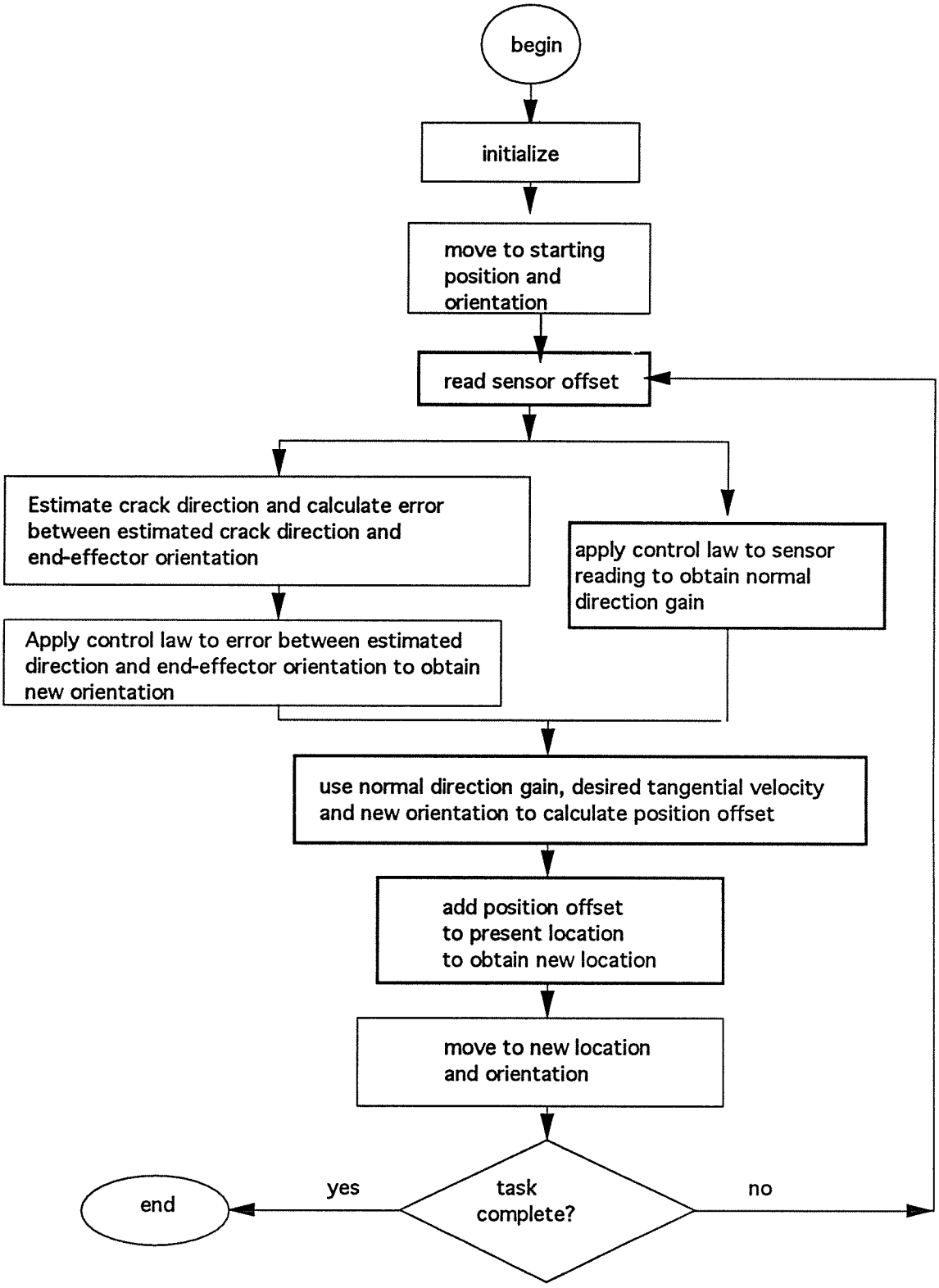


Figure 4.19 - Flowchart for hybrid control using the LSS

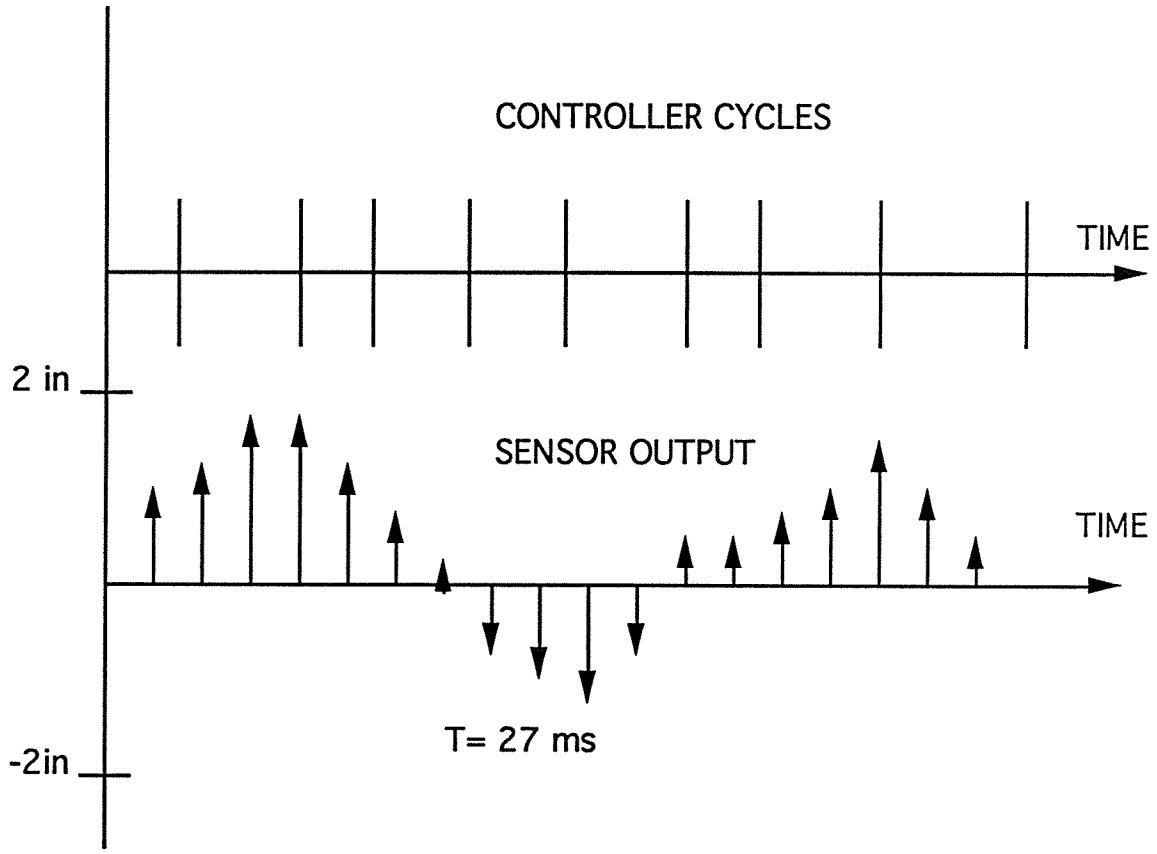


***Figure 4.20** Comparison of LSS and controller update times.*

processed from raw sensor data using an IBM PC (see chapter 3). The error information is passed from the PC to the controller via an RS-232 serial line at a transmission rate of 4800 baud. For ease of transmission each error reading was scaled to be transmitted as a single byte. Since the LSS generates error data faster than it can be used by the controller, a 'software handshaking' protocol was implemented so that excess error data will not accumulate on the serial port buffer of the controller. When the controller is ready to receive a piece of data, it sends a signal byte to the IBM PC being used by the LSS. Each time the LSS determines a new error signal, it checks to see if the controller has requested information. If the controller has requested information, the error signal is sent. If the controller is not ready for new information, the LSS loops through another scan until it determines a new error signal and again checks the serial line for a data

request from the RPS. Due to the time it takes for the LSS program to loop and the time required to both receive and send data bytes, time delays due to communications can be as long as 40 milliseconds. This delay could be substantially reduced by implementing a service interrupt routine for serial communication on the PC processing the LSS data (Krulewich and Velinsky, 1992). Details on operation of the LSS are given in Chapter 3.

4.7.3 TUNING THE CONTROL LOOPS

The hybrid control algorithm uses two separate control loops. One loop is the normal direction control loop that is closed around the LSS and the other loop is the orientation control loop which is closed by estimating the crack direction. These control loops are dynamically coupled due to the relative nature of the proximity sensing as described in section 4.2.3. In order to deal with the coupling effects, both control loops should be tuned to have a high damping ratio to insure high stability. If both loops are very stable, oscillations in one loop will not propagate to the other loop and cause the system to become unstable.

It is not possible to rigorously apply control theory to the control loops in the hybrid control algorithm. This is primarily due to the fact that point inputs to the controller can't be updated once the motion has begun as described in section 4.4.1. Also, the cycling time of the control program varies between 50 and 500 milliseconds. If the cycling time of the control program were constant it would be possible to model the system as a digital system with a sampler and zero-order hold. The LSS data could be treated as a sampled input and move commands to the motion controller could be treated as discrete output followed by a zero-order hold. The dynamics of the plant could be roughly modeled by empirically measuring motion of the manipulator and fitting the data to a second-order system model. However, since the sensor updates at approximately 33 Hz and the controller cycles at anywhere between 20 Hz and .5 Hz, the resulting system is a multi-rate discrete system where the time of the hold for the output varies with each cycle. No existing control theory can reasonably handle such a system in a rigorous manner.

Therefore, the control loops of the hybrid control algorithm had to be tuned using trial-and-error and applying general principles of control theory rather than specific laws.

The parameters for tuning the normal position control loop are dictated in a large part by the nature of the crack sealing operation. First, extreme accuracy is not required. Errors of up to an inch are perfectly acceptable for the task. Secondly, the type of cracks to be addressed by the ACSM will not make sharp changes in direction (greater than 25 degrees). The path of the end-effector should also be smooth, therefore the system should not respond to small oscillations in crack direction (which will appear as high frequency signals to the LSS). Finally, the system should be tuned to be as stable as possible due to inherent instabilities induced by position-orientation coupling and a variable cyclic rate. Therefore, the compensator for the normal position direction should emphasize stability and noise rejection at the expense of response time and steady state error.

The control law for the orientation error of the end-effector should act as low-pass filter. A sample of the orientation error is given in figure 4.21. Since the crack direction is estimated by using what essentially is a numerical differentiation algorithm, the

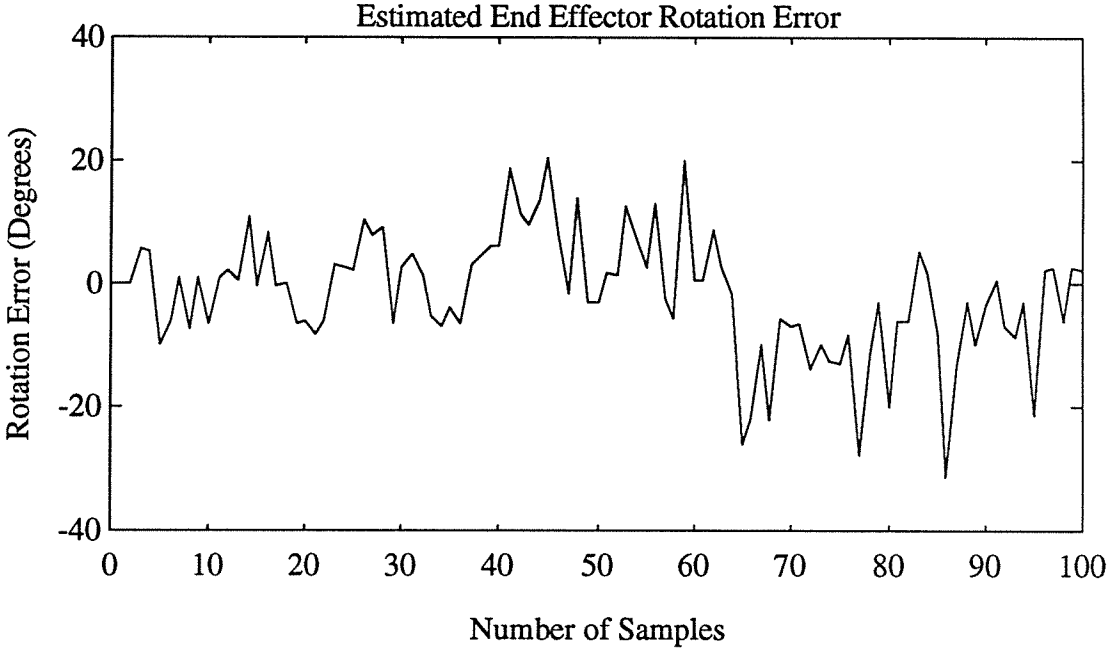


Figure 4.21 *End-effector orientation error during hybrid control.*

estimated crack direction tends to be a very noisy signal. Once again, stability and disturbance rejection in the control loop should be emphasized over response time and accuracy.

4.7.4 FILTERING OUTPUT OF THE LOCAL SENSING SYSTEM

It was determined that system performance using hybrid control could be improved by applying a low pass filter to the output of the LSS. All of the data read by the LSS is filtered with a second-order filter. The most recent filtered error is sent to the RPS each time data is requested. There are several reasons why the filter will improve system performance.

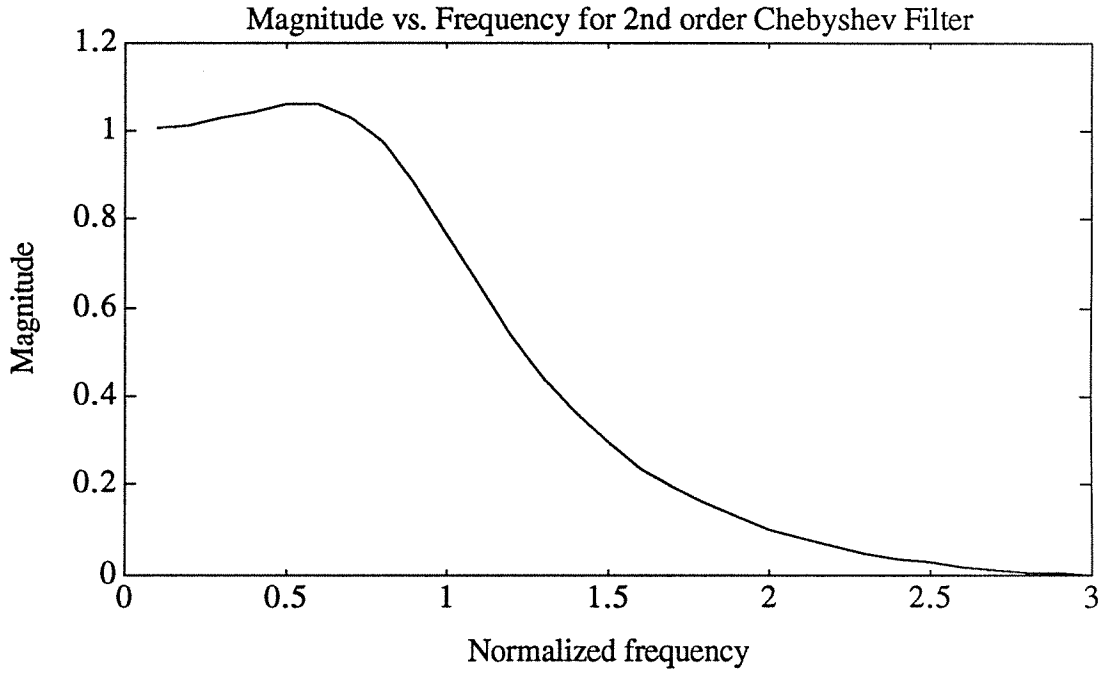


Figure 4.22 *Magnitude-frequency plot of a type-1 second order Chebyshev filter. The cutoff frequency for this filter was set at 4 Hz assuming a sampling rate of 33 Hz. The normalized frequency is such that*

$$\omega_{normalized} = \frac{\omega\pi}{\omega_s}$$

where ω_S *is the sampling frequency.*

Without the filter, the controller uses the most recent output for each cycle in the algorithm. The sensor can update many times faster than the controller can accept data. Therefore, much of the data from the LSS is lost if a filter is not used.

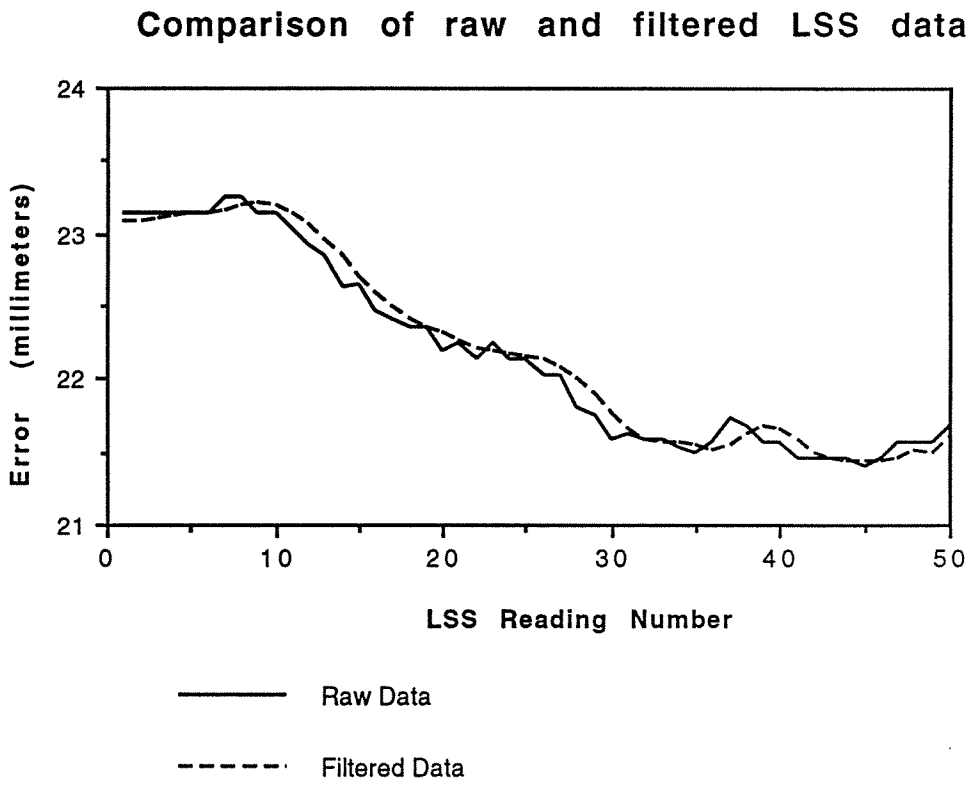


Figure 4.23 *Comparison of filtered and unfiltered sensor data from a test of the hybrid control algorithm. The test was run using a crack routed in plywood.*

A smooth path of the end-effector is desirable for the crack sealing operation. Pavement cracks are often rough and have many small deviations. In order to achieve a smooth end-effector motion these small deviations must be filtered out as if they were high frequency noise.

An integral gain was originally used in the normal position control loop for the

hybrid control algorithm. However, due to the dynamic coupling with orientation and the multi-rate effects, the system often became unstable with even small integral gains. It was therefore decided to eliminate the integral gain for the normal control direction at the controller and achieve disturbance rejection characteristics and a high damping ratio by filtering input from the LSS.

For testing purposes on cracks routed in plywood, the filter cut-off frequency was set at 4 Hz. For operation on pavement, the optimal filter cut-off frequency can be determined from frequency analysis of LSS data while operating in a crack following operation. The cut-off frequency is a function of end-effector speed as well as crack geometry. Frequency analysis of test on pavement cracks may also reveal the need to use a higher order filter.

4.7.5 ESTIMATING CRACK DIRECTION GEOMETRICALLY

An alternate method of calculating the end-effector orientation through geometric means was also tested. Problems arise with the control algorithm for end-effector orientation due to the fact that noise is tends to be amplified and the time step for the control law is unknown and varies with each step. The geometric method looks at previous points and finds the tangent through a interpolation or curve fit in Cartesian space. This method has been tested by using a least squares linear regression on the five most recent points along the crack to calculate a tangent angle.

4.8 CRACK SEALING USING HYBRID CONTROL

The following section will describe how the compliant motion algorithm with the local sensor is used on the ACSM. A control architecture is developed that combines all available information from the Vision Sensing System, the Path Planning system as well as the Local Sensing System. The object of the architecture is to implement crack following as quickly and efficiently as possible while making allowances for failures or errors in individual subsystems.

4.8.1 FEEDBACK VERSUS OPEN-LOOP CONTROL

Two types of control are available for the controller. The first type is open-loop control using the data from the VSS. Open-loop control assumes that the information from the VSS is available and correct. The data from the VSS will be processed to produce Cartesian locations in the workspace of the manipulator. The location data will consist of points along a crack as well as the crack direction at each point. Open-loop control has no way of the accuracy of the data.

The second type of control is closed-loop or feedback control. This is the type of control that is done using the output of the LSS. Feedback data can only reveal past locations and has no a-priori knowledge of the crack. Feedback control can correct errors in the manipulator position by feeding an error signal back to the controller. The LSS is only capable of returning the offset position of the crack in its field of view with respect to the end-effector frame of the manipulator. The crack direction in feedback control can be estimated by looking at previous points along the crack.

Controlling the manipulator can involve open-loop control, feedback control or a combination of both. There will be 3 distinct modes of control based on the type of control. Mode 1 will use open-loop data and verify it with the LSS data. Mode 2 will use feedback data from the LSS to control both position and orientation and Mode 3 will use feedback control to control position and open-loop control to control orientation.

4.8.2 - SOURCES OF ERROR

Two likely sources of errors for sensing cracks are the Vision Sensing System (VSS) and the Vehicle Orientation and Control System (VOC). The purpose of the VSS is to detect cracks in front of the support vehicle by using an area scan camera. The algorithm used by the VSS divides the scanning area into 2x2 inch pixels. The VSS compares gray shades to determine if a crack exists in each pixel. Errors of greater than an inch are possible simply due to the resolution of the pixels. These errors are added on to hardware errors such as camera resolution and vibration of the camera mount.

The purpose of the VOC is to kinematically transform the cracks detected by the

VSS in front of the vehicle to the workspace of the robot which is behind the vehicle as the vehicle moves forward. There will be more than a 40 foot separation between the front and rear of the vehicle. The VOC will determine vehicle motion by monitoring encoder wheels attached to the vehicle frame. Sources of error for the VOC include slippage of the encoder wheels on the pavement and the resolution of the encoder wheels. Errors from the VOC will be added to the errors already present from the VSS.

After a crack is detected, the vehicle moves forward until the crack is in the workspace of the robot. Due to the errors described above position errors of several inches are possible. It is also possible that no crack actually exists since the VSS can only detect gray shades and therefore can be fooled by grease stains, shadows and already sealed cracks. The field of view of the LSS is only 3 inches, so it is highly probable that once the manipulator moves to the start of the crack, the error will be great enough that the crack will not even be in the field of view of the LSS. It is also possible that the detected crack does not even exist.

4.8.3 - USE OF CLOSED-LOOP CONTROL

Many of the problems associated with errors from the VSS and the VOC can be eliminated by incorporating a compliant motion algorithm with the LSS. However, crack following using the compliant motion algorithm with LSS limits the manipulator speed. The speed limitations are due to the fact that the algorithm must define many points to constantly control the manipulator motion. The use of a higher point density causes more joint accelerations and decelerations and hence tends to slow the speed of the end-effector.

4.8.4 MODE 1-OPEN-LOOP CONTROL

The concept for the control architecture is to use a pre-planned path generated from VSS data as much as possible. The compliant motion algorithm will be used to search for the crack if the errors in the pre-planned path become larger than the field of view of the LSS. Once the crack is located by the compliant motion algorithm, the actual crack

location can be sent back to the ICU. The error in the crack location can then be used to help correct errors in the pre-planned path.

4.8.5 MODE 2-FEEDBACK CONTROL FOR POSITION AND ORIENTATION

The compliant motion algorithm can also completely override use of a pre-planned path in the event of a system failure of the VSS, the VOC or the ICU. In this case, the end-effector can be moved to the beginning of the crack and aligned with the starting direction of travel using the teach pendant. The compliant motion algorithm will then be executed for the crack following operation. The end-effector speed using the compliant motion algorithm will be slower than if pre-planned path data were available.

4.8.6 MODE 3-FEEDBACK CONTROL FOR POSITION AND OPEN-LOOP CONTROL FOR ORIENTATION

A scheme that uses the LSS to control the position of the end-effector and data from the VSS to control the orientation of the end-effector can also be used. This scheme will work well if there are errors in the location of the crack from the VSS, but its shape is generally known. The direction of the crack will be sent to the RPS according to its location along the 'x' axis of the world coordinate system. The control algorithm will use the LSS to follow the crack, but the direction information from the VSS will replace the orientation control loop of the algorithm. This scheme will require the approximate Cartesian location of the starting of the crack.

4.8.7 ERROR HANDLING

An error handling routine will be called any time the crack is lost from the field of view of the local sensor. The same error handling routine will be used regardless of whether the motion was being controlled by data from the VSS or the LSS. The search algorithm will search at right angles for the crack according to the most recent crack direction. The searching algorithm is essentially the same as the compliant motion algorithm except that the tangential velocity is set to zero. If the crack is not in the field of view of the sensor, an extreme error value must be given to indicate which direction to

start the search. If the crack is still not found, a search will be conducted in the opposite direction as well. Limits are set on the distance to be searched.

4.8.8 - CONTROL ARCHITECTURE

The control architecture will be defined by the following algorithm:

(1) The location of crack will be determined by the VSS, VOC etc. and a path consisting of Cartesian points within the workspace of the robot will be calculated and sent to the RPS.

(2) The manipulator will move to the beginning of the crack using the path generated by the ICU and path planning.

(3) The local sensor will be checked to determine if a crack is present. If no crack is found, the RPS will signal the ICU and begin searching for the crack at right angles using the LSS. The result of the search--either 'no crack found' or the location of the crack will be sent to the ICU. The RPS will then wait for an updated path to be sent from the ICU and will resume motion with step (2).

(4) Once the location of the crack has been verified by the LSS and the path has been updated (if necessary) the manipulator will follow the crack and the LSS output will be monitored. The LSS data can be used to determine if the manipulator runs off of the crack. If this occurs, the ICU will be signaled and a search for the crack will begin as in step (3). Once again, the result of the search will be sent to the ICU and the RPS will wait for an updated path before resuming motion as in step (2).

(5) When the manipulator reaches the last point defined on the path it will signal the ICU to indicate that the end of the crack has been reached.

Additional options:

(a) It will be possible to define a maximum offset within the field of view of the sensor. If the offset from the end-effector to the crack exceeds the maximum offset, the manipulator will center itself on the crack using the LSS and signal the ICU in similar manner to the signaling for the search routine described above.

(b) It will be possible to override all data from the ICU and follow cracks using only the RPS and the LSS. This override will require the manipulator to be moved to the start of the crack and aligned in the direction of travel using the teach pendant. The motion speed for this override will be much slower than if global data were used.

(c) Crack following can also be conducted using crack direction data from the VSS and using closed-loop control with the LSS for position control.

4.8.9 - COMMUNICATIONS

Communication to and from the RPS will be accomplished by means of serial lines. Two separate lines will be used, one for communication with the LSS and one for communications with the ICU. Specifics on the data transfer can be found in the documented code for integrated control in Appendix G.

4.9 - SUMMARY AND CONCLUSIONS

This chapter has addressed the problem of developing a motion control algorithm for hybrid control of an industrial robot. Hybrid control algorithms for both force and relative proximity sensing were developed. The purpose of the hybrid control algorithm is to follow cracks in the pavement with an industrial robot using the local sensor as part of the ACSM.

The first section of this chapter defined the problem to be addressed and divided it into three tasks, two of which implement control loops around the robot controller. The

second section discussed previous work that has been done in the area of hybrid control. Most of the work that has been done with hybrid control involves endpoint force sensing rather than proximity sensing. This section also discussed the advantages of using control loops closed around an industrial controller. The task frame for the problem was presented and the difference between implementing hybrid control with force and relative proximity sensing was also discussed.

The third section of chapter 4 presented a numerical simulation of hybrid control using MATLAB. The MATLAB simulation revealed important qualitative results on hybrid control using relative proximity sensing. The simulation showed that there is a coupling effect between position and orientation control when using relative proximity sensing. Therefore, an algorithm for hybrid control using the LSS should have compensators for orientation control as well as control of position normal to the path. Both compensators should be designed to yield a system that is heavily damped.

The fourth section began actual implementation of hybrid control using industrial robots. The hybrid control algorithm was implemented on the Adept-3 robot using both endpoint force sensing and endpoint relative proximity sensing with the LSS. Hybrid control was also implemented on a slide-mounted GMF A-510 manipulator using the LSS.

Due to the limitations of an industrial controller, it is difficult to rigorously apply control theory to control loops that are closed around an industrial controller. However, the best performance is achieved when the gains were adjusted so that the system is heavily damped. Also performance was increased by filtering the error data from the LSS with a low-pass filter.

The final section of the chapter discussed how the hybrid control algorithm fits into the overall architecture of the ACSM. The architecture attempts to combine off-line machine vision data along with closed-loop control using the LSS to create an efficient and robust crack-following capability for the ACSM.

CHAPTER - 5 SIMULATION AND TEST RESULTS

This chapter will discuss the results of implementing hybrid control on industrial controllers. Results of the hybrid control simulation programmed on MATLAB will also be presented. The results will show how the hybrid control algorithm with the LSS was refined progressively from the MATLAB simulation to implementation on the Adept-3 to its final implementation on the slide-mounted GMF-A510 robot. The results presented will be mostly qualitative in nature. The main criteria for the results are that the algorithm is capable of performing the desired operation of crack following. The gains for the controllers were adjusted until performance was satisfactory before data was collected for results in this section. Because control theory cannot be rigorously applied to the hybrid control algorithm, it is impossible to determine what optimal performance of the system should be (section 4.6.3). However, the control laws should take anticipated crack geometry into account (such as how quickly cracks change direction and the amount the cracks jog from side to side).

Section 5.1 will discuss results of the MATLAB simulation. Section 5.2 will discuss results of hybrid control using an endpoint force sensor on the Adept-3 robot. Results of hybrid control using the LSS will be presented in section 5.3.

5.1 - RESULTS OF NUMERICAL SIMULATION OF HYBRID CONTROL

A simulation of the hybrid control algorithm was done using MATLAB. There were several reasons for performing the simulation. First, the simulation was to test the validity of the compliant motion algorithm using endpoint force sensing and obtain some idea of how well it would perform. Secondly, the simulation was to determine what differences exist between using force sensing and relative proximity sensing for hybrid control. Finally, it was hoped that the simulation could be used to help determine optimal control gains for the hybrid control algorithm for implementation with an actual robot.

The simulation was done prior to the arrival of the Adept-3 robot and therefore provided a head-start for implementing hybrid control.

5.1.1 - SIMULATION OF HYBRID CONTROL WITH FORCE SENSING

The simulation was first performed for compliant motion using force sensing. The force sensor is simulated such that it will accurately return the distance between the end-effector and the desired path as well as the direction of the desired path. This simulates using a force sensor with a flexible tool attachment. The distance measurement can be obtained by dividing the measured force magnitude by the estimated stiffness between the end-effector and the environment at the contact point. The direction of the desired path can be obtained by taking the inverse tangent of the measured forces along the axes of the world coordinate system.

The hybrid control algorithm with force sensing uses one control loop to control motion in the force-controlled direction based on the force reading. Velocity and orientation control are done open-loop. The gains for the force control loop were chosen such that the system has a heavily damped response. This was done because stability was determined to be the most important criteria for crack following.

5.1.2 - SIMULATION OF HYBRID CONTROL WITH RELATIVE PROXIMITY SENSING

After the hybrid control algorithm was simulated for force control, the same algorithm was tested using a simulation of the LSS. The simulation of the LSS measures the distance between the end-effector and the crack along the scan line of the LSS and is not capable of measuring the direction of the path directly. The direction of the path using the LSS is determined by looking at previous positions of the end-effector and taking secant lines. This method of estimating the tracking direction tends to make the system less stable. Applying a low-pass filter to the estimated tracking direction improves system stability. Although a low pass filter for orientation was never used in the simulation, the need was recognized and a filter was used for hardware

implementation. Filtering allows the end-effector to be aligned with the tracking direction while not responding to oscillations in end-effector motion due to disturbances.

The hybrid control algorithm for the LSS uses two control loops; one for the normal direction to the path and one for end-effector orientation. Overall, the hybrid control algorithm simulation using the LSS has a less stable response than the algorithm using force sensing because the LSS cannot directly sense path direction. The response of the algorithm using both simulated force sensing and a simulation of the LSS is shown in figure 5.1. Furthermore, the simulation revealed a coupling effect between the two control loops of the algorithm using the LSS. Instability in one loop will tend to propagate to the other loop. Therefore, the external control loops for hybrid control using the LSS must be chosen such that the system response is heavily damped.

Code for the MATLAB simulation and the models that were used are contained in Appendix A.

5.2 - HYBRID CONTROL WITH ENDPOINT FORCE SENSING

This section will discuss the results of implementing hybrid control on the Adept-3 manipulator using an Adept A-series controller with an integrated 6-axis force sensor. The work that was done in force sensing is rather brief since it took place while waiting for a hardware problem with the LSS to be corrected. No attempt was made to optimize the external control loop gains on the algorithm since priority was placed on implementing hybrid control with the LSS. Hybrid control using loops closed around the Adept-3 controller did in fact work, but a certain amount of passive compliance between the end-effector and the environment was necessary. The need for a certain amount of passive compliance when implementing counter following using force control is mentioned in De Schutter (1988). The algorithm used was as described in section 4.6.

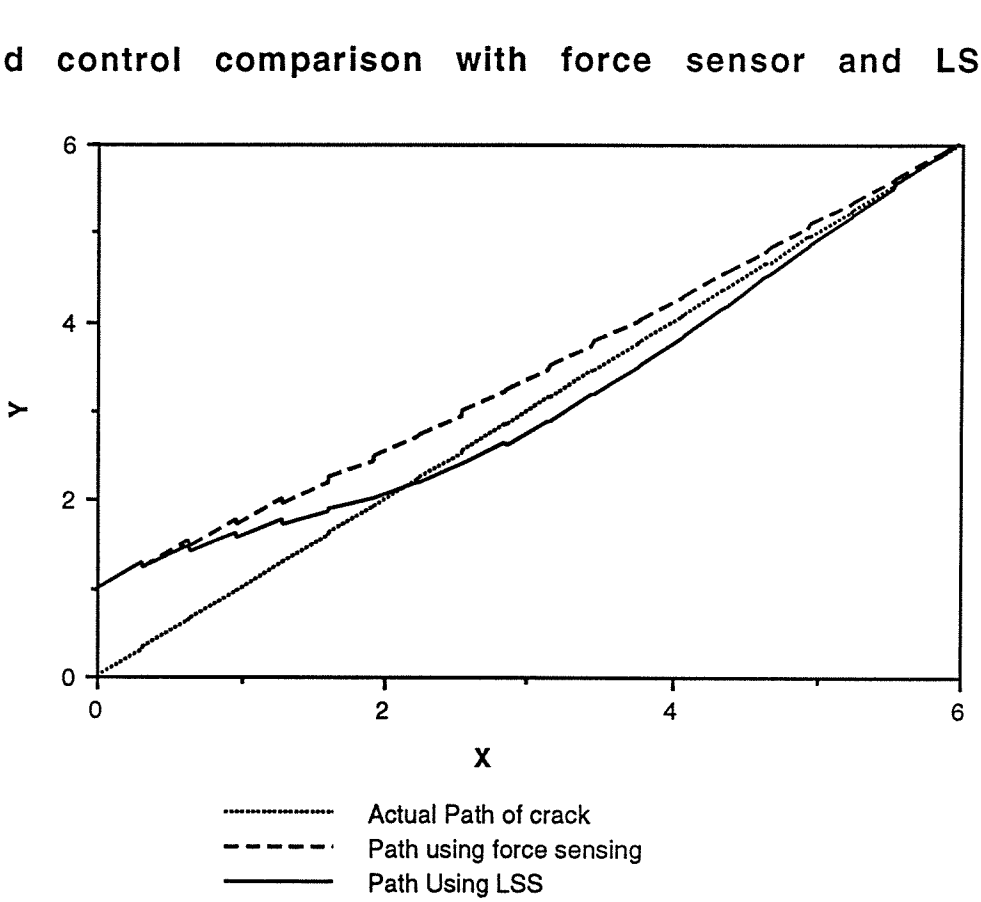


Figure 5.1 *The simulation results for hybrid control using simulated force sensing and the LSS. The end-effector was started with a one inch offset in both cases.. For the same system, the algorithm is less stable and has higher overshoot when using the LSS.*

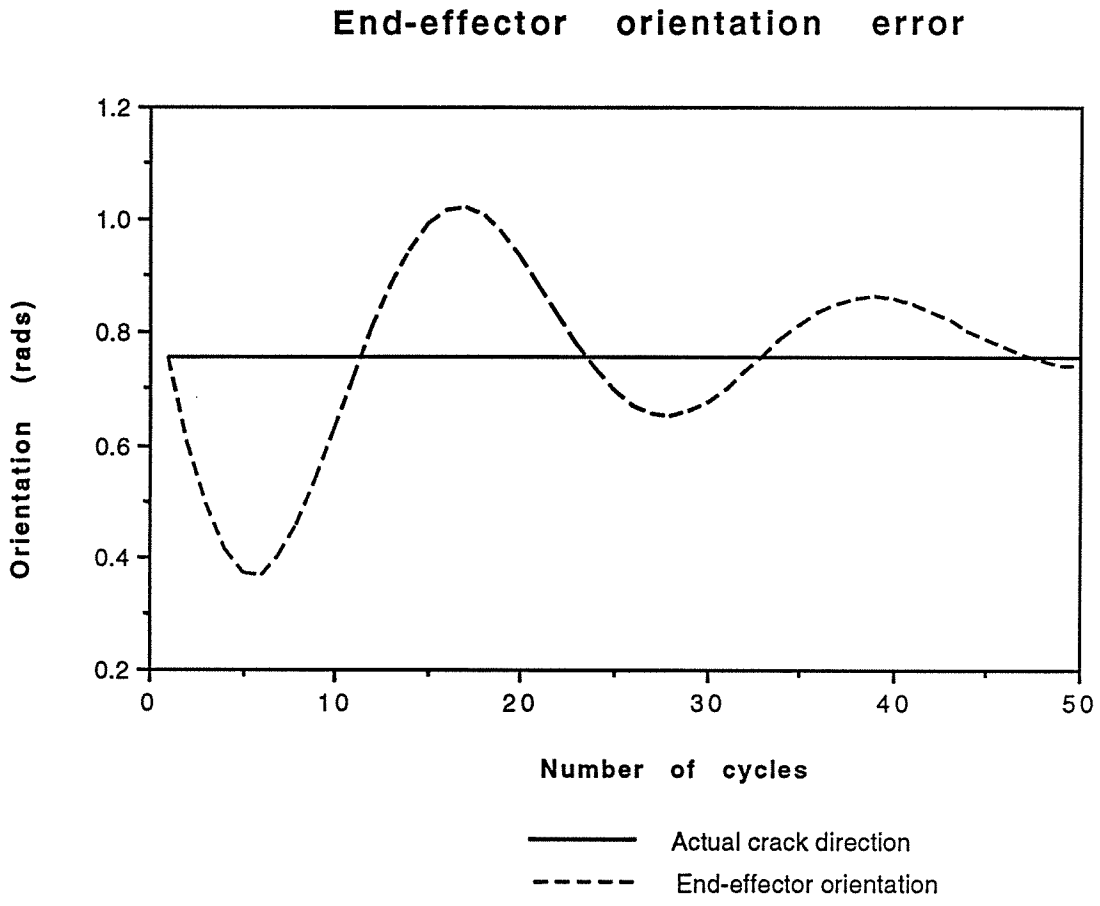


Figure 5.2 *The end-effector orientation error from hybrid control simulation using the LSS. This result shows the need for a low-pass filter to damp out oscillations in the estimated tracking direction.*

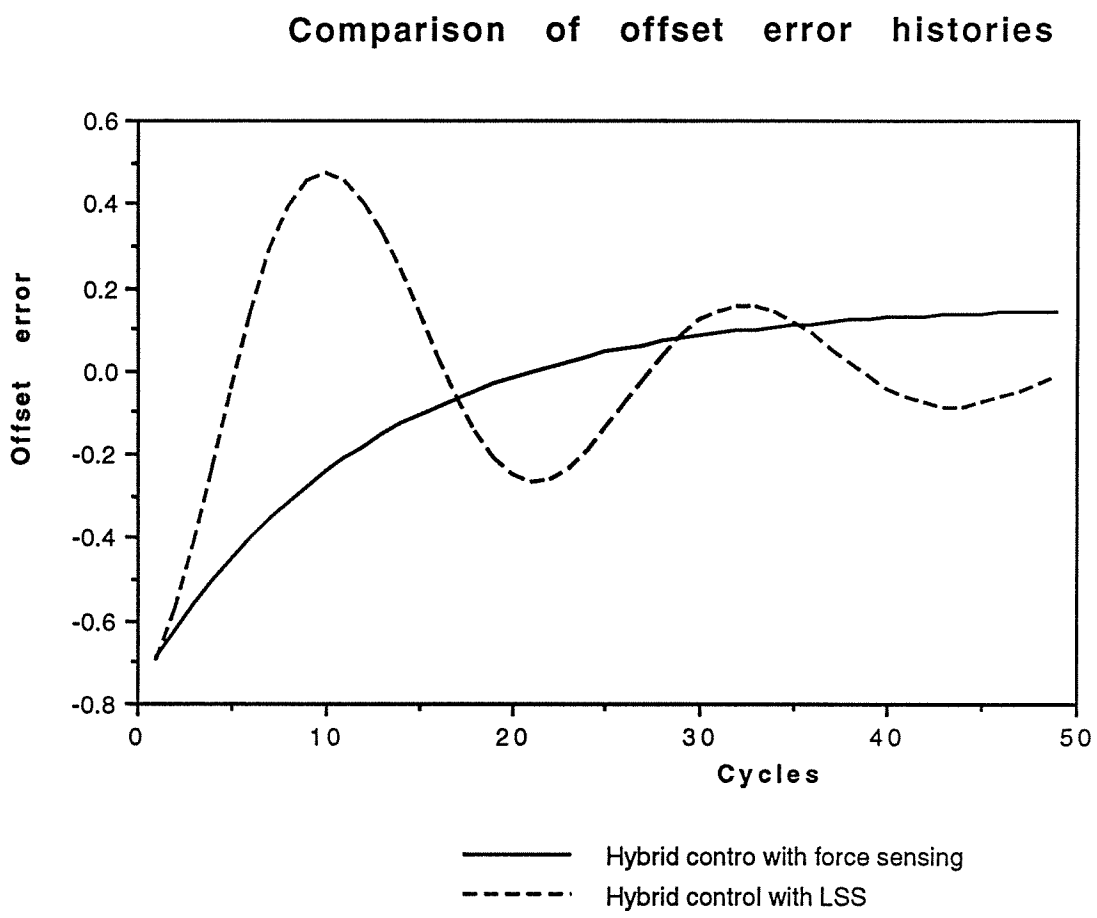


Figure 5.3 *Comparison of the offset errors for the hybrid control simulation using a force sensor and the LSS. The results show that the external control loops must heavily damped to achieve stable tracking when using the LSS for hybrid control.*

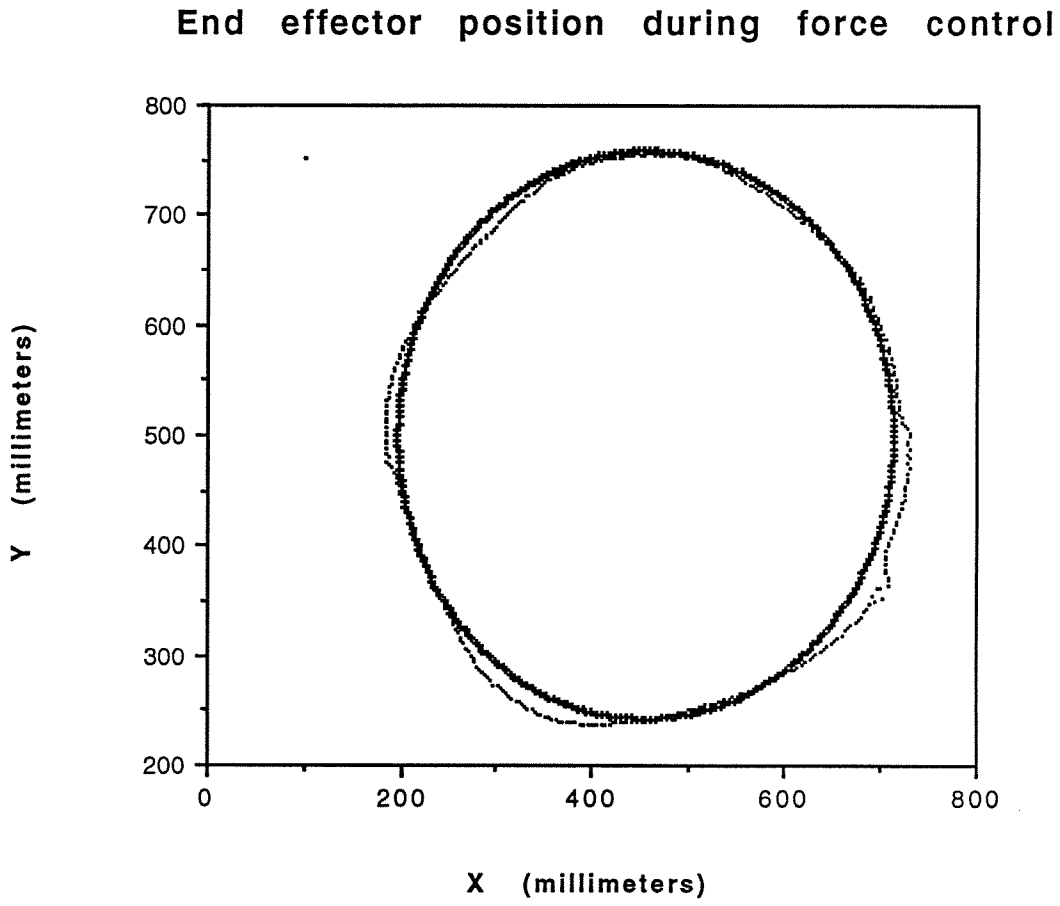


Figure 5.4 *Result of 2-dimensional contour following using endpoint force sensing with the Adept-3 robot. The contour consisted of a flexible plastic circle. The undeformed contour and the actual path of the end-effector are shown.*

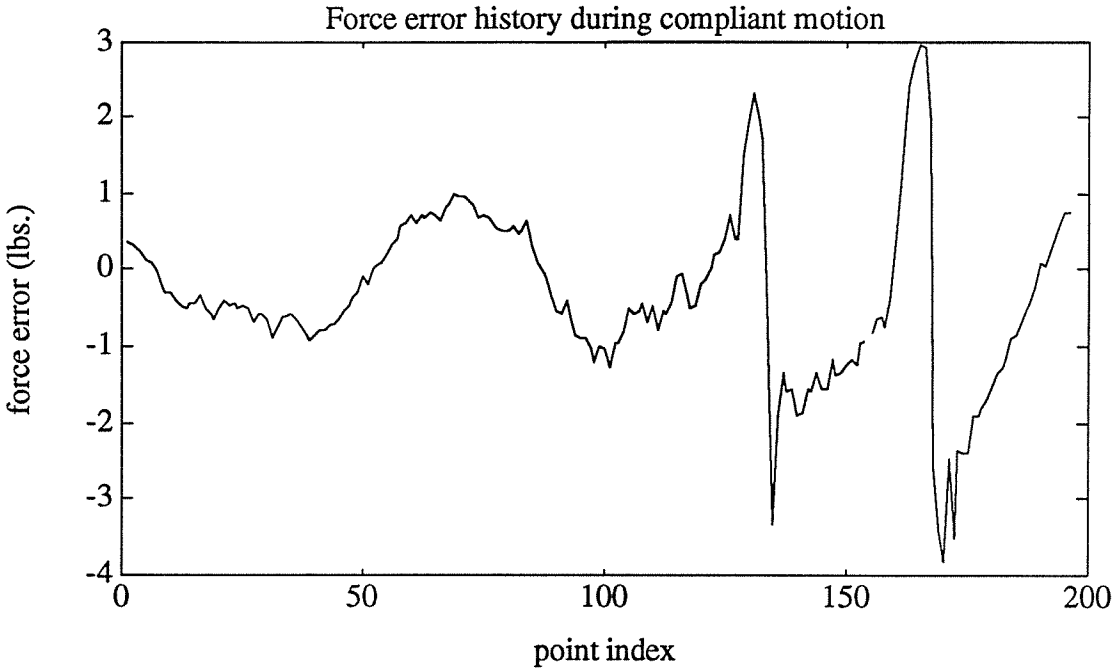


Figure 5.5 *Error history from contour following shown in Figure 5.4. The sharp spikes are disturbances due to friction.*

5.3 - COMPLIANT MOTION USING THE LSS

The hybrid control algorithm was implemented on the Adept-3 robot with the algorithm given in section 4.6.1. There were two main problems encountered with the algorithm. The first problem simply involved adjusting the gains of the control loops to achieve good performance. The second problem was how to handle cases where the crack moved out of the field of view of the LSS.

The control laws used for hybrid control are digital implementations of PID control. One difficulty of implementing a digital control law in this case is the fact that the sampling rate isn't accurately known. Therefore, an estimated time step is used which is approximately the time it takes for the manipulator to make a single point-to-point move in the algorithm. The difference between the actual and estimated move time varies with each move.

Stability is an important criteria for the algorithm, since overshoots can cause the

crack to move out the field of view of the local sensor. Originally, both the position and orientation control loops used digitally implemented PID compensators. Better performance was obtained when the position control loop was reduced to a PD compensator since this tends to make the system more stable. The orientation control loop was left as a PID compensator since the estimated orientation tends to be a very noisy signal (see Figure 5.7).

Tests with the Adept-3 were also the basis for adding the sensor saturation features and the low-pass filter features to the LSS (see sections 3.3.2 and 3.3.3).

The tests for crack following were run on cracks routed in plywood. Data from tests of crack following with the LSS and the Adept-3 are given in Figures 5.6 and 5.7.

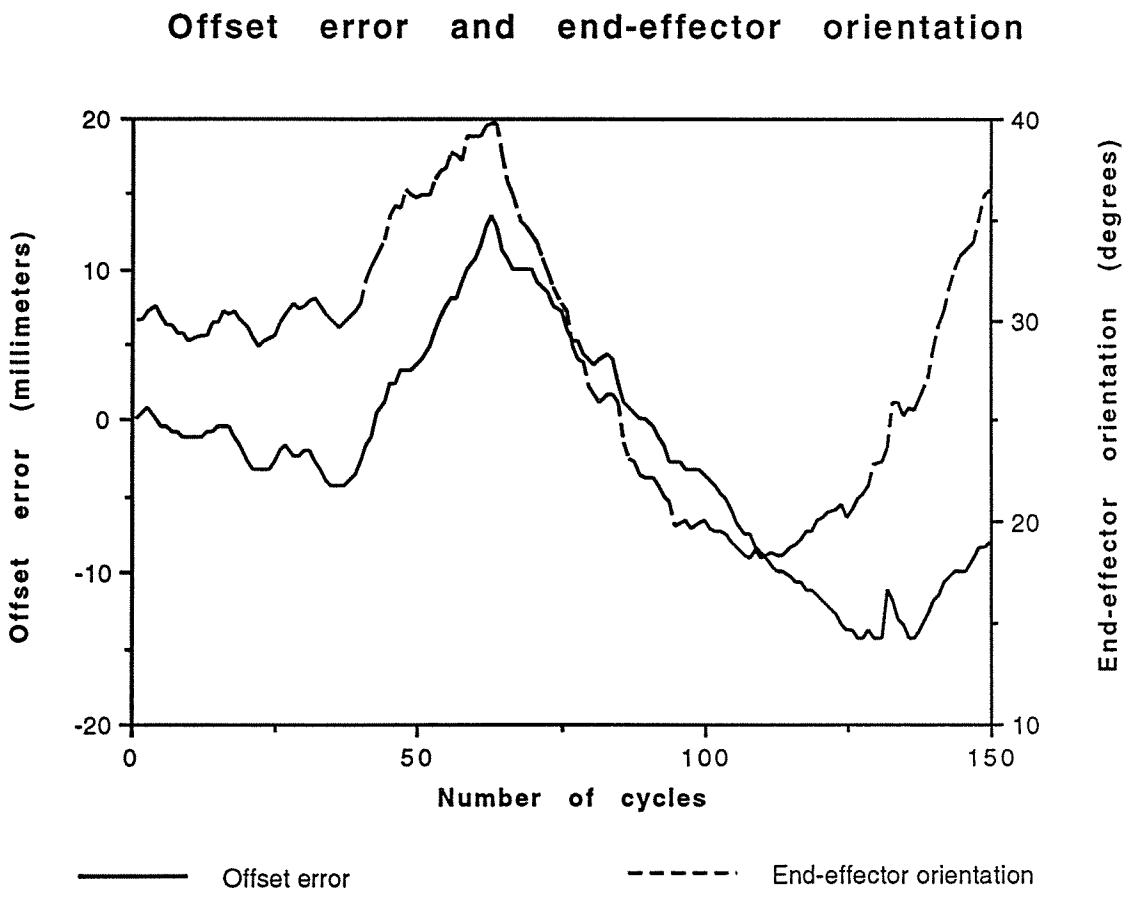


Figure 5.6 *Offset error and end effector orientation for crack following with the Adept 3. The graphs illustrate how the orientation is adjusted to follow the direction of travel of the end-effector.*

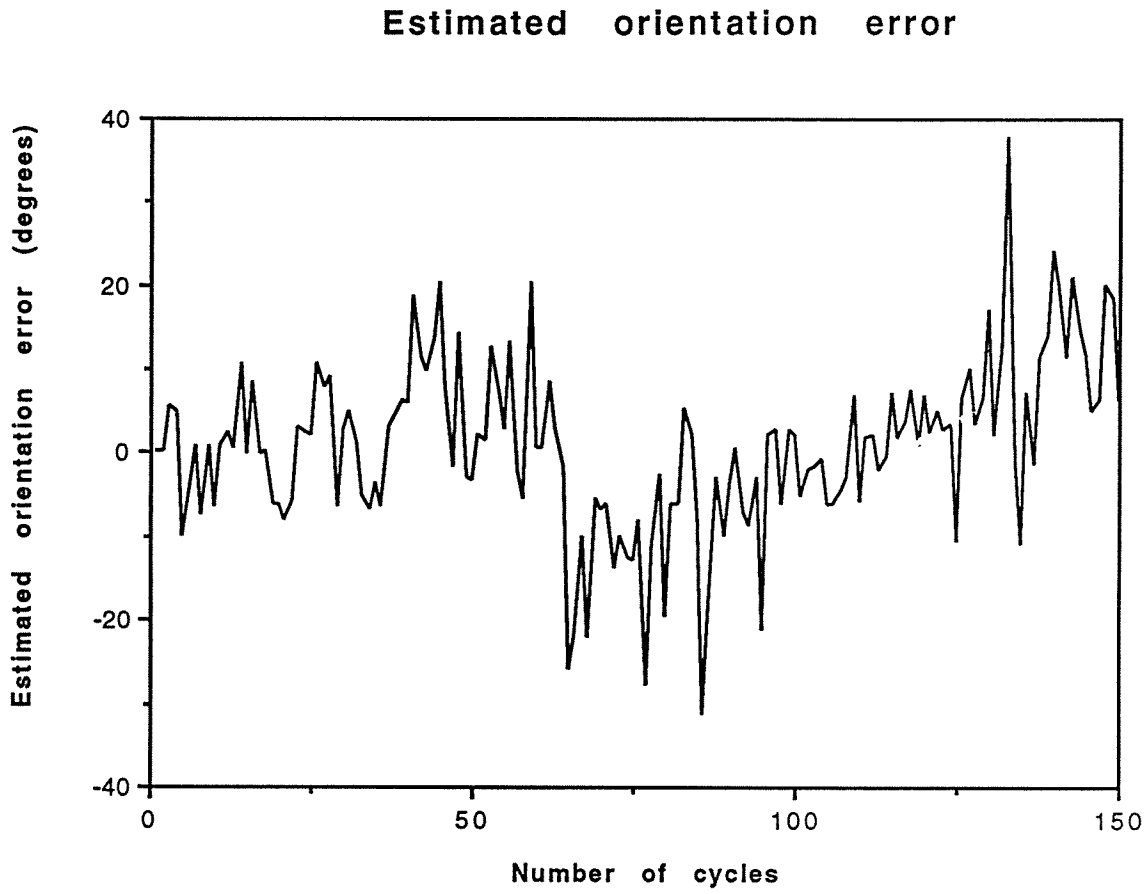


Figure 5.7 *Estimated orientation error for crack following using the Adept-3.*

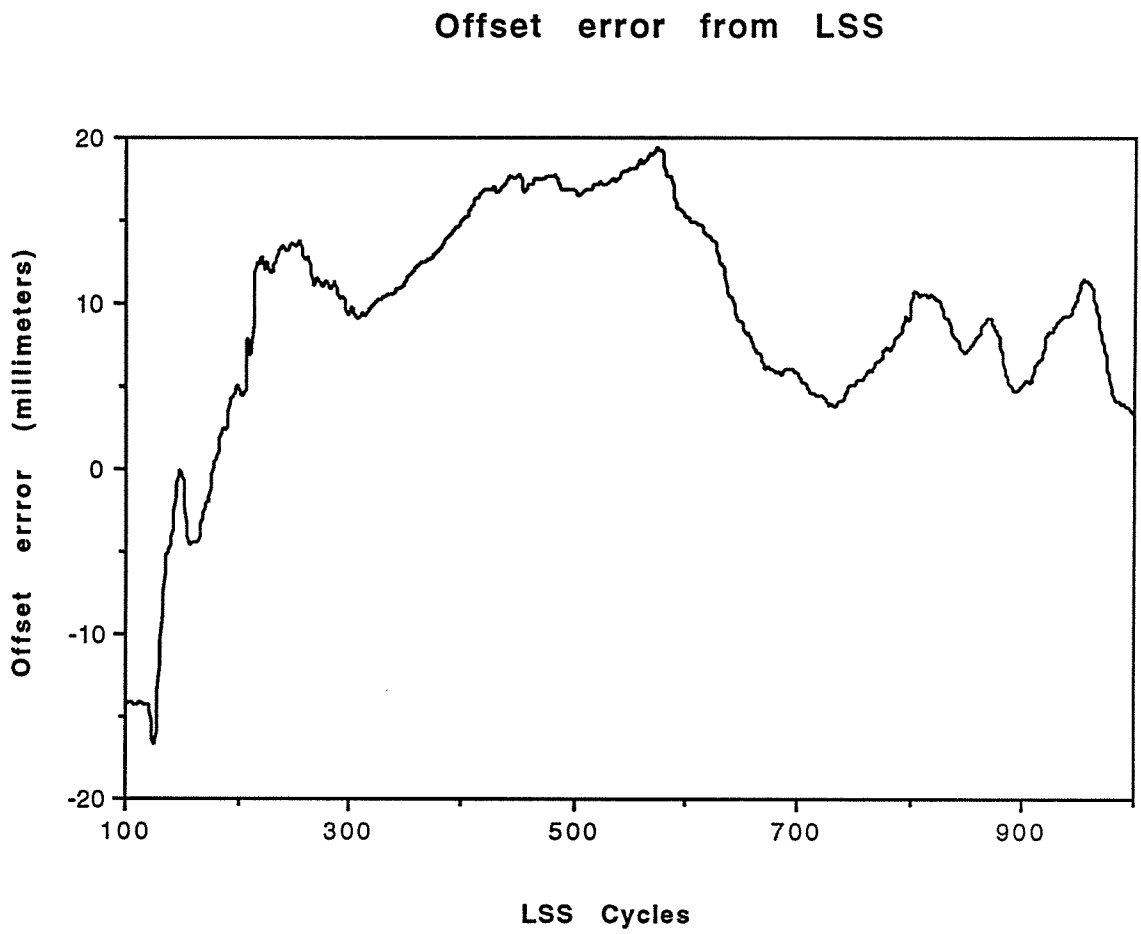


Figure 5.8 *Local sensor output for crack following using the GMF A-510 manipulator.*

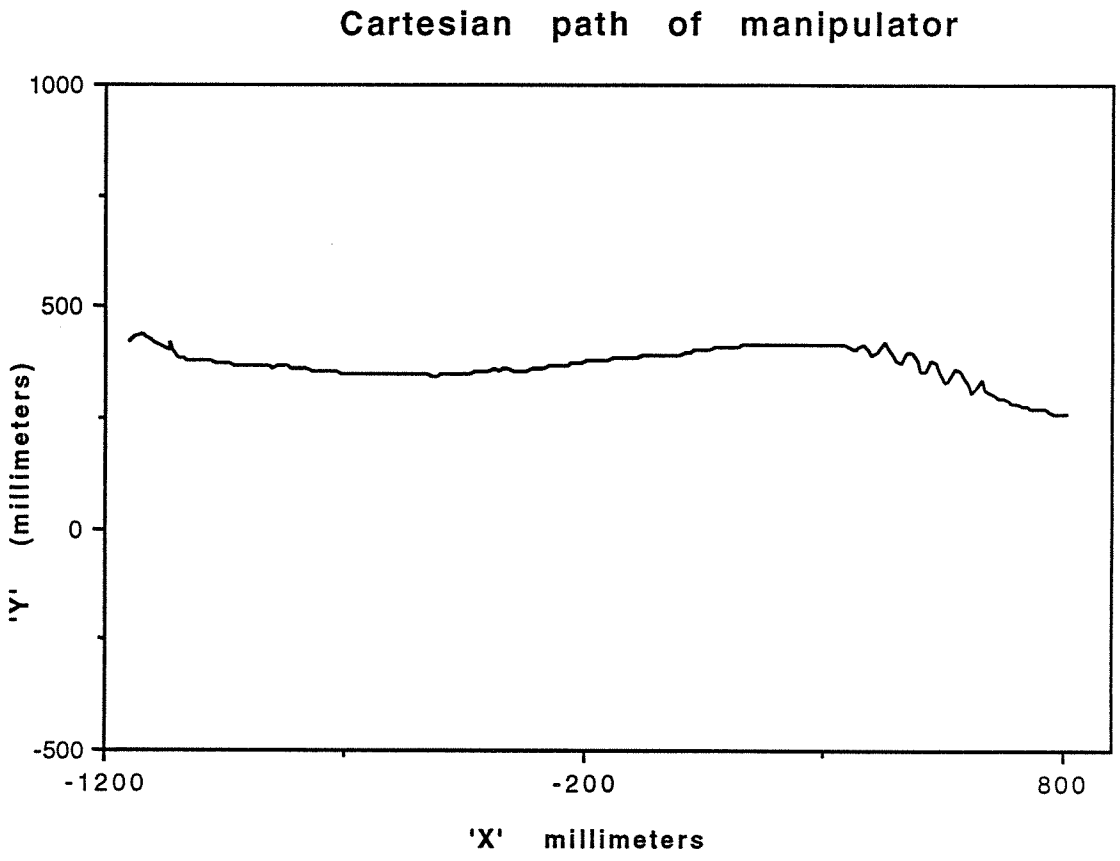


Figure 5.9 *Cartesian path of manipulator. for crack following with the GMF A-510. The starting point is at the right-hand side of the graph. Some oscillation was present in end-effector motion.*

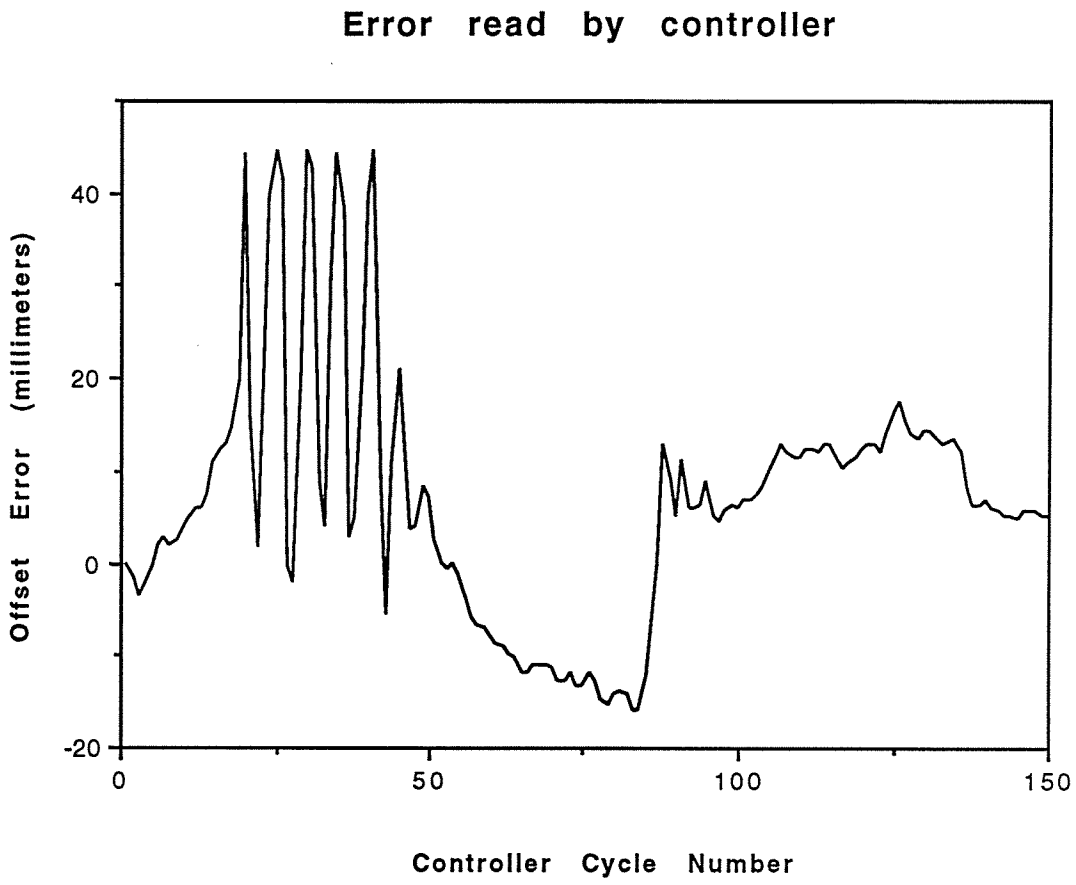


Figure 5.10 *The LSS error as read by the controller for the same run as shown in figure 5.9. Oscillations in position can be seen in the offset error.*

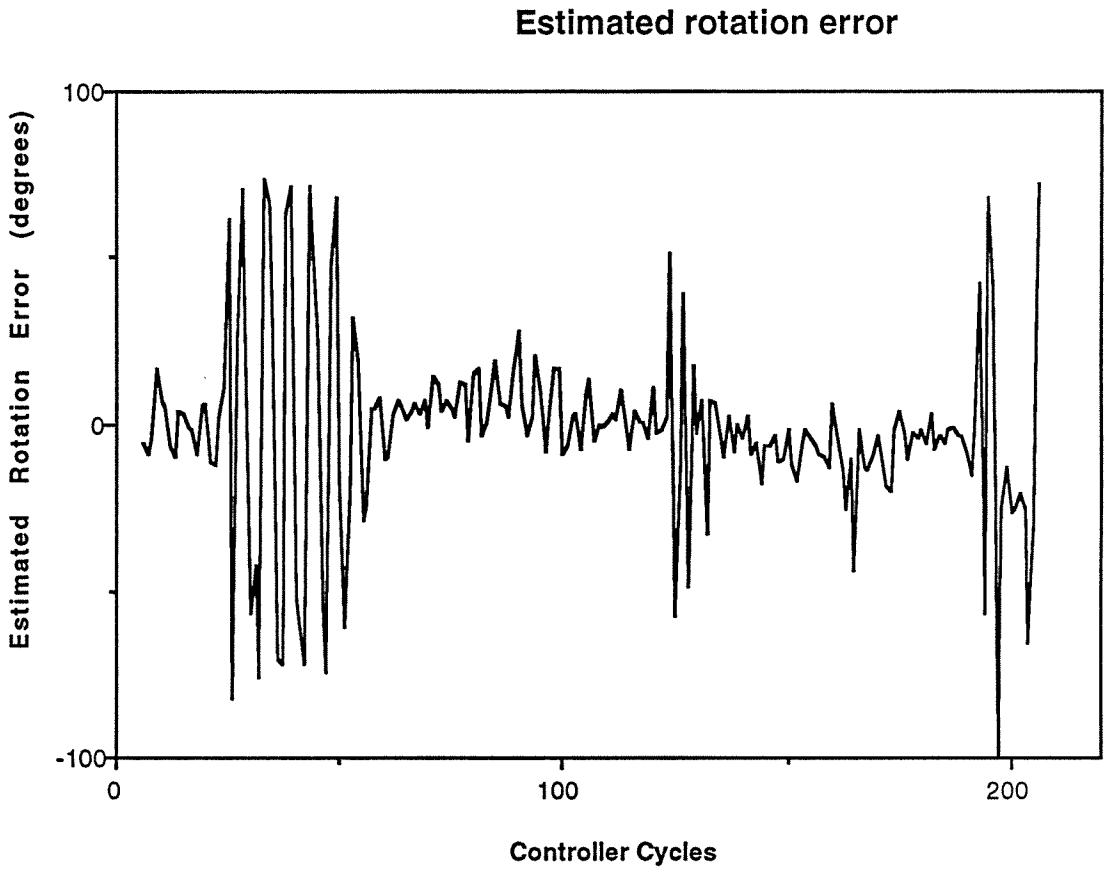


Figure 5.11 *Estimated rotation error for same run as above. The oscillations in orientation error show the coupling effects between position and orientation control.*

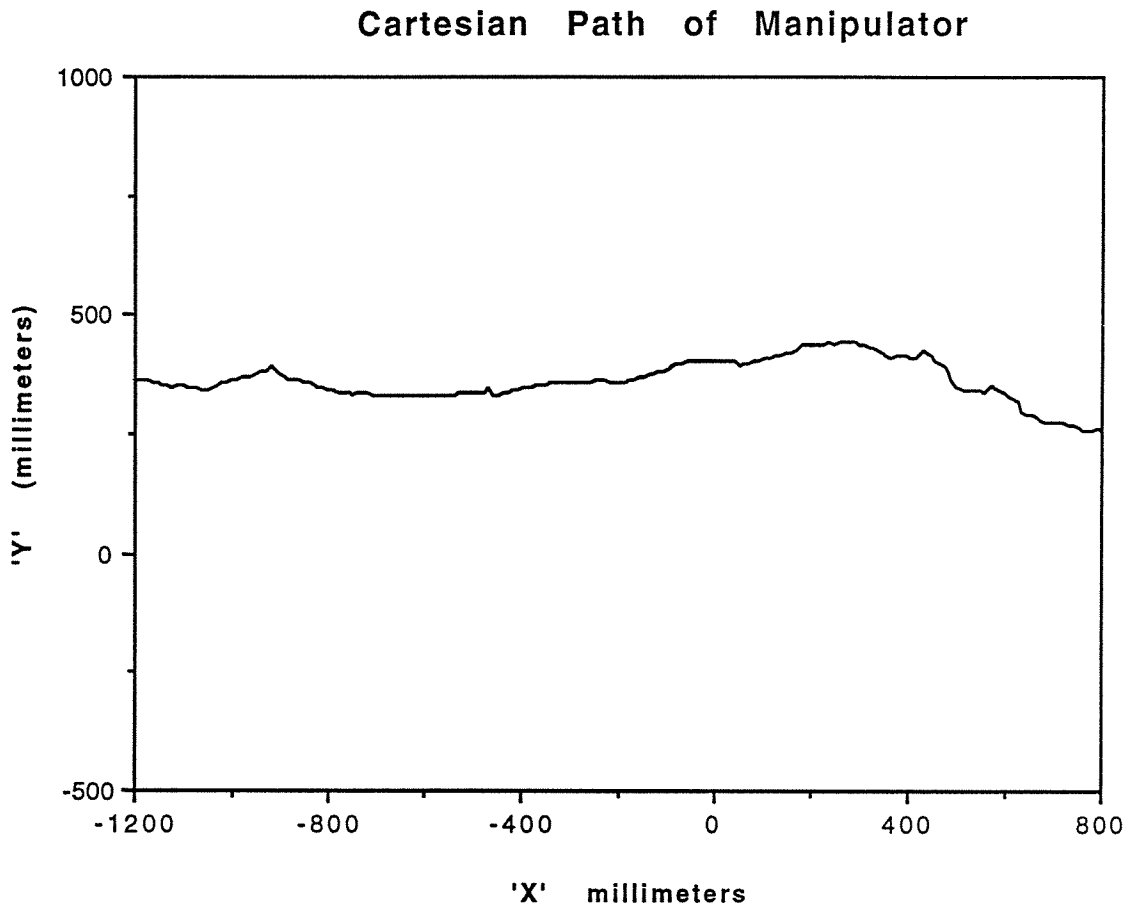


Figure 5.12 *Cartesian path of manipulator for same crack used in Figure 5.11 The speed of the manipulator was reduced to obtain a more stable response. The starting point was near (800,200).*

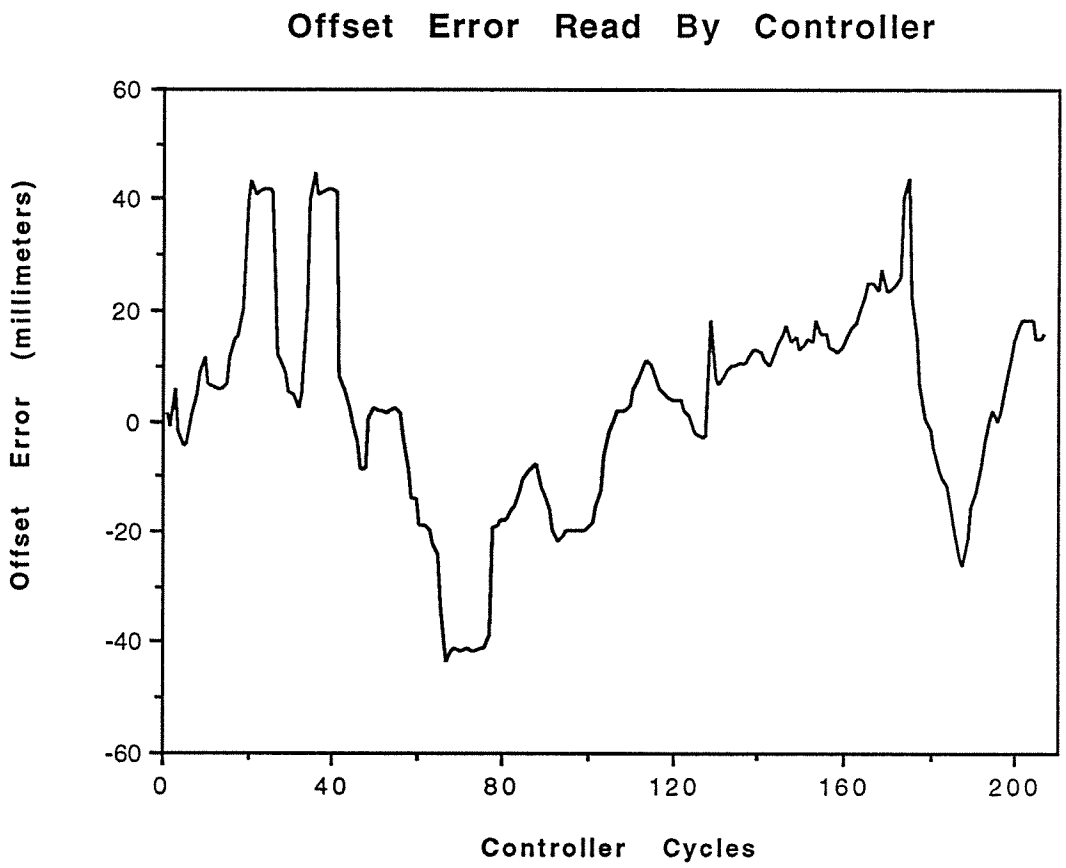


Figure 5.13 *Offset error from LSS read by controller for same run as above.*

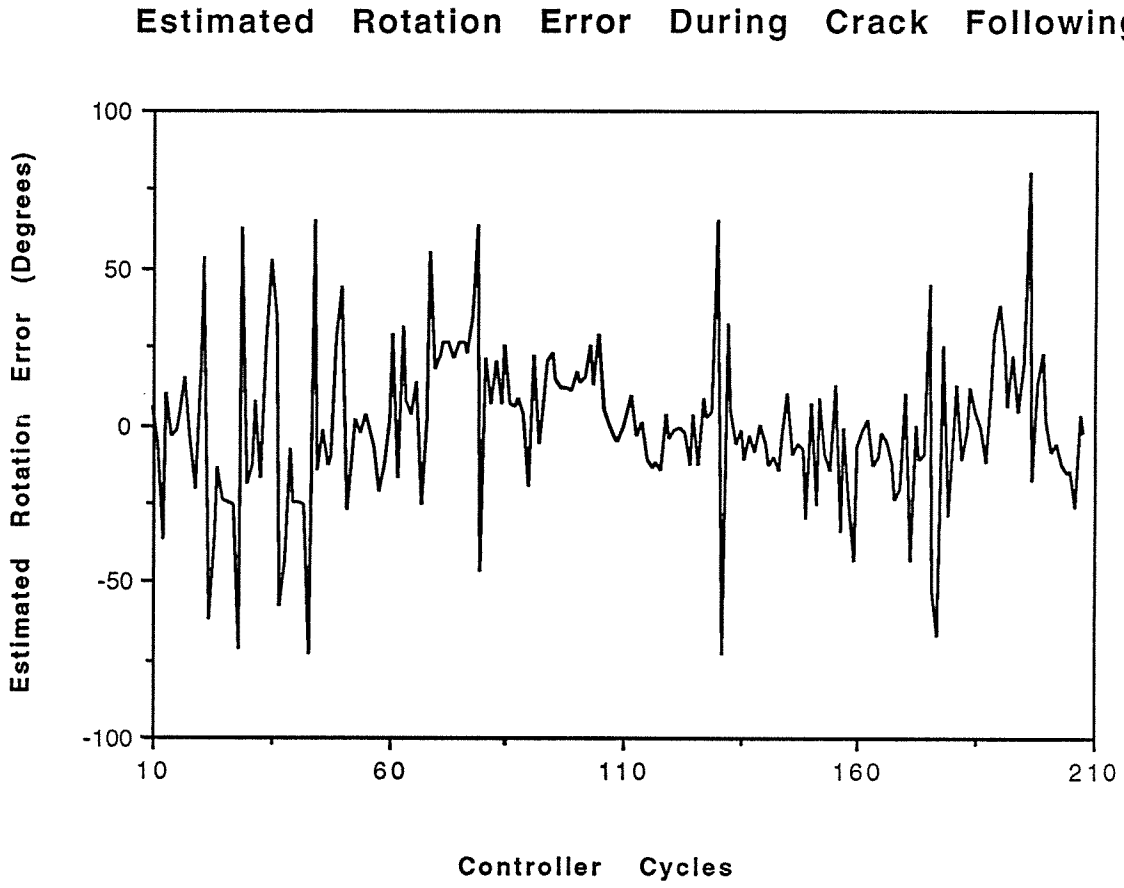


Figure 5.14 Estimated rotation error for same run as above.

5.4 - SUMMARY AND CONCLUSIONS

This chapter has presented results from implementing hybrid control using a computer simulation, force sensing and relative proximity sensing. Two different industrial robots were used. The most important result is that hybrid control using the LSS can be used for on-line control in crack following for the ACSM. The largest problems encountered were maintaining stable control loops while using an uncertain time step in digital control law and the limited field of view of the LSS. In order to maintain stable control loops the gains must be adjusted so the system response is heavily damped. The field of view of the local sensor is effectively increased by simulating sensor saturation in software.

CHAPTER 6 - CONCLUSIONS AND FUTURE WORK

This paper has developed algorithms for crack following using an industrial robot and a relative proximity sensor. The algorithm is successful in following cracks of similar geometry to those which will be addressed by the ACSM. Limitations on system performance are largely a function of the capabilities of industrial controllers (in this case, the Karel controller) and the limited field of view of the LSS. Also, since the LSS cannot directly measure crack direction, it is very difficult to follow cracks which make sharp changes in direction (> 45°).

6.1 - RECOMMENDATIONS

6.1.1 - CUSTOM HARDWARE

The GMF-A510 manipulator was selected for use on this project because it was determined to be the best commercially available manipulator for the task. Time and cost constraints eliminated the use of a custom manipulator for the ACSM. For a commercialized version of the ACSM prototype, it would be desirable to have a manipulator that had the ability to reach of full 12 feet of highway lane from an eight foot truck bed. The manipulator should also have a high payload capacity to handle dynamic loads caused by accelerating the mass of the process equipment. Since no commercially available manipulator meets the above specifications, it will be probably be necessary to build a custom manipulator. The custom manipulator should have full dexterity in the plane of the road. The manipulator should be easily controllable in two-dimensions in order to assure that on-line Cartesian path planning can be implemented.

There are several robotics companies in the United States who make custom manipulators. Two of these include Schilling Development of Davis, California and Odedics, Inc. of Anaheim California. Odedics has designed and built a SCARA robot for military use. It has a payload of 300 lbs, a reach of 8 feet and a top speed of 50

inches/sec. The manipulator has a weight of 350 lbs. Estimated price as of 1991 is $175,000.

Schilling Development produces a titanium, hydraulic-driven manipulator which has a 6 foot reach. The manipulator has a 200 lb. payload and a maximum speed of 3 feet/sec. Due to the fact that this is a hydraulic manipulator, it cannot be servo-controlled like many industrial manipulators. Estimated 1991 price for the manipulator is $105,000.

The controller for a custom manipulator can be a controller provided by the manipulator manufacturer, a custom controller or an available industrial robot controller that can be ported over to control a different type of manipulator. Adept Technology of San Jose, California produces both manipulators and controllers. Adept will port their controller to any manipulator that can be servo-controlled (.i.e.. is run by electric motors).

6.1.2 - LINEAR SLIDE INTEGRATION

It may be desirable to mount the manipulator on a linear slide or similar device to increase the workspace. The addition of the linear slide will add a redundant degree-of-freedom. On-line control of a manipulator with a redundant degree-of-freedom represents a serious problem. The integration of the linear slide and the A-510 manipulator was done assuming the path of the cracks to be relatively simple (see Appendix E). The control problem for a generalized path in two dimensions is much more complicated. The complexity of the redundant degree-of-freedom integration problem will depend largely on the types of crack paths to be addressed by the commercialized ACSM. If the cracks are mostly transverse in direction the problem will be relatively simply. If a meandering crack is to be followed the problem will be very complex.

6.1.3 - HARDWARE CONFIGURATION

Presently, the A-510 manipulator is mounted on the linear slide such that the motion of the linear slide is parallel to the 'x' axis of the manipulator. The workspace of the system would be enhanced if the manipulator were rotated 90° with respect to the slide

such that the linear slide motion is parallel to the 'y' axis of the manipulator. This would allow the linear slide motion to account for the joint stops in the rear of the manipulator workspace.

6.1.4 - PROCESS EQUIPMENT DESIGN

The weight and geometry of process equipment for the crack sealing process affects the motion capabilities of the manipulator. As the process equipment becomes larger and heavier the speed at which the manipulator can move will become slower. Also, the more inertia due to the process equipment the more difficult it will be to accurately control the end-effector. Optimization of the crack sealing machine must take into account the fact that the amount of process equipment present will greatly affect the efficiency of moving that equipment with a manipulator.

6. 1.5 - TUNING FOR THE ACSM

Adjusting the gains for the hybrid control loop on the robot controller will be an ongoing process throughout development and testing of the ACSM prototype. Other parameters that will have to be adjusted that affect system dynamics are the specified velocities accelerations for the manipulator and the linear slide. The filter order and cut-off frequency on the LSS will also have to be adjusted according to crack geometry and end-effector speed. The external factors that will affect the system dynamics are end-effector loading and crack geometry. The addition of process equipment will add a large inertial element to the system. The control gains must be adjusted to obtain good performance with the payloads present. The speed at which the end-effector moves over cracks and the nature of the cracks being sealed will determine the frequency content of the output of the LSS. This frequency content should be analyzed to determine desirable filter characteristics for the LSS low-pass filter.

6.2 - TUNING AND CONTROL ALGORITHMS

Determining the optimal control laws and gains for the hybrid control algorithm is a difficult task. The difficulty is primarily due to the fact that most industrial controllers

perform point-to-point moves that cannot be updated once the move command has been issued. If a system cannot be updated continuously or at least at known time intervals, then most control theory does not apply.

Val II, a programming language used by Unimation, Inc. has the capability to update end-effector trajectories while a move command is being executed. A command 'ALTER' can be used to change the present position of the end-effector as well as the destination position. The ALTER command can take position changes in terms of incremental changes in Cartesian coordinates in 28 ms cycles. ALTER is an extremely useful tool for real time path control of a robot (Loughlin, 1983). Unfortunately, neither GMF nor Adept has an equivalent of the ALTER command and Unimation no longer makes robot controllers. However, if a controller with a command similar to ALTER can be found, it could greatly improve the process of on-line control.

Even with the limitations of an industrial controller, it may be possible to determine an optimal range of gains for the hybrid control algorithm by estimating the time step. The average value of the time for each move will depend on the specified end-effector velocity and acceleration, the tangential direction velocity gain and end-effector loading. If the average value of each move time can be found (perhaps by measuring move-times with a given set of parameters) and if the move time does not vary to far from the average, it is possible to model the system as a discrete-time system with a sampler and zero-order hold connected to the plant. The input from the sensor can also be treated as a sampled signal. Difficulties will arise from determining an exact model for the robot-controller system comprising the plant. It is possible to obtain an approximate model of the plant by empirically measuring plant dynamics.. The control problem will be further complicated by the fact that the local sensor sampling rate will be faster than the robot cycle rate and a multi-rate system will result. In short, modeling the hybrid control algorithm using discrete control theory would be a difficult and tedious process and the quality of the results will be questionable.

6.2.1 - GEOMETRIC BASED ORIENTATION CONTROL

It is possible to control the orientation of the end effector by using the LSS and incorporating geometric based (rather than dynamic based) control. For the dynamic control scheme, the orientation of the crack is calculated by taking a secant line through the previous two points. The difference between this estimated orientation and the actual end-effector orientation is used to generate an error signal which is filtered through a digital proportional plus integral (PI) compensator. A difficulty arises from selecting the dynamics of the PI compensator because the time step is invariant and unknown.

The position of the end-effector can be found at any time along with the local sensor reading at any time. This data gives us fairly accurate information about the past positions of the crack. The true orientation of the crack can be found by looking at past points, creating a curve based on the past points and determining the calculated direction of the curve at the last end-effector orientation. This method was implemented in section 4.7.3. by using a linear regression of the previous five crack points to find the crack direction. This allows for orientation control without the need for a known time step.

It is possible to expand this methodology and to fit higher order functions to previous crack points. Potential problems with the control algorithm may arise from manipulator dynamics which are not taken into account. If the dynamics for the degree-of-freedom being controlled has a small time constant compared to the external control loops, it may be safe to ignore dynamics. It may also be possible to implement geometric based control for position control of the manipulator, but position dynamics are likely to have a much larger effect than orientation dynamics. One other factor affecting geometric control is the fact that the LSS data may be older than its corresponding end-effector position data. This would introduce a time lag effect into the geometric control scheme.

The suggestions made in this chapter are intended to improve the performance of controlling a manipulator for the ACSM. Many of the suggestions will hopefully be

useful in any application involving hybrid control with an industrial robot.

APPENDICES

APPENDIX A - CODE FOR SIMULATION OF HYBRID CONTROL WITH MATLAB.

```
%this program will simulate hybrid control for crack-
%following.
%the robot must be given a starting location and orientation
%for the crack in global coordinates.
%  take input for starting parameters:
x1(1,1)=input('starting global x');
x1(1,2)=input('starting global y');
%  input for starting initial orientation (in radians)
f(1)=input('starting orientation, rads');
%an estimate of the robot-controller system is given in the
%form of a transfer function.  The numerator coefficients are
%given in num and the denominator coefficients in den.
num=36864;
den=[1 105.6 36864];
%convert the transfer fucnction to state-space form.
[a,b,c,d]=tf2ss(num,den);
% specify the state-space output matrix c:
c=[0 1];
%specify the tangential velocity gain, v.
v=1000000;
%initialize variables to be used
e=0;
g=0;
k=1;
dp(1)=0;
dp(2)=0;
% specify the time step for the sampling rate, dt:
dt=.02;
g(1)=f(1);
%move in the specified orientation with velocity gain v
%for duration of time step dt.
% vel is a function that returns and x and y position, p(2x1)
% and an x and y velocity, dp(2X1) for a given starting
% position X(k) (2X1) and direction f(k) (rads)
% the state space matrices time step and initial velocities
% and positions are also required inputs.
[p,dp]=vel(a,b,c,d,dt,v,x1,f(k),k,dp);
k=k+1
x1(k,1)=p(1);
x1(k,2)=p(2);
%check distance between end effector and crack function
%given by err
l=lerr(x1,f,k);
e(2)=l;
e(1)=0;
%begin loop. end of crack will be signalled when err=128
while e(k)<128
 %obtain gain for velocity in force direction by applying
```

```
 %PID control law contained in gain(e,k,dt).
 fg=10000*gain(e,k,dt);
 %move with specified velocity in force direction for time
 %step dt.
 %fvel is same as vel except that the ee moves perpendicular
 %to the crack with a direction and magnitude determined
 %by the force-velocity gain fg.
 [p,dp]=fvel(a,b,c,d,dt,fg,x1,f,k,dp);
 k=k+1;
 x1(k,1)=p(1);
 x1(k,2)=p(2);
 %calculate orientation of end effector
 f=ro(x1,k);
 g(k)=f;
 %move in velocity direction
 [p,dp]=vel(a,b,c,d,dt,v,x1,f,k,dp);
 k=k+1;
 x1(k,1)=p(1);
 x1(k,2)=p(2);
%take sensor reading and go to beginning of loop.
 e(k)=lerr(x1,f,k);
end
%x,y locations at each time step are contained in matrix x.
%g contains the orientation for all odd values of k.
'end'

function fe=err(x,k,dt)
%simulates sensor output
%returns magnitude based on present position x
%and desired position function xd
%Use this routine for FORCE sensing.
x1=x(k,1)
y1=x(k,2)
y2=xd(x1)
x3=ixd(y1)
a=y1-y2
b=x1-x2
c=(a^2 + b^2)^.5
err=(a/b)*c
end

function fe=err(x,k)
%simulates sensor output
%returns magnitude based on present position x
%and desired position function xd
%  returns end of crack flag for specified value of k.
if k==100
 fe=128;
else
%calculates distance between end effector present
%location and crack, along normal to crack.
%Use this routine for FORCE sensing.
x1=x(k,1);
y1=x(k,2);
```

```
y2=xd(x1);
x3=ixd(y1);
a=y1-y2;
b=x3-x1;
c=(a^2 + b^2)^.5;
if b==0
fe=0;
else
fe=-a*sin(atan(b/a));
end
end
end


function l=lerr(x,f,k)
%simulates visual line scanner output
%gives position of crack function relative to sensor
%with position x(k) and orientation f.
% if f=0 then the line of the sensor is not an explicit
% function y=mx + b
x1=x(k,1);
if abs(f)<.005;
 x2=x1;
 y2=xd(x2);
end
% else express sensor line as y=mx + b
y1=x(k,2);
m=tan(1.5708+f);
b=y1-m*x1;
% (x1,y1) is location of center of sensor
% (x2,y2) is where sensor line intersects crack function
% solve for x2, y=f(x) must be known explicitly to solve
% and get the following equation.
x2=b/(1-m);
y2=xd(x2);
% sign convention: left side -, right side +
s=sign(xd(x1)-y1);
% distance formula
d=((y1-y2)^2+(x1-x2)^2)^.5;
l=s*d;
if k==100;
l=128;
end
end


function fo=force(x,xd,k,dt)
normal(1,1)=0
normal(1,2)=-1
normal(2,1)=1
normal(2,2)=0
dx=xd(k,1)-x(k,1)
dy=xd(k,2)-x(k,2)
d=(dx*dx+dy*dy)^.5
```

```
delx(1)=dx/d
delx(2)=dy/d
xn=gain(x,xd,k,t)*normal*error(x)*delx
fo=x+xn
end


function [p,dp]=fvel(a,b,c,d,dt,v,x,f,k,x0)
%returns position (2x1) and final condition (2x1)
%final condition =[dx dy]'
%for given velocity magnitude (v), direction (f)
%and starting position x(k) after time response dt.
%x0(2x1) contains the starting time derivatives for x and y.
%the new x location is given in p(1)
%new y locatin is given in p(2)
%new time derivatives of x and y are contained in vector dp (2x1)
%calculate gains in x and y for motion normal to the crack.
gx=-sin(f)*v;
gy=cos(f)*v;
t=0:dt/10:dt;
ic(2)=0;
ic(1)=x0(1);
% calculate points for move in x-direction
vx=lsim(a,b,c,d,gx*t,t',ic');
% specify initial conditions of states
ic(2)=0;
ic(1)=x0(2);
% calculate points for move in y-direction
vy=lsim(a,b,c,d,gy*t,t',ic');
%add old position to final values of vx and vy to calculate
%new values for x and y
p(1)=vx(11)+x(k,1);
p(2)=vy(11)+x(k,2);
%calculate time derivatives for x and y.
dp(1)=(vx(11)-vx(10))/(dt/10);
dp(2)=(vy(11)-vy(10))/(dt/10);
end


function ga=gain(e,k,dt)
%calculates the velocity gain for the force directions by using a
%PID control law applied to the position error given by e(k).
%integral, derivative and proportional gains are given by
%ki, kd and kp, respectively.
ki=10;
kd=.0;
kp=20;
%no error at time 0 when k=1 so g(k)=0
gi(1)=0;
e(1)=0;
if k==2
 %integrate between previous and present time step.
 %multiply by error and ki for case k=2.
 gi(k)=(ki*dt*.5)*e(k);
```

```
 gd=kd*e(k)/dt;
 else
 % if k>2, calculate gains:
e(k-1)=0;
gi(k-1)=0;
%calculate integral gain
gi(k)=(ki*dt*0.5)*(e(k)-e(k-2))+gi(k-2);
%calculated derivative gain
gd=kd*(e(k)-e(k-2))/dt;
end
%calculate proportional gain
gp=kp*e(k);
%total gains
ga=gi(k)+gd+gp;
end

function [p,dp]=vel(a,b,c,d,dt,v,x,f,k,x0)
%returns position (2x1) and final condition (2x1)
%final condition =[dx dy]'
%for given velocity magnitude (v), direction (f)
%and starting position x(k) after time response dt.
%x0(2x1) contains the starting time derivatives for x and y.
%the new x location is given in p(1)
%new y locatin is given in p(2)
%new time derivatives of x and y are contained in vector dp (2x1)
%calculate gains in x and y for motion parallel to crack.
gx=cos(f)*v;
gy=sin(f)*v;
t=0:dt/10:dt;
%set initial conditions for states:
ic(2)=0;
ic(1)=x0(1);
%calculates points for move in x-direction.
vx=lsim(a,b,c,d,gx*t,t',ic');
%set initial conditions for states:
ic(2)=0;
ic(1)=x0(2);
%calculate points for move in y-direction.
vy=lsim(a,b,c,d,gy*t,t',ic');
%add old positions to final values of vx and vy to calculate new
%values for x and y
p(1)=vx(11)+x(k,1);
p(2)=vy(11)+x(k,2);
%calculate time derivatives for x and y.
dp(1)=(vx(11)-vx(10))/(dt/10);
dp(2)=(vy(11)-vy(10))/(dt/10);
end


function a=xd(x)
%crack function
%gives y as a function of x.
%the crack is a line of the form Y=X.
%if x<=8
```

```
a=x;
%else
%a=4+.5*x;
%end


function a=ixd(x)
%inverse crack function
%(x as a function of y)
%the crack is a line of the form Y=X
%if x<=8
a=x;
%else
%a=2*(x-4);
%end


function f=ro(x,k)
%calculates the orientation of the end effector to be
%aligned with the direction of travel.  Assumes no
%no dynamics for end effector orientations.
%dx=x(k,1)-x(k-2,1);
%dy=x(k,2)-x(k-2,2);
%f=atan(dy/dx);
f=.754;
end
```

APPENDIX B - FORCE CONTROL WITH THE ADEPT-3 ROBOT

```
.PROGRAM fs_main()
                ;Main program for compliant motion
                ;with force control
                LOCAL point[], n, vg, dt, loc[], loc1, stat
                i = 1
                n = 10000
                vg = 0
                phi0 = 45
                dt = 0.2
                mag0 = 1
                CALL fs.initialize(stat) I
                F stat < 0 GOTO 50
                CALL init_guard()
                out[0] = 0
                DEVICE (2, 0, status, fs.ena.guard, 1) out[]
;               DEVICE (2, 0, status, fs.set.zero)

                    ACCEL 80, 80

                    SPEED 70 ALWAYS

                    err[i] = 0

                    phi[i] = phi0

                    CALL gen_point.sub(point[i], dt, vg)

;               DECOMPOSE loc[1] = point[i]

;               TYPE loc[1], loc[2], loc[3], loc[6]

                    MOVE point[i]

                    i = i+1

                    WHILE i < n DO

CALL read_fs()
                     CALL gen_point.sub(point[i], dt, vg)
                       MOVE point[i]
                       i = i+1
;       TYPE i
        END

        50 IF stat < 0 THEN
                TYPE "error ocurred while initializing force sensor"
                END
END .
```

```
PROGRAM gen_point.sub(point, dt, vg)
        LOCAL newx, newy, fg, x1, y1, z, loc.now, theta, x, tw, p
        CALL for_law.sub(err[], i, dt, fg)
        HERE loc.now
        DECOMPOSE x[1] = loc.now
        x1 = x[1]
        y1 = x[2]

            z = x[3]

            p = x[6]

        tw = 180-p-phi[i]

            newx = x1+vg*SIN(tw)+fg*COS(tw)

            newy = y1+(-1)*vg*COS(tw)+fg*SIN(tw)

            SET point = TRANS(newx,newy,z,0,180,45)

            RETURN


.END .


PROGRAM init_guard()
        LOCAL mode, ft, dim[], low[], up[], vec[,], status
        mode = 1
        ft = 0
        dim[0] = 3
        dim[1] = 0
        dim[2] = 0
        dim[3] = 0
        low[0] = -15
        up[0] = 15
        vec[0,0] = 0
        CALL fs.config(mode, ft, dim[], vec[,], low[], up[], status)
        IF status < 0 THEN
        TYPE "error configuring guarded move"
        END
        RETURN
.END .



PROGRAM read_fs()
        LOCAL in[], status, mag DEVICE (2, 0, status,fs.get.force) , in[]

            IF status < 0 GOTO 100
```

```
                mag = SQRT(in[0]*in[0]+in[1]*in[1])

                err[i] = mag-mag0

                phi[i] = ATAN2(in[1],in[0])

    100         IF status < 0 THEN

                TYPE "force system error"
                END
                RETURN
.END .
.END .

PROGRAM set_frf()
                LOCAL start.loc, pt[], i, status
                HERE start.loc
                DECOMPOSE pt[1] = start.loc
                 SET out.t = TRANS(pt[1],pt[2],pt[3],pt[4],pt[5],pt[6])
                DEVICE (2, 0, status, fs.set.frf, 0, 1) , , out.t
                RETURN
.END

PROGRAM for_law.sub(err[], i, dt, fg)
                LOCAL ki, kp, kd, gd, gp, gi[]
                ki = -0.1
                kp = -3
                kd = 0
                gi[1] = 0
                err[1] = 0
                IF i == 1 THEN
                        gi[i] = ki*dt*0.5
                         gd = kd*err[i]/dt
                ELSE
                        gi[i] = (ki*dt*0.5)*(err[i]+err[i-1])+gi[i-1]
                        gd = kd*(err[i]-err[i-1])/dt
                END
                gp = kp*err[i]
                fg = gi[i]+gd+gp
RETURN
.END
```

```
.PROGRAM a.user_fs()

; Non-AIM user initialization and utility routines for Force Sensing Module.

;* Version V1.8

;* Copyright (c) 1989 by Adept Technology, Inc.

;*************************************************************************
;* *
;* The information set forth in this document is the property *
;* of Adept Technology, Inc. and is to be held in trust and *
;* confidence. Publication, duplication, disclosure, or use *
;* for any purpose not expressly authorized by Adept Tech- *
;* nology in writing is prohibited. *
;* *
;* The information in this document is subject to change *
;* without notice and should not be construed as a commitment *
;* by Adept Technology. *
;* *
;* Adept Technology makes no warranty as to the suitability *
;* of this material for use by the recipient, and assumes no *
;* responsibility for any consequences resulting from such use. *
;* *
;*************************************************************************

; DESCRIPTION:
;
; This file contains Force Sensing Module routines for use in
; non-AIM applications. It contains initialization code that
; should be used to define global variables ("fs.initialize"), a
; routine to convert force system error messages to string form
; ("fs.error"), and several routines as examples of use of the
; force system.
;
; SPECIAL INSTRUCTIONS:
;
; This package requires V+ version 8.2 or above, with the stop-
; on-force option.
;
; The user should not directly edit the routines within this
; package. It is recommended that an archive copy be kept, and
; that the user modify a special copy of his own. For example,
; to copy this to a hard disk and make a user-modifiable copy
; (named FSM.V2), the following monitor commands may be issued:
;
; FCOPY C:USER_FS.V2 = A:USER_FS.V2
;Assuming FSM disk in drive A
; FCOPY C:FSM.V2 = C:USER_FS.V2
;Make customizable copy
;
; GLOBAL ENTRY POINTS:
;
; fs.auto.offset Clear force readings by offsetting current forces
```

```
; fs.buffer Enable/disable force signature buffer
; fs.config Configure guarded/monitor mode
; fs.error Determine detailed force error code and string
; fs.initialize Initialize global variables and force system
; fs.read.buffer Read force buffer data
;
; AUTHOR: John A. Tenney (890601)
;
; CHANGES:
;

                RETURN

; The CALLs below are never executed. They are used only to cause the
; programs included with this header to be saved with the command:
;
; STOREP/2 USER_FS.V2 = a.user_fs
;
; Programs added to this package should be included in the list.

                CALL    fs.auto.offset()

                CALL    fs.buffer()

                CALL    fs.config()

                CALL    fs.error()

                CALL    fs.initialize()

                CALL    fs.read.buffer()

                .END .

.END .


.PROGRAM fs.auto.offset(stat)

; ABSTRACT: Force system utility to offset current force level.
;
; This routine can be used before beginning a force-controlled
; operation to compensate for any force drift, weight of end-
; effector, or any other contribution to off-zero force readings.
; It zeros the current force readings by setting a force offset
; equal to the opposite of the current force readings. This
; offset can be removed after the end of the operation by
; resetting the offsets to zero.
;
; This routine performs a different function from that of opcode
; "fs.set.zero". This routine creates a temporary "zero" level
; using one force reading, while "fs.set.zero" sets a more reliable
```

```
; and permanent "zero" level by averaging many force readings.
;
; The advantages to "auto-offset" are that it is fast and non-
; permanent. Its speed allows it to be performed without slowing
; down an operation, so as to cancel end-effector weight or sensor
; drift while in motion. Being non-permanent, one can use a force
; offset in one part of an operation and cancel it later.
;
; INPUT PARM: None
;
; OUTPUT PARM: stat = Status of operation. Standard V+ error code.
;
; SIDE EFFECTS: Changes force readings until offset is removed.
;
; MISC: Identical copies of this routine exist in the files
; LIB_FS.V2 and USER_FS.V2.
;
;* Copyright (c) 1989 by Adept Technology, Inc.

            AUTO force[5], i, offset[5]

; Read current offset levels.

            DEVICE (2, 0, stat, fs.get.offset) , offset[]
            IF stat > 0 THEN

; Read current force levels.

                  DEVICE (2, 0, stat, fs.get.force) , force[]
                  IF stat > 0 THEN

; Subtract current offset from current force levels to obtain offset
; level that will cause current force reading to go to zero.

                        FOR i = 0 TO 5
                        offset[i] = offset[i]-force[i]
                        END
                        DEVICE (2, 0, stat, fs.set.offset, 6) offset[]
            END
            END

RETURN .
END


.PROGRAM fs.buffer(enable, reset, rate, wrap, record, stat)

; ABSTRACT: Utility routine to enable/disable the force signature buffer.
;
; INPUT PARM: enable = Flag: indicates whether to enable or disable the
; force signature buffer
; reset = Flag: indicates whether to reset the buffer before
; enabling it (used only when "enable" flag is set)
; rate = Real: buffer update rate in milliseconds
```

```
; (used only when "enable" flag is set)
; wrap = Flag: indicates whether to allow the buffer to wrap
; when full (an INPUT only when "enable" flag is set).
;
; OUTPUT PARM: wrap = Flag: indicates whether buffer wrapped (an OUTPUT
; only when "enable" flag is clear).
; record = Real: when enabling, this is the first record number
; in which data is being stored; when disabling, this
; is the number of the last record with valid data.
; stat = Status of operation. Standard V+ error code.
; (Other outputs are NOT defined if error occurs.)
;
; SIDE EFFECTS: None
;
; MISC: Identical copies of this routine exist in the files
; LIB_FS.V2 and USER_FS.V2.
;
;* Copyright (c) 1989 by Adept Technology, Inc.

        AUTO in[3], out[1]

        IF enable THEN
        ;Enable buffer

            IF reset THEN
            ;Copy "reset" bit
            out[0] = 1
            ELSE
            out[0] = 0 END
            IF wrap THEN
            ;Copy "wrap" bit
            out[0] = out[0]
            END
            out[1] = rate
            ;Copy sample rate

            DEVICE (2, 0, stat, fs.ena.buffer, 2) out[], in[]
            IF stat < 0 GOTO 100

            record = in[0]
            ;Note start index

        ELSE
        ;Disable buffer

            DEVICE (2, 0, stat, fs.dis.buffer) , in[]
            IF stat < 0 GOTO 100

            wrap = in[0]
                        ;Note if wrap  occurred

            record = in[1]                                          ;Note end index

        END
```

```
100 RETURN
.END .


PROGRAM fs.config(guarded, ft.flags, dim[], vec[,], lower[], upper[], stat)

; ABSTRACT: Force utility routine to configure Guarded or Monitor mode.
;
; This routine is typically executed before enabling Guarded or
; Monitor mode of operation. It packages trip-condition data that
; is sent (via the DEVICE instruction) to the force-sensing system.
; This routine does not, however, actually enable Guarded or
; Monitor mode.
;
; INPUT PARM: guarded = Flag indicating operating mode to configure:
; TRUE (=fs.guard) => configure Guarded Mode
; FALSE (=fs.monitor) => configure Monitor Mode
; ft.flags = Bit array of flags indicating the types of
; trip conditions to be used--torque (bit set)
; or force (bit clear). First bit corresponds
; to trip condition 0, second bit to condition
; 1, etc. For example, to set the first two
; conditions to be force, second two to torque,
; "ft.flags" should be set to ^B1100.
; vec[,] = Array of 3-element trip vectors. vec[0,0],
; vec[0,1], vec[0,2] contain the X, Y, and Z
; coordinates of the first vector, respectively.
; First array index goes from 0 to "fs.vec.num".
; dim[] = Real array of trip condition dimensions.
; dim[0] contains dimension of 1st condition, etc.
; If dim[i]==0, then condition "i" is ignored and
; vec[i,], upper[i], and lower[i] info is unused.
; upper[] = Real array of upper force or torque thresholds
; lower[] = Real array of lower force or torque thresholds
;
; OUTPUT PARM: stat = V+ error code. stat < 0 implies error occurred.
;
; SIDE EFFECTS: None
;
; MISC: Identical copies of this routine exist in the files
; LIB_FS.V2, TEST_FS.V2, and USER_FS.V2.
;
;* Copyright (c) 1989 by Adept Technology, Inc.

        LOCAL num.args, out[] AUTO v

; Ensure "out[]" array is defined, since some elements are not given
; values as a result of conditional statements below.

        IF NOT DEFINED(out[0]) THEN
            num.args = 2+6*fs.vec.num FOR v = 0 TO num.args-1
            out[v] = 0
            END
```

```
        END

; Load the input parameters into the array "out[]".

        out[0] = guarded                                    ;Load mode flag

        out[1] = ft.flags                                   ;Load trip-condition flags


        FOR v = 0 TO fs.vec.num-1
        ;Load threshold data...
            IF dim[v] THEN
                out[v+18] = upper[v] out[v+22] = lower[v] IF dim[v] < 3 THEN
                    out[2+3*v] = vec[v,0] out[3+3*v] = vec[v,1] out[4+3*v] = vec[v,2]
            END
            END
            out[v+14] = dim[v]
        END

; Send the information to the force system.

        DEVICE (2, 0, stat, fs.set.guard, num.args) out[]

RETURN
.END


.PROGRAM fs.error(stat, $error)

; ABSTRACT: Determines latest force system error code and string.
;
; Issues the "fs.get.status" DEVICE instruction to obtain system
; status, then returns an error string associated with that error
; using a lookup table.
;
; INPUT PARM: None
;
; OUTPUT PARM: stat = Real variable that receives the latest force
; system error. If no error occurred, "stat"
; is returned unchanged.
; $error = Error string associated with latest force
; system error. If no error occurred, the
; string is empty.
;
; SIDE EFFECTS: None
;
; MISC: Identical copies of this routine exist in the files LIB_FS.V2,
; TEST_FS.V2, and USER_FS.V2
;
;* Copyright (c) 1989 by Adept Technology, Inc.

        AUTO in[12], st

        $error = ""
```

```
                ;Initialize to no error

; Get error information from the force system.

                DEVICE (2, 0, st, fs.get.status) , in[] IF st < 0 THEN
                ;Status command failed
                    $error = $ERROR(st) stat = st GOTO 100
                END

                IF NOT in[3] GOTO 100
                ;Return if no error

; Return the appropriate error message string.

                stat = in[3]
                ;Report any error

                CASE in[3] OF
                    VALUE -6500:
                    $error = "*FPB/FIB ribbon cable disconnected*"
                    VALUE -6501:
                    $error = "*FIB internal error*"
                    VALUE -6502:
                    $error = "*FIB command error*" VALUE -6503:
                    $error = "*FIB voltage supply failure*"
                    VALUE -6504:
                    $error = "*FIB NV memory error*" VALUE -6505:
                    $error = "*FIB reset occurred*" VALUE -6506:
                    $error = "*Force sensor data error*"
                    VALUE -6515:
                    $error = "*FIB data timeout*" VALUE -6516:
                    $error = "*FIB command timeout*" VALUE -6517:
                    $error = "*FPB calibration load timeout*"
                    VALUE -6518:
                    $error = "*FPB calibration checksum error*"
                    VALUE -6519:
                    $error = "*FPB/FIB handshake error*" VALUE -6520:
                    $error = "*FPB internal error*"
                    VALUE -6521:
                    $error = "*Invalid data passed to force system*"
                    VALUE -6522:
                    $error = "*Illegal force operation while enabled*"
                    VALUE -6523:
                    $error = "*Invalid force mode configuration*"
                    ANY
                        $error = "*Unknown force system error"
                        $error = $error+" ("+$ENCODE(/I0,in[3])+")*"
                END

100 RETURN
.END
```

```
.PROGRAM fs.initialize(stat)

; ABSTRACT: Force utility routine for initializing the force system.
;
; This routine is used to set force system global variables and
; to initialize the force sensing system.
;
; NOTE: THE GLOBAL VARIABLES SHOULD NOT BE CHANGED BY THE USER,
; SINCE THEY ARE REFERENCED BY A VARIETY OF PROGRAMS.
;
; INPUT PARM: None
;
; OUTPUT PARM: stat = Status of operation. Standard V+ error code.
; If <0, more info may be obtained from
; "fs.get.status" DEVICE instruction.
;
; SIDE EFFECTS: None
;
; MISC: Identical copies of this routine exist in the files
; INIT_FS.V2 and USER_FS.V2
;
;* Copyright (c) 1989 by Adept Technology, Inc.

; Command table for FSM SETDEVICE instructions.

        fs.setd.init = 0            ;Initialize force sensing system

        fs.setd.reset = 1           ;Clear force system errors, reset

        fs.setd.sig = 4             ;Define signal set for Guarded/Monitor trips


; Command table for FSM DEVICE instructions.

        fs.get.status = 1           ;Read system status

        fs.set.parm = 2             ;Initialize sensor specific parameters


        fs.set.frf = 3              ;Set FRF

        fs.get.frf = 4              ;Read FRF


        fs.set.offset = 5           ;Set force offset

        fs.get.offset = 6           ;Read force offset


        fs.set.zero = 7
        ;Zero force sensor

        fs.get.force = 8            ;Read forces (in FRF)
```

```
fs.get.gauge = 9                ;Read strain gauges

fs.set.latch = 10               ;Reset data latches
fs.get.latch = 11               ;Read data latches

  fs.set.guard = 12             ;Configure guarded move or monitor
  fs.ena.guard = 13             ;Enable guarded move or monitor
  fs.dis.guard = 14             ;Disable guarded move or monitor

  fs.ena.buffer = 15            ;Enable force signature buffer
  fs.dis.buffer = 16            ;Disable force signature buffer
  fs.get.buffer = 17            ;Read force signature buffer

fs.ena.protect = 18             ;Enable protect mode
fs.dis.protect = 19             ;Disable protect mode

; Mnemonics for guarded-mode and monitor-mode configuration, enabling,
; and disabling.

        fs.guard = 1 fs.monitor = 0

; Bit masks for system status flags.

  fs.st.guard = ^H8                     ;Guarded Mode active
  fs.st.monitor = ^H10                  ;Monitor Mode active
  fs.st.protect = ^H20                  ;Protect Mode active
  fs.st.buffer = ^H40                   ;Force buffer active

; Number of trip vectors available to users.

        fs.vec.num = 4
        ;Number of trip vectors

; Initialize the force system.

        IF NOT SWITCH(FORCE) THEN
        ;If FORCE switch is not enabled...
        ENABLE FORCE
        END
```

```
                SETDEVICE (2, 0, stat, fs.setd.init)
                ;Initialize the sensor

RETURN .
END .


PROGRAM fs.read.buffer($f.t.pos, first, num, data[], stat)

; ABSTRACT: Data acquisition utility for force signature.
;
; Acquires force and position data from the FPB. Will acquire
; with repeated read-buffer commands until the entire amount of
; requested data is returned.
;
; INPUT PARM: $f.t.pos = String indicating the force, torque or
; position component to return. If this is
; not one of the accepted component names
; (see below), "num" is returned as zero.
; first = Record number of first record to be returned
; (#0 is oldest record in buffer)
; num = Number of records to be returned.
; (This is also an output parameter, so it
; MUST be specified as a variable.)
;
; OUTPUT PARM: num = Actual number of records returned. (Zero if
; "$f.t.pos" was invalid.)
; data[] = Real array of data returned.
; ("data[0]" contains first data element, ...,
; "data[num-1]" contains the last element.)
; stat = Status of operation. Standard V+ error code.
;
; SIDE EFFECTS: Temporarily changes setting of the UPPER system switch.
;
; MISC: Identical copies of this routine exist in the files
; LIB_FS.V2 and USER_FS.V2
;
;* Copyright (c) 1989 by Adept Technology, Inc.

                AUTO angs[6], bit, err, i, num.bits, out[2], pos
                AUTO received, upper
                LOCAL in[]

                received = 0
                ;Number of records successfully received

                upper = SWITCH(UPPER)                   ;Save setting of upper switch

                ENABLE UPPER                            ;Disable case-sensitivity


; Define data-handling information per request in "$f.t.pos".
```

```
num.bits = 1
;Assume 1 bit of data per access

CASE TRUE OF
    VALUE $f.t.pos == "FX":
    bit = ^H1
    ;Bit 0: FX VALUE $f.t.pos == "FY":
    bit = ^H2
    ;Bit 1: FY VALUE $f.t.pos == "FZ":
    bit = ^H4
    ;Bit 2: FZ VALUE $f.t.pos == "FR":
        bit = ^H7
        ;Bits 0, 1, 2: resultant force num.bits = 3
    VALUE $f.t.pos == "TX":
    bit = ^H8
    ;Bit 3: TX VALUE $f.t.pos == "TY":
    bit = ^H10
    ;Bit 4: TY VALUE $f.t.pos == "TZ":
    bit = ^H20
    ;Bit 5: TZ VALUE $f.t.pos == "TR":
        bit = ^H38
        ;Bits 3, 4, 5: resultant torque num.bits = 3
    VALUE $f.t.pos == "PX":
        bit = ^HFC0
        ;Bits 6-11: Position in X num.bits = 6 pos = 1
    VALUE $f.t.pos == "PY":
        bit = ^HFC0
        ;Bits 6-11: Position in Y num.bits = 6 pos = 2
    VALUE $f.t.pos == "PZ":
        bit = ^HFC0
        ;Bits 6-11: Position in Z num.bits = 6 pos = 3
    VALUE $f.t.pos == "J1":
    bit = ^H40
    ;Bit 6: Joint 1 angle
    VALUE $f.t.pos == "J2":
    bit = ^H80
    ;Bit 7: Joint 2 angle
    VALUE $f.t.pos == "J3":
    bit = ^H100
    ;Bit 8: Joint 3 angle
    VALUE $f.t.pos == "J4":
    bit = ^H200
    ;Bit 9: Joint 4 angle
    VALUE $f.t.pos == "J5":
    bit = ^H400
    ;Bit 10: Joint 5 angle
    VALUE $f.t.pos == "J6":
    bit = ^H800
    ;Bit 11: Joint 6 angle
    ANY
GOTO 100
;Undefined request
END
```

```
                out[0] = bit
                ;Set bits for requested data

; Loop until full amount of requested data is received or error occurs.

                WHILE received < num DO
                        out[1] = first+received out[2] = num-received

                        DEVICE (2, 0, stat, fs.get.buffer, 3) out[], in[] IF stat < 0 GOTO 100
                        ;Quit if error
                        IF in[0] == 0 GOTO 100
                        ; or no data

                        FOR i = 1 TO num.bits*in[0] STEP num.bits
                                CASE num.bits OF
                                   VALUE 1:
                                   ;Pass back data directly (1 bit)
                                   data[received] = in[i]
                                   VALUE 3:
                                   ;Resultant requested (3 bits)
                                   data[received] = SQRT(SQR(in[i])+SQR(in[i+1])+SQR(in[i+2]))
                                   VALUE 6:
                                   ;Position requested (6 bits)
                                      SOLVE.TRANS trans, err = in[i] DECOMPOSE angs[1] = trans
                                      data[received] = angs[pos]
                                END
                                received = received+1
                                ;Increment number received
                END
                END

        100     num = received                                  ;Pass back number received

                                                                SWITCH UPPER = upper
                ;Restore former case-sensitivity

                                                                MCS "DELETER @ in[]"
                ;Delete input array

RETURN .
END
```

APPENDIX C - COMPLIANT MOTION USING PROXIMITY SENSING WTH THE ADEPT-3 ROBOT

```
.PROGRAM vs_main()
;VISION SENSOR VERSION
;this program is used for compliant motion using the vision sensor
;FUNCTION: following cracks using the vision sensor
;by continuously generating points based on sensor readings
;and its present location.
;VARIABLES
; point[]-location generated at each step
; n - integer, number of times the loop will execute
;vg - real, velocity gain, defines tangential velocity
; along surface
;dt - real, estimated time step used for taking derivatives
; in PID control law
;status - integer, status of operations addressing sensor via serial port
;i - integer, index used for each point generated
;phi0 - real, the specified initial orientation of the end effector
;err[] -real,array, the magnitude of the offset error between the
; crack and the center of the sensor
;phi[] -real,orientation of end effector at any step i.
;dump -Variable used to "clean out" serial port buffer before
; starting communications with LSS.
;srot -Initial orientation given from mouse-interface startup program
;ti[] -Array containing values of system timer during each cycle
; used for off-line user analysis of program cycle time
;
;INITITALIZATION:
; declare variables which are local to main program only
                LOCAL point[], n, vg, status, dump ATTACH (10)
                ;attaches serial port user1 to this task
;Clear the buffer on the serial port by reading the port in "no wait"
;mode until an error is encounter that indicates no more data is left
                dump = 1
                WHILE dump >= 0 DO
                        dump = GETC(10,1)
                        IF dump < 0 THEN
                                WAIT
                                dump = GETC(10,1)
END
;buffer is cleared ;
        WRITE (10) $CHR(26)
                ;send character to LSS to indicate that
               ;program is ready to receive data

i = 1
;set the index to 1
; set the max number for i
n = 10000
; set gain corresponding to tangential velocity along surface
```

```
vg = 8
; set the end effector orientation
phi0 = 60
; set the time step for taking derivatives in force control law
                dt = 0.3
                ACCEL 80, 80
;set default acceleration, deceleration for the manipulator
; set arm speed
SPEED 12 ALWAYS
; don't null out small position errors with each move
NONULL ALWAYS
; set accuracy of positioning to coarse
COARSE ALWAYS
; initialize the first error = 0
err[i] = 0
; set initial orientation
phi[i] = phi0
; call subroutine to generate new location
; based on error and present location
CALL gen_point.sub(point[], vg)
; move to new point
MOVE point[i]
; increment index i by 1
i = i+1
;

                TIMER 1 = 0
                ;set timer to zero
;timer is used for user analysis of performance
;START LOOP: DO UNTIL i=n OR PROGRAM IS ABORTED
WHILE i < n DO
; call subroutine to generate new location based on error and
; present location
; offset error is generated in 'process_err' as global value
                        CALL sens_com() ;read error from sensor
                        CALL gen_point.sub(point[], vg)
                        ;generate new point
;       move to new point
                        MOVE point[i]
                        ti[i] = TIMER(1)
                        ;record the present time i = i+1
END
;END OF LOOP

.END



PROGRAM sens_com()
;This subroutine handles communications with the LSS.
;It reads the error information and stores it in err[i] for
;use by other programs.
;
;All errors are offset and scaled to be transimitted and received as values
```

```
; between 0 and 254. ;such that 0= -2 inches, 128=0 inches and 254=2 inches etc.
;error scaling allows transmission of each error as a single byte.
;Handshaking: ASCII Character 26 must be sent to the LSS before an error is
;sent to the controller. This prevents the serial port buffer from filling
;up. If the LSS encounters a "No crack found" condition, 255 will be sent.
;
;VARIABLES: ;lun - logical unit for the serial port
;bite - actual value read from serial port
;mode - read mode of serial port, default=0.
;scale - Scale factor between byte receivced and measured error
;offset- Offset between byte received and measured error
;err[] - Global array containing error for each step 'i'.
                LOCAL lun, mode, bite, scale, offset
                ;define local variables
                lun = 10
                mode = 1
                scale = 4*25.4/256
                offset = 128
                bite = GETC(10)
                WRITE (10) $CHR(26) IF bite == 255 THEN
                err[i] = err[i-1] ;use last error if "No crack found"
                ELSE
                err[i] = scale*(bite-offset)
                ;calculate error from 'bite'
                END
                RETURN
.END


.PROGRAM gen_point.sub(point[], vg)
;This subroutine generates the next for the manipulator to move to.
;The new point is based and the gain from the sensor reading 'fg',
;the tangential velocity gain, 'vg' and the present location 'loc.now[i]'
;the new location genrated is stored in 'point[i]' for use by the "vs_main"
;
;VARIABLES:
;vg -- Global, tangential velocity gain defined in "vs_main"
;fg -- Global, gain from offset error, calculated is "for_law.sub"
;point[] -- Array of generated points given to the manipulator
; at each step 'i'.
;loc.now[]- Array of actual locations measured at each step 'i'.
;a[] - Array containing Cartesian coordinates for 'loc.now[i-1]'
;b[] - Same as above for 'loc.now[i-2]' ;x[] - Ditto for 'loc.now[i]'.
;phi[] - Global, end effector orientation for each step 'i'.
;phi0 - Global, given initial crack orientation
;rerr[] - Global, orientation error between estimated crack direction
; and actual end-effector orientation
;rg - Global, gain taken from applying control law to 'rerr[]'.
;dx - Delta x between last two estimated crack locations
;dy - Delta y between last two estimated crack locations
; dx and dy are used with the ATAN2 function to estimate crack direction.
;x1p - These four variables
;y1p - are used in intermediate steps
;x2p - for calculating ;y2p - 'dx' and 'dy'
```

```
;newx - Cartesian x for new location 'point[i]'.
;newy - Cartesian y for new location 'point[i]'.
;x1 - These 3 variables
;y1 - store the first three elements
;z - of array 'x[]' respectively.
;
; Declare local variables ;
                LOCAL newx, newy, fg, dx, dy, x1, y1, x1p, y1p, x2p, y2p, z, loc.now[], a, b, x
                CALL for_law.sub(err[], fg) ; call subroutine to calculate offset
                                                                ; gain 'fg' from errors 'err[]'.

                HERE loc.now[i]
                ;store present location as 'loc.now[i]
                DECOMPOSE x[1] = loc.now[i]
                ;store the cartesian coordinates
                ;of 'loc.now in array 'x[]'.
                IF i <= 2 THEN
                phi[i] = phi0
                ELSE
;       store the cartesian coordinates from the previous 2 actual

;       manipulator locations into the arrays 'a' and 'b' respectively.

                        DECOMPOSE a[1] = loc.now[i-1]
                        DECOMPOSE b[1] = loc.now[i-2]
;calculate the actual cartesian location of the crack from the
;previous 2 steps by using the recorded manipulator location,
;orientation and sensor error.
;This allows estimation of actual crack direction

                        x1p = b[1]-err[i-2]*SIN(phi[i-2])
                        y1p = b[2]+err[i-2]*COS(phi[i-2])
                        x2p = a[1]-err[i-1]*SIN(phi[i-2])
                        y2p = a[2]+err[i-1]*COS(phi[i-2])

;dx and dy are the differences between the x and y coordinates
;of the crack from the previous two steps

                        dx = x2p-x1p dy = y2p-y1p
;       keep the ATAN2 function from becoming infinite if dx=0
                        IF ABS(dx) < 0.01
                        THEN
                        dx = 0.01*SIGN(dx)
                        END

;Calculate the error between the end effector orientation

;       and the estimated actual crack direction

rerr[i] = -phi[i-1]+ATAN2(dy,dx)
; call subroutine for control law for orientation error
CALL r_law(rerr[], rg)
; calculate new desired end effector orientation
                phi[i] = phi[i-1]+rg
```

```
                END
;           Calculate new location from 'fg', 'vg', 'phi[i]' & 'point[i-1]'
                x1 = x[1]
                y1 = x[2]
                z = x[3]
                newx = x1+vg*COS(phi[i])+fg*SIN(phi[i])
                newy = y1+vg*SIN(phi[i])-fg*COS(phi[i])
                SET point[i] = TRANS(newx,newy,z,0,180,(-1)*phi[i])
                RETURN

.END

.PROGRAM for_law.sub(err[], fg) ;
;This subroutine applies a PID control law to the position error
;The position errors are stored in the array "err[]"
;one position error is taken from the sensor for each time
the main program loops. The result of the error with the control
;law applied is returned as "fg". ;VARIABLES
; ki -- The coefficient for integral gain
; kp -- The coefficient for proportional gain
; kd -- The coefficient for derivative gain ; gi[] - The total value of the integral of the error
; calculated at each step
; gp -- Contribution from proportional gain at each step
; gd -- Contribution from derivative gain at each step
; i -- The number of the present step
; dt -- Global variable, the estimated time step used for
; differentiation and integration
;
LOCAL ki, kp, kd, gd, gp, gi[]
;declare local variables
; Set gains for control law
                ki = 0
                kp = 0.1 kd = 0
;           Initialize integral and derivative gains for first two time steps.

                gi[1] = 0
                err[1] = 0
                IF i == 1 THEN
                        gi[i] = ki*dt*0.5
                        gd = kd*err[i]/dt
ELSE
; Calculate integral and derivative of the error
                        gi[i] = (ki*dt*0.5)*(err[i]+err[i-1])+gi[i-1]
                        gd = kd*(err[i]-err[i-1])/dt
                END
                gp = kp*err[i]
                fg = gi[i]+gd+gp
                RETURN

.END



.PROGRAM r_law(rerr[], rg)
```

```
;THIS PROGRAM APPLIES A PID CONTROL LAW TO THE ORIENTATION ERROR BETWEEN
;THE ESTIMATED CRACK DIRECTION AND ACTUAL END EFFECTOR ORIENTATION.
;THE HISTORY OF ORIENTATION ERRORS ARE STORED IN ARRAY 'rerr[]'.
;
;VARIABLES:
;ki, kp, kd -- integral, proporitional and derivative gains, respectively. ;
gi[i] -- value of error integral at each step 'i'.
;gd, gp - contributions from derivative and proportional gain, respectively.
;rg -- total calculated gain ;
i -- index for present step
;
 ; Declare local variables
LOCAL ki, kp, kd, gd, gp, gi[]
; set gains
                ki = 0
                kp = 0.05
                kd = 0
;           initialize as appropriate and calculate gain
                gi[1] = 0
                gi[2] = 0
                rerr[1] = 0
                rerr[2] = 0
                IF i <= 2 THEN
                        gi[i] = ki*dt*0.5
                        gd = kd*rerr[i]/dt
                ELSE
                        gi[i] = (ki*dt*0.5)*(rerr[i]+rerr[i-1])+gi[i-1]
                        gd = kd*(rerr[i]-rerr[i-1])/dt
                END
                gp = kp*rerr[i]
                rg = gi[i]+gd+gp
                RETURN

.END .
```

APPENDIX D - COMPLIANT MOTION WITH THE GMF-A510 ROBOT

```
program lssint
--this program uses the lss to follow cracks.
--It requires the position at the start of the crack, 'spoint'
--to be taught beforehand.
--
--VARIABLES:
-- i, integer; counter for the number of times the main prog. loops.
--stat, integer; status for display of 'karelpage' screen.
--err, array; error from sensor for each step i.
--phi, array; calculated orientation for each step i.
--rerr, array; error between actual rotation and estimated
-- crack direction for each step i.
--x, array of world 'x' coordinates of positions.
--y, array of world 'y' coordinates of positions.
--gi, array; necessary to integrate rotation error for control law
-- for rotation, has to be declared in main to keep array
-- from being initialized each time control law routine is
-- called.
--axi, array;movement of linear slide for each step, calculated and used
-- in 'int2d' declared here to keep array from being
-- initialized each time 'int2d' is called.
--phi0; starting orientation of crack for position 'spoint' +/-180 degrees.
--dt; estimated time step for derivative and integral terms in control laws
--vg; gain for motion paralell to crack
--z; world z coordinate of a position
--wh; yaw of end effector (constant)
--ph; pitch of end-effector
--points; path consisting of all consecutive points generated by
-- looping through main routine.
--point; position generated in present step.
--spoint; taught starting position of crack
var
counter,i,stat:integer
err,x,y,phi,rerr,gi,axi:array[1000] of real
phi0,dt,vg,xh,yh,z,wh,ph,rh:real
configstring:string[12]
points:path
point,spoint:position
--
--Declared the rountine for integrated slide motion ('int2')
--from external program 'int2d'.
routine int2(positions:path;axi:array of real;i:integer)
from int2_arc
--
--normal direction control law
--This routine calculates the position gain perpendicular to the
--crack from the measured error. This is a function routine whose
--value is the total calculated gain.
```

```
--INPUTS:
--err; array of measured errors from sensor
--i; number of present step
routine for_law(err:array of real;i:integer):real
--VARIABLES:
--kp; proportional gain coefficient for control law.
--kd; derivative gain coefficient for control law.
--gd; calculated derivative gain.
--gp; calculated proportional gain.
--dt; estimated time step from main program
--fgl; total gain returned to main program.
var
kp,kd,gd,dtl,fgl,gp:real
begin
kp=.12
--set proportional gain
kd=0.035
--set derivative gain
err[1]=0
--set starting error to 0.
--take derivatives and calculated derivative gain.
if i=1 then
gd=kd*err[i]/dt else
gd=kd*(err[i]-err[i-1])/dt
endif
gp=err[i]*kp
--calculate proportional gain
fgl=(gp+gd)
--calculate total gain
return(fgl)
--return total gain to main program
end for_law
--
--calculate rotation gain
routine r_law(rerr:array of real;i:integer):real
--function routine to calculate rotation gain from estimated error.
--VARIABLES:
--kp; proportional gain coefficient
--gp; calculated proportional gain
--kd; derivative gain coefficient
--gd; calculated derivative gain
--rerr; array of estimated rotation error.
--gi; array used in calculating integral gain.
--ki; integral gain coefficient
--i; current step from main routine
--dt; estimated time step from main routine
--rg; total calculated gain returned to main routine
var kp,kd,ki,gd,gp,rg:real
begin
kp=.04
--set gains .....
kd=0
ki=0.007
gi[1]=0
```

```
gi[2]=0
rerr[1]=0
rerr[2]=0
if i<=2 then
   gi[i]=ki*dt*.5
   gd=kd*rerr[i]/dt
else
gi[i]=(ki*dt*.5)*(rerr[i]+rerr[i-1])+gi[i-1]
--integral gain
gd=kd*(rerr[i]-rerr[i-1])/dt
--derivative gain
endif
gp=kp*rerr[i]
--proportional gain
rg=gi[i]+gd+gp
--total gain returned to main
return(rg)
end r_law
--
--
--generate points
routine gen_point(err:array of real;i:integer):position
--This routine generates the a point for the next manipulator move
--The new point is based on the gain from the sensor reading, 'fg'
--the tangential velocity gain 'vg' and the present location as
--determined by the karel built-in routine 'curpos'. This is a function
--routine that returns a position.
--
--VARIABLES:
--vg; tangential velocity gain defined in 'lssint'
--fg; gain from calculated from sensor error, calculated in routine 'for_law'.
--dx; Delta x between last two estimated crack locations
--dy; Delta y between last two estimated crack locations
--(Crack direction is estimated using dx and dy with ATAN2 function)
--rg; Change in manipulator orientation, calculated from 'rlaw'
--x[]; Array, cartesian x for each step, i.
--y[]; Array, cartesian y for each step, i.
--w,p,z; Cartesian components of manipulator position that are constant
--newx; Cartesian x for new location.
--newy; Cartesian y for new location.
--x1p,y1p,x2p,y2p; These four variables are used in intermediate steps
-- in calculating 'dx' and 'dy'.
--phi[]; Array, end effector orientation for each step, i.
--phi0; starting orientation of crack
--
var
config:string[10]
dx,dy,fg,rg,x1p,x2p,y1p,y2p,newx,newy,w,p,z:real
begin
err[1]=0
w=180
p=0
fg=for_law(err,i)
--find normal direction gain from sensor data.
```

```
-- find cartesian components of current location
unpos(curpos,x[i],y[i],z,w,p,phi[i],config)
-- Estimate crack direction
if i<=2 then
phi[i]=phi0 else
   x1p=x[i-2]-err[i-2]*sin(phi[i-2])
   y1p=y[i-2]+err[i-2]*cos(phi[i-2])
   x2p=x[i-1]-err[i-1]*sin(phi[i-1])
   y2p=y[i-1]+err[i-1]*cos(phi[i-1])
dx=x2p-x1p
dy=y2p-y1p
if abs(dx)<0.01 then
dx=0.01
endif
rerr[i]=atan2(dx,dy)-phi[i-1]
-- Make sure direction is defined +/- 180 degrees.
if rerr[i]>360 then
rerr[i]=rerr[i]-360
endif
if rerr[i]<-360 then
rerr[i]=rerr[i]+360
endif
if rerr[i]>180 then
rerr[i]=rerr[i]-360
endif
if rerr[i]<-180 then
rerr[i]=rerr[i]+360
endif
rg=r_law(rerr,i)
--run difference between estimated crack direction
--through control law
phi[i]=phi[i-1]+rg
--new end effector orientation
   if phi[i]>180 then
   --define angle +/- 180.
   phi[i]=phi[i]-360
   endif
  if phi[i]<-180 then
  phi[i]=phi[i]+360
  endif
endif
--Calculate new cartesian location
newx=x[i]+vg*cos(phi[i])+fg*sin(phi[i])
newy=y[i]+vg*sin(phi[i])-fg*cos(phi[i])
return(pos(newx,newy,z,w,p,phi[i],config))
end gen_point
--
--
---- routine sens_com(err:array of real;i:integer)
--This routine handles serial communication with the local sensor.
--It sends ASCII character 26 each time to receive the present
--error which is scaled as a single byte signed integer (+/-127)
--
--VARIABLES:
```

```
--ibyte; error value read from sensor
--ireply; integer value sent to sensor to request data
--scale; converts scale value to offset in millimeters var ibyte,ireply:integer
scale:real
begin
ireply=26
scale=(4*25.4/256)
write f2(ireply)
read f2(ibyte::1)
write f2(ireply)
if ibyte=127 then
   if i=0 then
   err[i]=0
   endif err[i]=err[i-1]
endif
if ibyte<>127 then
err[i]=scale*(ibyte)
endif
end sens_com
--
--
--
begin
unpos(spoint,xh,yh,z,wh,ph,rh,configstring)
--Intitialize ....
phi0=rh
--set starting crack direction to current manipulator orientation
dt=.05
vg=10
i=1
err[i]=0
--
--Clear values of 'points' before starting program.
for i=1 to pathlen(points) do
delpathnode(points,1)
endfor
--
$termtype=nodecel
i=1
$speed=400
$use_config=false
--open serial port
open file f2('rw,field,readahead=10,uf,baud=4800,parity=odd,passall','c3:')
displaypg('karelpag',stat)
--move manipulator and slide to starting point, 'spoint'.
apndpathnode(points)
pospath(spoint,points,i)
int2(points,axi,i)
delay 1000
--begin first move
i=2
err[i]=0
point=gen_point(err,i)
apndpathnode(points)
```

```
pospath(point,points,i)
int2(points,axi,i)
--begin reading sensor and generating points
for i=3 to 10000 do
sens_com(err,i)
--read sensor
point=gen_point(err,i)
--generate next point
--append position 'point' to path 'points' and call routine 'int2'
--to perform integrated move.
apndpathnode(points)
pospath(point,points,i)
int2(points,axi,i)
endfor
--end of loop
close file f2
--close serial port
end lssint
```

APPENDIX E - WORKSPACE OF THE GMF-A510 MANIPULATOR

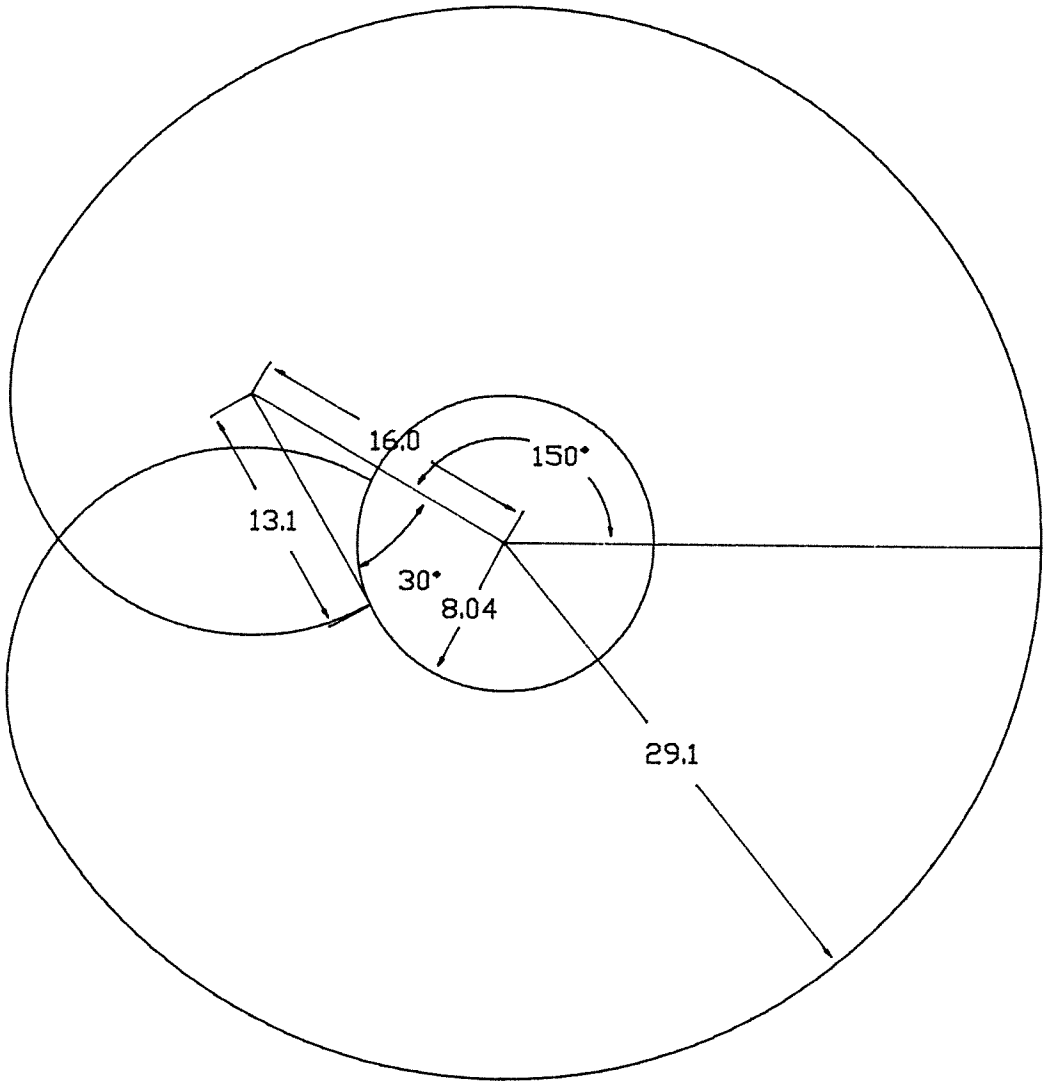


Figure E.1 *Workspace of the A-510 manipulator (dimensions in inches). The two link lengths 16.0 in. and 13.1 in. are shown.*

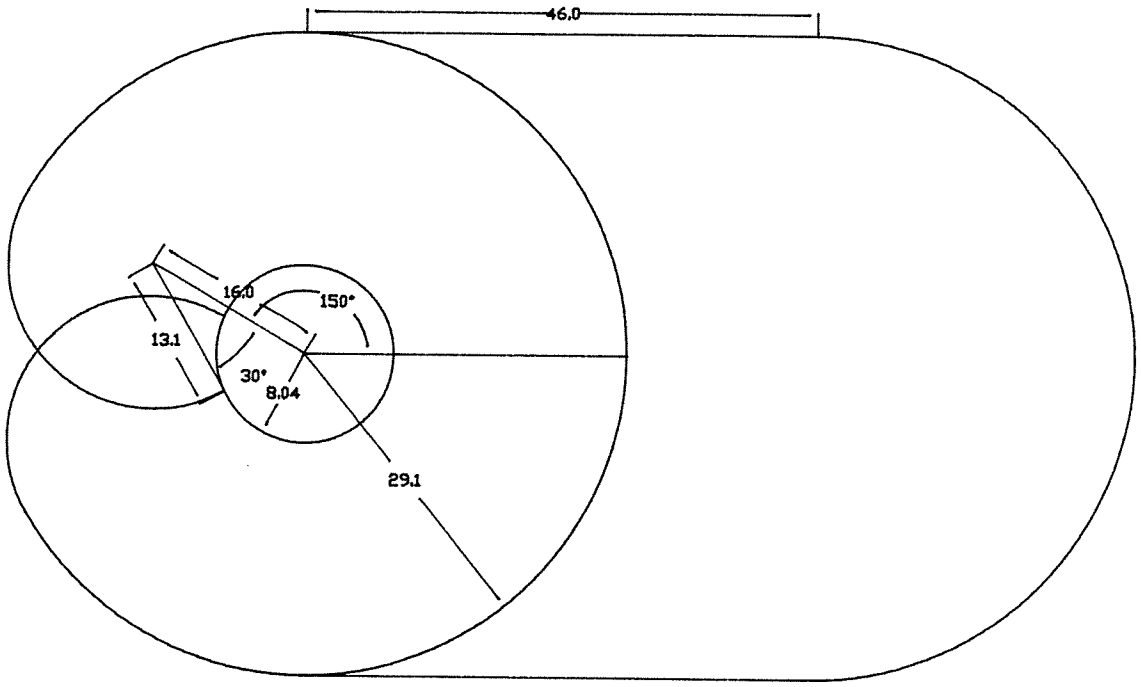


Figure E.2 *Total workspace of the GMF-A510 manipulator with the linear slide. The slide has 46 in. of travel.*

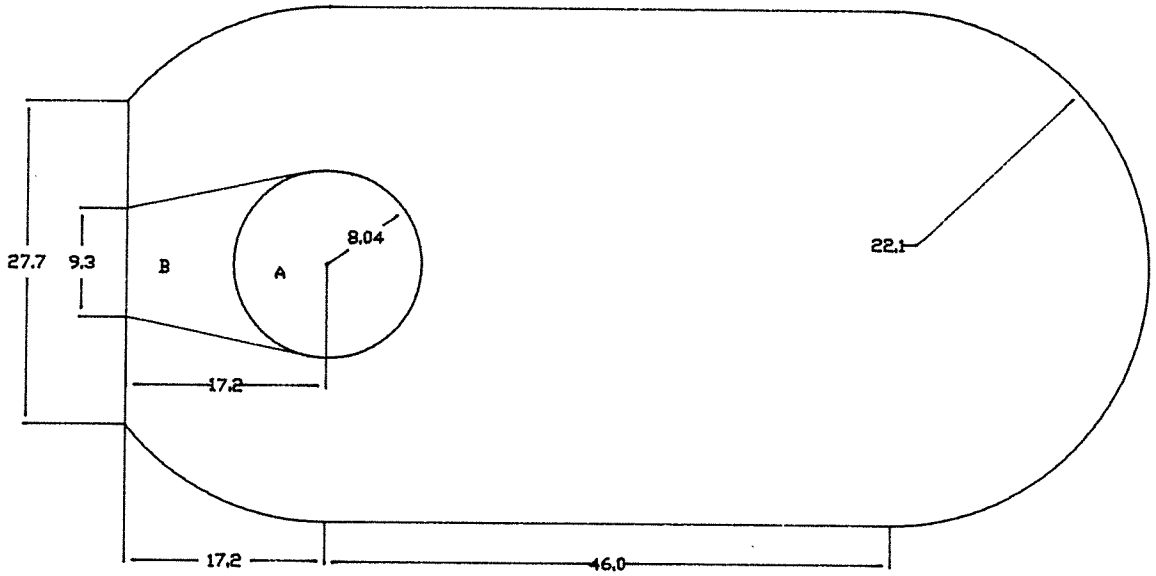


Figure E.3 *Usable workspace of the GMF A-510 manipulator with the linear slide. The usable workspace is defined as the workspace with the links extended to 75% of their maximum reach. The regions labeled* 'A' *and* 'B' *are restricted due to limitations of the linear slide motion. No path may have points inside region* 'A' *and no path may pass through two opposite boundaries of region* 'B'. *There are no singularities in the usable workspace.*

APPENDIX F - INTEGRATION OF THE GMF-A510 MANIPULATOR WITH THE LINEAR SLIDE

As mentioned in Chapter 2, the GMF A-510 manipulator is integrated with a linear slide which acts as a redundant degree-of-freedom. Code was written to integrate the linear slide into the inverse kinematics of the manipulator to allow for full Cartesian level programming. This is necessary because all of the sensing from the VSS is done with respect to a fixed Cartesian reference frame and so any positional data from the ICU will be sent to the RPS with respect to a fixed reference frame. Also, the hybrid control algorithm requires motion specified with respect to a fixed reference frame.

The A-510 manipulator has 4 independent joints and, therefore, 4 degrees of freedom. One of the joints is the ball screw in the column which moves paralell to the world 'z' axis. This leaves 3 degrees-of-freedom for motion in the x-y plane. Since we wish to control x and y positions as well as orientation in the x-y plane, we will utilize all 3 degrees-of-freedom. Therefore, the manipulator can move to any position and the end-effector can have an arbitrary orientation in the x-y plane.

The only redundant solutions that arise are due to configuration of the arm (whether the arm is a left-hand or right-hand configuration). The arm configuration must be specified for each position in addition to the usual Cartesian location and orientation. However, it is a simple matter to write code to determine the best arm configuration for various areas of the workspace. The KAREL controller calculates inverse solutions for all points within the manipulator workspace. Therefore, it is possible to input only Cartesian data, determine the configuration via a pre-determined algorithm and move to points within the workspace. This allows for full Cartesian level programming for application programs.

The linear slide moves the manipulator parallell to the x-axis in the x-y plane. If the linear slide is used as an additional degree-of-freedom, there are now an infinite number of inverse kinematic solutions for points that are within the workspace covered by the

manipulator and the slide.

The KAREL controller is capable of incorporating the position of the linear slide into the forward kinematics of the manipulator. This means, for example, that if all four robot axes are not moved and the linear slide is moved, the controller knows the change in the x position of the end-effector due to the motion of the slide. The controller is also capable of calculating inverse solutions for the end-effector within the manipulator workspace with respect to a fixed reference frame. If the controller is told to move the manipulator to a point defined in the world coordinate system, it will take the position of the linear slide into account and move the 4 joints to reach that point provided it is in the workspace of the manipulator.

The Karel controller does not use the linear slide in inverse kinematic solutions. *This means if the controller is given a point which is out of the workspace of the 4 robot axes but could be reached if the slide were moved, it will return a "point out of range" error.* This problem must be resolved in order to allow for full Cartesian programming in the workspace covered by the manipulator and the linear slide.

The problem of choosing inverse kinematic solutions for a manipulator with a redundant degree-of-freedom is very complicated in the general case. However, due to the nature of the crack sealing operation, constraints can be imposed on the desired motion of the end-effector which will simplify the problem of choosing an inverse solution. The two assumptions are:

(1) The manipulator will only be allowed to extend to 75% of its maximum reachable workspace (22.1 in. from center)

(2) Points will be given to the controller in series corresponding to following a single crack from one side of a lane to the other. All points in a series must be monotonic with respect to the previous point in the x direction. This means that any path must be constantly increasing or constantly decreasing in x. This constraint fits the types of cracks that are to be addressed by the ACSM.

Given the above constraints, the following approach can be used. Define a curve within the workspace of the manipulator such that when the curve is slid paralell to the motion of the slide, it will produce a desirable shape for the workspace. The workspace of the manipulator at each extreme of the slide can be added to the shape produced by sliding the curve to obtain the total workspace of the manipulator and the slide. One important constraint exists for the curve: the curve must be a single-value function of an axis fixed to the base of the manipulator paralell to the world 'y' axis. Using this

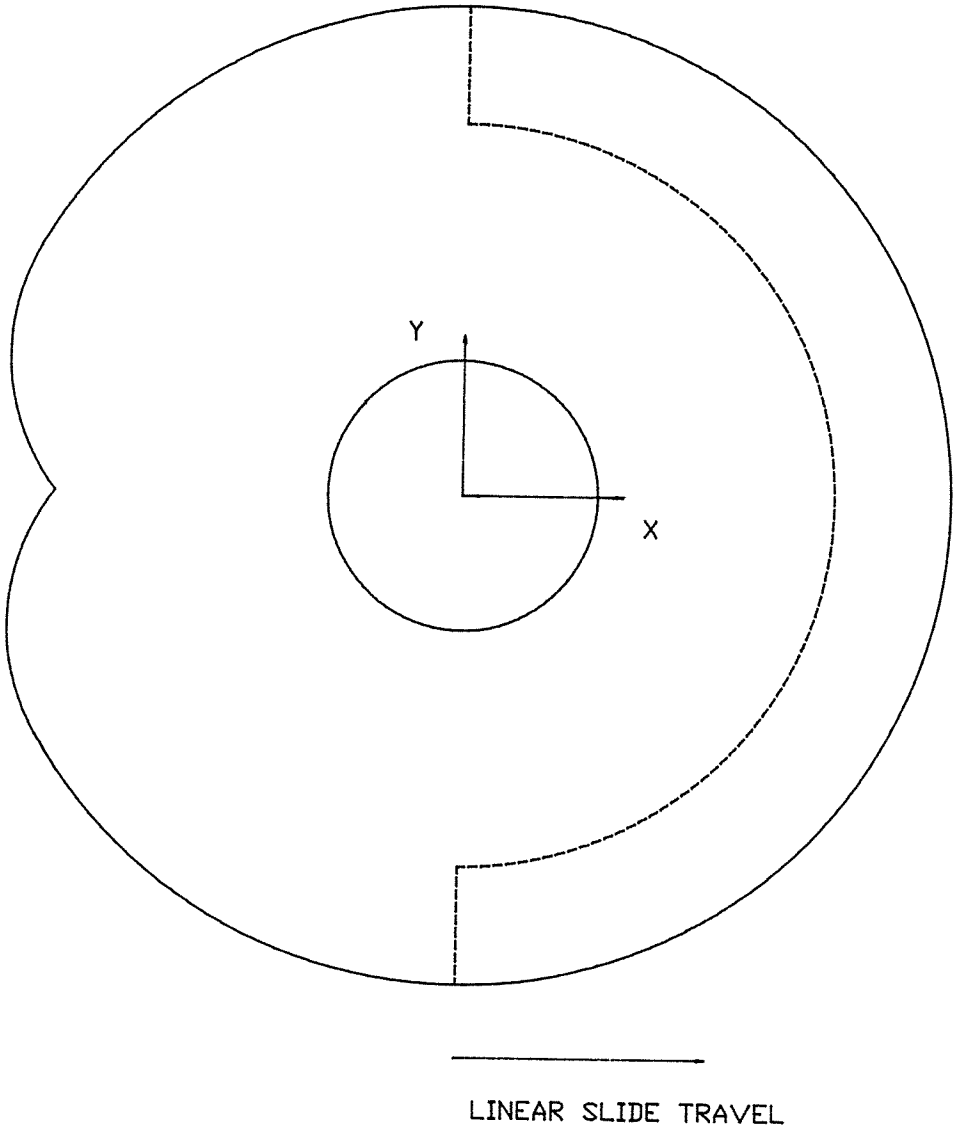


Figure F.1 *The dashed line represents the path on which the manipulator will move relative to its base for an integrated move with the linear slide.*

approach, unique inverse-kinematic solutions can be specified for all points within the workspace. For a move between any two points, the manipulator will maintain a trajectory along the curve relative to its base frame while the remainder of the motion is handled by the linear slide. As long as the curve is a function of 'y', there will be no redundant solutions (except configurations as discussed earlier) since the slide moves paralell to the x-axis.

This algorithm has two strong advantages. The first advantage is that it is simple to implement on the Karel controller. The controller can calculate inverse kinematics for the manipulator joints and calculating the necessary motion for the slide can be made quite simple by the proper choice of the pre-defined curve. Also, the workspace of the robot-slide system using the algorithm can be easily determined.

There are several disadvantages to the algorithm. First, it reduces the reachable workspace of the manipulator-slide system. This is not a real problem since it is desirable to stay away from the boundaries of the reachable workspace due to mechanical limitations. The second disadvantage is that any defined path must be monotonic in x. This constraint already exists for the crack sealing machine and therefore is not significant. The final constraint is that all points on a path must be defined relatively close together to avoid point out-of-range errors. This error will occur because the Karel controller must calculate the inverse solution for the manipulator beffore the linear slide moves. Therefore, if a point is given that is out of range of the robot joints with the current slide position, a position out-of-range error will result. This constraint requires the curve defined within the manipulator workspace to be a safe distance from the edge of the workspace.

The workspace of the A-510 manipulator is shown in figure E.1. The direction of motion of the slide is also shown. The curve in the workspace will be defined such that the manipulator will always be at 75% of maximum extension. Therefore, the curve will be a semi-circular arc with a radius of 22.1 in. This is shown if figue F.1. If the curve is

slid along the axis of the linear slide, and the workspace of the manipulator at the slide limits is added in, a workspace of the shape shown in figure E.2 will result. If we reduce the usable workspace of the manipulator at the slide limits to 75% of the reachable workspace, the shape shown in figure E.3 will result. Note that no provision has been made for the manipulator to reach directly underneath itself when at the right-hand side slide limit. For singularities and path constraints in this workspace see Appendix E.

A flowchart of the algorithm used to integrate the linear slide and the manipulator with the method that has been developed is shown in Figure F.2.

Points must be given to the integration routine as a path. A path is simply an array of positions. Using a path variable allows the integration routine to look at previous positions as well as the current position. The routine first checks to see if a new path is being started. If the path is new, the 'start_slide' routine is called which initializes motion for the linear slide. Otherwise, the program calculates where the given position lies on the arc that has been defined for the manipulator workspace. Once this is done, the program calculates where the arc intersects the x-axis. By taking the difference between the arc intersection of the current position and the last position, the program calculates how much the slide must be moved. Once this is done, the manipulator checks the current slide position relative to the slide limits. If the current point can be reached by the manipulator with the slide at its lower limit, the new slide position will be set at the lower limit. If the new calculated slide position exceedss the upper limit, then it will be set equal to the upper limit. Now that the slide move has beencalculated, the slide and the manipulator will move. The next path node will be read and the cycle will continue until the task is complete.

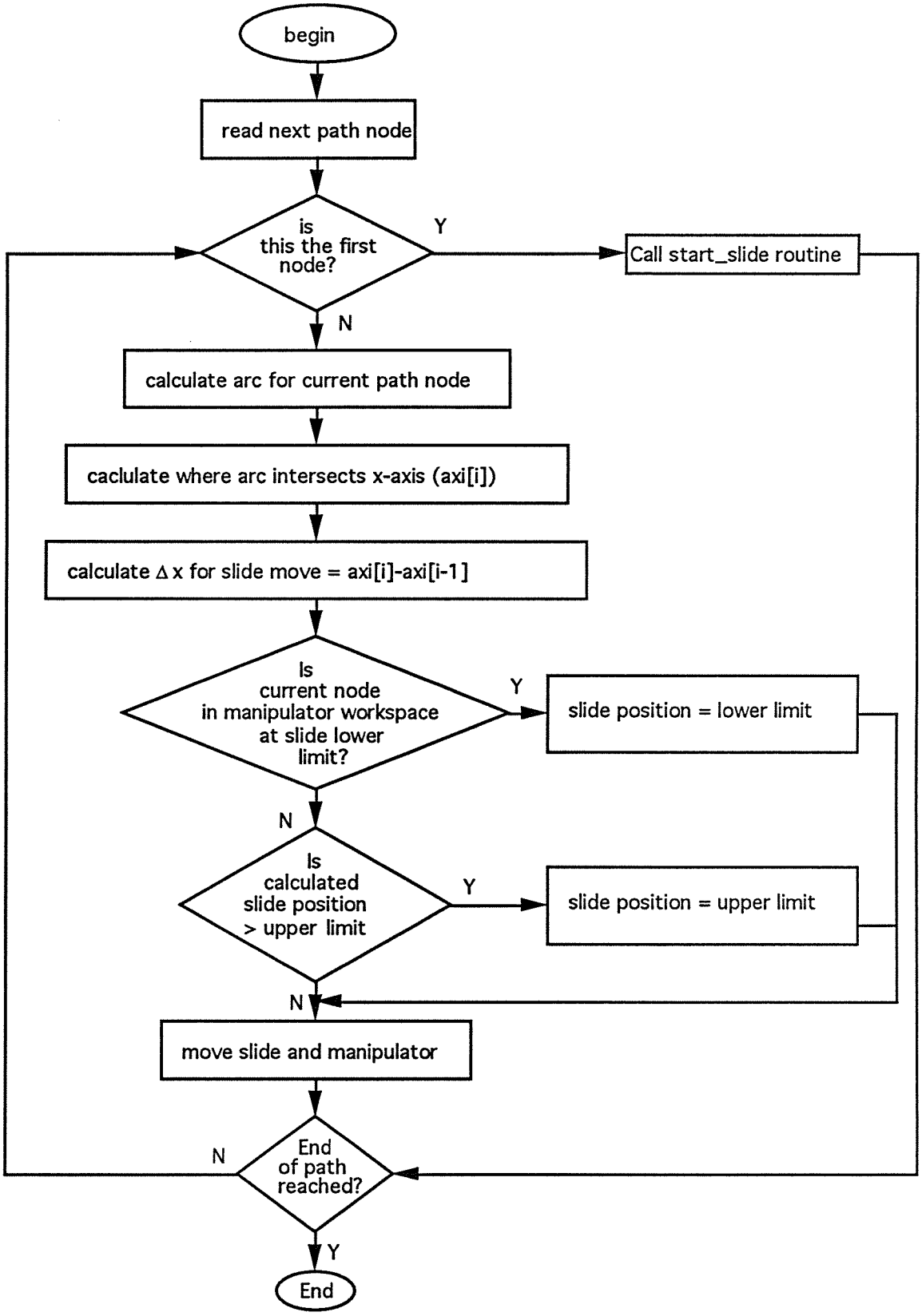


Figure F.2 *Flowchart for slide integration progam.*

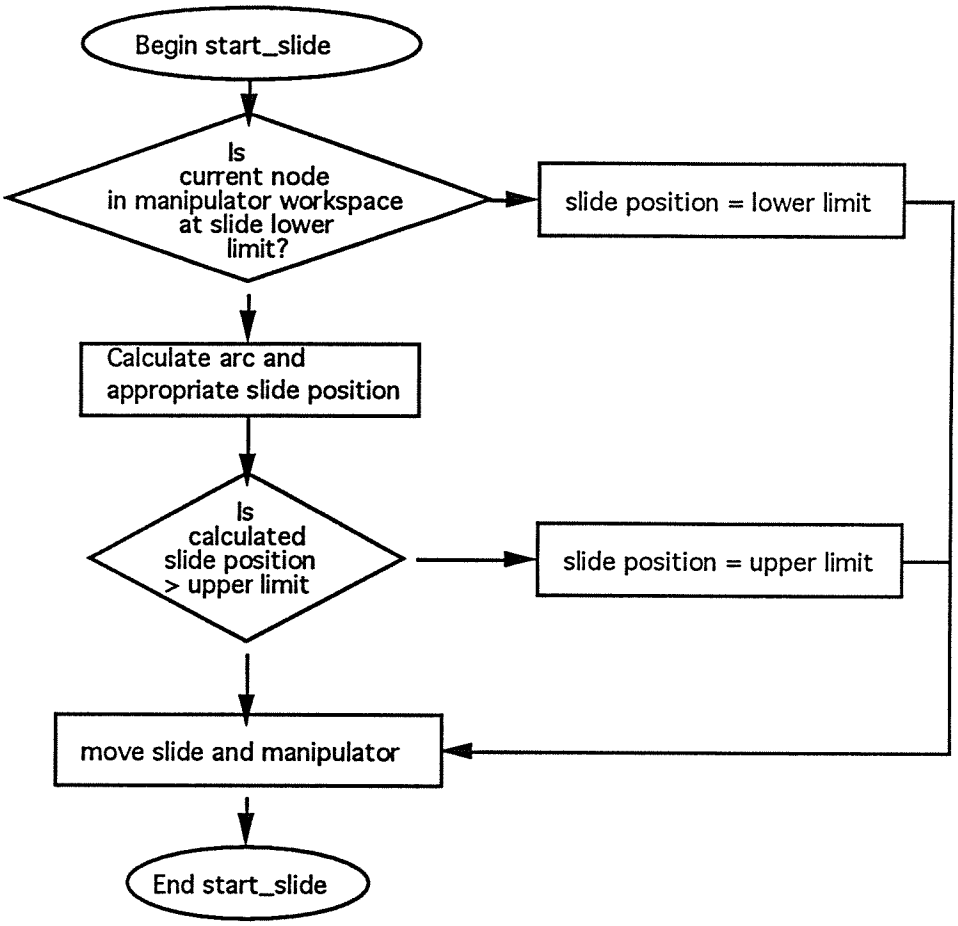


Figure F.3 *Flowchart subroutine 'start_slide' of program for integrating the A-510 manipulator with the linear slide.*

F-2 - CODE FOR LINEAR SLIDE INTEGRATION

```
program int2_arc
-- This program integrates the linear slide and robot axes for motions.
-- The manipulator moves along an arc described in its workspace and
-- remaining motion in the 'x' direction is performed by the slide.
-- All motion is performed by the robot axes at the slide limit at the rear
-- of the workspace.
--
-- INPUT PARAMETERS:
-- points: path, where points[i] is the current desired position
-- and the other nodes are previous desired positions.
-- axi: array of positions of the aux axis, this variable has
-- to be declared in the main program so the array won't be
-- initialized everytime 'int2' is called. All values in axi
-- are calulated in 'int2'.
-- i : index of how many times the program has been called
-- 'i' is necessary to convert the separate points into a path.
-- OUTPUTS
-- The program ouputs a move command to the manipulator along with
-- an auxmove command to the linear slide.
--
-- VARIABLES
-- point:position input
-- i:integer, index of number of times routine has been called.
-- axi:array of real, calculated slide position at each point.
-- config1: string, configuration of present position
-- config2: string, configuration of desired position
-- confign: string, used to change configuration if point is out of range
--
-- ROUTINES
-- startslide: performs initial movement of slide and manipulator for
-- the first point given
-- int2: performs slide and arm movement for all other points.
var
config1,config2,confign:string[12]
point:position
auxmove,auxvar:auxpos
--
--
routine startslide(startpos:position)
var
-- VARIABLES:
-- startpos: position to move to
-- x,y,z,w,p,r: cartesian components of startpos
-- auxdes: calculated desired aux axis position
-- auxhere: current location of slide
-- config: configuration of startpos
auxdes,auxhere:auxpos
x,y,z,w,p,r,x1,y1:real
config:string[12]
```

```
begin
$termtype=nodecel
unpos(startpos,x,y,z,w,p,r,config)
auxhere=curapos
-- find current aux axis position
-- determine which side of the slide the starting position is on and
-- calculate 'auxdes' such that the the arm will be 75% extended to reach
-- startpos.
 if x>=-53 then
    if Y>561 then
    y=561
    endif
    if Y<-561 then
   y=-561
   endif
   x1=sqrt(561*561-y*y)
   auxdes[1]=x-x1
else
auxdes[1]=-615
endif
-- move the aux axis and the manipulator to starting position.
move aux to auxdes
move tcp to startpos
end startslide
---
--- routine int2(points:path;axi:array of real;i:integer)
var
-- VARIABLES:
-- x1,y1,r1: Cartesian components of the last input position.
-- x2,y2,r2: Cartesian components of the current input position.
-- z,w,p,r: Cartesian components that are constant for all positions.
-- dx: Difference in x between last and current position.
-- config1,config2: configs of last and current positions, respectively.
-- confign: string, used to change config if point out of range.
-- points: path consisiting of all points sent to 'int2d'.
-- axi: array of the aux calculated aux axis position for each position.
-- point: position variable used to pull of a single position from
-- the path 'points'.
-- auxlim: value of limit of travel for aux axis.
auxlim,x1,y1,x2,y2,z,w,p,r,dx,r1,r2:real
config1,config2:string[12]
point:position
auxmove:auxpos
begin
-- set speed, termination type and aux axis limit.
$termtype=nodecel
$auxspeed=75
auxlim=-615
-- if this is the first call to routine, initialize with 'startslide'
if i=1 then
   point=pathpos(points,i)
   startslide(point)
else
-- find the aux position 'axi[1]' in order to calculate axi[2].
```

```
if i=2 then
   auxmove=curapos
   axi[1]=auxmove[1]
endif
-- Decompose the current point and the previous point to obtain their
-- Cartesian components.
point=pathpos(points,i)
unpos(point,x2,y2,z,w,p,r2,config2)
point=pathpos(points,i-1)
unpos(point,x1,y1,z,w,p,r1,config1)
-- Calulate the difference in x between previous and current point.
if y2>561 then
y2=561
endif
if y2<-561 then
y2=-561
endif
if y1>561 then
y1=561
endif
if y1<-560 then
y1= -561
endif
dx=x2-x1+sqrt(561*561-y1*y1)-sqrt(561*561Y2*y2)
axi[i]=axi[i-1]+dx
   if axi[i]<-615 then
axi[i]=-615
endif
   if axi[i]>(-auxlim) then
axi[i]=axi[i-1]
endif
auxmove[1]=axi[i]
-- assign the most recent path node to the position variable 'point' point=pathpos(points,i)
point=pos(x2,y2,z,w,p,r2,config2)
-- move robot and auxiliary axis.
move aux to auxmove tcp to point nowait
endif
end int2
-- int2d is a dummy program to hold procedures 'int2' and 'startslide'
begin
end int2_arc
```

APPENDIX G - PROGRAM FOR INTEGRATED CONTROL OF CRACK SEALING WITH THE A-510

```
program acsm_main
--This program controls communication with the LSS and the ICU
--to follow cracks. Data from the ICU will be used and verified
--by the LSS. If the crack disappears from the field of view of
--the LSS, a search will be conducted using the LSS.
--The ICU will constantly be updated on what task the robot is
--executing as well as the Cartesian location of the end-effector.
--
--VARIABLES
--
--err,x,y,phi: see module for 'lss_search'
-- axi : see slide integration routine 'int2' in 'int2_arc'
-- points: path use for searching with LSS
-- crack_path : points sent from ICU
-- nocrackfnd,update,beg-srch,beg_ctr,no_crack,end_crack - integer
-- values to signal icu of robot task
--
var
err,axi,x,y,phi:array[10000] of real
points,crack_path:path
nocrackfnd,counter,max_offset,update,beg_srch,beg_ctr,no_crack:integer
end_crack:integer
routine sens_com(err:array of real;counter:integer)
from lssint
--reads offset from LSS routine
int2(crack_path:path;axi:array of real;counter:integer)
from int2_arc
--performs integrated move with linear slide
routine icu_points
from icu_com
--reads in path of points from ICU routine search
-- searches for crack with LSS
routine center
from lss_search
-- centers on crack with LSS.
from lss_search
routine handl_err
from err_handl
-- sends Karel error number to ICU if error occurs.
routine send_point(dat_type:integer)
from point_send
--sends status and location to icu (status is dat_type)
--
routine main
begin
--
-- initialize serial port
open file f4 ('rw,field,readahead=100,uf,passall','c0:')
```

```
-- initialize signal values
nocrackfnd=127
max_offset=127
update=1
beg_srch=2
beg_ctr=4
no_crack=8
end_crack=32
--
--
-- signal icu if error occurs through 'handl_err'
condition[1]:
        when error[*] do unpause
        handl_err
endcondition
--
enable condition[1]
icu_points
--read in points
counter=1
int2(crack_path,axi,counter)
-- move to first point
sens_com(err,counter)
--check LSS
if abs(err[counter])=nocrackfnd then
        send_point(beg_srch)
        search
 endif
if abs(err[counter])>max_offset then
        send_point(beg_ctr)
        center
endif
for counter=2 to pathlen(crack_path) do
        int2(crack_path,axi,counter)
        --move to remaining points
          sens_com(err,counter)
        --continue to check sensor
        if abs(err[counter])=nocrackfnd then
          send_point(beg_srch)
          --signal icu and search if no crack found
          search
        endif
        send_point(update)
        --update ICU as each point on path is reached
endfor
send_point(end_crack)
--signal ICU when end of crack reached
close file f4
--main is a dummy program to hold routines
end main
begin
end
acsm_main
```

```
PROGRAM icu_com
--This program reads in the planned path from the icu.
--Variables:
--x,y,phi:Components of cartesian location for each point
--inumpoints - number of points in path
--crack_path: path formed by points
VAR
bite,CHECK,startflag:string[1]
X,Y,PHI:STRING[10]
NUMPOINTS:STRING[3]
counter,show_result,INUMPOINTS,count:INTEGER
crack_path:PATH
--
routine icu_points
BEGIN
$TERMTYPE=NODECEL
BITE='D'
STARTFLAG=BITE
CHECK='0'
OPEN FILE F3('RW,FIELD,READAHEAD=1000','C3:')
-- initialize serial port
DISPLAYPG('KARELPAG',show_result)
--display karelpage for crt output
READ F3(STARTFLAG::1::0)
--wait for signal to start
WRITE OUTPUT(STARTFLAG)
WRITE F3(BITE)
READ F3(NUMPOINTS::3::2)
--read in number of points
CONVSTR2INT(NUMPOINTS,INUMPOINTS)
--convert number of points to integer
FOR COUNT=1 TO INUMPOINTS DO
    --start loop to read in points
    READ F3(X::5::2)
    -- read x
    READ F3(Y::5::2)
    -- read y
    READ F3(PHI::4::2)
    -- read orientation
    WRITE OUTPUT(CR)
    WRITE OUTPUT(X)
    WRITE OUTPUT(CR,Y)
    WRITE OUTPUT(CR,PHI)
    build_path(x,y,phi,count)
    -- add point to path
endfor
write F3(bite)
-- signal icu that points have been received
CLOSE FILE F3
CLOSE FILE F4
END icu_points
--
--
```

```
ROUTINE build_path(x,y,phi:STRING;count:INTEGER)
-- adds each point to 'crack_path' as they are read in.
VAR
assign_pos:POSITION
RX,RY,RPHI,Z,W,P:REAL
c:STRING[3]
BEGIN
C="
z=100
W=180
P=0
CONVSTR2REAL(X,RX)
--convert all strings to real numbers
CONVSTR2REAL(Y,RY)
CONVSTR2REAL(PHI,RPHI)
APNDPATHNODE(crack_path)
-- add a node to the path
assign_pos=POS(RX,RY,Z,W,P,RPHI,c)
-- create position from x,y and phi.
POSPATH(assign_pos,crack_path,COUNT)
-- add position to path
RETURN
END build_path
begin
-- 'icu_com' is a dummy program to hold routines
end icu_com



program lss_search
--Contains routines to search for crack with LSS.
-- Variables
--x,y,phi: cartesian components of end-effector position during search.
--
var
x,y,phi,err:array[10000] of real
update_point:position
j,stat:integer vg:real
routine for_law(err:array of real;counter:integer):real
from lssint
--control law for sensor data
routine sens_com(err:array of real;counter:integer)
from lssint             -- reads data from local sensor.
routine icu_points      --reads new path from icu.
from icu_com
routine send_point(dat_type:integer)  --send status and location to icu.
from point_send
--
routine gpts_srch(err:array of real;i:integer):position
--generate points for move based on error data & current location.
var
newx,newy,w,p,z,fg:real
config:string[12]
```

```
begin
err[1]=0
fg=for_law(err,i)
--calculate gain from control law & sensor error
unpos(curpos,x[i],y[i],z,w,p,phi[i],config)
--take cartesian components
--of current location.
newx=x[i]+fg*sin(phi[i])
-- calculate new location
newy=y[i]-fg*cos(phi[i])
return(pos(newx,newy,z,w,p,phi[i],config))
--return new location
end gpts_srch
--
routine center
--centers end effector on crack
var
j:integer point:position
begin
vg=5
for j=2 to 100 do
    sens_com(err,j)
    --read sensor
    write output(err[j],cr)
    if abs(err[j])<20 then
       --check to see if error is small enough
       ret_posn
       --return position to icu
    endif
    point=gpts_srch(err,j)
    --generate new point
    move to point
    -- move to new point
endfor
end center
--
begin
-- initialize serial port
open file f2('rw,field,readahead=10,uf,baud=4800,parity=odd,passall','c3:')
displaypg('karelpag',stat)
--set page for crt display of output
$termtype=nodecel
--set motion type for path
err[1]=0
ret_posn
-- send crack location to icu
close file f2
end lss_search
--
routine ret_posn
var
crack_pos:integer
begin crack_pos=16
send_point(crack_pos)
```

```
icu_points
end ret_posn
--
routine search
--routine to search for crack
var
x,y,z,w,p,r:real
--Cartesian components of location
i,no_crack:integer err:array[10000] of real
config:string[12]
point,point1:position
begin
no_crack=8
point1=curpos
unpos(curpos,x,y,z,w,p,r,config)
for i=2 to 11 do
    x=x+10*cos(r)
    --move at right angle to current location
    y=y-10*cos(r)
    point=pos(x,y,z,w,p,r,config)
    move to point
    sens_com(err,i)
    --check sensor
    if abs(err[i])<>127 then
       --check to see if sensor detects crack
       center
       -- if crack found, then center on crack
    endif
endfor
move to point1
unpos(curpos,x,y,z,w,p,r,config)
for i=2 to 11 do
    --move back to starting location and search in
    --other direction.
    x=x-10*cos(r)
    -- (same as search above)
    y=y+10*cos(r)
    point=pos(x,y,z,w,p,r,config)
    move to point
    sens_com(err,i)
    if abs(err[i])<>127 then
      center
    endif
endfor
send_point(no_crack)
--if no crack found by now, signal icu.
end search


program err_handl
--routine to handle errors from the robot
--will send the karel error code to the icu.
var
err_code,sys_code:integer
```

```
msg_str:string[12]
--
routine main
from acsm_main
routine send_point(err_code:integer)          --send error code to icu
from point_send
--
routine handl_err
var
i:integer
begin
geterrorcode(err_code,sys_code)
-- (sys_code can be used to get a string describing the error from Karel
-- see Karel Refernce Manual)
send_point(err_code)
--send error code & current location to icu main
--send control back to main program
end handl_err
begin
--err_handl is a dummy program to hold routine handl_err
end err_handl


program point_send
-- Holds routine to send status and location to icu.
var
i:integer
--
routine send_point(data_type:integer)
-- 'dat_type' is signal bytes for icu eg. 'nocrackfnd'
var x,y,z,w,p,r:real
xi,yi,ri,start_bite:integer
config,xstring,ystring,rstring:string[12]
--
begin
start_bite=32767
unpos(curpos,x,y,z,w,p,r,config)
--take cartesian components of
--current location.
convreal2str(x,5,0,xstring)
--convert all cartesian components
convreal2str(y,5,0,ystring)
-- to strings and then to integers.
convreal2str(r,4,0,rstring)
convstr2int(xstring,xi)
convstr2int(ystring,yi)
convstr2int(rstring,ri)
write f4(start_bite::2)
-- send start bytes (all 1's)
write f4(data_type::2)
-- send robot status
write f4(xi::2)
-- send location data
write f4(yi::2)
```

```
write f4(ri::2)
write output(xi,cr,yi,cr,ri)
return
end send_point
begin
-- point_send is a dummy program to hold routines & variables.
end point_send
```

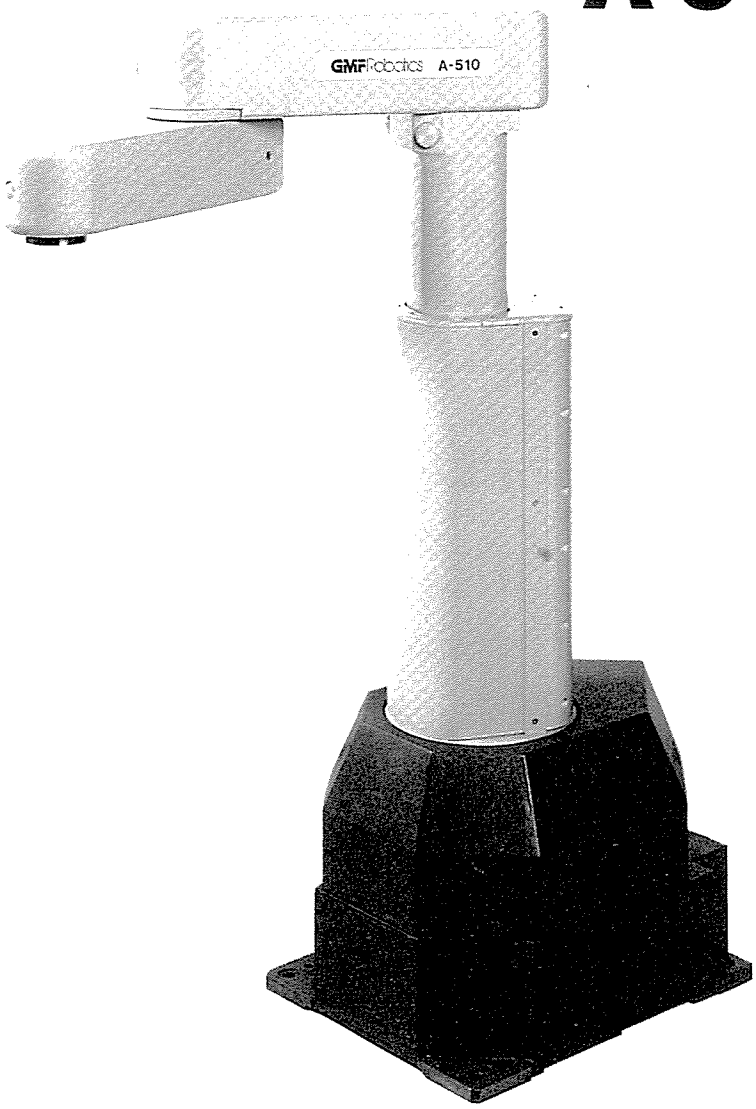
APPENDIX H - MANUFACTURER'S SPECIFICATIONS FOR THE GMF-A510 MANIPULATOR AND KAREL CONTROLLER

GMF Robotics

A-510

Basic Description

The GMFanuc A-510 is a four-axis horizontally articulated SCARA type robot featuring maximum versatility for electrical/electronic assembly, mechanical assembly, material handling, palletizing, and dispensing. In fact, it is also the highest performance robot of its kind. The ultra-reliable KAREL® controller utilizes the latest proven technologies in the industry. Flexibility is further enhanced by the powerful KAREL programming language and available INSIGHT™ vision system.



Features and Benefits

- **All-electric AC servo drive eliminates brush maintenance.**
- **Easy to install and program for quick pay-back.**
- **Rugged construction for long service life.**
- **Can operate in environments from the harshest to Class 10 clean room.**
- **17.2 ft³ (0.487 m³) work envelope* for real-world applications.**
- **Hollow design encloses all cable routing to eliminate snagging.**
- **99 lb. (45 kg) vertical force for pressing parts**.**
- **Integral fail-safe brakes for safety.**
- **Absolute position detection system provides automatic calibration.**
- **Hefty 44lb. (20kg) payload** for realistic EOAT design and part weight.**
- **Base-located Z-axis improves clearance for restrictive installations.**
- **Proximity switches for overtravel protection.**

* Floor Mount configuration only.
**Floor and Inverted Mount configurations.

Inverted Mount Configuration Features and Benefits

- Robot can now "double-back" on itself for increased access space.
- Placement directly over conveyors for maximum part manipulation.

Extended Reach Option Features and Benefits

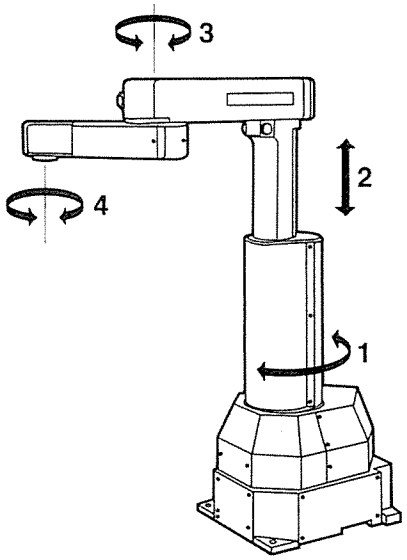
- Maximum reach increased by 28.4% for an even larger work envelope.
- Available in standard and inverted mount configurations.

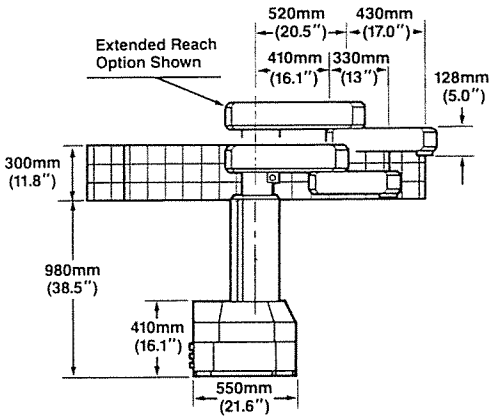
Clean Room Option Features and Benefits

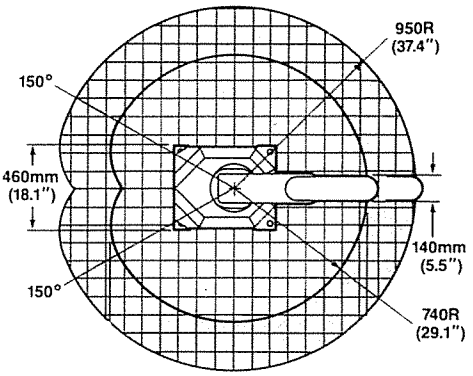
Class 10 configuration provides additional customer benefits:

- Meets IES-RP-CC-006--84-T and ASTM 50-83 clean room standards.
- Centrifugal blower and hosing for particulate evacuation.
- Z-axis in base reduces motion over the work area.
- Internal cable routing reduces contamination and laminar air-flow agitation.

Dimensions

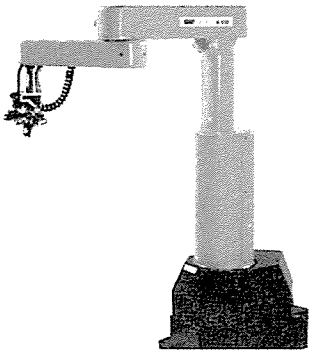


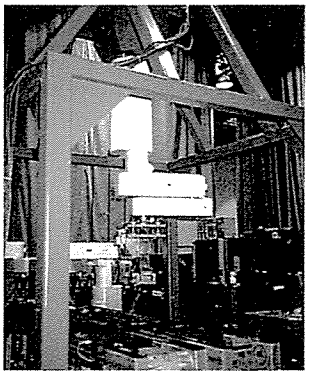




Grid Scale:
1 block = 100mm (3.9")

Specifications





*

Configuration/ Option Available	Floor Mount Standard and Clean Rm.	Floor Mount- Ext. Reach Standard and Clean Rm.	Inverted Mount* Standard
Payload	44lb (20kg)	22lb (10kg)	44lb (20kg)
Moment (Axis 4)	70 kg x cm	50 kg x cm	70 kg x cm
Repeatability	±0.002" (±0.05mm)	±0.003" (±0.065mm)	±0.002" (±0.05mm)
Work Envelope Maximum Reach	29.1" (740mm)	37.4" (950mm)	29.1" (740mm)
Axis 1 (Base) Rotation	300°	300°	300°
Speed	300°/sec	270°/sec	300°/sec
Axis 2 ("Z") Stroke	11.8" (300mm)	11.8" (300mm)	11.8" (300mm)
Speed	27.6"/sec (700mm/sec)	23.6"/sec (600mm/sec)	27.6"/sec (700mm/sec)
Axis 3 (Elbow) Rotation	300°	300°	300°
Speed	300°/sec	270°/sec	300°/sec
Axis 4 (Wrist) Rotation	540°	540°	540°
Speed	540°/sec	540°/sec	540°/sec
Weight-Mechanical Unit	330 lb (150kg)	340 lb (154kg)	330 lb (150kg)

Specifications subject to change without notice.

*Extended Reach Option available.

GMF Robotics

GMFanuc Robotics Corporation
2000 S. Adams Road
Auburn Hills, MI 48057-2090
Literature Request 1-800-634-0189
Main Office (313) 377-7000

GMFanuc Robotics Canada, Ltd.
6395 Kestrel Road
Mississauga, Ont. L5T 1S4
Canada
Main Office (416) 670-5755

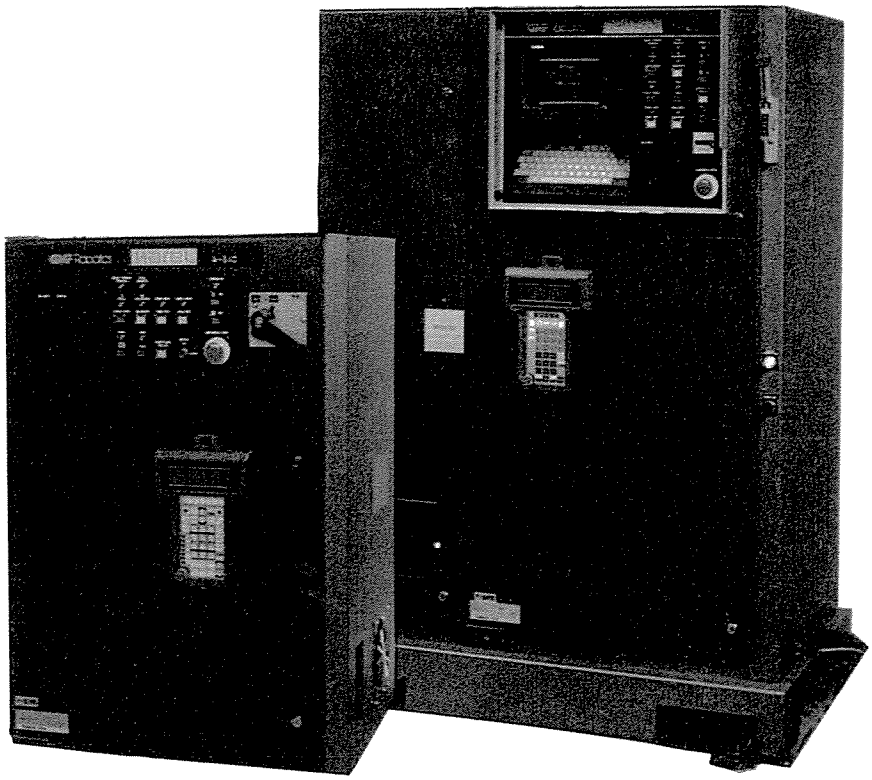
GMFanuc Robotics Europe (GmbH)
Heinrich-Hertz-Str. 4
4006 Erkrath-Unterfeldhaus
West Germany
Main Office 011-49-211-250040

5M 2-90 F & R LITHO IN U.S.A.

KAREL® Controller

BASIC DESCRIPTION

The GMF KAREL® Controller is a self-diagnostic robot control system with full program development and edit capabilities. It is available in two enclosure sizes, both of which feature teach pendant, multiple Motorola 68000 microprocessors, two optional I/O interface styles, built-in operator panel, and optional integral or remote keyboard/CRT. This state-of-the-art controller is designed for compatibility with all GMF robot mechanical units. The controller hardware has been developed from proven performance in over 300,000 NC controller and 10,000 robot controller installations.



KAREL features include:

- Powerful KAREL programming language featuring all new, easy to use operator interface.
- Menu driven operator functions
- Advanced Technology Digital Servos
- Up to nine axis simultaneous control
- Optional Integral MAP interface
- INSIGHT™ (Integral vision system option)
- Remote robot operation
- Remote keyboard/CRT option
- Up to 128 discrete digital I/O points with optional modular or fixed I/O interface
- Analog I/O
- Two enclosure sizes, both featuring single door access.
- Auxiliary axis control

KAREL OPERATOR PANEL

The KAREL operator's panel contains system status LED's along with the control switches needed for day-to-day operation of the robot.

The Operator Panel Features Include:

- Long-Life LED system status indicators
- Remote/Local Operation Keyswitch
- Overtravel Release Pushbutton
- Fault Indication LED and Reset Pushbutton
- Programmable "Cycle Start" Pushbutton
- Full Size Emergency Stop
- 2 User Programmable Pushbuttons and LED's
- Calibration Status LED and Calibrate Pushbutton
- Serial Port connections for Optional CRT unit (portable unit only) and disk drive unit or user device (printer, host computer, etc.)

KAREL I/O (INPUTS/OUTPUTS)

The KAREL controller suports either a Modular (rack) style I/O system or a Fixed (card) I/O system. The modular system is designed for applications requiring a wide variety or larger quantity of I/O points. The fixed I/O card is designed for applications with more limited I/O requirements.

Modular I/O System: Supports both analog and digital I/O modules which plug into a rack mounted within the controller enclosure. Modules are available in either 8 or 16 point configurations.

Discrete Inputs :
24 VDC, 120 VAC 50/60 HZ

Discrete Outputs :
24 VDC sink or source 120 and 240 VAC 50/60 HZ

Analog Inputs :
4 channels per module, voltage or current input, 12 bit resolution

Analog Outputs :
2 channels per module, voltage or current input, 12 bit resolution

Fixed I/O System:
Single system board which supports the following I/O on a simple quick disconnect connector:
24 point 24 VDC outputs
32 point 24 VDC inputs

Additional Modular I/O features include:

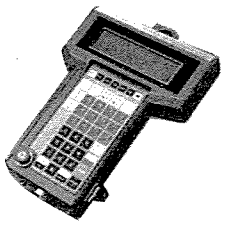
- Quick disconnect terminal strips
- Individual LED indicators on each I/O point
- Fused 120 VAC and 240 VAC output modules with blown fuse indicators.

KAREL TEACH PENDANT

The all new "T" style hand held teach pendant is used for jogging the robot and teaching positions. A large 8 line by 40 character display provides axis coordinates, robot program data, descriptive user messages and diagnostic information.

Additional features include:

- Ergonomically designed lightweight plastic case
- Integrated "Deadman" safety switch for programmer protection
- Sealed full travel keypad
- Hard-wired Emergency Stop switch
- Backlit LCD Display
- Menu driven function displays



KAREL KEYBOARD/CRT

The optional GMF KAREL Keyboard/CRT is used for programming and system setup of the KAREL controller. It can also be used to display operator messages from a KAREL applications program, as well as diagnostics and status information about a running robot system.

It has the following key features:

- Designed for Industrial Use, Fully Sealed
- Full Action Sealed ASCII Keyboard
- High Resolution 9-inch CRT (built-in), 12-inch CRT (remote)
- 80 Column x 24 Line Display to display operational and robot status information
- Available as a built-in unit (large cabinet only) or as a portable unit that sits conveniently on top of the small enclosure, or on a worktable.
- VT-100 and VT-220 compatibility, terminal emulation for IBM PC, optionally available.

KAREL CONTROLLER CPU (Central Processing Units)

The KAREL controller is based on Motorola 68000 microprocessors, and TMS 32C25digital signal processors for digital servo control.

Hardware features include:

- Digital servo control is used for all servo axes which eliminates temperature drifting and the need for analog adjustments.
- Up to 9 axes of simultaneous control for auxiliary axes. (3 auxilliary axes in addition to the base robot axes)
- Noise immune fiber optic links connect the modular I/O system to the main controller electronics.
- The KAREL controller makes extensive use of VLSI technology. This lowers the chip-count, and improves system reliability.
- Rugged industrial design Bubble Memory has been incorporated to store Controller and User Programs. This permits fast field updating of controller software.
- Up to 4 Mbytes of bubble memory (2.0 Mbytes standard)
- Up to 2 Mbytes of RAM memory(1.5 Mbytes standard)
- Optional Powerfail Recovery support
- Optional integrated grayscale Vision System INSIGHT™ implemented as a backplane plug-in
- Optional MAP interface implemented as a backplane plug-in

KAREL LANGUAGE

The KAREL controller incorporates the KAREL programming language: a general purpose robot and automation programming language which greatly simplifies robot application programming.

Programming features include:

- English-like programming statements.
- Basic and advanced arithmetic functions including trigonometric and logarithmic functions.
- Complete set of program control structures including branching and looping.
- Real-time I/O monitoring and user interrupts.
- Relative motion statements which reduce the number of taught positions.
- Coordinate transforms for position shift, offset, and rotation.
- Continuous path motion.
- English names for positions, variables, and I/O points.
- Operator messages displayed at either the CRT or Teach Pendant.
- User comments for documenting developed programs.

Positional data features include:

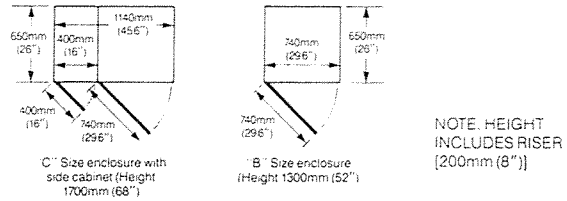
- Storage of data in cartesian coordinates.
- Separation of program logic and position point data. This permits:
- Referencing a position multiple times without using additional memory
- Sharing data between multiple programs
- Insert, change, or delete data without altering program logic.
- Use of the same program logic with different sets of positions. This is useful for different parts with a common process.
- Dynamic changing of program and positional data from the keyboard, teach pendant, sensors, or host computers.

SPECIFICATIONS

Ambient Temperature : 0-50 deg C, (32-122 deg F)
Power Requirements : 200 to 575 VAC, 50/60 HZ, 1.7 KWatts (including robot)

Weight:
Large ("C" size) Enclosure 400 Kg. (880 lbs.)
Side Cabinet 140 Kg. (308 lbs.)
Small ("B" size) Enclosure 250 Kg. (550 lbs.)

CONTROLLER CABINET DIMENSIONS



NOTE. HEIGHT INCLUDES RISER [200mm (8")]

GMFanuc Robotics Corporation
2000 South Adams Road
Auburn Hills, MI 48326-2800
Literature Request 1-800-47-ROBOT
Phone (313) 377-7000 Fax (313) 377-7366

GMFanuc Robotics Canada, Ltd.
6395 Kestrel Road
Mississauga, Ontario
Canada L5T 1Z5
Main Office (416) 670-5755

GMFanuc Robotics Europe GmbH
Heinrich-Hertz-Strasse 16
P.O. Box 3345
D-4006 Erkrath 1, West Germany
Main Office 211-20060

Specifications subject to change without notice

REFERENCES

Allen, Peter K. et al (1991) "Real Time Visual Servoing", *Proceedings of the IEEE International Conference on Robots and Automation* p. 851-856.

Atiya, Sami and Greg Hager (1991) "Real-Time Vision-Based Robot Localization", *Proceedings of the IEEE International Conference on Robots and Automation*, p. 639-644.

Bamba, T. et. al. (1984), "A Visual Seam Tracking System for Arc-Welding Robots", *Proceedings of the 14th Annual Symposium on Industrial Robots,* p. 365-74.

Chaumette, Francois et al (1991), "Positioning of a Robot with Respect to an Object, Tracking it and Estimating its Velocity by Visual Servoing", *Proceedings of the IEEE International Conference on Robots and Automation*, p. 2248-2253.

Craig, John J. (1989) *Introduction to Robotics*, 2nd Ed. Addison-Wesley Publishing Company, Inc. New York, NY.

Craig, Kevin (1991).. "Class Notes from Dynamics and Control of Multibody Systems", Rensselaer Polytechnic Institute, Spring, 1991.

De Schutter, J. (1990) "A Force Controlled Robot" in "Lecture Notes of the 1990 Integrated European Course in Mechatronics" "Computer Controlled Motion and Robotics" Katholieke Universiteit Leuven, Department of Mechanical Engineering. Heverlee Belgium. p. 213-230.

De Schutter, J. (1990) "An Introduction to PID Control and Its Application to Motion Control" Katholieke Universiteit Leuven, Department of Mechanical Engineering. Heverlee Belgium. p. 71-103.

De Schutter J. and H. Van Brussel (1988), "Compliant Robot Motion", *The International Journal of Robotics Research*, vol. 7, no. 4, p. 3-32.

Desa, S. and B. Roth (1985) "Synthesis of Control Systems for Manipulators Using Multivariable Robust Servomechanism Theory", *The International Journal of Robotics Research*, Vol. 4., No. 3, p 18-34.

Espiau, B., J.-P. Merlet and C. Samson (1990) "Force-Feedback Control and Non-Contact Sensing: A Unified Approach", *RoManSy 8, Proceedings of the Eighth CISM-IFToMM Symposium on Theory and Practice of Robots and Manipulators,* p. 176-185.

Farin, Gerald (1988) *Curves and Surfaces for Computer Aided Geometric Design,* Academic Press, Inc., San Diego, Ca.

Fihey, J-L. et. al. (1987) "A Prototype Track Based Compact Robotic System for In Situ Weld Repair of Hydraulic Turbine Runners", *Automated and Robotic Welding*, November, 1987, Paper 25, p. 233-245.

Jang, Won and Zeungnam Bien (1991), "Feature-based Visual Servoing of an Eye-in-hand Robot with Improved Tracking Performance", *Proceedings of the IEEE International Conference on Robots and Automation*, p. 2254-2260.

Jouaneh, M. and D. Dornfeld (1988) "A Kinematic Approach for Coordinated Motion of a Robot and Positioning Table", *Journal of Manufacturing System*, Vol. 7, No. 4 p. 307-314.

Kirschke, K. R. and S. A. Velinsky (1992) "Histogram-based approach for automated pavement-crack sensing", ASCE Journal of Transportation Engineering, vol. 118, p. 700-710, Oct. 1992.

Karezooni, H., Sheridan, T.B. and Houpt, P.K. 1986. "Robust Compliant Motion for Manipulation. Part I: The Fundamental Concepts of Compliant Motion", *IEEE Journal of Robotics and Automation* RA-2(2):83-92

Krulewich, D. A. and Velinsky, S. A. (1992) "Development of a High Resolution System for Automated Crack Sealing Machinery", Interim Report SHRP H-107A, Department of Mechanical, Aeronautical and Materials Engineering, University of California, Davis.

Lasky, T. and B. Ravani (1993) "Path Planning for Robotic Applications in Roadway Crack Sealing" Submitted to *Proceedings of the 1993 IEEE International Conference on Robotics and Automation.*

Lozano-Perez, Tomas (1982) "Robot Programming" *Proceedings of the IEEE*, Vol. 71, No. 7, p. 821-840.

Luh, J. Y. S. (1985) "Design of Control Systems for Industrial Robots" Chp. 11 of *Handbook of Industrial Robotics*, John Wiley & Sons, Inc. Englewood Cliffs, NJ. p. 169-202.

Maciejowski, J. M. (1989) *Multivariable Feedback Design,* Addison-Wesley, NY. 1989. pp. 1-36.

Mason, M. T. and J. K. Salisbury, Jr. (1985) *Robot Hands and the Mechanics of Manipulation*, MIT Press, Cambridge, Mass.

Mason, M.T. (1981) "Compliance and Force Control for Computer Controlled Manipulators" *IEEE Transactions on Systems, Man and Cybernetics* SMC-11(6):418-432.

Millman, Richard S. and George D. Parker (1977) *Elements of Differential Geometry*, Prentice-Hall Inc., Englewood Cliffs, NJ.

NBS (National Bureau of Standards) (1988) "Vision System in the AMRF (Automated Manufacturing Research Facility)", in *Journal of Research of the National Bureau of Standards*, vol. 93, p. 539-544.

Ogata, Katsuhiko. (1987) *Discrete-time Control Systems*, Prentice-Hall, Englewood Cliffs, NJ.

Ogata, Katsuhiko (1990) *Modern Control Engineering*, 2nd. Ed., Prentice Hall, Englewood Cliffs, NJ.

Panakolopoulos, N. et al (1991) "Vision and Control Techniques for Robotic Visual Tracking" *Proceedings of the IEEE International Conference on Robots and Automation*, p. 857-864.

Raibert, M.H., and Craig, J.J. (1981) "Hybrid Position/Force Control of Manipulators" *ASME Journal of Dynamic Systems, Meaurement and Control* 103(2):126-133.

Salisbury, J.K. (1980), (Albequerque, N.M.) "Active Stiffness Control of a Manipulator in Cartesian Coordinates" *IEEE Conference on Decision and Control.*

Schulteiss, E. D. and Velinsky, S. A. (1991) "On the Development of a Design Concept For Automated Pavement Crack Sealing Machinery" Interim Report of SHRP H-107A, Department of Mechanical Aeronautical and Materials Engineering, University of California, Davis, 1991.

Taylor, Russell, Ralph L. Hollis and Mark A. Lavin (1985) "Precise Manipulation with Endpoint Sensing" *IBM Journal of Research and Development,* Vol. 29 No. 4, p. 363-375.

Velinsky, S. A. and Kirschke, K. R.(1991), "Design Considerations for Automated Pavement Crack Sealing Machinery", Proceedings of the Second ASCE International Conference on Applications of Advanced Technologies in Transportation Engineering", pp. 76-80.

Velinsky, S. A. (1991) "Fabrication and Testing of Maintenance Equipment Used For Pavement Surface Repairs" Final Report of SHRP-107A, Phase I, University of California, Davis.

Velinsky, S. A. (1990) "Fabrication and Tesing of Maintenance Used for Pavement Surface Repairs", Proposal for SHRP H-107, University of California, Davis, June, 1990.\

Weiss, Lee E., Arthur C. Sanderson and Charles P. Neuman (1987) "Dynamic Sensor-Based Control of Robots with Visual Feedback", *IEEE Journal of Robotics and Automation,* Vol. RA-3, No. 5, p. 404-416.

Westmore, David B. and William J. Wilson (1991) "Direct Dynamic Control of a Robot Using and End-Point Mounted Camera and Kalman Filter Position Estimation", *Proceedings of the IEEE International Conference on Robots and Automation*, p. 2376-2384.

Wong, Phillip W. (1991) "Force Compliance Control of the UCI Finger System", Department of Mechanical and Aerospace Engineering, University of California, Irvine.