APPLICATIONS OF TELEROBOTICS TO HIGHWAY MAINTENANCE

T.C. HSIA, A.A. Frank
G.D. Benson, and R. Cobene

AHMCT Research Report
UCD-ARR-93-06-07

Final Report of Contract
IA65Q168-MOU 91-6

June 7, 1993

Abstract

This report covers telerobotics studies from the perspective of highway maintenance automation conducted in the UC Davis Robotics Research Laboratory over the last fourteen months. Here we document the four project objectives. First, we explore the current field of telerobotics to understand the underlying principles and current technology. Second, we examine the feasibility and benefits of telerobotics application. Third, we survey an array of "off the shelf" products that can be incorporated in a highway maintenance telerobotic system. Finally, we document the design and development of our telerobotics testbed that demonstrates the utility and feasibility of telerobotics for highway maintenance and to provide practical experience with the development process. The report is concluded with a recommendation of future research and development of a mobile telerobotic system for man-in-the-bucket maintenance operations.

Executive Summary

The main objective of this study is to investigate the applicability of telerobotics technologies to highway maintenance automation. The investigations were carried out through the following tasks:

1. Comprehensive literature survey of telerobotics principles, applications, and limitations.

2. Identify highway maintenance tasks which are feasible and beneficial for telerobotics applications. Establish telerobotic system characteristics meeting highway maintenance requirements.

3. Complete a comprehensive survey of commercially available robotic systems and assess their capabilities in terms of meeting the requirements for highway maintenance automation.

4. Set up a telerobotics test bed at UCD Robotics Research Laboratory to evaluate the functionality of, and human factors in, telerobotics manipulation. Demonstrate the feasibility of highway maintenance by performing a task (tree trimming was chosen) in the laboratory.

Our findings are documented in this final report. Major results of this study and our conclusions are summarized below:

- An appropriately designed telerobotic system using currently available technology is feasible for highway maintenance automation in a number of areas to improve maintenance crew safety, quality of work, and efficiency.

- A laboratory test bed has demonstrated the capability of a teleoperated industrial robot to perform tree trimming task. Through this study the human factor requirement and the corresponding control software and hardware characteristics are understood which would help us to specify the operational features of a telerobotic system for highway maintenance.

- A set of mechanical structure parameters for a flexible highway maintenance telerobotic system are defined. A survey of existing off-the-shelf systems shows that none meets all these requirements. The main problem is the lack of a single system which combines the payload capability of a construction robot and the computer-controlled high-precision manipulability, as well as the automated function capability, of an industrial robot.

- A highly skilled operator controlling a construction robot is capable of picking up and placing a large structure column in a hole in the ground, or placing a brick on a wall. However, manual operations of this type require intensive concentration on the part of the operator and the speed of operation is generally slow. In addition it would be extremely difficult to extend this capability to manipulations which require the end effector to follow some complicated trajectories while the tool is in contact with the environment. Examples of this kind in highway maintenance tasks are sign washing and tree trimming. Those tasks need computer control to aid the operator. An appropriately designed special telerobotic system would provide this capability, doing the task with greater speed and precision. Moreover the skill of the operator is less demanding so that more workers can be trained as operators.

- We recommend the development of a new telerobotic system which combines the mobility and large lifting capability of a "man-in-the-bucket" Caltrans maintenance truck with a dexterous industrial robot as end effector, to replace the workman for the man-in-the-bucket operations. One major technical challenge is to design a sensor-based computer control system which can automatically compensate for the unwanted vibrations and motions of the truck bed during task execution to achieve accurate robotic manipulation. We propose that the inertial stabilization concept be applied to stabilize the base of the end effector robot. Another technical problem is how to intelligently control the motions of the robot such that the vehicle will not tip over when lifting a heavy load.

Table of Contents

List of Figures

List of Tables

Disclosure/Disclaimer

The research reported herein was performed as part of the Advanced Highway Maintenance and Construction Technology Program (AHMCT), within the Department of Electrical and Computer Engineering at the University of California, Davis and the Division of New Technology and Materials Research at the California Department of Transportation. It is evolutionary and voluntary. It is a cooperative venture of local, state and federal governments and universities.

The contents of this report reflect the views of the authors who are responsible for the facts and the accuracy of the data presented herein. The contents do not necessarily reflect the official views or policies of the STATE OF CALIFORNIA or the FEDERAL HIGHWAY ADMINISTRATION and the UNIVERSITY OF CALIFORNIA. This report does not constitute a standard, specification, or regulation.

The contractor is free to copyright material , including interim reports and final reports, developed under the contract with the provision that Caltrans, and the FHWA reserve a royalty-free, non-exclusive and irrevocable license to reproduce, publish or otherwise use, and to authorize others to use, the work for government purposes.

Acknowledgments

We wish to acknowledge the generous donation of a miniature control arm by Schilling Development Inc. of Davis California. Their technical assistance on telerobotic systems technology is also appreciated.

Chapter 1
Introduction

Caltrans provides many services to our state's roadway system, including construction and maintenance. Caltrans highway maintenance is undoubtedly crucial in keeping our roads safe, efficient, and clean. As the state grows so do the demands on transportation, thereby increasing the need for effective highway maintenance. Highway maintenance is time consuming and in many cases it is dangerous to the maintenance crew despite all the precautions currently taken to ensure worker safety. Clearly, increasing highway maintenance efficiency and worker safety are two worthy pursuits. A relatively new technology called *telerobotics* appears to have great promise to improve current highway maintenance tasks both in terms of safety and efficiency. These objectives can be accomplished by removing workers from the work site, such as man-in-the-bucket operations, and replacing them by remotely controlled robot manipulators. Our primary goal is to explore telerobotics from the perspective of highway maintenance. By determining both the capabilities and limitations of telerobotics, we can ascertain which highway maintenance tasks are appropriate for this technology.

The underlying principle of telerobotics is to enhance and extend the capabilities of a human operator to a remote work site. A telerobotic system will typically consist of a local control station for human input, a controlling computer, and a remote manipulator (see Figure 1.1). By extending human dexterity to a remote environment, we can perform tasks too dangerous for direct human interaction and too complex for an automated robot. Thus we want to take advantage of both the human mind to make quick decisions based on complex sensory information and computer efficiency to perform repetitive calculations.

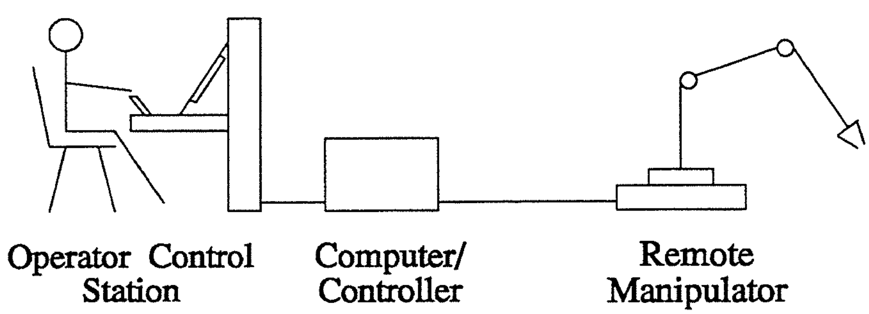


Figure 1.1 A Telerobotic System

Although the telerobotic concept is straightforward, applications research of this technology is rather limited. The largest amount of research in telerobotics is found in space telerobotics (NASA) and undersea applications. In space telerobotics the goal is to provide earth-based human control of remote space tasks (i.e. satellite repair and retrieval). Similarly, undersea telerobotics provides for such tasks as deep-sea exploration, sunken ship retrieval, and oil line repair and construction. More recently, telerobotics is used for nuclear waste cleanup and removal. Each of these applications share two important factors. First, the task environment for each application is either unsuitable or *extremely dangerous* for humans. Second, the tasks are *unstructured.* Humans can adapt to new situations quickly, whereas computer simulated intelligence (artificial intelligence) is still quite restrictive to be applied. Therefore, these systems strive to relay an accurate picture of the remote environment to allow the operator to make informed control decisions. The systems also provide a means for human input to allow the operator to carry out manipulation tasks. Figure 1.2 illustrates the concept of this transfer of information.

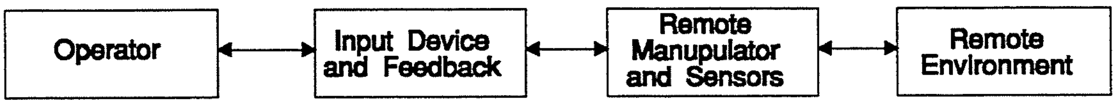


Figure 1.2 Basic Telerobotic System Structure

Many highway maintenance tasks are both dangerous and highly unstructured. Using these criteria it seems that highway maintenance tasks are well suited for telerobotics. As an example, consider the task of roadside tree trimming and maintenance. Currently, many trees are trimmed by a team with usually two or more workers. The team coordinates the tree maintenance process by having someone on the ground directing a worker in a bucket elevated by a large hydraulic crane (depending on the size of the tree). The worker in the bucket controls the position of the crane and has the necessary tools to cut and remove branches. Because the tree is located randomly relative to the roadside a safety region is formed by using cones to direct traffic around the work site. This safety zone is usually quite large to provide maximum worker and traffic safety. As a result traffic congestion is sometimes created.

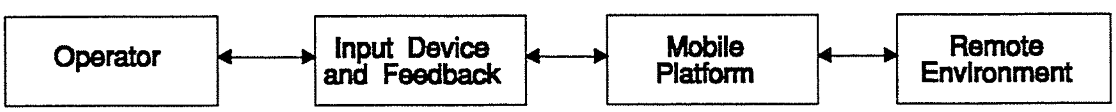


Figure 1.3 Mobile Telerobotic System

Now consider a telerobotic highway maintenance vehicle to perform the same tree trimming task just described. For our purposes we will replace the hydraulic crane with an articulated robot arm and replace the man-in-the-bucket with a device to cut and retrieve branches. The hydraulic crane and robot arm will take the place of the remote manipulator, changing Figure 1.2 to Figure 1.3. The robot arm and vehicle will be supplied with video cameras to provide vision to the operator in the cab. Instead of a standard truck cab, we add television monitors and an input device (i.e. a joystick) for controlling the robot arm. The worker uses the monitors to "see" the tree and uses the input device to control the cutting device. Our new tree maintenance vehicle would then require only one worker rather than a team, thus the time due to coordination is eliminated. Because the driver and operator are inside the vehicle at all times a much smaller safety zone is required, thus reducing the safety zone setup time and lessening the disturbance of traffic flow. Most importantly, nobody is physically located on the road. Although it is a highly simplified description, this example provides the setting and motivation for the application of telerobotics to highway maintenance.

Our research objective during this period was to assess the feasibility of applying the telerobotics concept and technology to highway maintenance. To accomplish this we have the following three major tasks. First, we conduct an in depth survey of the current field of telerobotics to understand the underlying principles and the present state of technology. Second, we identify highway maintenance operations which are feasible and beneficial for telerobotics application. Third, we survey an array of "off the shelf" products that can be incorporated in a highway maintenance telerobotic system. Finally, we document the design and development of our telerobotics testbed. The testbed serves to demonstrate the utility of telerobotics and to provide practical experience with the development process. Based on these experiences, recommendations for future research and development of highway maintenance telerobotic systems are outlined.

The report is structured as follows. Thus far we have given a simplified definition of a telerobotic system. Chapter 2 provides a detailed description of telerobotics and related concepts. Current areas of research are also explored. In chapter 3 we further examine the issues involved regarding applications of telerobotics to highway maintenance leading to the development of the requirements for an "ideal" highway maintenance system. Using the requirements developed in chapter 3 as guide we survey some of the current robotic and manually operated mechanical systems commercially available in chapter 4. The laboratory testbed developed to acquire a practical understanding of telerobotics is described in chapter 5. Finally, in chapter 6 we present some concluding remarks and directions for future research and development.

Chapter 2
Telerobotics Technology

The term "telerobotics" encompasses many ideas, therefore it can lead to ambiguity. Because telerobotics is relatively new, this ambiguity is compounded. This chapter attempts to clear up any confusion over the field of telerobotics. We discuss many related concepts and present some of the current research in the field. The information expounded here will form a basis for the remaining chapters.

In Section 2.1 we define telerobotics and related fields. We present the various facets of a telerobotic system in Section 2.2.

2.1 Overview of Telerobotics

Telerobotics is a relatively new field that has stemmed from research in robotics and teleoperation. The field of robotics is also vast, but holds to a central theme; to automate a task or process normally done by humans. Thus we have robots that can paint cars, mix chemicals, and assemble parts, to name a few. These examples require a human expert to preprogram the robot for the desired task. If all goes well the task will be completed successfully. If for some reason there is a problem (i.e. a car falls of the assembly line) then a human operator will have to intervene. Although robots are equipped with sensors and are controlled by powerful computers, a robotic system lacks the human ability to reason. Therefore we find that standard industrial robots are limited to repetitive and mundane tasks.

Humans are still required to perform tasks that are highly unstructured. There are many tasks that are both unstructured and hazardous to humans. For these tasks *teleoperation* was developed. Unlike telerobotics, teleoperation has been in use for quite some time, dating back to 1947 [JC71]. Sheridan [She88a] defines teleoperation as an extension of a person's sensing and manipulating capability to a remote location. A teleoperator typically consists of an input device, a remote manipulator, and some type of interconnection. Early teleoperators were mechanically coupled, but more recent teleoperation systems are electronically connected with local and remote controlling computers. In most cases teleoperators are used to take advantage of human dexterity in an environment that is unsafe or otherwise inaccessible.

Successful teleoperation is realized through *telepresence.* Telepresence refers to a system characteristic in which appropriate sensor feedback is presented in an efficient way so that a human operator feels physically present at the remote work site. The closer the local environment comes to imitating the remote environment the better a human operator will be able to perform remote tasks. Telepresence is achieved by various techniques. A visual display allows an operator to see the task at hand. Bilateral control (force reflection at the input device) allows the operator to feel proportionally similar forces that are imposed on the remote manipulator. Teletouch, remote sensing of force patterns applied to the manipulator, and teleproprioception, awareness of the manipulator position, are two other attributes of telepresence. For a complete discussion of telepresence see [WR88] and [She88b].

Telerobotics, also referred to as supervisory control, consists of both traded and shared control. In a traded mode an operator can control a manipulator manually, as in teleoperation, but also command the manipulator to perform autonomous tasks. In a shared mode the controlling computer, also referred to as the *host computer*, may coordinate some aspects of a task while the human operator controls other aspects of the same task. For example the controlling computer could be programmed to apply a specific force to a surface, while the operator controls the position along the surface. This configuration allows the human operator to solve analytically complex problems while letting the controlling computer perform computationally complex problems and mundane tasks (i.e. straight-line movements). The goal is to maximize the utility of both the operator and the controlling computer.

So far we have interpreted the telerobot as an extension of the human "arm". This makes sense because most complicated tasks are done with the arms and hands. Other researches, however, use telerobotics to describe remotely controlled land and space vehicles as well. These devices share many of the same ideas as the remote arm, but the implementation techniques used are quite different. Our approach to highway maintenance automation uses the arm paradigm. For these reasons we will only consider the telerobotic arm for the rest of this report.

2.2 Telerobotic Issues

A telerobotic system can have several attributes including control methods and physical components. This section describes the attributes of a telerobotic system. It is important to remember that the goal of a telerobotic system is to allow a human to complete a task quickly and efficiently in a remote location. To do so the human must have sufficient feedback (telepresence) to make good control decisions. To carry out these decisions the operator must also have a means of controlling the remote manipulator. Thus our goal as a telerobotic designer is to improve both sensory feedback and operator input. Research in the areas that follow attempt to work toward this goal.

2.2.1 Modes of Operation

A telerobotic system consists of both traded and shared control, therefore a reasonable system will have different modes of operation and a method to switch between modes. There are three basic modes of operation: *autonomous*, *manual*, and *shared*.

Autonomous mode is similar to a standard robotic system. The remote arm carries out some preprogrammed task possibly based on the current sensory input. In autonomous mode the operator can be a supervisor to the task, allowed to intervene at any time to stop the operation or to switch to manual mode. Although we have stated that telerobotics is best suited for unstructured environments, autonomous mode allows the operator to take advantage of any regularity in a task.

Manual mode is identical to teleoperation. Essentially the operator has total control of the remote arm with virtually no assistance from the controlling computer except to transmit the desired motions. In a telerobotic system the manual mode might also have a teach function. This allows the operator to program tasks to be used in autonomous mode.

The most complex mode in a telerobotic system is *shared* mode. In shared mode some aspects of the task are controlled by the operator while other aspects are handled by the controlling computer. The ratio of computer control to human control is variable. The operator may only indicate when to start and stop, thus the computer has more control. In contrast, the operator may control the position of the end effector, while the computer simply enforces a safety zone for the end effector.

These modes of operation are essential to any telerobotic system, although their degree of implementation may vary. Using these definitions we can pictorially represent the different modes and the interaction of the operator, machine (host computer plus the remote arm), and the environment [Wam88]. Figures 2.1 illustrates the basic telerobotic paradigm.

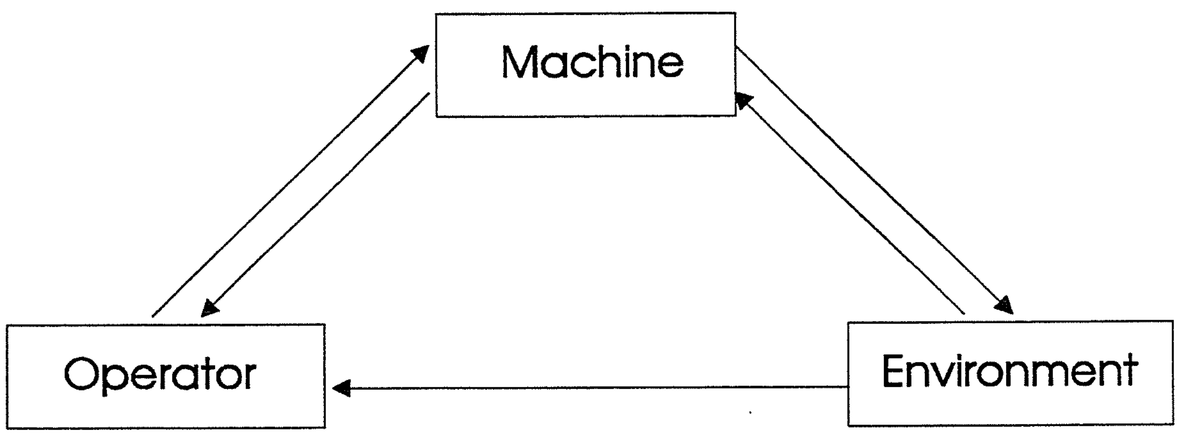


Figure 2.1 Telerobotic Interaction

The operator will perform tasks in the remote environment through the host computer. In Figure 2.1 the *machine* represents both the host computer and the remote arm. The environment (or task space) is directly affected by the machine. Although the operator may get direct feedback (visual) from the environment, he or she cannot directly interact with the task. It is easy to see that all three modes of operation (autonomous, manual, and shared) can be applied to this configuration. Next, we will concentrate on more specific aspects of a telerobotic system.

2.2.2 Control Methods

A telerobotic system can be "controlled" in many different ways. By *control* we mean the way the host computer interprets and processes operator input that determines the position and actions of the remote arm. Therefore there are two important aspects of a control method; the way the operator perceives to affect the remote arm and the way the host computer actually processes input and output. This section covers three important issues related to telerobotic control.

It should be noted that our use of the word "control" is somewhat broader than that used in robotics texts [Fu87, Cra87]. Control in the strict sense refers to the low-level algorithms based on control theory that actuate the motors in the robot. We refer to control as the combination of low-level algorithms and the high-level operator interface. Therefore control involves the host computer as well as the human operator.

2.2.2.1 Joint-to-Joint Mapping vs. Cartesian Mapping

To position the remote arm the operator will manipulate an input device. The input device can be of many different forms (see section 2.2.3). If the input device is a miniature version of the remote arm then we can either use *joint-to-joint mapping* or *Cartesian mapping*.

In *joint-to-joint mapping* the movement of a joint on the input devices causes a similar movement on the remote arm. If we use a one-to-one mapping then a movement of one degree on a given joint of the input device will move the corresponding joint on the remote arm one degree. This mode is easy to implement and requires little operator training time. If the input device and the remote arm are proportional then movements in the operator space will produce exactly proportional movements in the remote arm space.

Having a proportional arm as the input device is the ideal situation, unfortunately they are hard to find. In many cases the remote arm geometry could require the proportional input device to be physically awkward. If the input device is not proportional to the remote arm then although joint movements will be proportional, the end effector positions will not correspond. If the input device and the remote arm are relatively similar then this effect may not be a problem. The operator can adapt to the inconsistencies.

Cartesian mapping solves this problem by mapping the movements of the input device end effector to the remote arm end effector. Even if the two arms are not proportional, the Cartesian positions will directly correspond. For example we can have 1 mm to 1 cm mapping. If the input device moves 1 mm in the x-direction then the remote arm will move 1 cm in the x-direction.

By using Cartesian mapping we eliminate the need for proportional arms. However Cartesian mapping is more complicated to implement and incurs some problems not found in joint-to-joint mapping. First, we must now determine the Cartesian position of the input device instead of the joint positions. This requires that we solve the *forward kinematics* for the input device [Fu87, Cra87]. Solving the kinematics is not difficult, but it does add computational overhead. Second, we must control the remote arm in Cartesian mode which introduces the problem of *singularities*. To move the remote arm the joint motors must be actuated. We want to command a Cartesian position and not joint positions. To do so we must solve the *inverse kinematics* to obtain the joint positions that correspond to the given Cartesian position [Fu87, Cra87]. When solving the inverse kinematics we might reach a singularity. A singularity is a position that either the remote arm cannot physically reach or one that produces an infinite number of solutions. If a singularity is reached then the remote arm must stop and be guided out of the position.

2.2.2.2 Rate Control vs. Position Control

Whether we use joint-to-joint mapping or Cartesian mapping we need to specify how the input device will "move" the remote arm. There two basic methods of control: *rate control* and *position control.*

Under *rate control*, movement of the input device cause a change in velocity (direction and speed) of the remote arm. If we are using Cartesian mapping and joystick for input then a small displacement of the joystick in the x-direction will can the remote arm to move at a low speed in the x-direction. A larger displacement of the joystick will increase the rate. Rate control can be used both in joint-to-joint mapping and Cartesian mapping along with many different input devices.

Position control allows the operator to specify the location of the remote arm with the input device without regard to speed. Position control is ideal for miniature arms and joysticks. When the operator moves the input device to a new position the remote arm immediately tracks the movement. If the input device is moved to quickly then the remote may have a velocity limit which will cause the remote arm to lag behind the input device.

Both rate control and position control have been well studied. In [KTES87] position control and rate control are compared. Experiments were conducted to acquire the completion times of simple tasks using both control methods. The study determined that in most cases position control produced better results.

2.2.2.3 Bilateral Control (Force Reflection)

A very popular method of control is *bilateral control*, also called *force reflection.* Using this control method the operator feels the contact forces of the remote manipulator and its environment. If the remote arm presses against a wall, the operator will feel a proportion force in the input device. This method increases the level of telepresence by giving the operator another sense of the task environment. Force reflection is ideal for many tasks, but it is also complicated to implement and very expensive. Under force reflection the host computer must not only control the remote arm, but the input device as well. If the input device is a miniature arm, each joint will have a motor, thus making it a small robot arm.

There is much research in the area of force reflection, both in theory and implementation. See [AS89, HAN89] for more detailed information. Force reflective telerobotic systems are mostly limited to research laboratories.

2.2.3 Input Devices

Although the operator will interact with a telerobotic system through the host computer (i.e. keyboard) most teleoperation (manual control) will be done with the input device. Because the input device the is the primary interface tool, its design is critical. There exist many different input devices, both commercially available and custom made. In [FDS90] Fischer *et al.* present criteria for specification and design of input devices for teleoperation. In this section we look at three popular devices used for teleoperation: the joystick, the force/torque ball, and the miniature arm.

2.2.3.1 Joysticks

The joystick is a device that has many uses; from airplane control to video games. In the most general case a joystick consists of a base, a stick for grasping, and possibly some buttons. Some joysticks will only span two dimensions, while other will span three dimensions. Joysticks can usually operate in one of two modes. In the first mode the joystick returns to a "center" position when released. Thus the user will feel a force that is opposite the direction of movement. When used for teleoperation this mode is useful for rate control (see Section 2.2.2.2). In the second mode of operation the joystick maintains the current position. In this mode the user can position the joystick in space and it will remain there until it is moved again. This mode is applicable to position control using Cartesian mapping (see Section 2.2.2.1 and Section 2.2.2.2)

2.2.3.2 Force/Torque Balls

Force/torque balls offer another type of input for teleoperation. The force/torque ball is simply a ball, the size of an orange, mounted on a base (see Figure 2.1). The base may also have some buttons for input. When the ball is grasped it can sense forces in the x, y, and z directions, as well as moments around each axis. This input device serves well for rate control using Cartesian mapping (see Section 2.2.2.1 and Section 2.2.23.2). As the user pushes the ball in the x-direction, the remote manipulator will also move in the x-direction. The greater the force applied to the ball the greater the speed of the remote arm. The torque input from the ball is used to determine the orientation of the end effector.

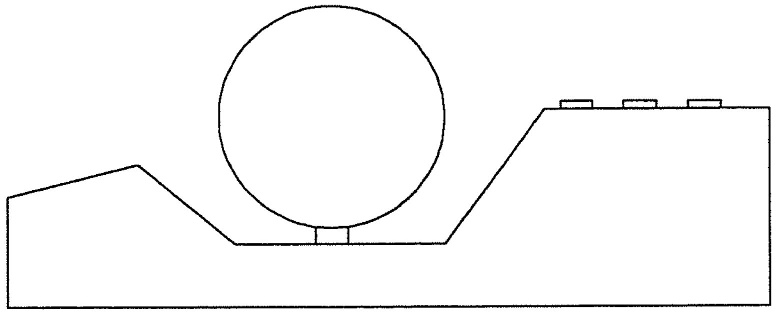


Figure 2.1 The Force/Torque Ball

2.2.3.3 Miniature Arms

The miniature arm is a useful device for teleoperation. In this case the input device is simply a small version of the remote manipulator. All movements of the miniature arm will correspond to similar movements in the remote arm. This device is preferred over others because it is very easy to use. The operator simply positions the miniature arm based on the current position of the remote arm, whether it is viewed directly or through television monitors. The miniature arm can be used for both joint-to-joint mapping and Cartesian mapping. Although the miniature arm could be adapted to work with rate control, it is ideal to use it with position control.

2.2.4 Remote Manipulators

The remote manipulator (or remote arm) is the device that interacts with the environment. Under teleoperation the remote manipulator becomes an extension of the operator. Along with many other properties the remote arm should be articulate, that is it should be able to realize several configurations. The remote arm should also have the appropriate reach and payload capacity for the task requirements. These aspects and more are discussed thoroughly in Chapter 3.

2.2.5 Visual Feedback

Visual feedback is an import element of a telerobotic system. If the operator cannot directly see the work space, visual information allows the operator to view the environment. In this situation the operator will make all movements and control decisions based on the visual feedback. It is important that the visual feedback provide an accurate representation of the task space and that it is presented in real time. The less accurate the visual feedback the slower the operator will be able to complete tasks [KS89]. If the visual information is delayed in some way the operator will have a difficulty compensating for the time delay [BKV90].

Visual feedback can be as simple as a closed-circuit television monitor connected to a remote camera. One camera/monitor pair can be used to view the task environment from a specific perspective. Several camera/monitor pairs can be used to obtain a variety of perspectives. In some cases it is desirable to attach the camera to the end of the remote manipulator to directly view the objects affected under teleoperation. Another option is provide another remote arm with a camera as the end-effector. This allows one camera/monitor pair to view the task from several different positions.

Using a single camera/monitor only provides monocular vision, thus it lacks depth perception. More sophisticated vision systems provide binocular vision by using two cameras that are directed at the environment in slightly different configurations [KS89]. The output signals from the cameras are fed back to either two monitors or a special binocular vision headset. This method allows the operator to perceive depth and thus provides a more accurate representation to the task environment.

Chapter 3
Telerobotics for Highway Maintenance

The main objective of this study is to investigate the applicability of telerobotics technology to highway maintenance automation, with the goal of performing the duties now performed by people on the road quicker, better, and with people removed from the hazards of the road. One important issue to be addressed is to identify specific areas within the scope of the Caltrans highway maintenance program in which it would be feasible and beneficial to apply telerobotics technology. A survey of Caltrans' Highway Maintenance Manual has led to the identification of the following areas of interest. The estimated maintenance costs in some areas are also listed based on data taken from Caltrans' maintenance budget of fiscal year 1990/91.

1. Roadside tree/bush trimming and brush removal ($17.4M for landscaped category and $7.5M for non-landscaped category).

2. Selective herbicide spraying for long range (up to 30 ft. reach) roadside weed control ($11.5M for landscaped weed control and $10.9M for non-landscaped weed control).

3. Sign and guide marker washing.

4. Safe pickup and disposal (sweeping, vacuuming, collecting, transporting) of hazardous material in hazardous spill clean-up.

The above list represents maintenance tasks which are the most promising candidates for telerobotics application. Other areas of potential can be identified later as we gain more experience with telerobotics technology. The basic capability that telerobotics can deliver in all these cases is that maintenance tasks can be accomplished by one worker in the cab through robotic arm teleoperation.

Many important technical problems need to be addressed. They include the following:

1. vehicle motion and dynamics;

2. lift capability;

3. precision capability; and

4. user control and human factors.

3.1 Feasibility

A fundamental issue in this research is: what is the feasibility of a telerobotics system for highway maintenance at this time? To answer such a question, an assessment of the current "state of the art" technology capabilities relative to the requirements of a highway maintenance system is necessary.

As cited later in this section, industrial robotics have evolved for the past 31 years and the success has been impressive. Autonomous industrial robots operate for 24 hours at full capacity for thousands of hours before any maintenance on the system is needed. Therefore the reliability in robots has proven to be excellent. Similarly, through the fields of space and sea applications, development of fully functional and environmentally resistant robotic/telerobotic systems have been successful. The current level of technology in controllers and software for a telerobotic system have also evolved into user friendly and responsive systems.

The issue of safety is also a major concern of robotic systems development history. Since the beginnings of robotics, schemes for "safe" work areas has been seriously addressed. The idea of a user defined safe workspace can now be realized through innovative software means. Development of a functional and safe telerobotic system is essentially a reality.

3.2 Benefits

The most important aspect of telerobotics for highway maintenance is "getting people off the road" and into a much safer position. By appropriate highway maintenance automation, not only are lives saved but economic benefits can also be realized.

With automation, improvement of task quality is also possible. By allowing the operator of a highway maintenance system to focus on higher functions of maintenance, repetitive operations by robotics can be achieved in a much more accurate and consistent manner. The speed of maintenance operations can also be improved because the operator can focus on the "big picture", of a maintenance task, and let the telerobotic system perform repetitive work at full speed.

In conclusion improved safety, quality, economics, and speed of highway maintenance operations are feasible when applying a fully functional telerobotics system. In order to realize a telerobotic system for highway maintenance, the basic characteristics of an "ideal" system must be determined. These characteristics are defined and analyzed below.

3.3 The Ideal Telerobotic System

3.3.1 Controllers and Software

At the crux of a telerobotic system lies the controller and corresponding software. The controller is essentially the computer that coordinates the entire telerobotic system. The controller handles everything from high-level processing to low-level processing. In many systems there will be more than one controller (computer). At the lowest level the controller contains a tight control loop that determines the position and movement of the remote manipulator. The low-level also includes the ability to read the signals from the input device and the force sensors. At the higher levels the controller will execute operator commands and provide a user interface.

The controller is basically a computer, therefore high-speed processing and large amounts of memory are desirable in the ideal system. The controlling computer should be able to interface with all of the elements of the telerobotic system, thus it best to choose an architecture that supports a large amount of "off the shelf" interface products. The most popular computer bus today for real-time interfacing is the VME bus. If the system you want to use does not have a VME bus, adapter cards are available for many bus types.

The operating system should also be suited for interfacing and real-time processing. Thus is it desirable to use a real-time operating system such as VxWorks or QNX. All of the standard software development tools are valuable to any software project. Hardware debuggers may also be useful during the design phase of a telerobotic system.

Most commercially available manipulators also come with a proprietary controller system. These controllers usually provide mechanisms to program the arm to carry various tasks. Unfortunately most of the systems are designed to simply repeat programmed tasks. For teleoperation it is necessary to have direct access to the low-level controller so that the trajectory of the remote arm can be updated in real-time to follow the position, speed and acceleration of the input device. When choosing a manipulator a major consideration will be whether or not it allows an external computer to have access to the low-level control loop. If the manipulator does not allow access to the low-level control loop or does not allow real-time path modification, then the controller will have to be replaced [BH88].

3.3.2 Manipulators

In this section the mechanical parameters for the structural design of the ideal telerobotic manipulator is discussed. Various basic coordinate frames and arms with different degrees of freedom are used for robotic manipulators. Their selection in the design stage is dictated by specifications on the basic parameters of the robot. In this case, the ideal system, the robot is subject to a broad range of specifications. Therefore, the specifications are somewhat general but do define the necessary properties of the mechanical manipulator for this ideal situation.

Before discussing the principal structural parameters of the ideal highway maintenance manipulator, it is important to describe eight basic factors governing the mechanical structure: [Riv87, Sto85]

1. **Payload**
2. **Mobility,** i.e., number of degrees of freedom (DOF)
3. **Workspace**
4. **Agility** (effective speeds of execution of prescribed motions)
5. **Accuracy** and repeatability of positioning in various DOF
6. **Structural stiffness** (compliance's) payload to weight ratio
7. **Environmental resistance**
8. **Economics** (cost, reliability, maintainability, etc.)

Many of these parameters are interrelated, depending on the point at which they are being defined. However, some standards need to be set for the ideal manipulator and these parameters generally describe it.

The rated **Payload** is the maximum mass the robot can handle in any configuration of its linkages. Of course, some orientations allow the robots payload to be greater than others but with different orientations it is more appropriate to discuss torques on the manipulator instead of the masses. However, mass is a basic parameter that adequately describes or quantifies a robots payload capabilities. For the most general description, it is necessary to define a high value for the payload, thereby ensuring that the ideal system is flexible and will not be limited in certain applications.

For the four candidate application areas of telerobotics mentioned earlier, the typical payload would be roughly equivalent to that of a man-in-the-bucket system of a maintenance truck. That is, a payload capacity at the range of 400 lbs. would be necessary.

The **Mobility** of a robot is determined by the number of DOF which can be performed by all of its links. It should be noted that the end effector has its own degrees of freedom but are omitted when describing manipulators. A telerobotic system needs at least 6 DOF, which is often found in industrial robots, to ensure full articulation potential of a system.

The **Workspace** of the manipulator is the space composed of all points which can be reached by its arm end or some point on its wrist but not by the end effector. The reason for such a standard is the workspace will change depending on the size and shape of the end effector. The volume and shape of the workspace are very important for this application since they determine capabilities of the robot. The use of a robot might be severely limited for some applications since the workspace usually has voids or "dead zones" which cannot be reached by the robot. Therefore, the workspace should be large enough to perform all tasks necessary for highway maintenance. Yet, the dead zones should be utilized as those that are not being used, e.g., the space occupied by the cab of a truck mounted system.

The **Agility** of the robot is interrelated to many other factors but is primarily the rate at which the robot can accelerate from one position to another. One of the governing factors for agility are the masses of the links, or more appropriately the moments of inertia of the links. The structure should be relatively lightweight to ensure quick motions when necessary yet strong enough to handle high payloads without excessive deflections. A high degree of agility is perhaps not essential for a maintenance system, but for an ideal system it is an important parameter.

Accuracy is important in both point to point and trajectory operations. It can be defined as the difference of the target coordinate and the actual coordinate which the robot reaches. Accuracy and repeatability can be influenced by friction, hysterisis, backlash, compliance in links or joints, joints and drive interfaces, etc.. Accuracy is influenced by the design and kinematics of the linkages, as well as by its payload, types of drives, and overall system configuration. In highway maintenance, a high level of accuracy (i.e. industrial robotics quality) is required for delicate operations; however, a high level of accuracy is not required for most large scale, high payload operations. The concept of slow powerful systems with agility could be accomplished by a dual robot, a manipulator as an end effector on the master manipulator.

Structural stiffness, which is akin to structural dynamics, is characterized by masses and moments of inertia, stiffness, damping constants, natural frequencies, and modes of vibration which is critical for several reasons. Large masses and moments of inertia lead to the need for large drive actuators and not a very responsive system. Low stiffness leads to excessive deflections from the robot in turn decreasing accuracy and repeatability. Similarly, low damping constants and low natural frequencies of the system will lead to problems with oscillations and "overshoot". Of course, all of these factors explain the modes of vibration which are interrelated to the accuracy and repeatability of the system. Although a system with less than adequate stiffness properties can be somewhat compensated for through the software, the best design dictates that the robot have high structural integrity (see Section 3.3). As with the possible high payload needs of a maintenance system, high structural stiffness is of paramount importance.

Environmental resistance is the ability of the system to withstand harsh conditions (i.e., extreme cold, rain, snow, and heat without appreciably decreasing functionality). Similarly, the system also needs to be able to maintain a high level of reliability without being effected by harsh environments.

Economics is always a factor that governs the practicality of any system. Therefore, it is of utmost importance to understand and regard economics as a major parameter in a telerobotic system. Cost, of course, is an important parameter because the price of a system or even sub-systems will affect the overall system. Reliability is an economic consideration within the maintenance costs of operating a telerobotic system. Maintainability is closely related to reliability in that a system that is low maintenance is often also easily maintained. As such, an ideal system should encompass reasonable component cost with a low maintenance schedule. In this section it should also be mentioned that the ideal telerobotic system for maintenance should be able to endure adverse environments without excessive system maintenance (low maintainability).

3.3.3 End Effectors

End effectors usually are designed for specific applications or operations. For a maintenance system, there are a large number of tasks or applications that such an end effector would have to perform. Current industrial systems use an interchangeable end effector scheme where the robot arm selects the appropriate end effector (tool) from a turret housing a variety of tools.

For an ideal highway maintenance system, a selection of end effectors would be necessary. In this way, the system is robust and can accommodate a variety of applications. These tools may simply be stored on the truck or platform. Essentially, the highway maintenance robot or telerobot is one which has high power as well as high accuracy capabilities.

Chapter 4
A Survey of Commercially Available Systems

4.1 Commercially Available Telerobotic Systems

It would be extremely desirable to find a commercially available telerobotic system that could simply mount to the back of a Caltrans truck. However, such systems, for the parameters needed for highway maintenance, have not been discovered in this survey. It should also be noted here that the data shown is representative of the available systems currently on the market. During the initial survey, many robotic systems were found to be very similar if not the same; therefore, we will present representative systems in this section.

In the survey data it is evident that several possible areas exist and should be grouped into telerobotics systems and sub-systems. The basic robotics fields surveyed were:

1. Industrial Robots
2. Hazardous Environmental Robots
3. Space Application Robots
4. Deep-Sea Application Robots
5. Agricultural and Food Service Robots
6. Construction Manipulators

Many of these systems are interrelated and as such they have been combined. The most advanced, as expected, robotic systems are those found in the **aerospace and marine** fields--these fields are very specialized and only a limited number of manufacturers that market commercially available systems exist. The next most applicable area to telerobotics is the **industrial robotics** field. An abundance of commercially available systems exist in industrial robots, more so than any other field. In this survey a relatively small number of industrial robot data is presented but it should be noted that the basic industrial robotic system is the same independent of manufacturer. Finally, the **construction** field offers manipulators with the high payload capability with low cost yet primitive controls. Construction systems rely totally on the dexterity of the operator for repeatability and accuracy. Therefore, evaluation of construction systems using the characterization parameters based on aerospace/marine and industrial systems is difficult.

4.1.1 Aerospace/Marine Systems

These systems already have telerobotic capabilities due to the high technology applications. Schilling Development, a local company, is developing large scale telerobotic systems for primarily marine applications. They have been adapted to space and hazardous waste removal applications.

Controllers
These telerobotic systems generally include a controller with sophisticated software that enables operator override during an operation as well as teaching functions. The control arm may also come with force feedback capability. Force feedback control enables the operator to "feel" the remote arm's interaction with the environment. This function, called telepresence, gives the operator a true sense of the interaction forces. However, it is presently quite expensive and still in the development stages.

Environmental resistance
The systems for marine applications are subjected to severe environmental effects, namely corrosion and high pressures from great depths of operation. Thus the use of expensive corrosion resistant and high strength materials drives up the price of such systems.

End effectors
The majority of end effectors available for marine applications are four bar linkage, two jawed grippers. In addition, interchangeable tooling end effector systems that will enable the system to change end effectors in adverse environmental conditions are being developed.

4.1.2 Industrial Systems

The industrial robotics field is the most developed robotics field to date. These systems are built for repetitive tasks in the application of automated manufacturing. They are used for high accuracy (e.g. servo motors with encoders, for absolute position control) throughout the industrial robotics industry. However, a fundamental problem exists with these systems: low payload capability. The need for high payload is often not an issue in automated production systems. The low payload capability is also due to the basic design of these robots. The large forces transmitted through the joints of the robot cause extremely large servo motors to be used. Servo motors are limited by the amount of torque necessary to resist the internal loads induced by the payload. At some point, the torque at a joint will be high enough to require a servo motor disproportionate in size to the robot arm. Due to the nature of automation, industrial robots are designed for high accuracy not high payload applications.

The **service record** of these robotics systems are "the best in the business" because they have been used so extensively in factory automation. Panasonic U.S.A. quotes a 15-25,000 hour service period between maintenance checks for a 24 hour a day work schedule. This impressive reliability is partially due to a controlled environment. The environment for a highway maintenance system is not so kind to the robot. Therefore, some modification would be necessary to ensure adequate environmental resistance if industrial robots are used.

The **controllers** for industrial robots lend themselves easily to telerobotic operations. A typical controller includes operator interrupt and path teaching capabilities. These controllers typically have a joystick input from the operator or a direct coordinate input via. a teach pendant. Although these controllers are easily adaptable, they are primarily designed for repetitive tasks, such as those in automated manufacturing

End effectors for industrial robots are totally based on the particular application. Most end effectors are either pneumatically actuated or driven by a servo motor. Pick-and-place grippers are the same as those being used in the aerospace/marine fields. Much research and development has been done for end effectors in the industrial robotics field. As such, an existing "off the shelf" end effector could be selected or modified for some highway maintenance applications.

4.1.3 Construction Systems

The construction industry offers manipulators that can easily meet high payload capabilities and severe environmental requirements for a highway maintenance telerobotics system. However, these manipulators are controlled by "primitive" controllers relative to the controllers for the other systems in this survey. In fact, construction manipulators are not robotic systems at all. Construction manipulators are predominantly hydraulic actuated and not servo motor driven. The operator must manipulate levers that control the rate of fluid into the hydraulic cylinder of a link in the manipulator either directly or via a proportioning valve. The pressure in the system may be held constant so that if the operators level of dexterity is not high, the manipulator moves in discrete steps and not a smooth continuous motion, sometimes characterized as a "jerking" motion. However, IMT Inc. manipulators use proportioning valves to control the fluid pressure to each actuator so that the system operates more smoothly. IMT also has fine actuator controls which allow the operator to position the manipulator at a slower and more accurate manner. Other proportional control valves are available which could be used for a highway maintenance robot.

Typically, an operator controls one lever at a time, each lever only controls one actuator on the manipulator. Thus, in order for an operator to control a construction manipulator similar to an industrial robot, he must operate multiple levers at one time. Obviously, the level of dexterity of such an operator would need to be excellent, almost an art form because computer control is absent. Comparison of accuracy and agility of these systems with the others in this survey is not applicable because of the direct operator input dependence.

Construction manipulator technology has been tested and proven over the past 50 years. The reliability of these systems is quite high. Similarly, construction systems have been developed to operate in harsh environments so, their environmental resistance is excellent.

End effectors for construction manipulators often include their own DOF which is usually the wrist section of a robotic manipulator. For this reason, in this survey, the DOF for the construction manipulators is stated as only 4. Construction end effectors are mainly used for digging, scraping, drilling, and demolition applications--these applications require high payload capabilities with accuracy of +/- 100mm or more depending on the operators skill.

4.2 Survey Evaluation

From a **controller** standpoint, aerospace/marine and industrial robotics systems have the best controllers and require little or no modification for teleoperation. In fact, the basic controller scheme used for both aerospace/marine and industrial robotics is the same. However, the input devices and software for aerospace/marine systems are more evolved. The more evolved systems use operator input devices that minimize the required dexterity level so that the operator can concentrate on higher order operations. Similarly, the more similar the controller is to human movement, the higher the quality of operator control.

In construction systems, the controllers currently used are not conducive to teleoperation. Teleoperation for a construction manipulator would require extensive research and modification to the system. Most industrial controllers also require modifications, mostly software, to be used in teleoperation. The best choice of a controller scheme for teleoperation is definitely those used in the more advanced aerospace/marine systems.

Of course, controllers are only one parameter in a telerobotic system. Evaluation of these systems with respect to the ideal highway maintenance system hardware is based on the seven other basic factors mentioned earlier. Mobility is the next factor and all of these systems are already capable of 6 DOF. However, these robots are not capable or easily adaptable to highway mobility. Highway mobility is more complex because the robotic system must be portable. One indication of highway mobility is in the payload to weight ratio. A high value for the payload to weight ratio relates to a system with high mobility.

Even though the aerospace/marine systems have the highest value for the payload to weight ratio, they are also the most expensive. Therefore, construction manipulators are the most adaptable for highway mobility. In fact, the IMT manipulators surveyed are truck mounted systems, making them the best for a highway maintenance system.

In the survey data, **workspace** varies from system to system so evaluation of maximum reach is based on either maximum vertical or horizontal reach, listed in Table 4.1. A comparison of the average maximum reach among systems clearly shows that the construction systems have the largest reach (see Figure 4.1). Figure 4.1 also shows that the construction systems have the highest average payload capability. In a similar manner, system vs. cost is graphed on figure 4.1, indicating that construction systems have the lowest cost.

Table 4.1 Telerobotic Systems Survey Data

Company	Model	Payload (Kg.)	Accuracy (+/- mm)	Mobility DOF	Reach (mm)	Weight (Kg)	Payload/Wt	Cost ($x1000)
Aerospace/Marine								
Schilling Dev.	Titan II	108	--	6	1940	79	1.37	140
Schilling Dev.	Titan 7F	113	--	6	1980	67	1.69	112
Industrial								
Kawasaki Heavy Industries Ltd.	UZ-100	100	0.3	6	--	1300	0.08	98
	EH120	120	0.5	6	--	1500	0.08	89
Panasonic	AW-8100	100	0.4	6	2580	1600	0.06	111
	AW-8060	60	0.2	6	2840	1200	0.05	92
Motoman Inc.	K150S	150	0.5	6	2390	1600	0.09	107
	K60S	60	0.3	6	2000	980	0.06	99
	K100S	100	0.5	6	2387	1600	0.06	98

Table 4.1 Continued

Company	Model	Payload (Kg.)	Accuracy (+/- mm)	Mobility DOF	Reach (mm)	Weight (Kg)	Payload/Wt	Cost ($x1000)
Construction Equipment								
IMT Co. Inc.	2115	636	--	4	4560	544	1.17	10
	6425	680	--	4	9450	1792	0.38	20
	20017	5216	--	4	5180	3467	1.50	33

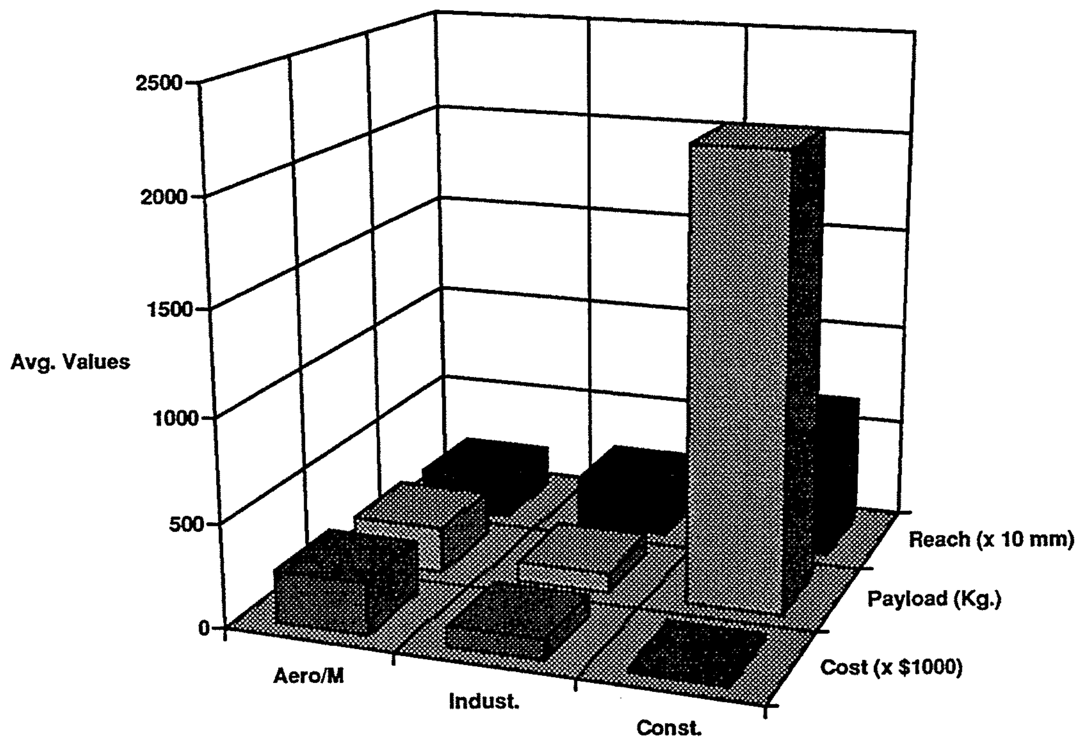


Figure 4.1 System Comparison

Based on the reach, payload, and cost parameters it is apparent that the construction system is best for an ideal highway maintenance manipulator. However, the controller issue still exists in that computer controls are not part of the construction systems. As such, it does not make sense to compare accuracy between the other two systems.

If the aerospace/marine and industrial systems are compared, see Figure 4.1, the industrial systems have lower average price, higher average reach, but lower average payload. For environmental resistance, the aerospace/marine systems are far better.

Another parameter to consider is the structural stiffness. One possible way to quantify this parameter is by the payload to weight ratio of the manipulator. Typically, the higher the ratio value the higher the strength of the manipulator. As stated earlier, if a structure is rigid or stiff then it will have high natural frequencies and problems like overshoot during operation are not substantial. The average values of the payload to weight ratio from Table 4.1 are shown in Figure 4.2.

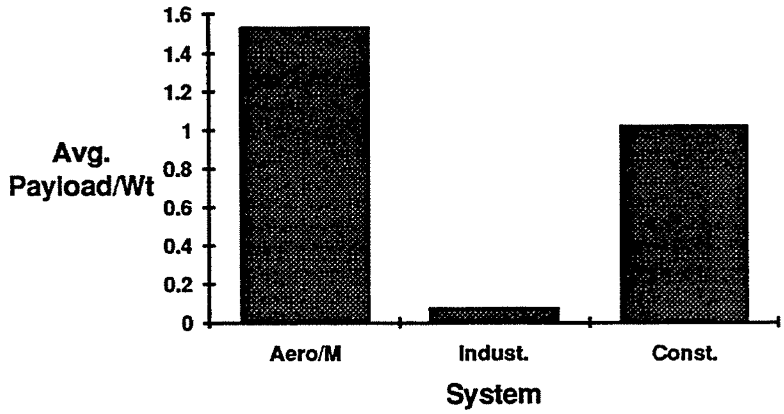


Figure 4.2 Average Payload to Weight Ratio

The aerospace/marine systems have the highest effective stiffness of all the systems surveyed because of the requirements of such a system. As an overall evaluation, the construction system manipulators are the best when compared to an ideal highway maintenance system. However, they lack the flexibility of automated control. The best robotics systems are the aerospace/marine systems but they lack the large workspace needed for highway maintenance. Aerospace/marine systems are also the most expensive but they have the most user friendly control systems.

Therefore, an "off the shelf" flexible highway maintenance telerobotic system is not commercially available based on the manufacturers in this survey.

Chapter 5
The Telerobotics Testbed

To demonstrate the capabilities of a telerobotic system we have developed an experimental testbed. Our goal was to build a small-scale system that uses standard components whenever possible. This approach not only allowed us to quickly implement a working system to gain valuable first hand experience in telerobotics, but also enabled us to keep research and development costs down. The design and implementation process revealed many practical considerations that can be directly applied to large-scale systems. The testbed also provides a foundation for future telerobotics research and application development.

This chapter presents a detailed description of the telerobotics testbed. First, we describe the overall system configuration. Second, the system hardware is presented, including the input device, the controlling computers, the robot arm, and the grippers. Third, we cover the system software; concentrating on both the low-level drivers and the high-level interface. In the final section we discuss the demonstration tasks.

5.1 System Configuration

Before we present the hardware and software details we will review the overall system configuration. The testbed is composed of five basic components: the input device, the host computer, the robot controller, the robot arm, and the end effector (see Figure 5.1).

The user guides the robot arm through the input device and the host computer. The input device used is manufactured by Schilling Inc. and was provided as a donation. This input device (as shown in Figure 5.2) is kinematically similar to the Puma 560 which is the remote robot arm used in our testbed. The primary function of the input device is to allow the operator to manipulate the position of the robot arm. The tip of the input device is equipped with three buttons (see Figure 5.2), each of which can be programmed for different actions. For our purposes we dedicate one button for an engage/disengage function. While disengaged, movements of the input device have no effect on the remote arm. Once the engage/disengage button is pressed, the remote arm will track the movements of the input device. A second button is used to activate the end effector. If the end effector is a gripper, pressing this button will cause the hand to open/close if the hand was previously closed/opened.

The host computer provides another means for the operator to control the robot arm. In general, the host computer is used to switch between different modes of operation and to execute high-level tasks. For instance the user might want to switch from a joint-to-joint mode (motions of the remote arm joints track those of the input arm joints) to a Cartesian mode (end effector motion of the remote arm tracks that of the input device). The user can also execute high-level tasks such as teach and play back or a preprogrammed operation. The host computer is also used to interface the input device to the robot controller. At each joint in the Schilling arm there is a potentiometer. By connecting the potentiometers to an ADC (analog to digital converter) board we can read the absolute position of each joint. Not only must we supply the host computer with an ADC board, but we must have software drivers to process the input and a connection to the robot controller. In our testbed we connect the host computer directly to the robot controller via a serial line. This connection allows the host to send position and end effector commands to the robot controller.

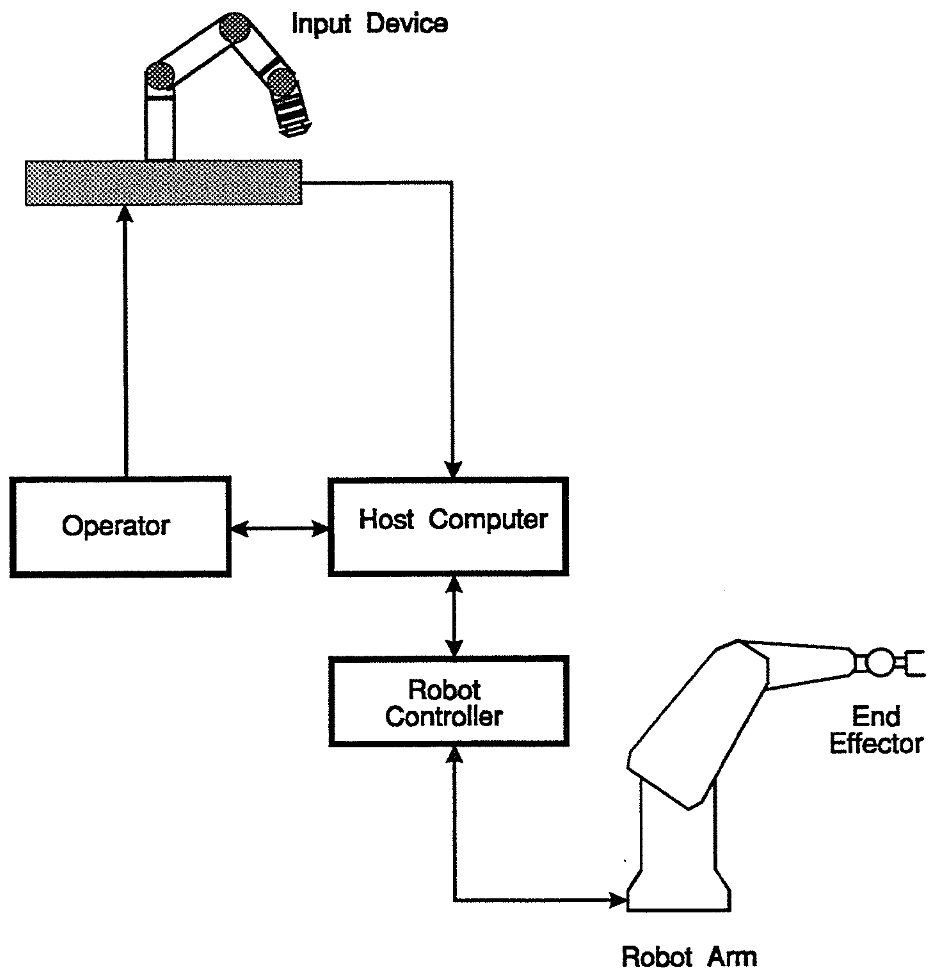


Figure 5.1 System Configuration

The robot controller used in the testbed is the Unimate computer/controller running the VAL II language. Together the Unimate controller and VAL II coordinate the movements and actions of the Puma 560 robot arm. A set of programs written in VAL II enables the controller to accept commands from the host computer. These commands are interpreted and then issued to the robot arm. Conceptually the VAL program is a simple infinite loop that moves the arm to each commanded position read from the host. The VAL program also activates the end effector as commanded by the host.

The Puma 560 robot arm is used as the remote arm to interact with the environment. The Puma is a widely used research tool and serves well for a small testbed as ours. The work envelope of the Puma, however, is quite small and poses some limitations on telerobotic tasks. For the purpose of demonstrating the feasibility of telerobotic technology for highway maintenance, we have designed and fabricated two end effectors that attach to the Puma 560. The first end effector is a simple parallel gripper that can grasp small objects. The second end effector is a clipper that is used to cut thin sticks and branches. The end effectors are discussed further in Section 5.2.4.

It should be noted that the operators only feedback is from directly viewing the task space. This does not present any problem other than the input device and host computer must be relatively close to the task environment. For remote work-site operation, a vision feedback system must be used. A more realistic system might have some sort of binocular vision as discussed in Chapter 2.

5.2 System Hardware

The system hardware forms the foundation of the telerobotics testbed. In this section we will examine each component mentioned in Section 5.1.

5.2.1 The Schilling Input Device

The Schilling Inc. miniature manipulator serves as our input device. Because the Schilling arm is kinematically similar to the Puma 560 the task of integrating it into the system is simplified. The kinematic similarity also eases the difficulty of remote manipulation because movements in the input arm space correspond closely to movements in the remote arm space. Figure 5.2 illustrates the Schilling arm.

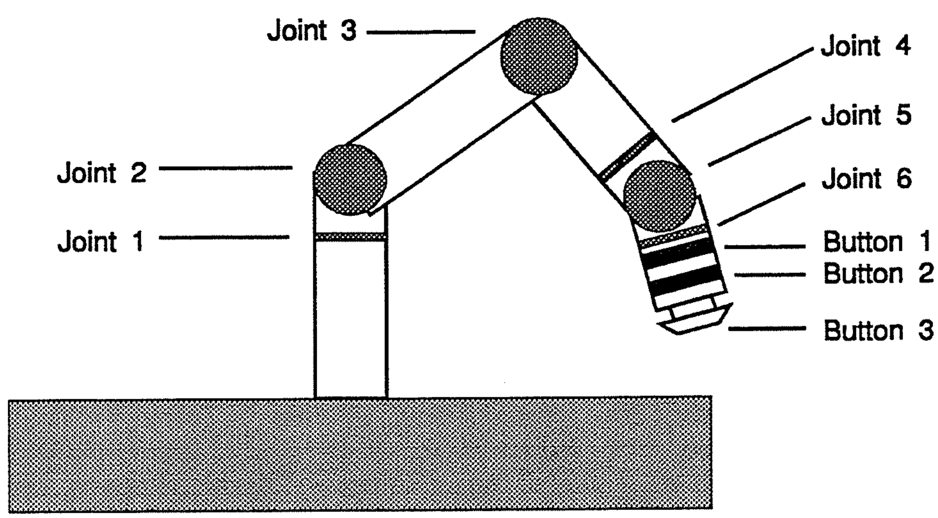


Figure 5.2 The Schilling Arm

Like the Puma 560, the Schilling arm has 6 DOF (degrees of freedom) consisting of three positional joints and three orientation joints. Just as the Puma has joint limits so does the Shilling arm. Table 5.1 lists the Schilling arm joints and there respective ranges. Although the joint ranges of the Puma and the Schilling arm are not identical, they are close enough to make almost all of the Puma work envelope accessible. Each joint of the Schilling arm is equipped with a potentiometer that supplies absolute position data.

Joint	Range (in degrees)
1	0-270
2	0-270
3	0-270
4	0-330
5	0-180
6	0-330

Table 5.1 Schilling Arm Joint Ranges

The Schilling arm is mounted on an aluminum case. The case contains a 5V power supply, some minor wiring, a power switch, and a connector. The connector provides lines to the six potentiometers (the joints) and the three digital inputs (the buttons). The connector is a standard 37 pin D type. The pinouts are wired specifically for the CIO-AD16JR board for the IBM PC/AT. The CIO-AD16JR is discussed in the next section.

5.2.2 The Host Computer

The host computer is an i486 IBM PC/AT type computer running at 33MHz. The computer has 8 MB of RAM and a 120 MB hard disk. This platform was chosen because of its low cost and wide availability. In addition, the UCD Robotics Research Lab has extensive experience with the IBM PC/AT platform [Las90][Ben90].

To interface with the Schilling arm we use a ADC board, the CIO-AD16JR from ComputerBoards Inc. The CIO-AD16JR is capable of high sampling rates (120 KHz) and is compatible with many popular data acquisition boards (Metrabyte DAS-16, Metrabyte DAS-16/F, and Advantech PCL718). To interface with the Schilling arm we use the 6 of the 8 differential analog inputs to read the potentiometers and 3 of the 4 digital I/O lines to read the buttons values.

One of the two serial ports is used to interface with the Unimate controller. We use a standard RS232C cable running at 19200 baud with 8 data bits, no parity, 1 start bit and 1 stop bit. To allow for high speed communication interrupt driven I/O is used on both the host computer and the Unimate controller, see Section 5.3.

The operating system is MSDOS and the development compiler is Borland C++ 3.1. All the software for the testbed is written in 80x86 assembly and C. The Borland development environment is comprehensive and is easy to work with.

5.2.3 The Unimate Controller and Puma 560

The remote Puma 560 arm uses the standard Unimate controller running VAL II [Uni85a][Uni85b]. The Puma 560 is a 6 DOF (degree of freedom) anthropomorphic robot manipulator, see Figure 5.3. Each joint consists of a servo motor, a potentiometer, and an optical encoder. Like the potentiometers on the Schilling arm, the potentiometers on the Puma provide absolute joint positions. The potentiometers, however, are unreliable for accurate joint positions. Instead the optical encoders are used to provide extremely accurate relative joint positions.

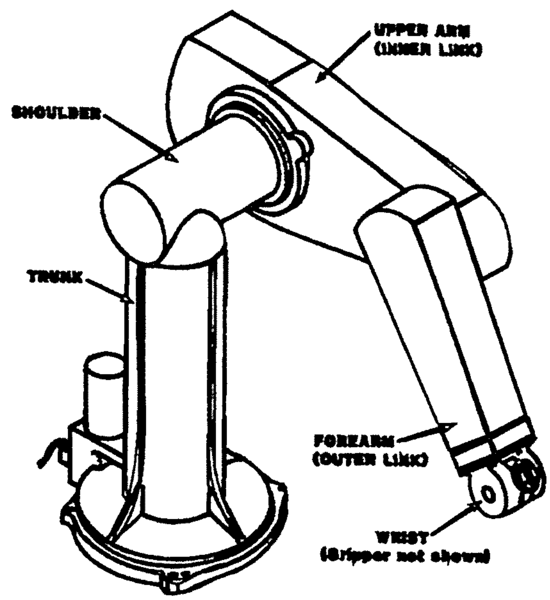


Figure 5.3 The Puma 560

The Puma is traditionally used for industrial applications, but it is also popular for researchers to use it for telerobotics [LLNL91]. The Puma is versatile, but has a limited work envelope. When operating path control in Cartesian mode, we should be aware of the problem of kinematic singularities. There are basically two types of singularities: joint limited and mathematical. Joint limited singularities occur when the commanded end effector Cartesian position requires a joint to move beyond its limit. Mathematical singularities occur when the inverse kinematic solution for a given Cartesian position is undefined (or infinite solutions exist). This type of singularity occurs when joints 4 and 6 become aligned. This problem can be avoided when we only allow the Schilling arm to manipulate the Puma in joint mode. From a software perspective using joint mode is less complex than Cartesian mode. We further note that the Puma and the Schilling arm are only kinematically similar, not proportional. This means that a straight-line movement in the Shilling arm space may not necessarily produce a straight-line in the Puma arm space, see Figure 5.4. For many tasks this does not present a problem if the operator bases movements on the current position of the Puma arm and not on the Schilling arm.

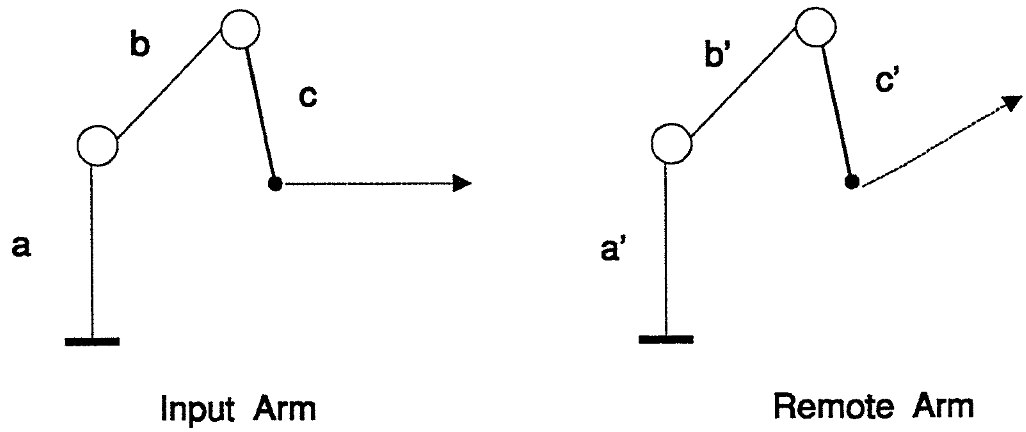


Figure 5.4 The Alignment Problem

5.2.4 The End Effectors

Although the existing PUMA end effector is a limited travel gripper, a more versatile end effector is needed to demonstrate the different capabilities of the telerobotic system. Two basic operations have been chosen for the demonstration: pick-and-place and a clipping.

An end effector is a device attached to the end of a robotics manipulator system which will perform one or more functions, in our case pick-and-place and clipping. The term "end effector" is used interchangeably with the term "gripper" (a tool mounted on the manipulator wrist), but "end effector" covers a wide range of movable tooling devices, not just grippers.

Grippers are usually associated with industrial robots. Grippers can perform pick-and-place operations or hold tools during other operations. In most cases, grippers are designed to perform grasping operations through the use of magnets, suction cups, or articulated mechanisms. The gripper must also be able to grasp objects and hold them without damaging them, so the gripper system must be designed to perform without exerting excessive force.

Many gripper designs perform four actions: parallel-jaw, two-finger, and multi-fingered gripping. Parallel-jaw grippers contact the work piece over a relatively large area, bringing two flat surfaces together on opposite sides of the object being grasped (see Figure 5.5).

For both the pick-and-place and clipping operations, simple parallel-jawed type grippers are used. The PUMA robot is the test bed for the demonstration and the design of the grippers are subject to the constraints of this system. One constraint is a maximum weight of the gripper and payload of 5 lbs.. The other determining factor for operation is the use of a pneumatic actuator for the gripper.

5.2.4.1 The Pick-and-Place Gripper

This pick-and-place operation for the demonstration requires the gripper to pick-up 1.25 inch sq. blocks, yet a gripper with more versatility is preferable. As such, an existing Robohand Inc. RP-100 pneumatic actuator is used for this operation because it has a linear stroke motion, as opposed to two and three bar linkage actuators. The RP-100 uses a pneumatic cylinder as the primary actuator which rotates two spur gears. The two spur gears in turn move the jaw mounts in a linear horizontal motion (see Figure 5.5). As a safety factor, the maximum force the actuator can deliver is 7 lbs. at the jaws. This ensures that the object being grasped will not be damaged during the pick-and-place operation.

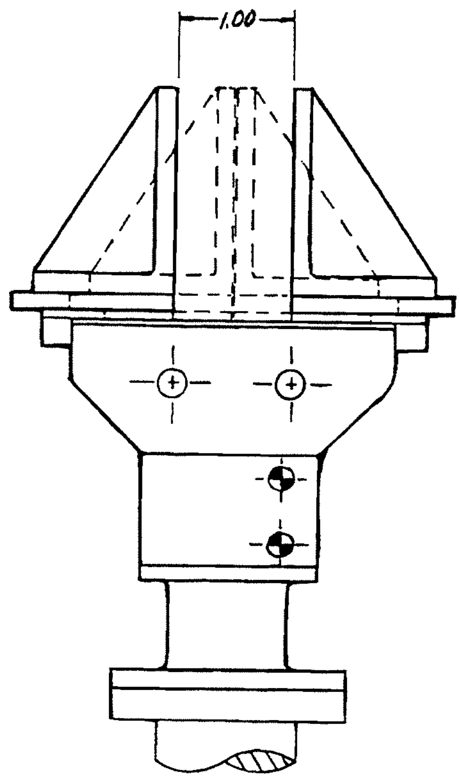


Figure 5.5 The Pick-and-Place Gripper

In the interest of versatility, adapter plates have been built to allow adjustability of the jaws an extra 1 inch. This effectively gives the gripper a total stroke of the 2 inches by means of extra bolt holes for the 0.25 inch slots in the base of the jaws.

The jaws are very similar to those used on the Schilling Titan grippers (see Appendix A). These type of jaws are tapered at the end, or at the tip of the fingers, which gives greater sensitivity for small objects being moved. Small grooves are milled in the faces of the jaws to enhance the gripping friction between the opposite sides of the object being grasped. The jaws are mounted to the adapter plates which are mounted to the actuator.

Unfortunately, the RP-100 actuator can not be directly attached to the PUMA robots wrist therefore, design of an adapter base is necessary. The adapter base simply is a mechanical interface between the two bolt patterns for the actuator and robot. A clearance of 0.80 inches is necessary between the bottom of the actuator and the top of the robot wrist for tooling clearance.

With the gripper assembled and the air lines from the robot attached, the gripper performs its operations satisfactorily. Although the gripper design for the pick-and-place operation is straight forward, the clipping operation utilizes a parallel-jaw mechanism with shearing edges instead of flat edges.

5.2.4.2 The Clipper

For the clipping operation, another type of end effector or gripper type system needs to be designed. In this case, a simple shear type device will serve the purposes of the demonstration. As such, the clipper design is very similar to that used in industrial robots for grasping. This clipper utilizes a three-bar linkage to close the jaws (see Figure 5.6).

The jaws are simply two offset plates with steel cutting edges, very similar to everyday scissors. The jaws are connected to the actuator linkage by two small intermittent linkage bars. Due to the relatively small stroke of the pneumatic actuator, the linkage bars are designed so that the jaws will open to approximately 1.5 inches (see Figure 5.6).

The actuator used is modified from an old gripper design. The decision to modify the actuator was made based on the interests of time and fabrication. With a total stroke of only 0.38 inches, the jaw radius and linkage bar lengths are optimized so that the jaws will open as wide as possible. Similarly, the angles through which the jaws and linkage bars swing limit the shearing force to approximately 10 lbs.

The limited shearing force is in the interest of safety and dictated by the limited stroke of the actuator. As such, the clipper effectively shears small twigs and small, yet low strength, members effectively.

Some question has been posed as to the further development of a clipping end-effector in that another type of shearing mechanism could be designed. This question is based on the amount of dexterity the operator must have in order to cut objects consistently in a plane. With this in mind, an end effector with multiple shearing edges or a rotating blade is an alternative that would solve part of the dexterity problem. However, another possible solution could be that the software would limit the motion of the clipper to a pre-defined plane, thereby reducing the level of dexterity of the operator so that he can concentrate on higher order functions.

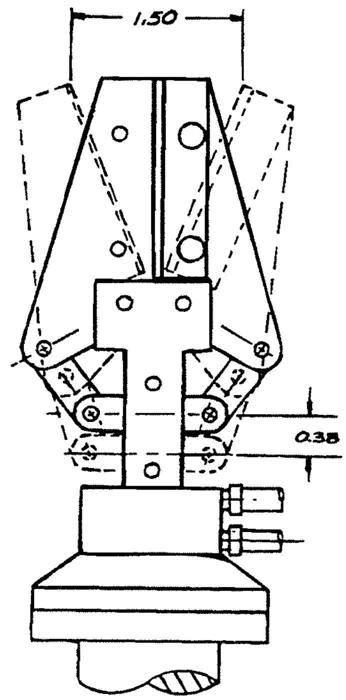


Figure 5.6 The Clipper

5.3 System Software

The telerobotics testbed is controlled by a set of programs that run on the Unimate controller and the host computer. This section describes the source files that comprise the testbed software and shows how to start the telerobotics demo. The complete source code is listed in Appendix C.

5.3.1 The Unimate Programs

The Unimate programs reside on the Unimate controller. These programs. provide an interface between the i486 computer and the Puma robot. Using a serial link the following programs communicate with the i486 computer by accepting commands to move the Puma arm and actuate the end effector. The three VAL II programs are:

- usercd.pg - interrupt driven serial I/O routines (machine code)
- pcg.pg - background process that sends and receives commands over the serial line
- main.pg - initializes variables, starts pcg.pg, and executes the main control loop

5.3.2 The Host Programs

The Host programs reside on the i486 computer. These programs enable the user to manipulate the Puma arm with the Schilling miniature arm. The telerobotics programs are divided into two modules: the device driver and the application. The device driver provides low-level routines to read the potentiometers on the Schilling arm and to communicate over the serial line. The application program provide the high-level user interface by accessing the device driver.

The device driver consists of several source files that are described as follows:

- ddi.h - device driver interface header file
- ddi.c - device driver interface
- ddiinit.c - device driver initialization (loader)
- intentry.asm - software interrupt interface to application programs
- drivers.h - header file that lists the different devices available
- adcdrv.h - ADC (Schilling arm potentiometers) device driver header file
- adcdrv.c - ADC (Schilling arm potentiometers) device driver

- comdrv.h - serial communications device driver header file
- comdrv.c - serial communications device driver
- ibmcom.h - serial communications core routines header file
- ibmcom.h - serial communications core routines

Besides the ddiinit program which contains all of the device drivers, there are two additional device driver programs.

- ddicheck.c - determines whether or not the device drivers are loaded
- ddiquit.c - disables the device drivers

All three programs given above can be built using the "makefile". Simply type:

make

This will generate ddiinit.exe, ddicheck.exe, and ddiquit.exe.

The application program consists of the following source files:

- robot.h - general robot data structures
- kingen.h - forward kinematics for the Puma 560
- trinput.c - the telerobotics application program
- trinput.mak - the application program makefile

As with the device drivers the application program can be built with the makefile, "trinput.mak". Simply type:

make -f trinput.mak

This will generate tinput.exe.

5.3.2 Using the Programs

The programs are easy to use, but you must start the programs in the correct order. Here are the steps needed to run the telerobotics testbed software.

First, you must start the programs on the Unimate controller. To do so, first bring up VAL and make sure that the programs, usercd.pg, pcg.pg, and main.pg are currently loaded. To start the VAL programs first type:

ex usercd

Then type:

ex main

This will enable the Unimate controller to accept command from the i486 computer. Now we must start the programs on the i486. First we need to load the device drivers by typing:

ddiinit

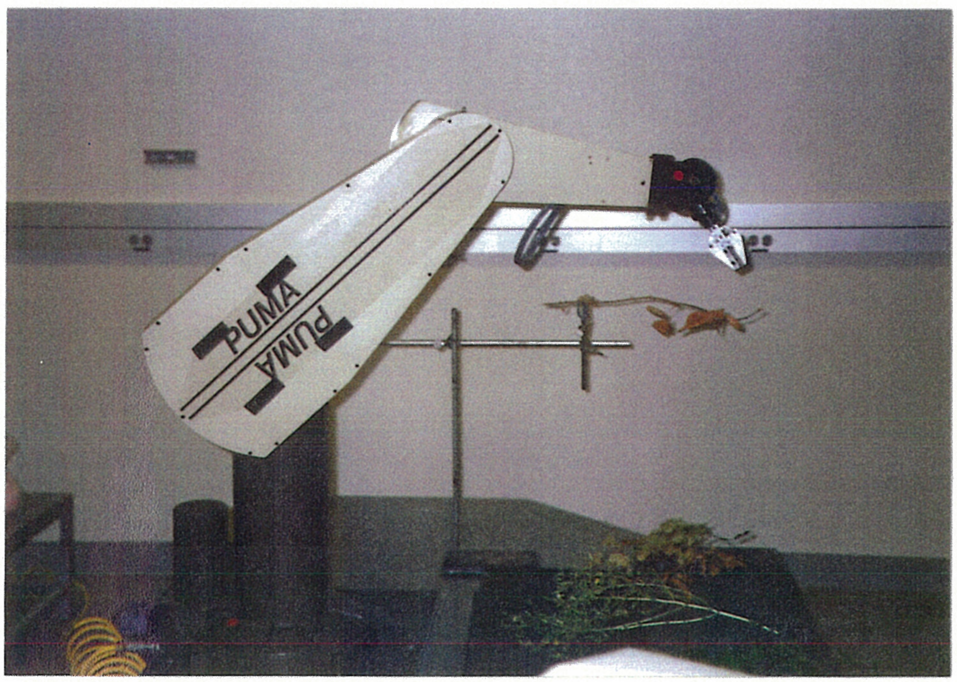
Then we simply run the application program by typing:

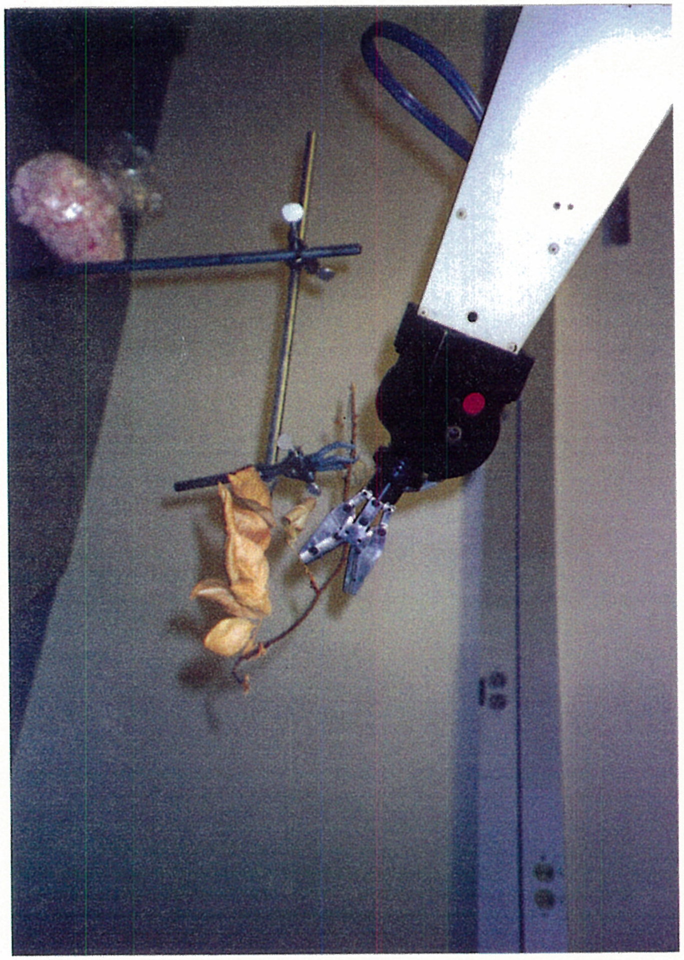
trinput

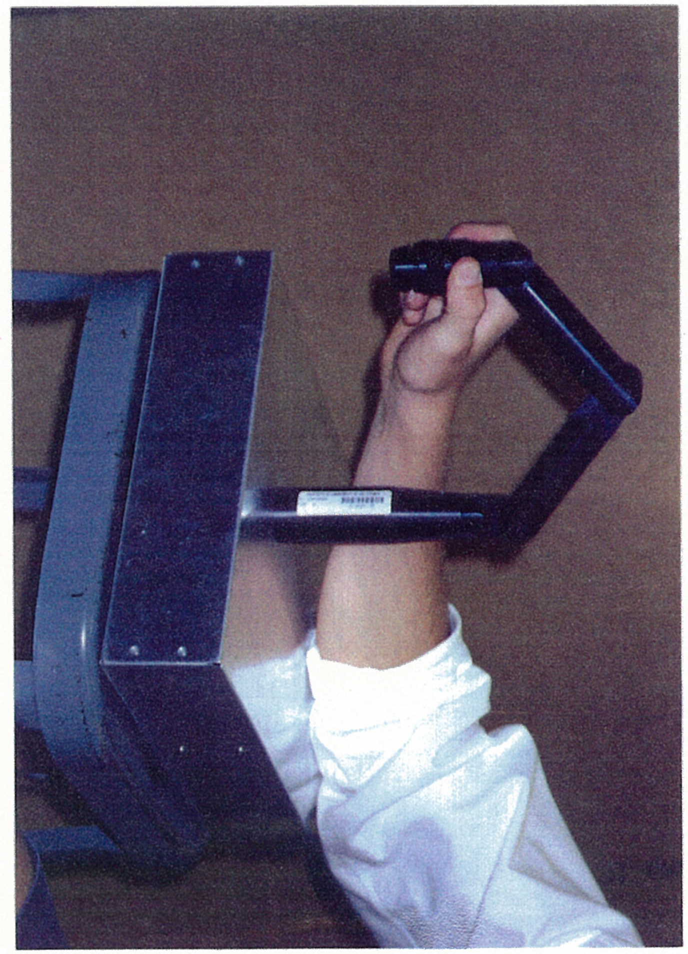
You will be presented with a menu of commands that control system. The commands are straightforward.

5.4 Testbed Demonstration

The pictures in this page and the next show the testbed in action. The remote manipulator is equipped with the clipper. The pictures illustrate the operator using the Schilling input device to clip a branch.







Chapter 6
Conclusions and Future Research

In this study, we have investigated the benefits and feasibility of applying telerobotics technology to highway maintenance automation. The primary goal of automation is to improve safety by removing workers from the hazards of roads, and to perform maintenance tasks quicker, better, and more economically. A survey of Caltrans' Highway Maintenance Manual shows that telerobotics technology is compatible with the following maintenance operations: roadside tree/bush trimming and brush removal, long range roadside weed control, sign and guide marker washing, and safe pickup and disposal of hazardous materials. Experimental investigations with a laboratory testbed have demonstrated that tree/brush trimming tasks can be implemented using a telerobotic system.

Important mechanical structure parameters that are required by an ideal telerobotic maintenance system are defined: they are payload, mobility, workspace agility, accuracy, structural stiffness, environmental resistance, and economics. A survey of commercially available systems has been made to assess their capabilities. Based on the survey data, it is evident that no complete "off the shelf" telerobotic system exists for the highway maintenance parameters defined. The existing commercial systems either are very precise but lack high payload capacity, highway mobility, and large workspace or they are powerful and economical but lack computer control.

It is clear that a hybridized system can be constructed in the interest of retaining the parameters of an ideal telerobotic highway maintenance system while not sacrificing economics. One possible way to hybridize construction and industrial robotic systems is through the use of existing hydrodynamic fluid drive systems. These systems use a electro-servo hydraulic valve to control the hydraulic actuator. The valve is computer controlled and the position of the actuator is fed back into the computer/controller (see Figure 6.1). This type of system is versatile and adaptable to existing construction systems because the existing hydraulic systems do not need modification.

Another possible hybridization of construction and industrial systems is a hydrostatic fluid drive system. A servo motor drives a gear type fluid pump which moves the actuator. A position sensor monitors the actuator motion and feeds information back to the computer. The computer keeps track of the actuator position and the pump, it then sends information to the controller that operates the servo motor (see Figure 6.2). This type of system would allow an industrial controller to be modified to work with a construction manipulator much like that of a hydrodynamic fluid drive system. Through the gear type fluid pump, the hydraulic actuators force, acceleration, and position can be controlled very accurately.

Both of hydrodynamic and hydrostatic fluid drive systems enable a large construction manipulator to operate like an industrial robot. Similarly, the basic controller from an industrial controller system could be adapted to make the system telerobotic. With the combination of industrial controllers and a hybridized construction manipulator, a system could have all the attributes of the ideal telerobotic system.

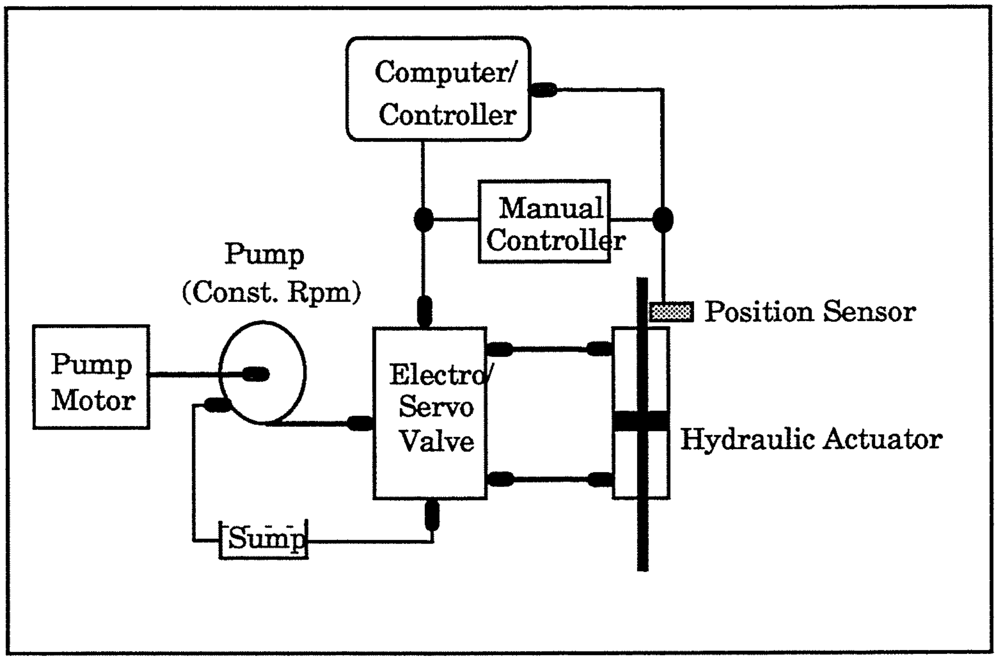


Figure 6.1 Hydrodynamic Fluid Drive System

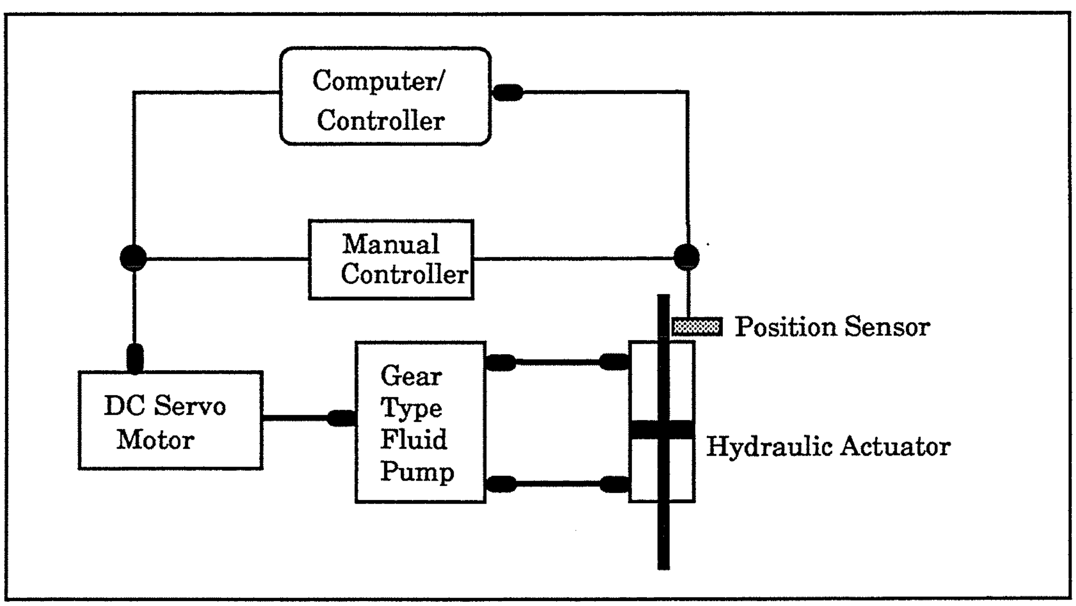


Figure 6.2 Hydrostatic Fluid Drive System

One foreseeable problem is the compliance of the truck or mobile platform the system is mounted on. Even though the manipulator is rigid and the accuracy and repeatability are high, the mobile platform is not rigid. During operations requiring high accuracy, the global accuracy is dependent on that of the telerobotic manipulator and mobile platform's compliance.

A solution to this problem is an end effector that is basically another small robot manipulator with an inertially stabilized platform at its base (see Figure 6.3). This intelligent robotic platform would allow the smaller robot's end effector to remain in a fixed position relative to the ground (globally) and not relative to the mobile platform.

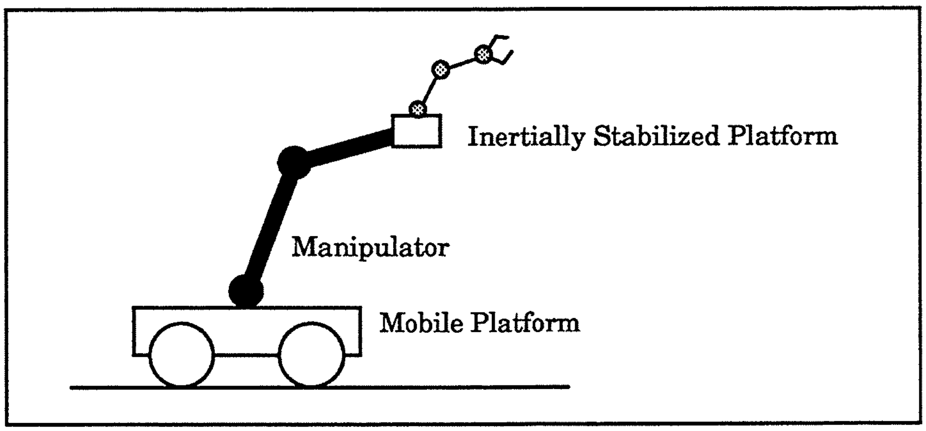


Figure 6.3 Intelligent Platform System

The base of the intelligent platform would incorporate an inertial sensing system that would counteract the unwanted motions of the rest of the system as illustrated in Figure 6.4. In effect, the system's compliance is reduced by active feedback control.

The actual inertial sensing is obtained from accelerometers. They "sense" the accelerations of the base of the platform and cancel out or reduce the motion at the end effector. This kind of system has the capability of high accuracy, with the inertially stabilized platform. The overall system also has high payload capabilities.

Another important problem to be considered is vehicle stability. Instability would occur if the center of gravity of the systems goes outside the vehicle platform. Therefore the reach of the robotic arm must be constrained by the payload associated with the robotic end effector. Using a main-in-the-bucket system as a reference, the maximum payload would be around 400 lbs. This limits the maximum extensions of the robot arm in various directions that are permissible in order to maintain vehicle stability (assuming vehicle stabilization feet are deployed when they are available). These limits can be conveniently maintained by control software.

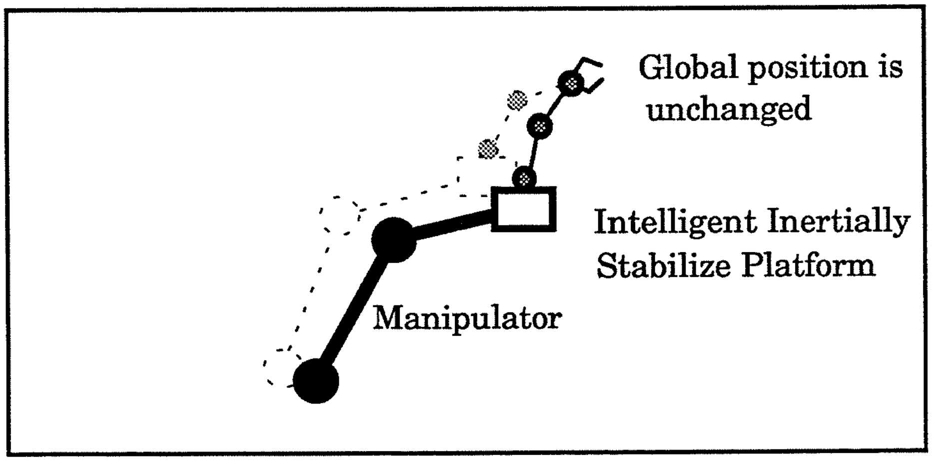


Figure 6.4 Intelligent Platform Illustration

Research by others at MIT [Dub92] have been concerned with similar problems for the past 10 years. They have proposed specific solutions and made demonstrations for NASA and the DOE. Their program has been running at a level of about $500K/year for the last 10 years, illustrating the difficulty of the problem from their approach. Their method relies on sensors placed at the base of the mobile platform. The sensors provide feedback used to compensate the position of the mobile base relative to the robot end effector. We have alternative concepts which show that the problem can be solved in a more sophisticated and less expensive manner.

Bibliography

[AS89] R. J. Anderson and M. W. Spong, "Asymptotic Stability for Force Reflecting Teleoperators with Time Delay," in *Proceeding of the IEEE International Conference on Robotics and Automation*, pp 1618-1625, 1989.

[BT90] P. G. Backes and K. S. Tso, "UMI: An Interactive Supervisory and Shared Control System for Telerobotics," in *Proceedings of the IEEE International Conference on Robotcs and Automation*, pp. 1096-1101, 1990.

[Bej87a] A. K. Bejczy, "Man-Machine Interface Issues in Space Telerobotics: A JPL Research and Development Program," in *Proceedings of the Workshop on Space Telerobotics*, pp. 361-369, 1987.

[Bej87b] A. K. Bejczy, "Teleoperators," in *Encyclopedia of Artificial Intelligence,* pp. 1100-1101, New York: John Wiley & Sons, Inc., 1987.

[BKV90] A. K Bejczy, W. Kim, and S. Venema, "The Phantom Robot: Predictive Displays for Teleoperation with Time Delay," in *Proceedings of the IEEE International Conference on Robotics and Automation*, pp. 546-551, 1990.

[Ben90] G. D. Benson, *The PUMA 560 IBM PC/AT Control Environment*, Tech. Rep. RRL-91-2, Robotics Research Laboratory, Department of Electrical Engineering and Computer Science, University of California, Davis, June 1990.

[BH88] D. G. Bihn and T. C. Hsia, "A Universal Six Joint Robot Controller," in *IEEE Journal of Robotics and Automation*, vol. 8, pp. 31-36, Feb. 1988.

[BoHa88] P. T. Boissiere and R. W. Harrigan, "Telerobotic Operation of Conventional Manipulators," in *Proceedings of the IEEE International Conference of Robotics and Automation*, pp. 576-583, 1988.

[CFGM88] G. Clement, R. Fournier, P. Gravez, and J. Morillon, "Computer-Aided Teleoperation: From Arm to Vehicle Control," in *Proceedings of the IEEE International Conference on Robotics and Automation*, pp. 590-593, 1988.

[Cra85] J. J. Craig, *Introduction to Robotics: Mechanics and Control.* Reading, Massachusetts: Addison-Wesley, 1 ed., 1985.

[Dub92] S. Dubowsky, "On the Dynamics and Control of Mobile Telerobotics Systems in Unstructured Environments," Seminar, MAME Dept., University of California, Davis, Dec. 4, 1992.

[FDS90] P Fischer, R. Daniel, and K. Siva, "Specification and Design of Input Devices for Teleoperation," in *Proceedings of the IEEE International Conference on Robotics and Automation*, pp 540-545, 1990.

[FDB86] C. P. Fong, R. S. Dotson, and A. K. Bejczy, "Distributed Microcomputer Control System for Advanced Teleoperation," in *Proceedings of the IEEE International Conference on Robotics and Automation*, pp. 987-995, 1986.

[FGL87] K. S. Fu, R. C. Gonzalez, and C. S. G. Lee, *Robotics: Control, Sensing, Vision, and Intelligence*. New York: McGraw-Hill, 1987.

[FP91] J. Funda and R. P. Paul, "Remote Control of a Robotic System by Teleprogramming," in *Proceeding of the IEEE International Conference on Robotics and Automation*, addendum, 1991.

[Han89] B. Hannaford, "Stability and Performance Tradeoffs in Bi-lateral Telemanipulation," in *Proceedings of the IEEE International Conference on Robotics and Automation*, pp 1764-1767, 1989.

[HV89] S. Hayati and S. T. Venkataraman, "Design and Implementation of a Robot Control System with Traded and Shared Control Capability," in *Proceedings of the IEEE International Conference on Robotics and Automation*, pp 1310-1315, 1989.

[HLTBL90] S. Hayati, T. Lee, K. Tao, P. Backes, and J. Lloyd, "A Testbed for a Unified Teleoperated-Autonomous Dual-Arm Robotic System," in *Proceedings of the IEEE International Conference on Robotics and Automation*, pp. 1090-1095, 1990.

[HHL89] G. Hirzinger, J. Heindl, and K. Landzettel, "Predictive and Knowledge-Based Telerobotic Control Concepts," in *Proceedings of the International Conference on Robotics and Automation*, pp. 1768-1777, 1989.

[Hog89] N. Hogan, "Controller Impedance at the Man/Machine Interface," in *Proceeding of the IEEE International Conference on Robotics and Automation*, pp. 1626-1631, 1989.

[HHL92] T. C. Hsia, K. Han, and T. A. Lasky, "A Simple Robust Cartesian Space Controller for Robot Manipulators," in *Submitted to IEEE International Conference on Robotics and Automation*, (Nice, France), 1992.

[Jen85] L. M. Jenkins, "Telerobotics Work System - Space Robotics Application," in *Proceedings of the IEEE International Conference on Robotics and Automation*, pp 804-806, 1985.

[JC71] E. G. Johnson and W. R. Corliss, *Human Factors Applications in Teleoperator Design and Operation.* New York: John Wily & Sons, Inc., 1971.

[Las91] T. A. Lasky, "Progress Toward a Compliant Robot Controller," Tech. Rep. RRL-91-1, Robotics Research Laboratory, Department of Electrical Engineering and Computer Science, University of California, Davis, Oct. 1991.

[KTES87] W. S. Kim, F. Tendick, S. R. Ellis, and L. W. Stark, "A Comparison of Position and Rate Control for Telemanipulators with Consideration of Manipulator System Dynamics," *IEEE Journal of Robotics and Automation*, vol. 3, pp. 426-436, Oct. 1987.

[KS89] W. S. Kim and L. W. Stark, "Cooperative Control of Visual Displays for Telemanipulation," in *Proceedings of the IEEE International Conference on Robotics and Automation*, pp. 1327-1332, 1989.

[KaSa87] I. Kato and K. Sadamoto, *Mechanical Hands Illustrated*, Hemisphere Publishing Corporation, 1987.

[Lum89] R. Lumia, "Space Robotics: Automata in Unstructured Environments", in *Proceedings of the IEEE International Conference on Robotics and Automation*, pp. 1467-1471, 1989.

[MBB89] O. Y. Ming, D. V. Beard, and F. P. Brooks, Jr., "Force Display Performs Better than Visual Display in a Simple 6-d Docking Task", in *Proceedings of the IEEE International Conference on Robotics and Automation*, pp. 1462-1466, 1989.

[MMYA86] F. Miyazaki, S. Matsubayashi, T. Yoshimi, and S. Arimoto, "A New Control Methodology Toward Advanced Teleoperation of Master-Slave Robot Systems," in *Proceedings of the IEEE International Conference on Robotics and Automation*, pp. 997-1002, 1986.

[Pen86] J. Pennington, "Space Telerobotics: A Few More Hurdles," in *Proceedings of the IEEE International Conference on Robotics and Automation*, pp 813-816, 1986.

[PS85] L. Pao and T. H. Speeter, "Transformation of Human Hand Positions for Robotic Hand Control," in *Proceedings of the IEEE International Conference on Robotics and Automation*, pp. 1758-1763, 1989.

[Riv87] E. Rivin, *Mechanical Design of Robotics*, McGraw-Hill, 1987.

[She85] T. B. Sheridan, "Human Supervisory Control of Robot Systems," in *Proceedings of the IEEE International Conference on Robotics and Automation*, pp. 808-812, 1985.

[SCMR88] T. B. Sheridan, L. Charny, M. B. Mendel, and J. B. Roseborough, "Supervisory Control, Mental Models and Decision Aids," in *Workshop on Shared Autonomous and Teleoperated Manipulator Control*, (Philadelphia, PA), IEEE International Conference on Robotics and Automation, 1988.

[She88] T. B. Sheridan, "Teleoperation, Telepresence, and Telerobotics: Research Needs for Space", in *Workshop on Shared Autonomous and Teleoperated Manipulator Control*, (Philadelphia, PA), IEEE International Conference on Robotics and Automation, 1988.

[SKT88] L. W. Stark, W. S. Kim, and F. Tendick, "Cooperative Control in Telerobotics," in *Proceedings of the IEEE International Conference on Robotics and Automation*, 1988.

[Sto85] K. Stonecipher, *Industrial Robotics: A Handbook of Automated Systems Design*, Hayden Book Company, 1985.

[Wam88] C. Wampler, "Teleoperator, Supervisory Control," in *International Encyclopedia of Robotics*, pp. 1740-1747, New York: John Wiley & Sons, Inc., 1988.

[WR88] J. G. Webster and B. Ravani, "Teleoperator, Control Using Telepresence,", in *International Encyclopedia of Robotics*, pp. 1710-1718, New York: John Wiley & Sons, Inc., 1988.

Appendix A Commercially Available Systems Survey Data

Schilling Dev.
Titan II, Titan 7F, and Gamma 7F

Kawasaki
EH-120, UZ100

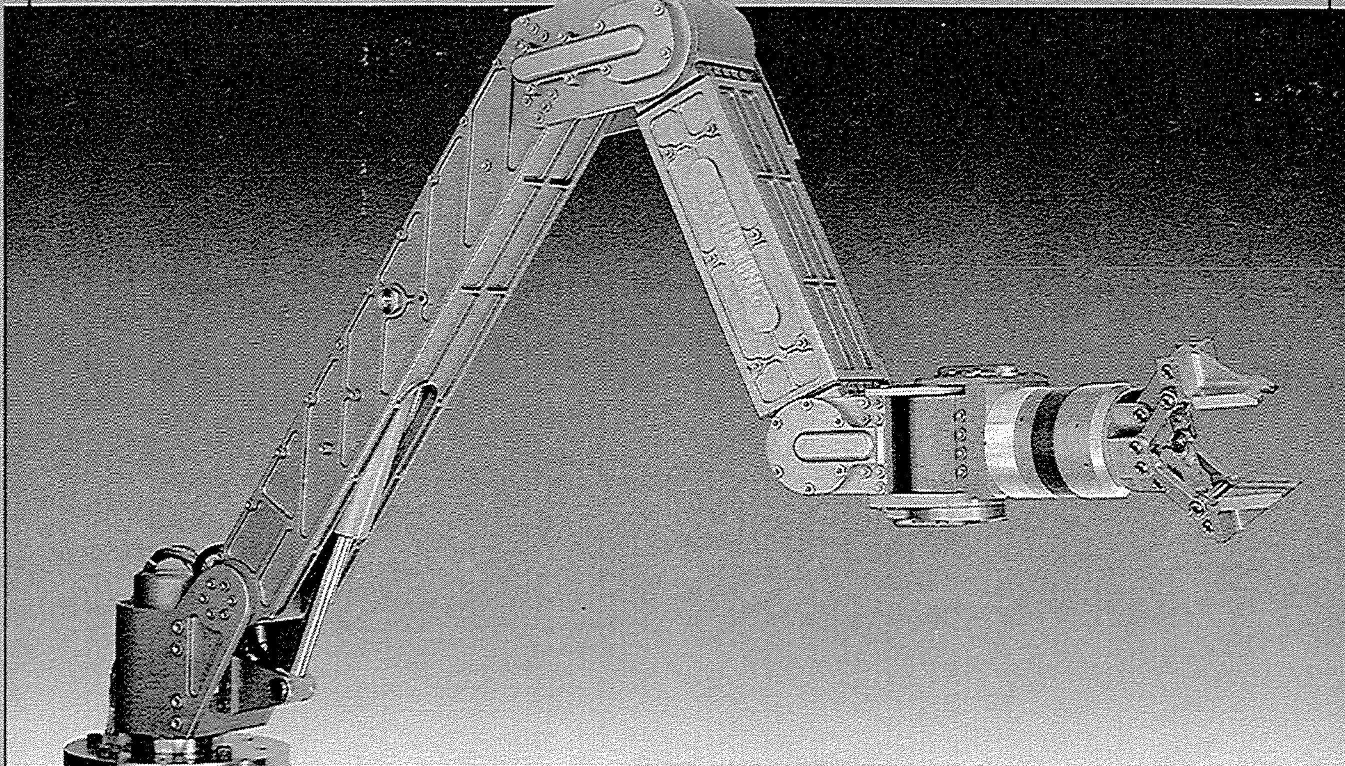
Panasonic
AW-8100, AW-8060, and AW-8030

Motoman Inc.
60S, K100S, and K150S

IMT Inc.
20017, 2115, and 6425

TITAN II

TELEROBOTIC MANIPULATOR SYSTEM

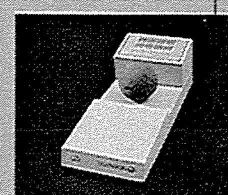


To meet the increasing demands of remote manipulation in hostile environments, Schilling Development, Inc. has developed a second-generation TITAN II — a dexterous, servo-hydraulic telerobotic manipulator with six degrees of freedom. Schilling's seven-year experience in producing equipment for extreme environments has driven the design refinements available in the TITAN II, such as high-resolution bilateral force feedback, advanced telerobotic control, and end effector interchange. Offered in a number of configurations, the TITAN II addresses applications in radioactive, deep-ocean, toxic-chemical and high-voltage environments.

The variety of remote work tasks requires the telemanipulator to be highly reliable, field maintainable, and adaptable. The TITAN II provides a turn-key solution for manipulative tasks in a wide range of hostile environments. The standard configuration features titanium construction, high payload, wide range of motion, intuitive control, and smooth, fluid operation.

To aid the operator in performing tasks that might otherwise be impossible or impractical, the TITAN II is available with options such as force feedback, advanced telerobotic control, and tool interchange. Force feedback combines visual feedback with direct experience of motion and forces acting on the slave arm by reflecting the forces to an electrically actuated master arm. The Advanced Telerobotic Controller (ATC) offers true robot control of the slave arm. The ATC incorporates a modular control architecture to accommodate a number of input devices, graphical interface, real-time sensor inputs, and data transmission types. Instantaneous transition from teleoperated control to robot control and complex trajectory generation greatly enhance capability and productivity. Tool interchange allows a practical means of deploying various tools at the remote worksite.

For radioactive environments, Schilling has designed a hardened version of the TITAN II to tolerate an accumulated exposure of 10^7 RAD with no loss in performance.

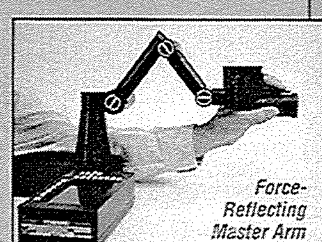


Cartesian Controller*

FEATURES:

- High dexterity
- Heavy lift
- Titanium construction
- Comfortable and intuitive control
- Compatible with multiple fluid types
- Optional force feedback, advanced telerobotic control, tool interchange, and radiation hardening

Master Arm



Force-Reflecting Master Arm

SCHILLING

GENERAL DESCRIPTION

Modes of Operation
Master/Slave Position Controlled .... standard
Cartesian/Tool Frame Control .... optional
Force Control .... optional
Bilateral Force Feedback .... optional

Input Devices
Passive Master Arm .... standard
Cartesian Controller .... optional
Force-Reflecting Master Arm .... optional
Graphical Interface and Trajectory Generation .. optional

Degrees of Freedom
Six Plus Grip .... standard
Tool Interchange .... optional

Power System
Hydraulic .... Multiple fluid compatible

STANDARD DIMENSIONS AND SPECIFICATIONS
Maximum Reach .... 76.3 in.
Lift Capacity (maximum) .... 1200 lb
Lift Capacity (full extension) .... 240 lb
Wrist Torque .... 75 ft-lb (peak)
Jaw Capacity .... 4.0 in.
Weight .... 175 lb

RANGE OF MOTION

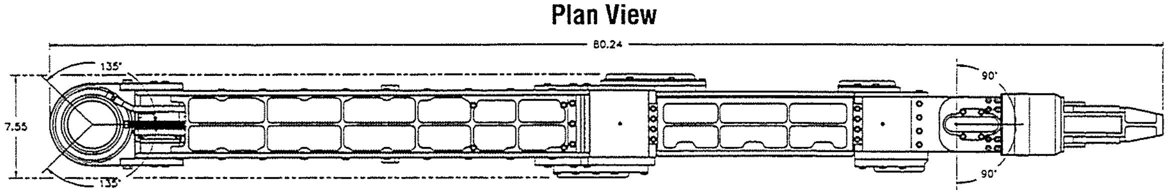
Hardware Range

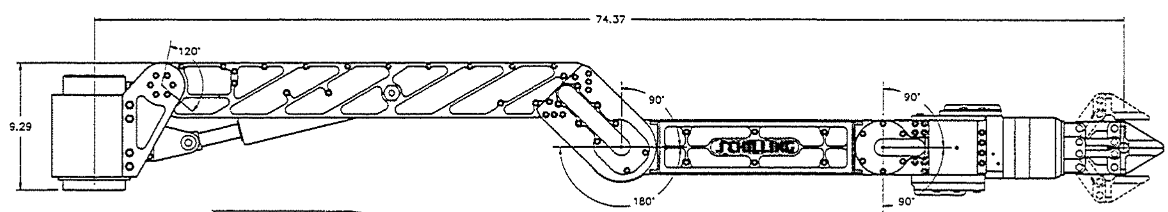
Waist Yaw .... 270°
Shoulder Pitch .... 120°
Elbow Pitch .... 270°
Wrist Pitch .... 180°
Wrist Yaw .... 180°
Wrist Rotate
- Slaved .... 270°
- Continuous .... 0-55 rpm

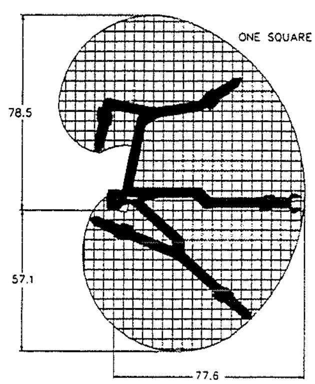
HYDRAULIC REQUIREMENTS
Fluid Type .... Hydraulic Oil
Optional Fluid .... Consult Factory
Flow .... 1.5 to 5 gpm
Pressure .... 3000 psi nominal

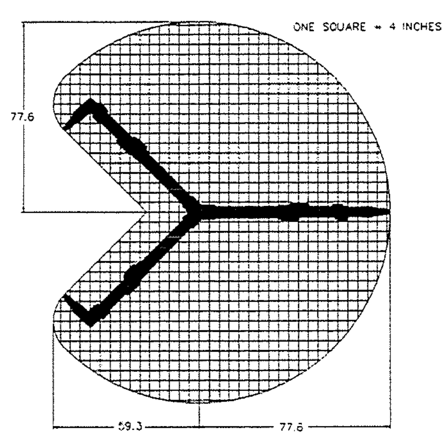
ELECTRICAL AND TELEMETRY REQUIREMENTS
Consult Factory

Description and specifications are subject to change without notice.
Contact Schilling Development for latest information.

Plan View


Elevation


Elevation




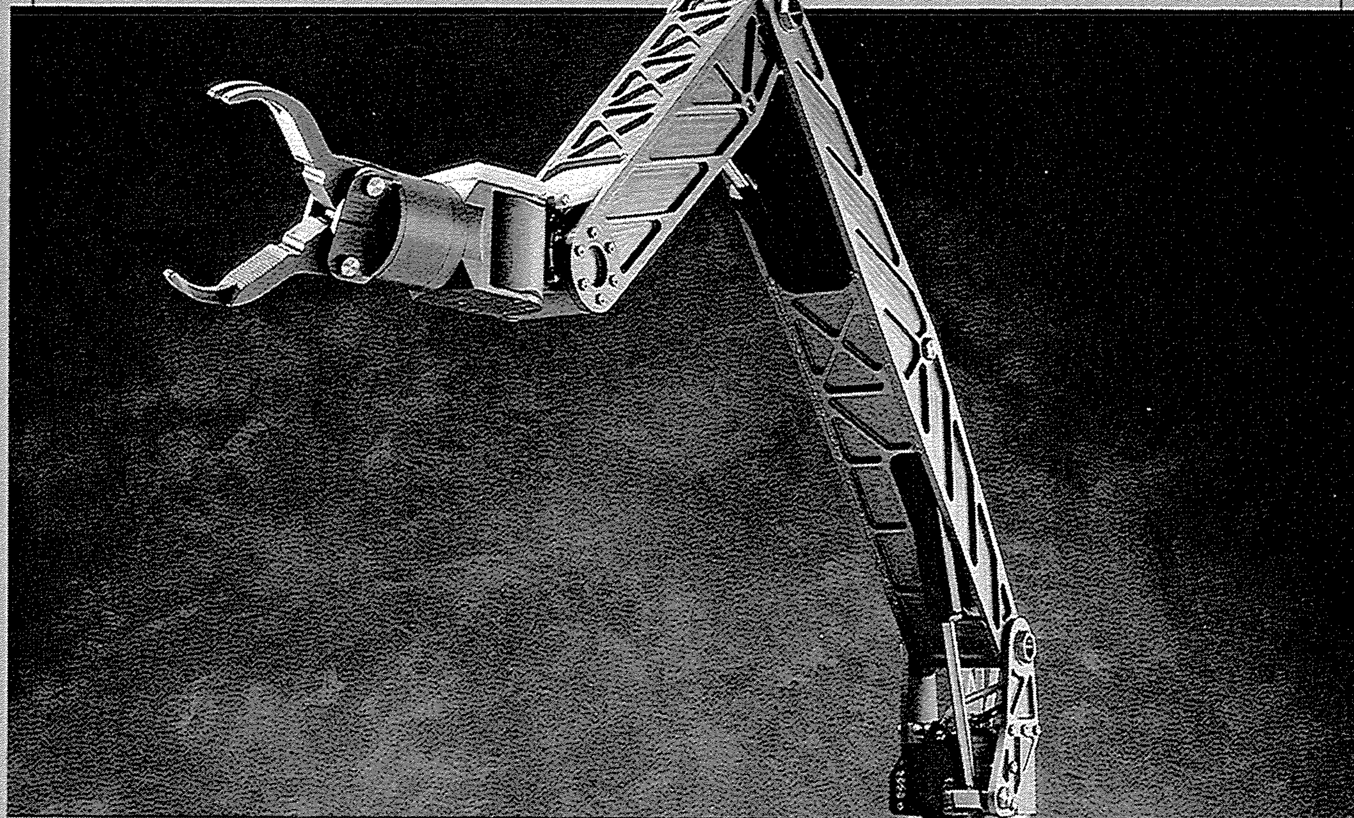
Plan View

SCHILLING
DEVELOPMENT
1632 DA VINCI COURT, DAVIS, CA 95616
(916) 753-6718 FAX (916) 753-8092

*Dimension 6 force/torque control ball, reproduced with permission of CIS Graphics, Inc.

TITAN 7F and GAMMA 7F

REMOTE MANIPULATOR SYSTEMS



Schilling Development, Inc. meets user requirements for seven-function remote manipulators with two highly dexterous and powerful servo-hydraulic, Master/Slave systems - The TITAN 7F and GAMMA 7F. Each are constructed primarily of 6-4 titanium and employ advanced electronic and mechanical design for use whenever manipulative tasks must be performed in locations or environments where man cannot safely or practically venture. Applications range from undersea to radioactive environments.

The TITAN 7F has acquired an impeccable reputation since its introduction in 1987 and is seeing service for a variety of commercial, scientific and military users. Typical tasks range from undersea salvage, maintenance and construction to ordnance handling and toxic cleanup.

Building upon the TITAN 7F's established record of reliability and performance, the GAMMA 7F provides for operation in radioactive environments. Through careful selection of radiation resistant materials, the GAMMA 7F is designed to tolerate an accumulated exposure of 10^7 RAD gamma radiation. Now a standard, commercially proven product is available that eliminates costly development, rework costs and reliability concerns associated with custom designed equipment.

The TITAN 7F and GAMMA 7F are controlled by compact master arms that provide for comfortable and intuitive manipulator control. Advanced hydraulic and control system technology combine smooth and fluid operation with an extremely tight control loop giving the manipulators human-like speed and accuracy.

FEATURES:

- Available for undersea and terrestrial applications.
- Microprocessor, servo controlled.
- Dexterous and powerful.
- 250 lb. payload at full arm extension.
- Radiation hardened to 10^7R.
- Oil hydraulic and silicon base fluid compatible.
- Titanium construction.
- Portable master controllers that are simple to learn, easy and comfortable to operate and require little space.
- Compact and powerful three-axis wrist assembly.

SCHILLING

GENERAL DESCRIPTION

Mode of Operation .... Spatially Correspondent
Input Device .... Compact Master Control Arm
Number of Functions .... Seven
Power System .... (TITAN 7F) Oil Hydraulic
(GAMMA 7F) Multi Fluid Compatible

DIMENSIONS AND SPECIFICATIONS

SLAVE ARM

- Maximum Reach .... 78 inches
- Lift capacity at Full Extension .... 250 lbs.
- Jaw Capacity .... 4.0 inches (standard)
- Jaw Closure Force .... 350 lbs. max.
- Weight in Air .... 147.0 lbs.
- Weight in Water .... 113.0 lbs.

MASTER CONSOLETTE

- Height .... 10.0 inches
- Width .... 6.0 inches
- Length .... 19.0 inches
- Weight .... 10.0 lbs.

SLAVE CONTROLLER ASSEMBLY

- Height .... 4.0 inches
- Width .... 7.5 inches
- Length .... 16.0 inches
- Weight in Air .... 27.0 lbs.
- Weight in Water .... 14.5 lbs.

PERFORMANCE

	Hardware Range	Max. Slew Rate
Waist Yaw	270°	90°/sec.
Shoulder Pitch	90°	90°/sec.
Elbow Pitch	120°	90°/sec.
Wrist Pitch	180°	400°/sec.
Wrist Yaw	180°	400°/sec.

Wrist Rotate
- Slaved .... 270°
- Continuous .... 0 to 55 rpm

Wrist Torque .... 70 ft. lbs. (peak)

HYDRAULIC REQUIREMENTS

3000 psi - 3.0 gpm nominal

ELECTRICAL REQUIREMENTS

25 watts nominal powered by 120/240 VAC or 20-30 VDC

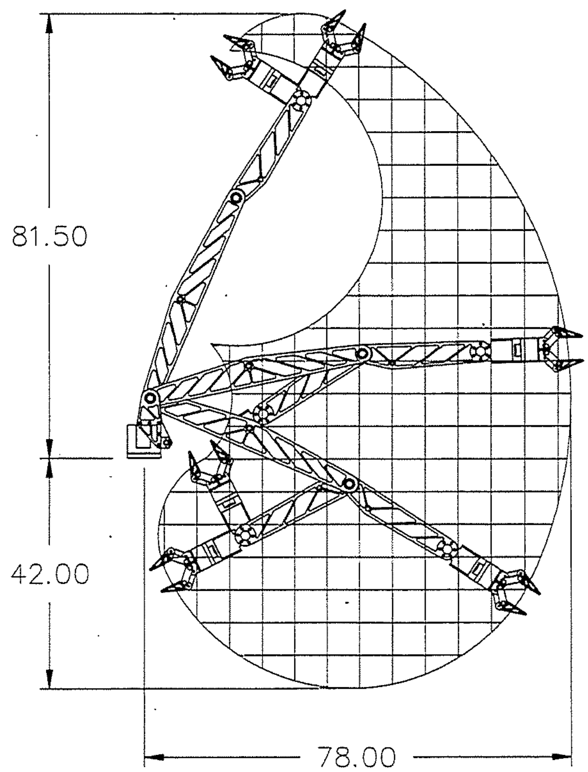
TELEMETRY REQUIREMENTS

RS-422 type media - Single twisted wire pair;
RG-108 or equivalent

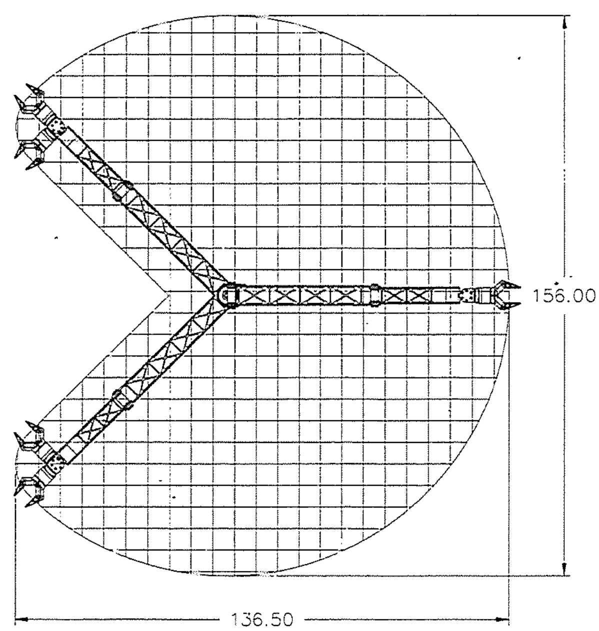
OPTIONS

Contact Schilling Development, Inc. for details.
Available in single and dual manipulator configurations.

Side Elevation View

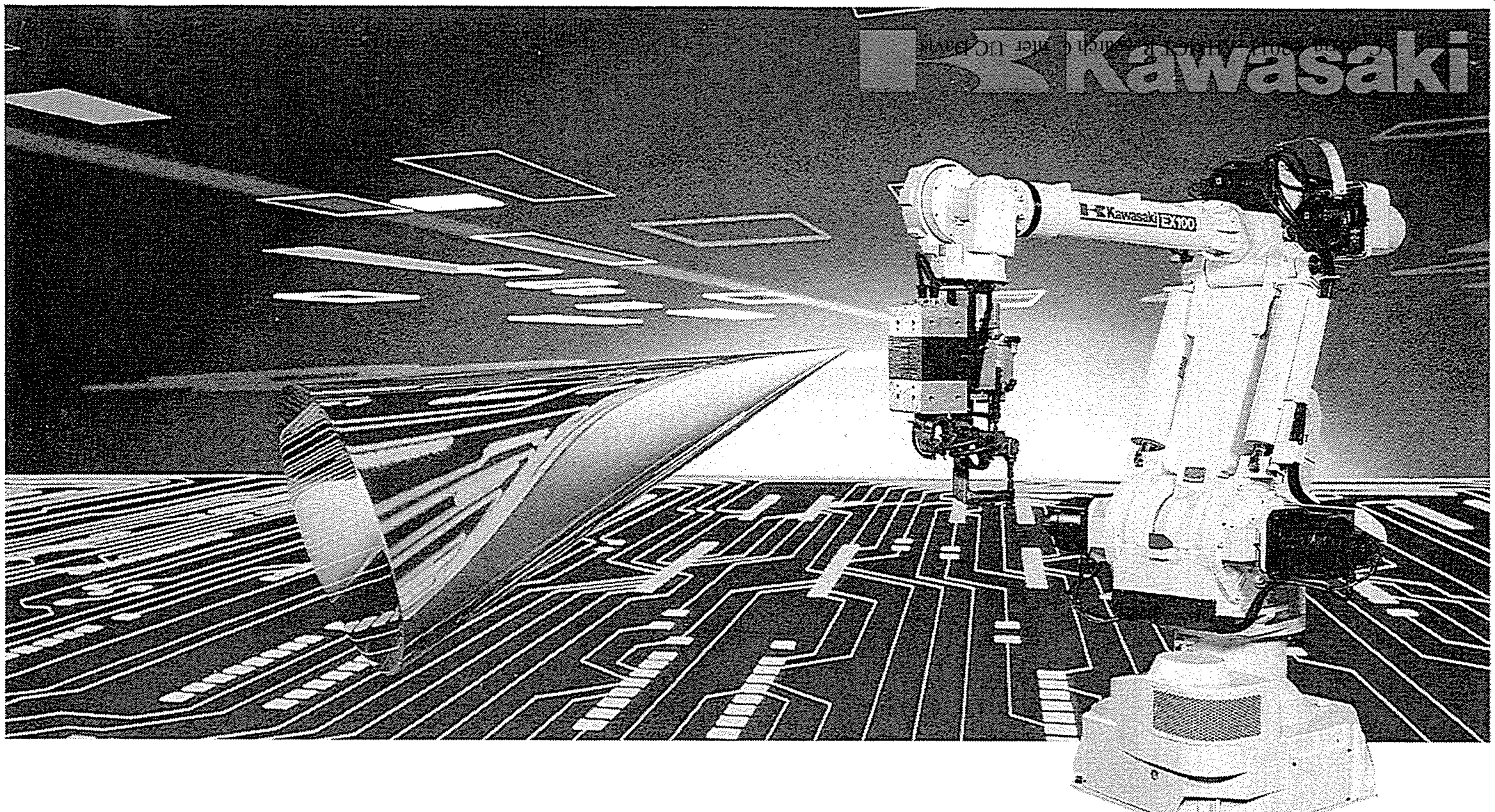


Plan View



SCHILLING
DEVELOPMENT
1632 DA VINCI CT., DAVIS, CA 95616
(916) 753-6718 • FAX 753-8092

Description and specifications are subject to change without notice. Contact Schilling Development for latest information.



Kawasaki Robot
LINE UP

KAWASAKI
HEAVY INDUSTRIES, LTD.

Robots focusing on the 21st century

In Japan, Kawasaki Heavy Industries was the pioneer in the field of industrial robots, and has 20 years experience in their development and manufacture.

Robots aim to liberate people from tedious, repetitive work, heavy labor and work under disagreeable conditions, and at the same time are effective in improving productivity.

In recent years our robots have been accepted throughout industry, and today are fulfilling their early promise. Kawasaki is prepared to meet a variety of needs with the four series which form the core of our product line.

The powerful "Kawasaki E series" has a large working envelope and high level path control functions, and is designed for all types of handling and welding work.

The "Kawasaki P series" has flexible action similar to that of the human hand and can be readily adapted to assembly and numerous other jobs.

The "Kawasaki J series" can be mainly adapted to assembling and arc-welding.

The new "Kawasaki U series" can be applied in wide variety of applications including handling, spot-welding, transport between pressing machines.

An outline of the Kawasaki Robot line is given in this catalog, but what we are aiming for is not only to take over the work of human beings. We are also working to expand horizons and technology, and to develop robots to do work which humans cannot.

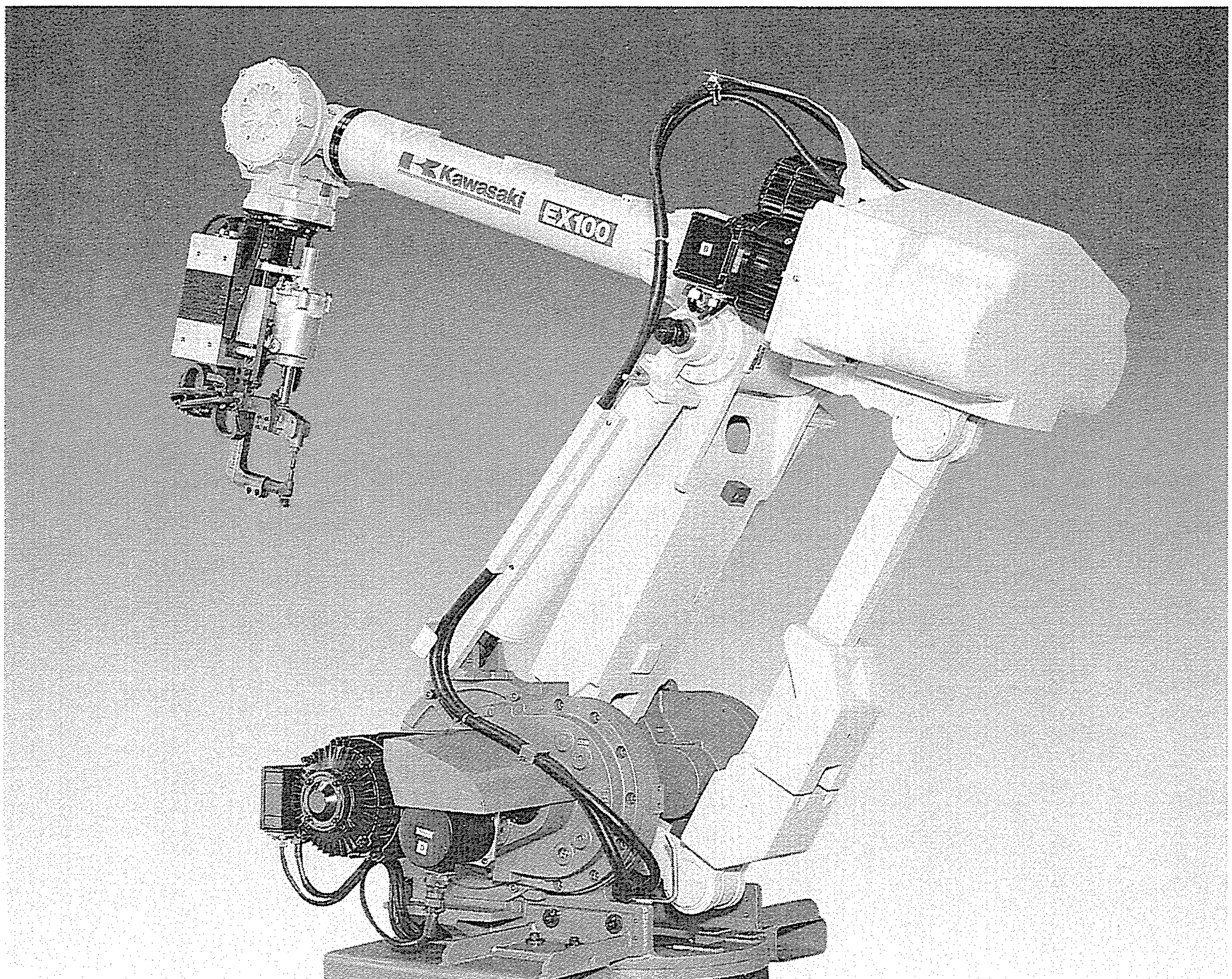
For example, robots are being applied in fields such as assembly of ultra-precise, ultra-miniature devices in super clean rooms, jobs relating to nuclear power utilization and medical therapy support tasks.

Kawasaki has been steadily making progress. For instance, we have completed a class 10 clean specification assembly robot, and in the medical field have developed a robot to aid in rehabilitation.

While striving for harmonious coexistence between people and robots, and focusing on the kinds of robots that are truly necessary, Kawasaki is constantly reviewing its technology so that we may realize our goal of being the world's leading maker of industrial robots.

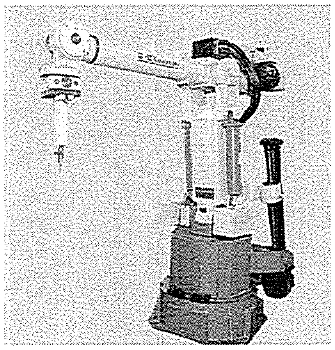
To provide overseas customers with after-sales and technical services, we have established a worldwide network including Kawasaki Robotics (USA), Inc. in Detroit and Bourne End Robot Office in London.

Kawasaki is marching steadily into the world and the 21st century.



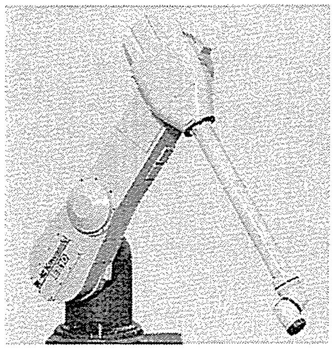
U series

ES65



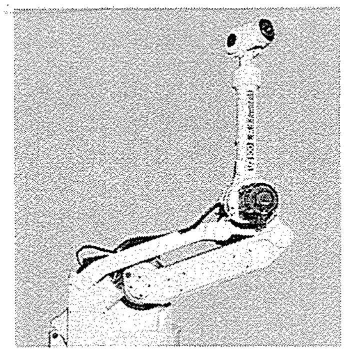
Withstand large counter-pressure-fource.
Series-stud-welding.
Performing a double role by tool changing.

EE10



Apply to both "Spray painting" and "Sealant Dispensing"
Large envelope.
Traversing devices are available.

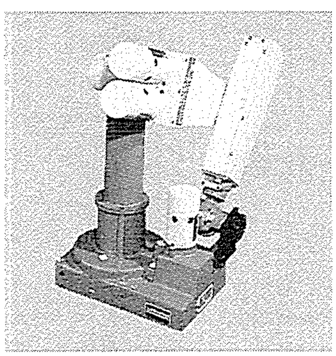
UZ100



Reversible arm enables efficient operations in the back side. Installation on ceiling possible. Wide variety of applications, such as handling and spot-welding.

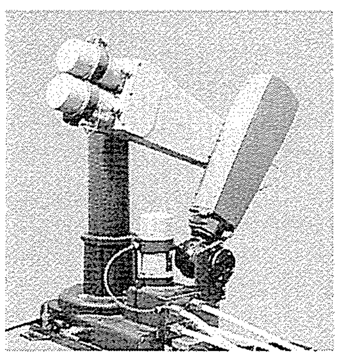
Features	ES65	EE10	UZ100
Payload	65 kgf	10 kgf	100 kgf
Repeatability	±1 mm	±0.5 mm	±0.3 mm
Max. Speed	Rotation: 60°/sec. Out-In: 55°/sec. Up-Down: 55°/sec.	2,000 mm/sec.	Rotation: 120°/sec. Out-In: 200°/sec. Up-Down: 110°/sec.
Weight	1,390 kgf	580 kgf	1,300 kgf

PH260



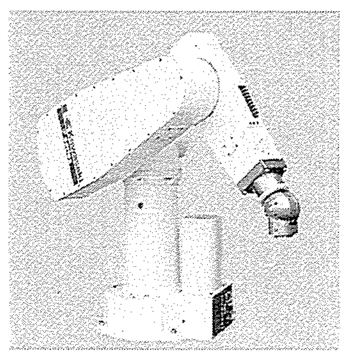
KL's program commands simplify operation.
Compact and light-weight.

PH260-CR



KL's program commands simplify operation.
Satisfying class 10.
Suit for small-precision-parts' handling.

PH561-CR



KL's program commands simplify operation.
Satisfying class 10.
Suit for use in all of the clean rooms.

Features	PH260	PH260-CR	PH561-CR
Payload	1 kgf	1 kgf	2.5 kgf (5 kgf)
Repeatability	±0.05 mm	±0.05 mm	±0.1 mm
Max. Speed	500 mm/sec. 1,450 mm/sec.	500 mm/sec. 1,450 mm/sec.	1,000 mm/sec. 2,700 mm/sec.
Weight	15 kgf	20 kgf	120 kgf

※ T.W.C.=Typical Working Configuration, F.O.C.=Full Out Configuration

Remarks

- Spot welding
- Arc welding
- Assembling
- Spray painting
- Sealant dispensing
- Handling

—Kawasaki has impeccable records and technologies

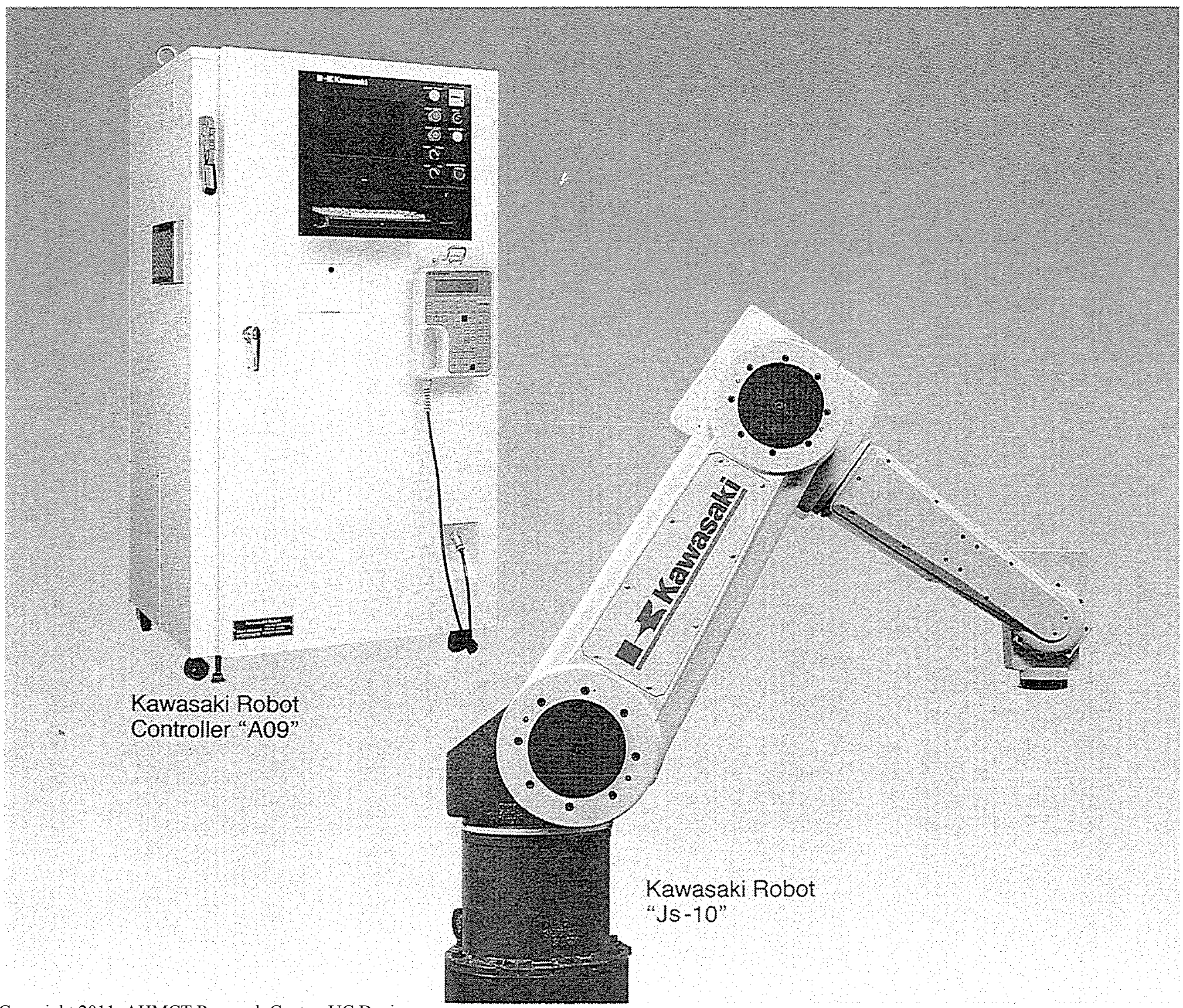
Brief History

as of April 1991

1968 • Kawasaki takes the lead in Japan in manufacture of industrial robots.
1970 • Exclusive robot plant set up.
• Domestic sales started.
1972 • Exports started.
1980 • Robot production reached 1,000.
1981 • Production of P-series robots started.
1983 • Electric robot E-series developed.
1986 • Robot production reached 5,000.
• Detroit Robot Center opened.
1989 • J-series robot developed.
1990 • Robot production reached 10,000.
• Kawasaki Robotics (USA), Inc. in Detroit.
1991 • Bourne End Robot Office established.
• U-series robot developed.

Nations with export records

Australia
Brazil
Canada
China
Finland
Germany
India
Italy
Korea (R.O.K.)
Malaysia
Mexico
Singapore
South Africa
Spain
Taiwan (R.O.C.)
Turkey
U.K.
U.S.A.
U.S.S.R.

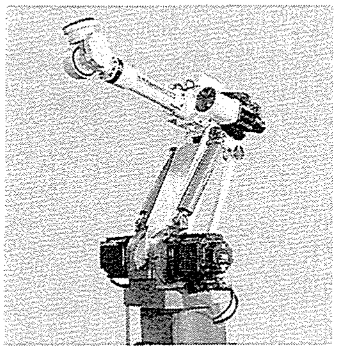
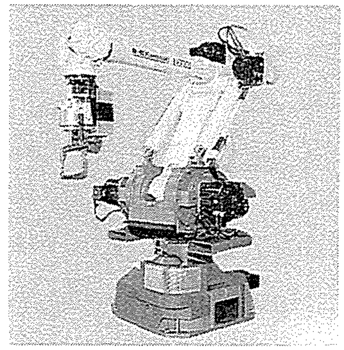
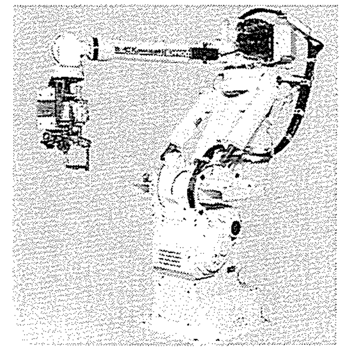


Kawasaki Robot Controller "A09"

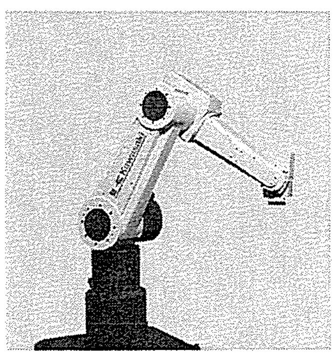
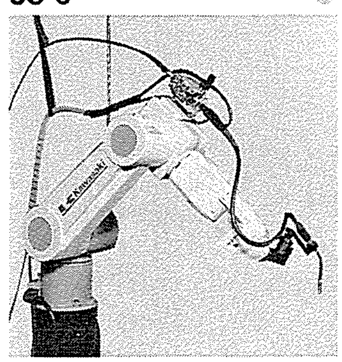
Kawasaki Robot "Js-10"

full line-up of models for any type of work, from assembly

E series

	EX30/40	EX100/120/150	EH120
			
Features	Versatile robot with medium payload and medium size. Large motion angle of each axis performs wide operation envelope.	Wide envelope, less backward dead space. Internally housed piping of cooling water and air for welding gun.	Horizontally rotating axis is adopted as the third axis. Small side-and-backward dead space enables high density layout.
Payload	30/40 kgf	100/120/150 kgf	120 kgf
Repeatability	±0.3 mm	±0.5 mm	±0.5 mm
Max. Speed	Rotation: 110°/sec. Out-In: 110°/sec. Up-Down: 110°/sec.	Rotation: 90/110/90°/sec. Out-In: 90/110/90°/sec. Up-Down: 90/110/90°/sec.	Rotation: 90°/sec. Out-In: 90°/sec. Up-Down: 90°/sec.
Weight	650 kgf	1,600/1,800/1,800 kgf	1,500 kgf

J series

	Js-10	Js-6
		
Features	Compact size robot with high speed, high accuracy and wide envelope. Advanced controller "A09" having high level programming language "AS".	Easy to teach and operate arc-welding robot with high accuracy and wide envelope. Advanced controller "A22" having high level programming language "AS".
Payload	10 kgf	6 kgf
Repeatability	±0.1 mm	±0.1 mm
Max. Speed	5,000 mm/sec.	1,500 mm/sec. (in air-cut motion) 100 mm/sec. (in welding motion)
Weight	150 kgf	140 kgf

P series

Features
Payload
Repeatability
Max. Speed※ at T.W.C. / at F.O.C.
Weight

INQUIRIES

KAWASAKI HEAVY INDUSTRIES, LTD.
ROBOT DIVISION

Tokyo Head Office
World Trade Center Bldg., 4-1 Hamamatsu-cho 2-chome, Minato-ku, Tokyo 105 Japan
Phone: Tokyo (03) 3435-6908
Telex: (NTT) 242-4371 KAWAJU J
Fax: (03) 3578-1573

Akashi Works
1-1 Kawasaki-cho, Akashi 673 Japan
Phone: (078) 921-1551
Telex: (NTT) 5628-951
Fax: (078) 923-6548

OVERSEAS OFFICES

Beijing Office
Room No. 2602-05, China World Tower, China World Trade Center, No. 1, Jian Guo Men Wai Avenue, Beijing 100004, People's Republic of China
Phone: (1) 505-1350 Telex: 22385 KHI CN
Fax: (1) 505-1351

Taipei Office
15/F, Fu-Key Bldg., 99 Jen-Ai Road, Section 2, Taipei
Phone: (2) 322-1752 Telex: 28393 KHI TPE
Fax: (2) 322-5009

Bangkok Office
20th Floor, Thaniya Plaza Business Complex No. 52, Silom Road, Bangkok, 10500, Thailand
Phone: (2) 231-2360~2 Telex: 82800 KAWAJU TH
Fax: (2) 231-2363

Manila Office
20th Floor, Metrobank Plaza Bldg., Sen. Gil J. Puyat Avenue, Makati, Metro Manila, Philippines
Phone: (2) 818-2786
Fax: (2) 818-2787

Jakarta Office
7th Floor, Skyline Bldg., Jalan M.H. Thamrin 9, Jakarta, Indonesia
Phone: (21) 320737 Telex: 61549 KAWAJU IA
Fax: (21) 321049

Sydney Office
Suite 6, Level 8, Barrack House, 16-20 Barrack Street, Sydney, N.S.W. 2000, Australia
Phone: (2) 262-4412 Telex: 177443 KAWAJU AA
Fax: (2) 262-4409

Cairo Office
9th Floor, Abu El Feda Bldg., 3 Abu El Feda Street, Zamalek, Cairo, Egypt
Phone: (2) 3411361 Telex: 92659 KAWAJU UN
Fax: (2) 3411358

OVERSEAS SUBSIDIARIES

Kawasaki Heavy Industries (USA), Inc.
599 Lexington Avenue, Suite 2705, New York, N.Y. 10022, U.S.A.
Phone: (212) 759-4950 Telex: 237004 KHINY UR
Fax: (212) 759-6421

Houston Branch
Suite 3670, 601 Jefferson Street, Houston, Texas 77002, U.S.A.
Phone: (713) 654-8981 Telex: 203309 KHI UR
Fax: (713) 654-8187

Kawasaki do Brasil Industria e Comercio Ltda.
Avenida Paulista, 1294/1318-5 Audar, São Paulo, CEP 013010 Brazil
Phone: (11) 289-2388 Telex: 1122171 KAWA BR
Fax: (11) 289-2788

Kawasaki Heavy Industries (UK) Ltd.
4th Floor, 3, St. Helen's Place, London EC3A 6EB, United Kingdom
Phone: (71) 628-9915~7 Telex: 886303 KAWAJU LNG
Fax: (71) 628-8907

Kawasaki Heavy Industries GmbH
5th Floor, Wehrhahn Center, Oststrasse 10, 4000 Düsseldorf, F.R. Germany
Phone: (211) 350441 Telex: 8587421 KHI D
Fax: (211) 161844

Kawasaki Heavy Industries (Europe) B.V.
7th Floor, "River Staete", Amsteldijk 166, 1079LH Amsterdam, Netherlands
Phone: (20) 6446869~70 Telex: 15115 KHI NL
Fax: (20) 6425725

Kawasaki Heavy Industries (Singapore) Pte, Ltd.
6 Battery Road, No. 18-04 Singapore 0104
Phone: 2255133~4 Telex: 25487 KAWAJU RS
Fax: 2249029

Kawasaki Heavy Industries (H.K.), Ltd.
16th Floor, Jardine House, Connaught Road, Central, Hong Kong
Phone: 522-3560 Telex: 75690 KHIHK HX
Fax: (845) 2905

KAWASAKI ROBOTICS(USA), INC.

Head Office
24402 Sinacola Court, Farmington Hills, MI 48331
Phone: (313) 474-6100
Fax: (313) 474-6101

KAWASAKI HEAVY INDUSTRIES (UK), LTD.

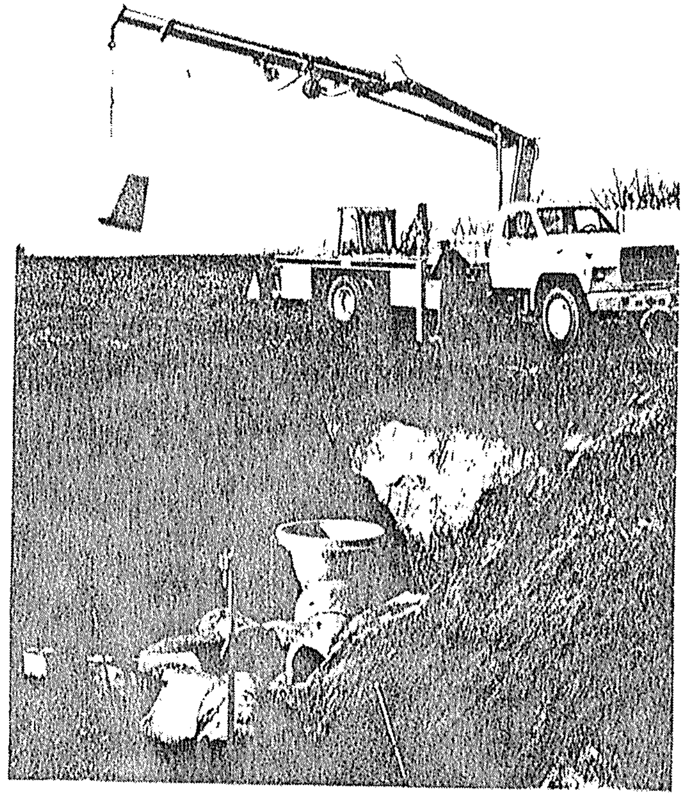
Bourne End Robot Office
1 Dukes Meadow, Millboard Road, Bourne End, Bucks SL8 5XF, England
Phone: (0628) 851288/851388
Fax: (0628) 851352

AGENT

★ Materials and specifications are subject to change without notice.
★ Kawasaki Robot is equipped with plenty of safety features; however, the surely measures for the operating staff should be taken by each customer concerned by taking into account each specific working environment, installation conditions, application requirements, etc.

Cat. No.3L1484 May. '92Ⓜ
Printed in Japan

6425



Lifting capacities

6′ 4″ (1.93 m)	10,000 lbs. (4,536 kg)
8′ (2.44 m)	8,000 lbs. (3,629 kg)
10′ (3.05 m)	6,400 lbs. (2,903 kg)
15′ (4.57 m)	4,250 lbs. (1,928 kg)
20′ 4″ (6.20 m)	2,950 lbs. (1,338 kg)
25′ 8″ (7.82 m)	2,200 lbs. (998 kg)
31′ (9.45 m)	1,500 lbs. (680 kg)

Minimum 26,500 lb. (12,020 kg) GVW chassis

IMT's 6425 offers lift capacities from 2,200 lbs. at 25′ 8″ to 10,000 lbs. at 6′ 4″. The optional manual 64″ extension to 31′ handles up to 1,500 lbs. at over 40′.

For traveling and maximum payload space, the 6425 stores compactly between cab and body in a figure-four position.

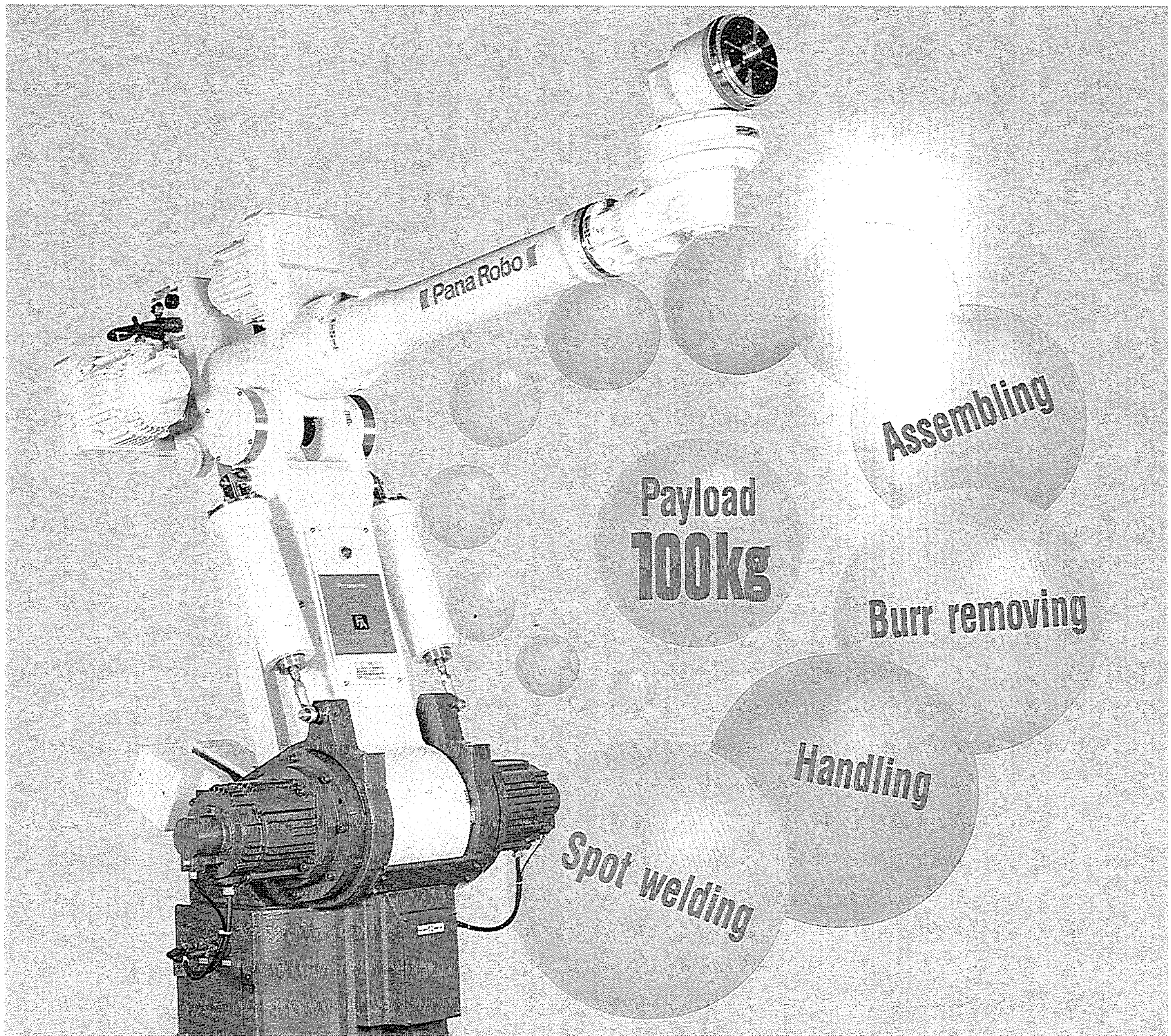
Specifications

Crane rating	64,000 ft.-lbs. (8.85 ton-m)
Standard boom length	25′ 8″ (7.82 m)
Max. horizontal reach	31′ (9.45 m)
Max. vertical lift	40′ 3″ (12.27 m)
Mounting space	30″ (76.2 cm)
Crane weight	3,950 lbs. (1,792 kg)
Stowed height	11′ 2″ (3.40m)
Rotation	370° (6.46 rad.)
Outrigger span	12′ 3″ (3.73 m)
Outrigger style	Out and down
Working pressure	2,750 PSI (193.3 kg/cm²)
Pump capacity	9 GPM (34 liter/min.)
RBM required	900,000 in.-lbs. (10,373 kg-m)

Panasonic
Industrial Robot

6 axis
Articulated arm robot

Pana Robo AW-8100



- Compact size but large work envelope.
- High-speed operation of 112°/sec. and elaborate speed control shortening tact time.
- Position repeat accuracy as high as ±0.4 mm.
- Easy teaching with many kinds of teaching coordinate system (Positions can be corrected from CRT screen.)
- No need of origin return because of absolute value encoder.
- No need of brush replacement because of AC servo motor.
- Sufficient care for safety.

			Specification	
Model			YA8101AM	
Structure			Multiple articulation	
Degree of freedom			6 axis	
Operation range	Arm	Rotation	±150° (Front standard)	
		Upper arm	+65°–60° (Vertical standard)	
		Front arm	+30°–115° (Horizontal standard)	
	Wrist	Rotation	±240°	
		Bending	±190° (Front arm standard)	
		Twisting	±350°	
Work envelope	Arm operation sectional area		3.6m²x360°	
	Arm fore-back operation distance		+480 ~ +2420mm (From rotation axis center to bending axis center)	
	Arm up-down operation distance		−57 ~ +2580mm (From robot bottom to bending axis center)	
Momentary max. speed	Arm	Rotation	112°/sec	
		Upper arm	112°/sec	
		Front arm	112°/sec	
	Wrist	Rotation	140°/sec	
		Bending	140°/sec	
		Twisting	240°/sec	
Max. pay load			100 kg	
Wrist allowable load Moment/inertia		Rotation	60 kgm	150 kgfm²
		Bending	60 kgm	150 kgfm²
		Twisting	36 kgm	80 kgfm²

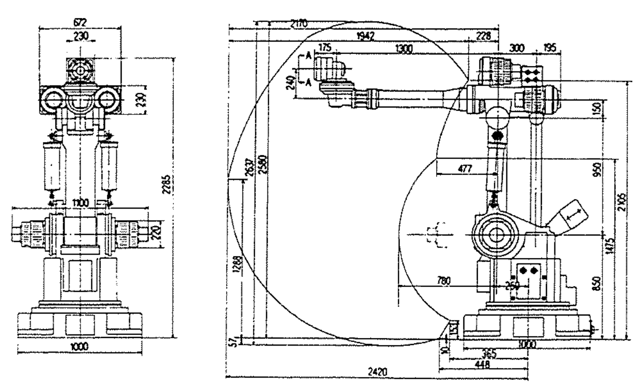
			Specification
Position repeat accuracy			Within ±0.4 mm
Position detector			Absolute encoder
Drive power	Arm	Rotation	4.5 kW (AC servo motor)
		Upper arm	3.6 kW (AC servo motor)
		Fore arm	3.6 kW (AC servo motor)
	Wrist	Rotation	1.6 kW (AC servo motor)
		Bending	1.6 kW (AC servo motor)
		Twisting	1.6 kW (AC servo motor)
Brake			All axis with brake
Ambient temp. & humidity			0-45°C, 20-90%RH (no dew)
Operational limit protection			1. Soft limit
			2. Hard limit (rotation axis)
			3. Mechanical stopper (Standard axes)
Operation lamp			Lighting when servo ON
Installation			Horizontal floor, hanging from ceiling
Outer dimensions			See the dimensional diagram
Grounding			Exclusive Class 3 grounding for robot via control unit.
Total weight of main unit			1,600 kg

Robot Specification/Controller

			Specification
	Model		YA8102AC
Control system	Teaching system		Teaching playback
	Route control system		PTP & CP (Linear or circular interpolation)
	Control axis		6 axis. Option: external 6 axis (PTP)
	Position detection		Absolute encoder
	Position control		Digital closed loop system
	Speed control		Linear speed constant control (in CP control mode)
Memory Display	Memory system		IC memory (battery back-up)
	Memory capacity		Standard 4000 points (2000 steps, 2000 sequences) Possible to increase up to 16,000 points (8,000 steps, 8000 sequences) by optional unit.
	Operation mode		1. Operation 2. Teaching 3. Editing 4. System Setting 5. Cassette Printer 6. Control Data
	Program divisions		254
	Job divisions		127
	Edit protecting function		Provided (write inhibit mark)
	Display mehtod	Operation panel	9-inch CRT & LED
		Teaching box	3-digit x 2-digit display & LED
	External memory		Specified cassette recorder, floppy disc (option)
	Printer		Printer interface (option) Printer (option, specified printer)
Teaching	Interpolating function		1. Linear 2. Circular (3-dimensional plane) 3. PTP 4. Palletizing (option)
	Coordinate system selection		1. Joint 2. Cylinder 3. Cartesian 4. Tool 5. User
	Speed setting method		1 ~ 999 mm/s (direct figure) or 1 ~ 99(%) 5-step direct selection and detail setting possible in teaching
	Welding menu selection		Welding start sequence 15 types (selectable from teaching box)
	Welding gun		Gun ON & OFF
	Pitch feed		0.5, 1.0, 2.0, 5.0, 10 mm/pitch (5 types)
Checking operation	Operation unit		Job unit, program unit, step unit
			Address search
	On-line fine adjustment		1. Speed 2. Condition output 1 & 2 (analog output)
	Step forward/backward		Step forward & step backward
	Correcting function		1. Change (Position, Speed, weld sequence) 2. Addition 3. Deletion
Edition	Type of command	37	1. Output 2. Branch
			3. Counter processing
			4. Time waiting 5. Sub-routine
			6. Others
	Editing function		Copy, division, connection, deletion, addition, change, etc.
	Edit during operation		Edition of job/program except those in operation is possible
Operation	Operation condition command		Job, program, robot lock, input/output lock
	Stop condition command		Job, program, step
	Reserve function		Up to 16 jobs which are not in operation or reserve
	Control function		No. of job executions, robot operation time
	In-fence operation changeover		Speed can be controlled in the safety speed 1-300 m/sec.
External control	Job select input		7-bit input system (max. 127 selectable)
	Input/output for general use	Input	16 points (64 max. by option)
		Output	16 points (64 max. by option)

			Specification
External control	Input/output for exclusive use	Input	1. Start 2. Stop 3. Emerg. Stop 4. Job reserve cancel 5. Error release 6. In-fence operation 7. Teaching permit 8. Teaching select 9. Operation mode select
		Output	1. During operation 2. During stop 3. During emerg. stop 4. Operation mode 5. Teaching mode 6. Mode selectable
	Input/output specification	Input	Photocoupler (DC 24V 12 mA ON/OFF)
		Output	Relay contact (Contact Spec. DC24V 1A)
	External communication		Option (RS232C)
	Analog output		2 ports (condition 1, 2)
Welding control	Welding input/output	Input	1. Hold end 2. Trouble 1 3. Trouble 2 4. Chip sticking 5. Step up 6. Insufficient water pressure
		Output	1. Weld start 1, 2, 4, 8 2. Gun pressure
	Welding sequence setting		15 types of welding start sequence are stored as library
Protection function (Self diagonosis)			1. Mechanical stopper 2. Soft limit 3. CPU trouble monitor 4. Cable connection monitor 5. Power trouble 6. Panel temp. abnormal 7. Servo trouble (overspeed, overcurrent, detector trouble, overload) 8. Welding trouble 9. Operation error
Structure	Structure		Box type hermetic
	Cooling system		Indirect air cooling
	Ambient temp. & humidity		0-45°, 20-90% RH (no dew)
	Input power supply		3-phase AC 200/220V±10% 50/60Hz 15 kVA or over (Tap change needed for 220V)
	Grounding		Exclusive Class 3 grounding for robot
	Outer dimensions		700x560x1600 (WxDxH)
	Weight		approx. 180 kg (Teach pendant, exclusive cable included.)
	Robot cable		Exclusive cable 5m (option 10m) with conector
	Teaching cable		7m (option 10 m) (from CRT console)
	CRT console		Uni-structural with control unit Separable by option (with cables)

Robot Body/Outer Dimensions & Work Envelope



Japan: Matsushita Industrial Equipment Co., Ltd.
Overseas Department
1-1, 3-chome, Inazu-cho, Toyonaka OSAKA 561 JAPAN
TEL: 06(862)1121 FAX.06(866)0709

THIS PRODUCT MAY BE SUBJECT TO EXPORT CONTROL REGULATIONS

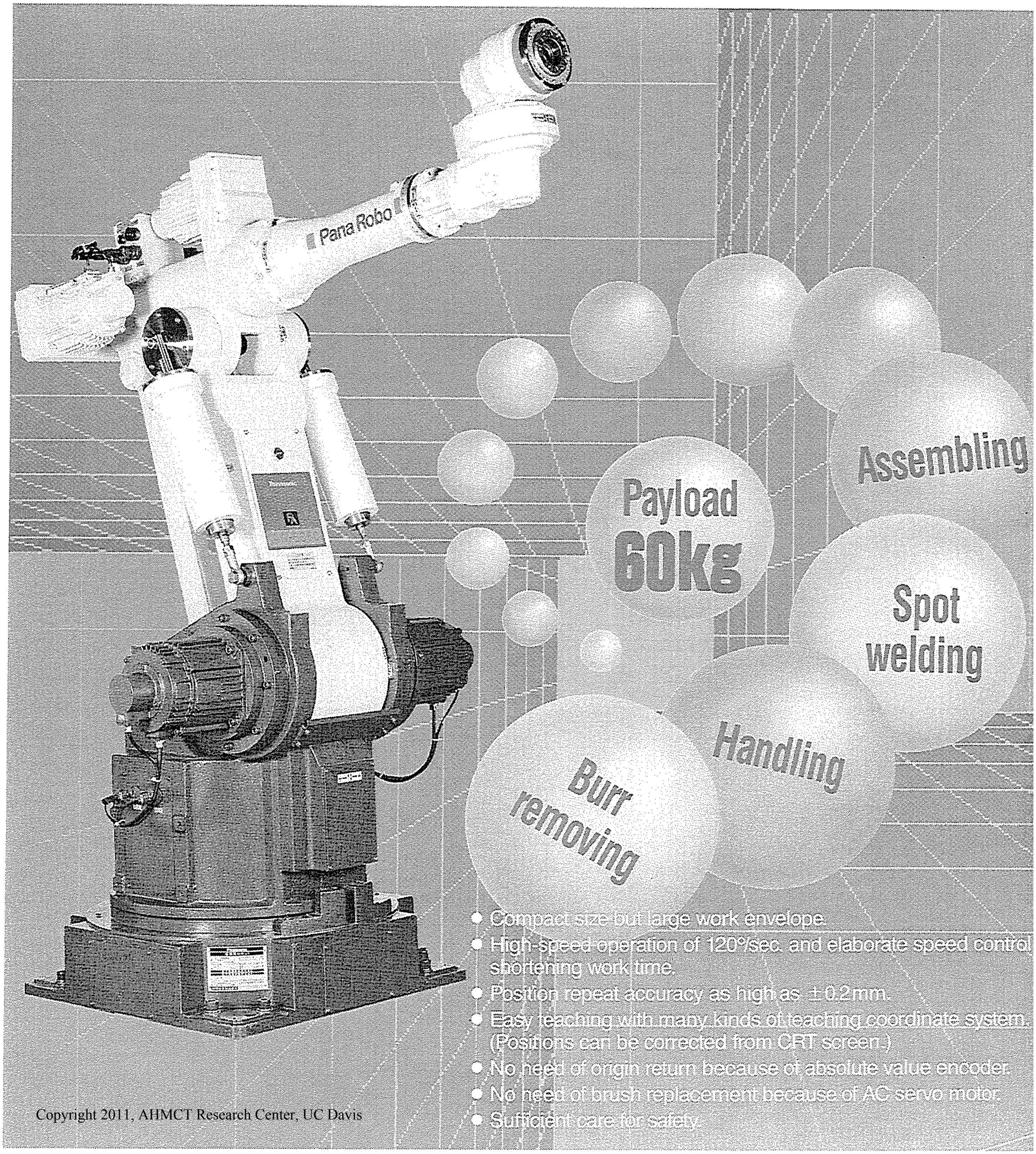
Panasonic
Industrial Robot

Printed in Japan

Panasonic
Industrial Robot

6 axes
Articulated arm robot

Pana Robo AW-8060



- Compact size but large work envelope.
- High-speed operation of 120°/sec. and elaborate speed control shortening work time.
- Position repeat accuracy as high as ±0.2mm.
- Easy teaching with many kinds of teaching coordinate system. (Positions can be corrected from CRT screen.)
- No need of origin return because of absolute value encoder.
- No need of brush replacement because of AC servo motor.
- Sufficient care for safety.

			Specification
Model			YA-8061AM
Structure			Multiple articulation
Degree of freedom			6 axes
Operation range	Arm	Rotation	±150° (Front standard)
		Upper arm	+90° −75° (Vertical standard)
		Front arm	+90° −130° (Horizonal standard)
	Wrist	Rotation	±240°
		Bending	±190° (Front arm standard)
		Twisting	±190°
Work envelope	Arm operation sectional area		$4.2 m^2 \times 300°$
	Arm fore-back operation distance		−720～+2080 mm (From rotation axis center to bending axis center)
	Arm up-down operation distance		−200～+2640m (From robot bottom to bending axis center)
Momentary max. speed	Arm	Rotation	120°/sec
		Upper arm	120°/sec
		Front arm	120°/sec
	Wrist	Rotation	140°/sec
		Bending	160°/sec
		Twisting	240°/sec
Max. pay load			60 kg
Wrist allowable load Moment/inertia		Rotation	35 kgm
		Bending	25 kgm
		Twisting	20 kgm

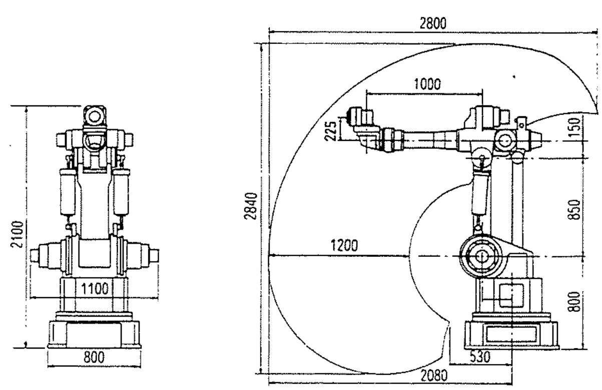
			Specification
Position repeat accuracy			Within ±0.2 mm
Position detector			Absolute encoder
Drive power	Arm	Rotation	3.6 kW (AC servo motor)
		Upper arm	3.6 kW (AC servo motor)
		Fore arm	3.6 kW (AC servo motor)
	Wrist	Rotation	1.2 kW (AC servo motor)
		Bending	1.2 kW (AC servo motor)
		Twisting	1.2 kW (AC servo motor)
Brake			All axes with brake
Ambient temp. & humidity			0—45°, 20—90%RH (no dew)
Operational limit protection			1. Soft limit
			2. Hard limit (rotation axis)
			3. Mechanical stopper (except for wrist axis)
Operation lamp			Lighting when servo ON
Painting color			Munsell 10R5/12 (semi-lustered orange), partially Munsell N3 (black)
Installation			Horizontal floor, hanging from ceiling
Outer dimensions			See the dimensional diagram
Grounding			Exclusive Class 3 grounding for robot via control unit.
Total weight of main unit			1,300 kg

Rated Specification/Controller

			Specification
	Model		YA8061AC
Control system	Teaching system		Teaching playback
	Route control system		PTP & CP (Linear or circular interpolaiton)
	Control axes		6 axes. Option: external 6 axes (PTP)
	Position detection		Absolute encoder
	Position control		Digital closed loop system
	Speed control		Linear speed constant control (in CP control mode)
Memory Display	Memory system		IC memory (battery back-up)
	Memory capacity		Standard 4000 points (2000 step, 2000 sequence) Possible to increase up to 16000 points (8000 step, 8000 sequence) by optional unit.
	Operation mode		1. Operation 2. Teaching 3. Editing 4. System setting 5. Cassette printer 6. Control data
	Program divisions		255
	Job divisions		127
	Edit protecting function		Provided (write inhibit mark)
	Display method	Operation panel	9-inch CRT & LED
		Teaching box	3-digit × 2-digit display & LED
	External memory		Specified cassette recorder, floppy disc (option)
	Printer		Printer interface (option) Printer (option, specified printer)
Teaching	Interpolating function		1. Linear 2. Circular (3-dimensional plane) 3. PTP 4. Palletizing
	Coordinate system selection		1. Joint 2. Cylinder 3. Cartesian 4. Tool 5. User
	Speed setting method		1—999 mm/s (direct figure) or 1—99(%) 5-step direct selection and detail setting possible in teaching
	Welding menu selection		Welding start sequence 15 types (selectable from teaching box)
	Welding gun		Gun ON & OFF
	Pitch feed		0.5, 1.0, 2.0, 5.0, 10 mm/pitch (5 types)
Checking operation	Operation unit		Job unit, program unit, step unit
			Address search
	On-line fine adjustment		1. Speed 2. Condition output 1 & 2 (analog output)
	Step forward/backward		Step forward & step backward
	Correcting function		1. Change (position, speed, weld sequence) 2. Addition 3. Deletion
Edition	Type of command	37	1. Input 2. Branch
			3. Counter processing
			4. Time waiting 5. Sub-routine
			6. Others
	Editing function		Copy, division, connection, deletion, addition, change, etc.
	Edit during operation		Edition of job/program except those in operation is possible
Operation	Operation condition command		Job, program, robot lock, input/output lock
	Stop condition command		Job, program, step
	Reserve function		Up to 16 jobs which are not in operation or reserve
	Control function		No. of job executions, robot operation time
	In-fence operation changeover		Speed limit possible in the range of safety speed 1—300 m/sec.
External control	Job select input		7-bit input system (max. 127 selectable)
	Input/output for general use	Input	16 points (64 max. by option)
		Output	16 points (64 max. by option)

			Specification
External control	Input/output for exclusive use	Input	1. Start 2. Stop 3. Emerg. stop 4. Job reserve cancel 5. Error release 6. In-fence operation 7. Teaching permit 8. Teaching select 9. Operation mode select
		Output	1. During operation 2. During stop 3. During emerg. stop 4. Operation mode 5. Teaching mode 6. Mode selectable
	Input/output specification	Input	Photocoupler (DC 24V 12 mA ON/OFF)
		Output	Relay contact DC 24V 1A
	External communication		Option (RS232C)
	Analog output		2 ports (condition 1, 2)
Welding control	Welding input/output	Input	1. Hold end 2. Trouble 1 3. Trouble 2 4. Chip sticking 5. Step up 6. Insufficient water pressure
		Output	1. Welding start 1, 2, 4, 8 2. Gun pressure
	Welding sequence setting		15 types of welding start sequence are stored as library
Protection function (Self diagonosis)			1. Mechanical stopper 2. Soft limit 3. CPU trouble monitor 4. Cable connection monitor 5. Power trouble 6. Panel temp. abnormal 7. Servo trouble (over-speed, overcurrent, detector trouble, overload) 8. Welding trouble 9. Operation error
Structure	Structure		Box type hermetic
	Cooling system		Indirect air cooling
	Ambient temp. & humidity		0—45°, 20—90%RH (no dew)
	Input power supply		3-phase AC 200/220±10% 50/60 Hz 15 kVA or over (Tap change needed for 220V)
	Grounding		Exclusive Class 3 grounding for robot
	Painting color		Munsell 5Y8/1
	Outer dimensions		750 × 600 × 1520 (W × D × H)
	Weight		Approx. 80 kg (teaching box, cable included)
	Robot cable		Exclusive cable 5 m with connector
	Teaching cable		7 m (from CRT console)
	CRT console		Uni-structural with control unit Separable by option (with cables)

Robot Body/Outer Dimensions & Work Envelope



Japan: Matsushita Industrial Equipment Co., Ltd.
Overseas Department
1-1, 3-chome, Inazu-cho, Toyonaka OSAKA 561 JAPAN
TEL. 06 (862) 1121 FAX. 06 (862) 1441

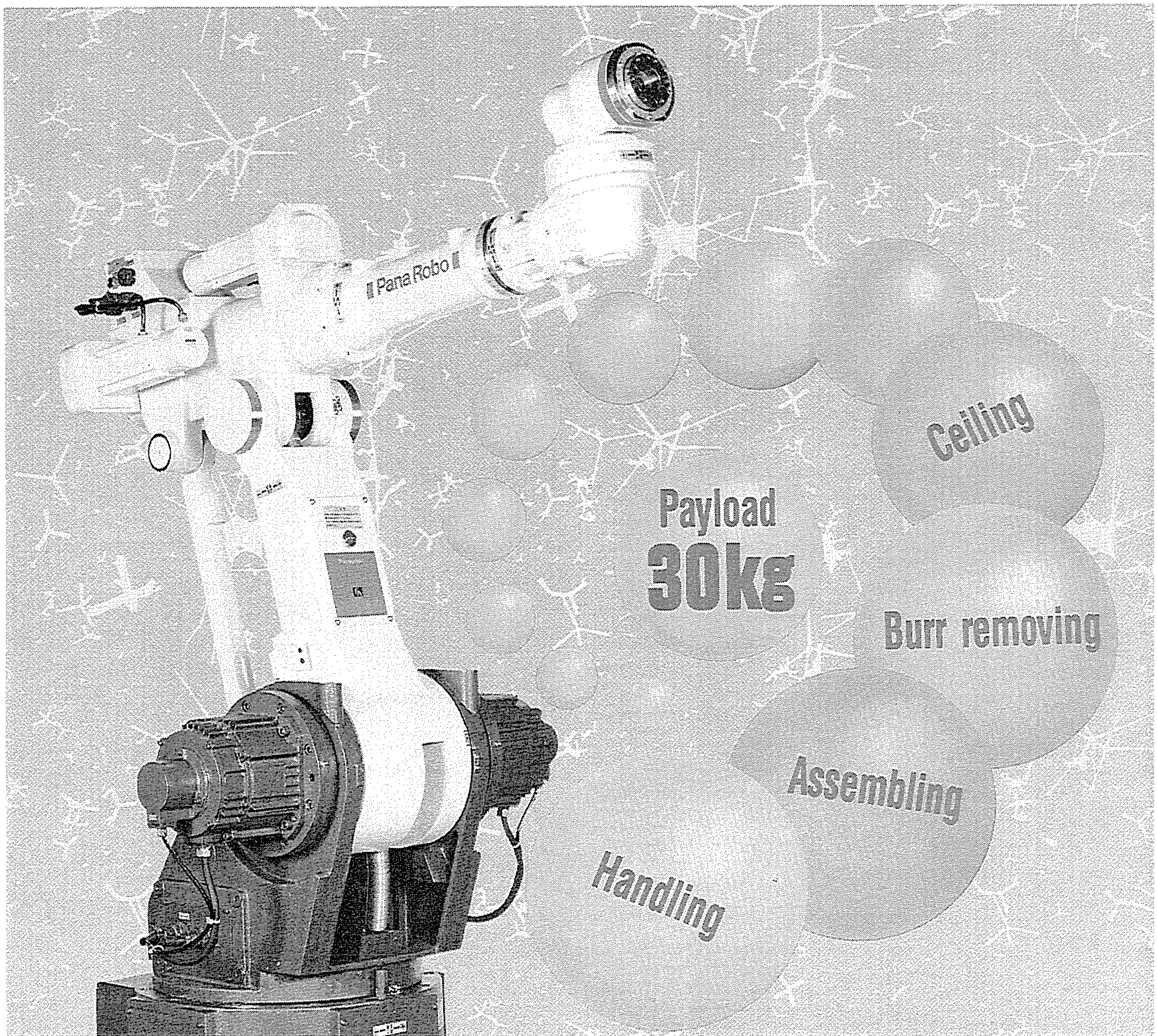
Panasonic.
Industrial Robot

Printed in Japan

Panasonic
Industrial Robot

6 axis
Articulated arm robot

Pana Robo AW-8030



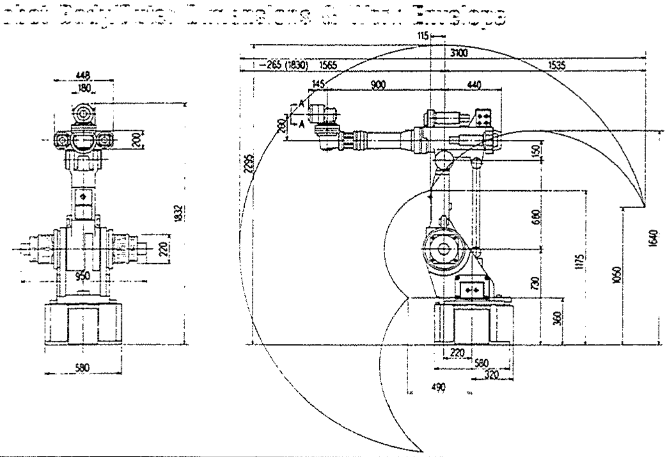
- Compact size but large work envelope.
- High speed operation of 120°/sec. and elaborate speed contro shortening tact time
- Position repeat accuracy as high as ±0.15 mm
- Easy teaching with many kings of teaching coodinate system. (positions can be corrected from CRT screen.)
- No need of origin return because of absolute value encoder.
- No need of brush replacement because of AC servo motor.
- Sufficient care for safety.

			Specification	
Model			YA8031AM	
Structure			Multiple articulation	
Degree of freedom			6 axis	
Operation range	Arm	Rotation	±180° (Front standard)	
		Upper arm	+150°–75° (Front standard)	
		Front arm	+135°–110° (Horizontal standard)	
	Wrist	Rotation	±240°	
		Bending	±190° (Front arm standard)	
		Twisting	±350°	
Work envelope	Arm operation sectional area		$3.9m^2$x360°	
	Arm fore-back operation distance		−1400 ~ +1565mm (From rotation axis center to bending axis center)	
	Arm up-down operation distance		−770 ~ +2295 mm (From robot bottom to bending axis center)	
Momentary max. speed	Arm	Rotation	120°/sec	
		Upper arm	120°/sec	
		Front arm	120°/sec	
	Wrist	Rotation	200°/sec	
		Bending	200°/sec	
		Twisting	300°/sec	
Max. pay load			30 kg	
Wrist allowable load Moment/inertia		Rotation	25 kgm	40 $kgfm^2$
		Bending	15 kgm	30 $kgfm^2$
		Twisting	12 kgm	20 $kgfm^2$

			Specification
Position repeat accuracy			Within ±0.15 mm
Position detector			Absolute encoder
Drive power	Arm	Rotation	2.5 kW (AC servo motor)
		Upper arm	2.5 kW (AC servo motor)
		Fore arm	2.5 kW (AC servo motor)
	Wrist	Rotation	0.72 kW (AC servo motor)
		Bending	0.72 kW (AC servo motor)
		Twisting	0.72 kW (AC servo motor)
Brake			All axis with brake
Ambient temp. & humidity			0-45°C, 20-90%RH (no dew)
Operational limit protection			1. Soft limit
			2. Hard limit (rotation axis)
			3. Mechanical stopper (except for wrist axis)
Operation lamp			Lighting when servo ON
Installation			Horizontal floor, hanging from ceiling
Outer dimensions			See the dimensional diagram
Grounding			Exclusive Class 3 grounding for robot via control unit.
Total weight of main unit			600 kg

			Specification
	Model		YA8032AC
Control system	Teaching system		Teaching playback
	Route control system		PTP & CP (Linear or circular interpolation)
	Control axis		6 axis. Option: external 6 axis (PTP)
	Position detection		Absolute encoder
	Position control		Digital closed loop system
	Speed control		Linear speed constant control (in CP control mode)
Memory Display	Memory system		IC memory (battery back-up)
	Memory capacity		Standard 4000 points (2000 steps, 2000 sequences) Possible to increase up to 16,000 points (8,000 steps, 8000 sequences) by optional unit.
	Operation mode		1. Operation 2. Teaching 3. Editing 4. System Setting 5. Cassette Printer 6. Control Data
	Program divisions		254
	Job divisions		127
	Edit protecting function		Provided (write inhibit mark)
	Display mehtod	Operation panel	9-inch CRT & LED
		Teaching box	3-digit x 2-digit display & LED
	External memory		Specified cassette recorder, floppy disc (option)
	Printer		Printer interface (option) Printer (option, specified printer)
Teaching	Interpolating function		1. Linear 2. Circular (3-dimensional plane) 3. PTP 4. Palletizing (option)
	Coordinate system selection		1. Joint 2. Cylinder 3. Cartesian 4. Tool 5. User
	Speed setting method		1 ~ 999 mm/s (direct figure) or 1 ~ 99(%) 5-step direct selection and detail setting possible in teaching
	Welding menu selection		Welding start sequence 15 types (selectable from teaching box)
	Welding gun		Gun ON & OFF
	Pitch feed		0.5, 1.0, 2.0, 5.0, 10 mm/pitch (5 types)
Checking operation	Operation unit		Job unit, program unit, step unit
			Address search
	On-line fine adjustment		1. Speed 2. Condition output 1 & 2 (analog output)
	Step forward/backward		Step forward & step backward
	Correcting function		1. Change (Position, Speed, weld sequence) 2. Addition 3. Deletion
Edition	Type of command	37	1. Output 2. Branch
			3. Counter processing
			4. Time waiting 5. Sub-routine
			6. Others
	Editing function		Copy, division, connection, deletion, addition, change, etc.
	Edit during operation		Edition of job/program except those in operation is possible
Operation	Operation condition command		Job, program, robot lock, input/output lock
	Stop condition command		Job, program, step
	Reserve function		Up to 16 jobs which are not in operation or reserve
	Control function		No. of job executions, robot operation time
	In-fence operation changeover		Speed can be controlled in the safety speed 1-300 m/sec.
External control	Job select input		7-bit input system (max. 127 selectable)
	Input/output for general use	Input	16 points (64 max. by option)
		Output	16 points (64 max. by option)

			Specification
External control	Input/output for exclusive use	Input	1. Start 2. Stop 3. Emerg. Stop 4. Job reserve cancel 5. Error release 6. In-fence operation 7. Teaching permit 8. Teaching select 9. Operation mode select
		Output	1. During operation 2. During stop 3. During emerg. stop 4. Operation mode 5. Teaching mode 6. Mode selectable
	Input/output specification	Input	Photocoupler (DC 24V 12 mA ON/OFF)
		Output	Relay contact (Contact Spec. DC24V 1A)
	External communication		Option (RS232C)
	Analog output		2 ports (condition 1, 2)
Welding control	Welding input/output	Input	1. Hold end 2. Trouble 1 3. Trouble 2 4. Chip sticking 5. Step up 6. Insufficient water pressure
		Output	1. Weld start 1, 2, 4, 8 2. Gun pressure
	Welding sequence setting		15 types of welding start sequence are stored as library
Protection function (Self diagonosis)			1. Mechanical stopper 2. Soft limit 3. CPU trouble monitor 4. Cable connection monitor 5. Power trouble 6. Panel temp. abnormal 7. Servo trouble (overspeed, overcurrent, detector trouble, overload) 8. Welding trouble 9. Operation error
Structure	Structure		Box type hermetic
	Cooling system		Indirect air cooling
	Ambient temp. & humidity		0-45°, 20-90% RH (no dew)
	Input power supply		3-phase AC 200/220V±10% 50/60Hz 15 kVA or over (Tap change needed for 220V)
	Grounding		Exclusive Class 3 grounding for robot
	Outer dimensions		700x560x1600 (WxDxH)
	Weight		approx. 180 kg (Teach pendant, exclusive cable included.)
	Robot cable		Exclusive cable 5m (option 10m) with conector
	Teaching cable		7m (option 10 m) (from CRT console)
	CRT console		Uni-structural with control unit Separable by option (with cables)



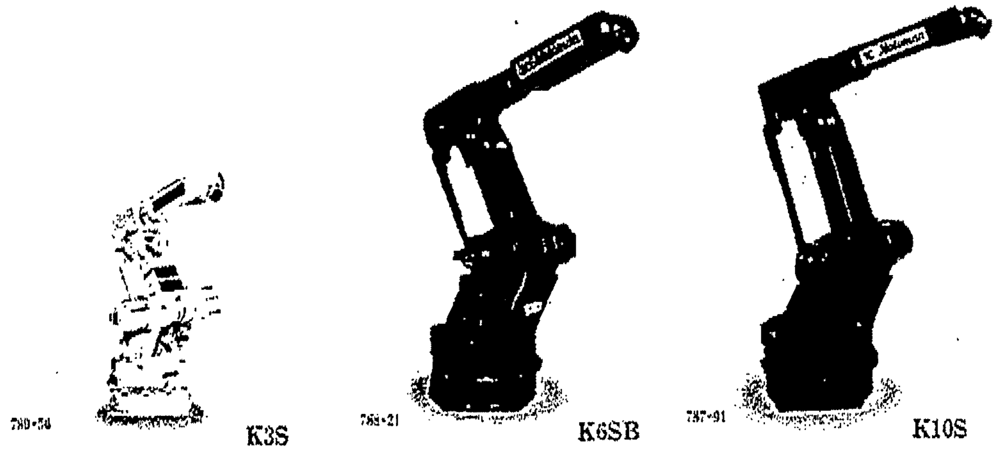
Japan: Matsushita Industrial Equipment Co., Ltd.
Overseas Department
1-1, 3-chome, Inazu-cho, Toyonaka OSAKA 561 JAPAN
TEL: 06(862)1121 FAX.06(866)0709

Panasonic
Industrial Robot

Printed in Japan

MOTOMAN SERIES—CONSTANTLY A

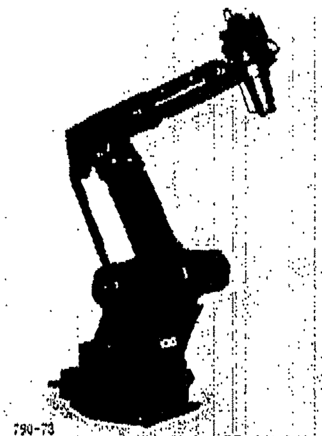
K Series



K3S K6SB K10S

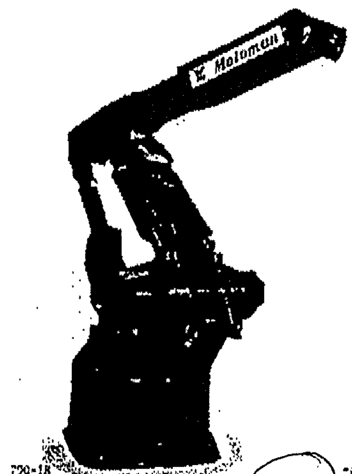
Specifications			Manipulator K3S	Manipulator K6SB	Manipulator K10S
Applications	Arc Welding		APPLICABLE	APPLICABLE	APPLICABLE
	Assembly		APPLICABLE	APPLICABLE	APPLICABLE
	Dispensing			APPLICABLE	APPLICABLE
	Material Cutting (Laser, Water, Plasma, Gas)			APPLICABLE	APPLICABLE
	Material Handling		APPLICABLE	APPLICABLE	APPLICABLE
	Material Removal				APPLICABLE
	Spot Welding				
	Surface Finishing				APPLICABLE
	Thermal Coating			APPLICABLE	APPLICABLE
Features			•Floor, ceiling & optional wall mount. •Fast cycle rates 2.5m/s 98.4"/s •Small footprint 0.11m² 1.2ft²	•Floor, ceiling & optional wall mount. •Integrated arc welding package. •Optional extended reach (MS) version 1775mm 69.94"	•Floor, wall or ceiling mount. •Integrated arc welding package •Optional extended reach (MS) version 2577mm 101.53"
Controlled Axes			6 degrees of freedom, vertical jointed-arm type	6 degrees of freedom, vertical jointed-arm type	6 degress of freedom, vertical jointed-arm type
Maximum Motion Range and Speed	Arm		S-axis turning: 340°, 2.61 rad/s (150°/s)	S-axis turning: 340°, 1.91 rad/s (110°/s)	S-axis turning: 340°, 2.09 rad/s (120°/s)
			L-axis lower arm movement: 240°, 3.49 rad/s (200°/s)	L-axis lower arm movement: 240°, 1.57 rad/s (90°/s)	L-axis lower arm movement: 240°, 2.09 rad/s (120°/s)
			U-axis upper arm movement: 260°, 3.49 rad/s (200°/s)	U-axis upper arm movement: 270°, 1.92 rad/s (110°/s)	U-axis upper arm movement: 275°, 2.09 rad/s (120°/s)
	Wrist		R-axis roll: 360°, 4.89 rad/s (280°/s)	R-axis roll: 360°, 4.19 rad/s (240°/s)	R-axis roll: 360°, 4.59 rad/s (263°/s)
			B-axis pitch/yaw: 270°, 4.89 rad/s (280°/s)	B-axis pitch/yaw: 270°, 4.19 rad/s (240°/s)	B-axis pitch/yaw: 270°, 4.59 rad/s (263°/s)
			T-axis twist: 400°, 7.33 rad/s (420°/s)	T-axis twist: 400°, 6.98 rad/s (400°/s)	T-axis twist: 400°, 6.98 rad/s (400°/s)
Reach			859mm 33.82"	1322mm 52.01"	1555mm 61.22"
Repetitive Positioning Accuracy			±0.1 mm ±0.004"	±0.1 mm ±0.004"	±0.1 mm ±0.004"
Payload			3 kg 6.6 lb	6 kg 13 lb	10 kg 22 lb

VANCING, ALWAYS GETTING BETTER



790-73 K10ASB

789-05 K30S



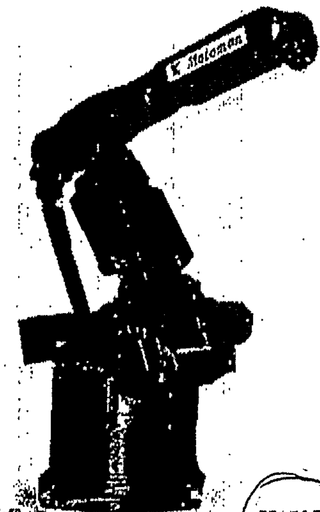
790-18 K60S

788-03 K100S

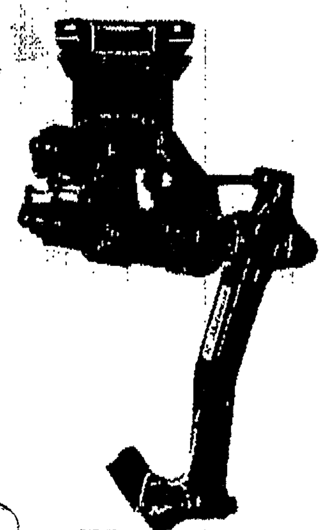
Manipulator K10ASB	Manipulator K30SB	Manipulator K60S	Manipulator K100S
	APPLICABLE		
	APPLICABLE		
APPLICABLE	APPLICABLE		
	APPLICABLE	APPLICABLE	APPLICABLE
d 6th & 7th Axes for cutting	APPLICABLE	APPLICABLE	APPLICABLE
using		APPLICABLE	APPLICABLE
l floor or wall mount	APPLICABLE	APPLICABLE	APPLICABLE
	APPLICABLE		
pe | Dimensions | Max. Speed ied) | 2 to 30 mm 0.0787" to 1.181" | 9.9 m/min 389.8"/min | Full Working Range | 2.0 to 2.5m/min 78.7" to 98.4"/min	•Floor, wall or ceiling mount •Durable cycloidal wrist design. •Optional extended reach (WS) version 1971mm 77.66"	•Durable cycloidal wrist design. •Optional shelf mount version. •Optimized medium payload design (Versus "Modular").	•Durable cycloidal wrist design. •Optional shelf mount version.
of freedom, vertical m type	6 degrees of freedom, vertical jointed-arm type	6 degrees of freedom, veritcal joint ed-arm type	6 degrees of freedom, vertical jointed-arm type
ing:)9 rad/s (120°/s)	S-axis turning: 300°, 2.09 rad/s (120°/s)	S-axis turning: 300°, 2.00 rad/s (115°/s)	S-axis turning: 300°, 1.92 rad/s (110°/s)
er arm movement:)9 rad/s (120°/s)	L-axis lower arm movement: 240°, 2.09 rad/s (120°/s)	L-axis lower arm movement: 115°, 2.00 rad/s (115°/s)	L-axis lower arm movement: 115°, 1.92 rad/s (110°/s)
er arm movement:)9 rad/s (120°/s)	U-axis upper arm movement: 260°, 2.09 rad/s (120°/s)	U-axis upper arm movement: 140°, 2.00 rad/s (115°/s)	U-axis upper arm movement: 140°, 1.92 rad/s (110°/s)
:)9 rad/s (263°/s)	R-axis roll: 450°, 3.49 rad/s (200°/s)	R-axis roll: 380°, 2.79 rad/s (160°/s)	R-axis roll: 380°, 2.44 rad/s (140°/s)
	B-axis pitch/yaw: 270°, 3.49 rad/s (200°/s)	B-axis pitch/yaw: 270°, 2.79 rad/s (160°/s)	B-axis pitch/yaw: 260°, 2.44 rad/s (140°/s)
h/yaw: 9 rad/s (263°/s)	T-axis twist: 700°, 5.24 rad/s (300°/s)	T-axis twist: 700°, 4.18 rad/s (240°/s)	T-axis twist: 700°, 4.19 rad/s (240°/s)
61.22"	1787mm 70.35"	2003mm 78.86" 6.5'	2387mm 93.98" 7.75'
±0.004"	±0.2 mm ±0.008"	±0.3 mm ±0.012"	±0.5 mm ±0.020"
b	30 kg 66 lb	60 kg 132 lb	100 kg 221 lb
33 lb	600 kg 1323 lb	980 kg 2160 lb	1600 kg 3527 lb

Ceiling

788-63 K120S

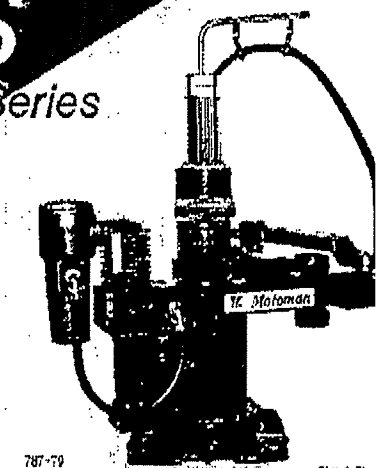


788-63 K150S



787-39 K205SB

S Series



787-79 S50S

Manipulator K120S	Manipulator K150S	Manipulator K205SB
APPLICABLE	APPLICABLE	APPLICABLE
APPLICABLE	APPLICABLE	
APPLICABLE	APPLICABLE	
APPLICABLE	APPLICABLE	
•Durable cycloidal wrist design. •Optimized performance for payload (Versus "Modular" design).	•Durable cycloidal wrist design. •Largest payload in its class. •Optimized performance for payload (Versus "Modular" design).	•Floor and ceiling mount. •Fast cycle rates 1.5 sec in/out •Durable dedicated design.
6 degrees of freedom, vertical jointed-arm type	6 degrees of freedom, vertical jointed-arm type	5 degrees of freedom, vertical jointed-arm type
S-axis turning: 300°, 1.75 rad/s (100°/s)	S-axis turning: 300°, 1.57 rad/s (90°/s)	S-axis turning: 270°, 2.61 rad/s (150°/s)
L-axis lower arm movement: 115°, 1.75 rad/s (100°/s)	L-axis lower arm movement: 115°, 1.57 rad/s (90°/s)	L-axis lower arm movement: 75°, 1.95 rad/s (112°/s)
U-axis upper arm movement: 140°, 1.75 rad/s (100°/s)	U-axis upper arm movement: 140°, 1.57 rad/s (90°/s)	U-axis upper arm movement: 90°, 2.43 rad/s (139.5°/s)
R-axis roll: 380°, 2.44 rad/s (140°/s)	R-axis roll: 380°, 2.44 rad/s (140°/s)	B-axis pitch/yaw: 90°, 3.25 rad/s (186°/s)
B-axis pitch/yaw: 260°, 2.44 rad/s (140°/s)	B-axis pitch/yaw: 260°, 2.44 rad/s (140°/s)	
T-axis twist: 700°, 4.19 rad/s (240°/s)	T-axis twist: 700°, 4.19 rad/s (240°/s)	T-axis twist: 360°, 4.14 rad/s (237°/s)
2387 mm 93.98"	2387 mm 93.98"	1670 mm 65.75"
±0.5 mm ±0.020"	±0.5 mm ± 0.020"	±0.3 mm ±0.012"
120 kg 265.2 lb	150 kg 331.5 lb	20 kg 44 lb
1630 kg 3602 lb	1600 kg 3536 lb	700 kg 1543 lb

Specifications		Manipul
Applications	Arc Welding	
	Assembly	
	Dispensing	
	Material Cutting (Laser, Water, Plasma, Gas)	
	Material Handling	APPL
	Material Removal	APPL
	Spot Welding	APPL
	Surface Finishing	APPL
	Thermal Coating	
Features		•Compact S(•Modular wri •High S Valu
Controlled Axes		4 degrees of horizontal join
Maximum Motion Range and Speed		S-axis horizon 210°, 1.68 r
		L-axis horizon 98°, 1.68 ra
		U-axis vertical 150 mm 5.9 256 mm/s 1
		R-axis turning 450°, 2.09 r
Reach		1315mm 51.
Repetitive Positioning Accuracy		±0.2 mm ±
Payload		50 kg 110 lb
Weight		420 kg 926

6425

IMT's 6425 offers lift capacities from 2,200 lbs. at 25′ 8″ to 10,000 lbs. at 6′ 4″. The optional manual 64″ extension to 31′ handles up to 1,500 lbs. at over 40′.

For traveling and maximum payload space, the 6425 stores compactly between cab and body in a figure-four position.

Lifting capacities

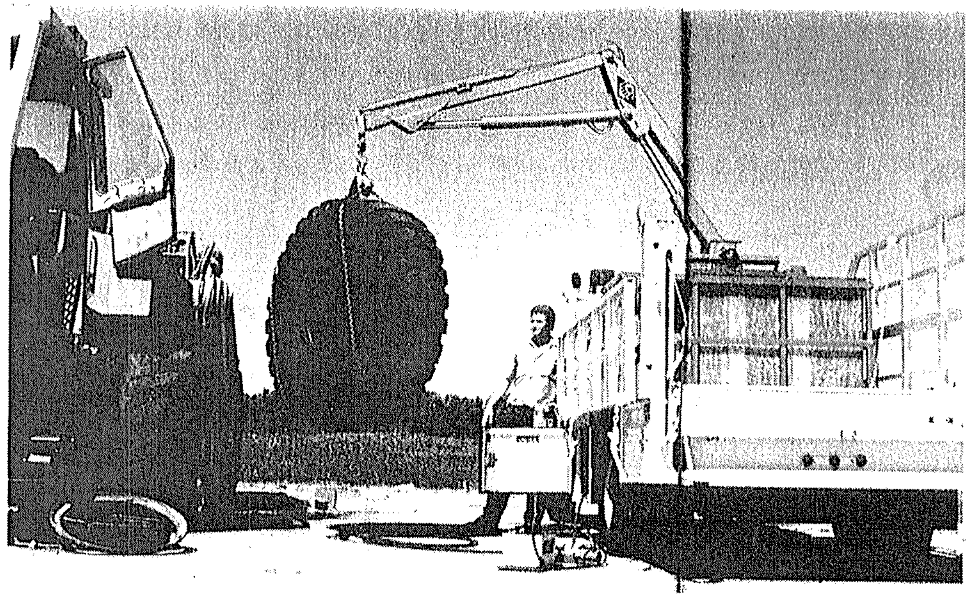
6′ 4″ (1.93 m)	10,000 lbs. (4,536 kg)
8′ (2.44 m)	8,000 lbs. (3,629 kg)
10′ (3.05 m)	6,400 lbs. (2,903 kg)
15′ (4.57 m)	4,250 lbs. (1,928 kg)
20′ 4″ (6.20 m)	2,950 lbs. (1,338 kg)
25′ 8″ (7.82 m)	2,200 lbs. (998 kg)
31′ (9.45 m)	1,500 lbs. (680 kg)

Minimum 26,500 lb. (12,020 kg) GVW chassis

Specifications

Crane rating	64,000 ft.-lbs. (8.85 ton-m
Standard boom length	25′ 8″ (7.82 m
Max. horizontal reach	31′ (9.45 m
Max. vertical lift	40′ 3″ (12.27 m)
Mounting space	30″ (76.2 cm)
Crane weight	3,950 lbs. (1,792 kg)
Stowed height	11′ 2″ (3.40m)
Rotation	370° (6.46 rad.)
Outrigger span	12′ 3″ (3.73 m)
Outrigger style	Out and down
Working pressure	2,750 PSI (193.3 kg/cm^2)
Pump capacity	9 GPM (34 liter/min.)
RBM required	900,000 in.-lbs. (10,373 kg-m)

2115



Designed for trucks one ton and larger, the 2115 handles up to 3,500 lbs. at 6′. Additionally, it has a standard boom reach of 15′ where it will handle up to 1,400 lbs.

Smooth lifting and handling are accomplished with the 2115's hydraulic power system and the high degree of articulation allows for extremely close-in load placement.

Standard features include remote control with 25-foot cable, dual outriggers, a 36 inch hydraulic boom extension, and a full 400 degree rotational system.

Lifting capacities

6′ (1.83 m)	**3,500 lbs. (1,590 kg)**
12′ (3.66 m)	**1,750 lbs. (795 kg)**
15′ (4.57 m)	**1,400 lbs. (636 kg)**

Minimum 11,500 lb. (5,216 kg) GVW chassis

Specifications

Crane rating	21,000 ft.-lb. (2.91 ton-m)
Standard boom length	15′ (4.57 m)
Max. horizontal reach	15′ (4.57 m)
Max. vertical lift	23′ 1″ (7.03 m)
Mounting space	22″ (55.9 cm)
Crane weight	1,200 lbs. (544 kg)
Stowed height	8′ 2″ (2.49 m)
Rotation	400° (6.96 rad.)
Outrigger span	8′ 10″ (2.69 m)
Outrigger style	Out and down
Working pressure	2,350 PSI (165.2 kg/cm^2)
Pump capacity	3 GPM (11.4 liter/min.)
RBM required	290,000 in.-lbs. (3,342 kg-m)

20017

With lift capacities ranging from 24,000 lbs. at 8′ to 11,500 lbs. at 17′, the 20017 is IMT's heavy-duty tirehandler.

Teamed-up with the IMT Tirehand #12, the 20017 will handle 36:00 x 51 tires weighing up to 7,700 lbs.

Standard crane features include hydraulic outriggers with 15′ of stability, operator controls on both sides of the crane for operator convenience and optimum load visibility, and hydraulic 370 degree rotational system.

Lifting capacities

8′ (2.44 m)	**24,000 lbs. (10,886 kg)**
10′ (3.05 m)	**20,000 lbs. (9,072 kg)**
13′ 8″ (4.17 m)	**14,500 lbs. (6,577 kg)**
17′ (5.18 m)	**11,500 lbs. (5,216 kg)**

Minimum 54,000 lb. (24,494 kg) GVW chassis

Specifications

Crane rating	200,000 ft.-lb. (27.66 ton-m
Standard boom length	17′ (5.18 m
Max. horizontal reach	17′ (5.18 m
Max. vertical lift	28′ 2″ (8.59 m
Mounting space	36″ (91.4 cm
Crane weight	7,630 lbs. (3,467 kg
Stowed height	12′ 2″ (3.71 m
Rotation	370 (6.46 rad
Outrigger span	15′ (4.57 m
Outrigger style	Fold-ove
Working pressure	2,500 PSI (175.7 kg/cm
Pump capacity	16 GPM (60.61 liter/min
RBM required	3,000,000 in.-lbs. (34,575 kg-m

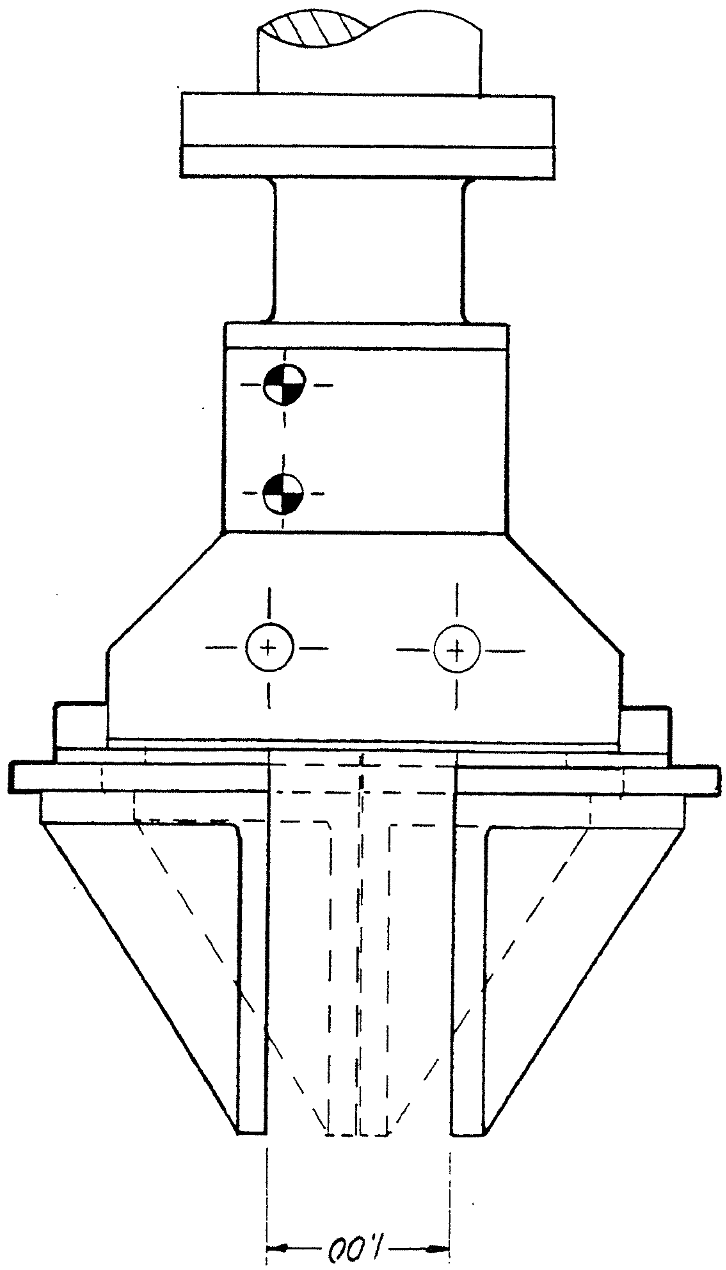
Appendix B Puma End Effector Working Drawings

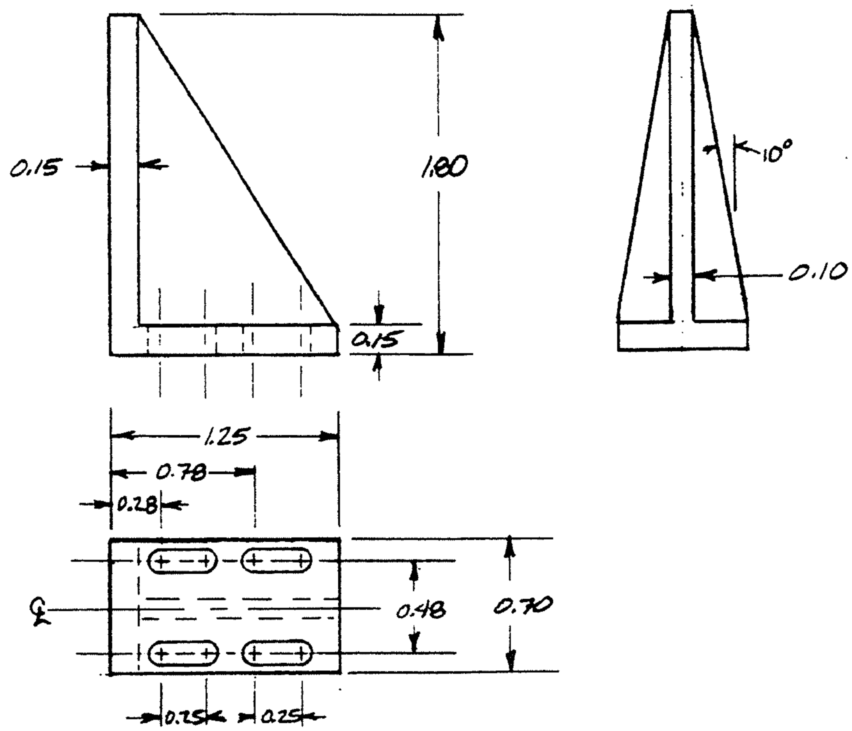
Gripper

Clipper

UNIVERSITY of CALIFORNIA at DAVIS UCD ENGINEERING

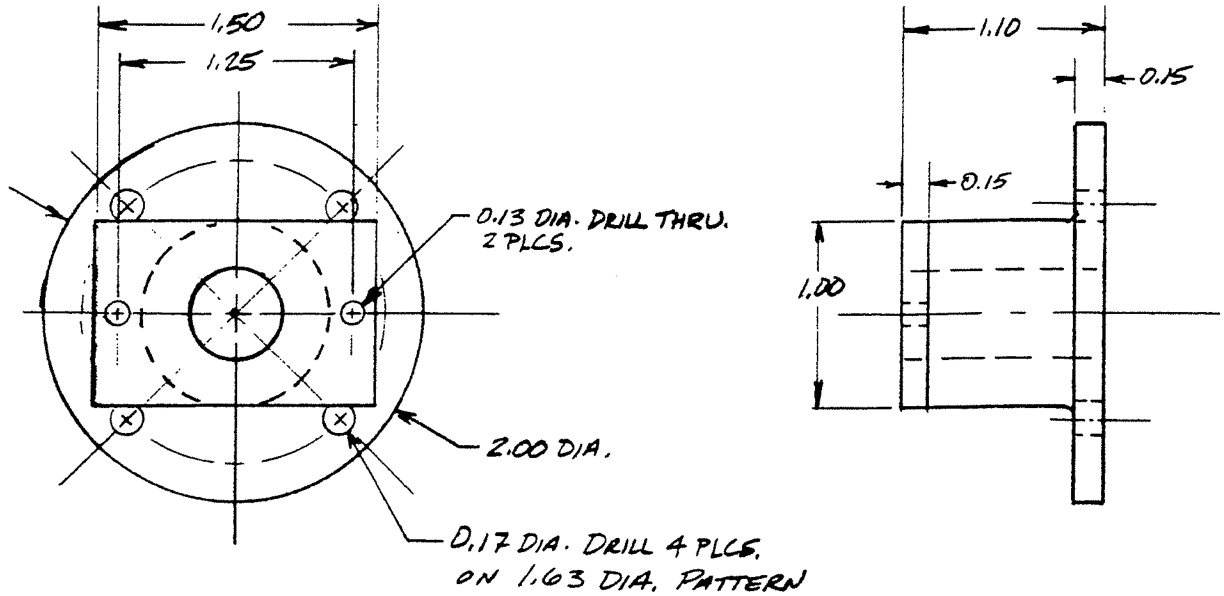
TITLE: GRIPPER ASM	SCALE: FULL
DESIGN: B. COBENE	REV: 1
DR: B.C.	PAGE: 1 of 3
	DATE: 6/26/92





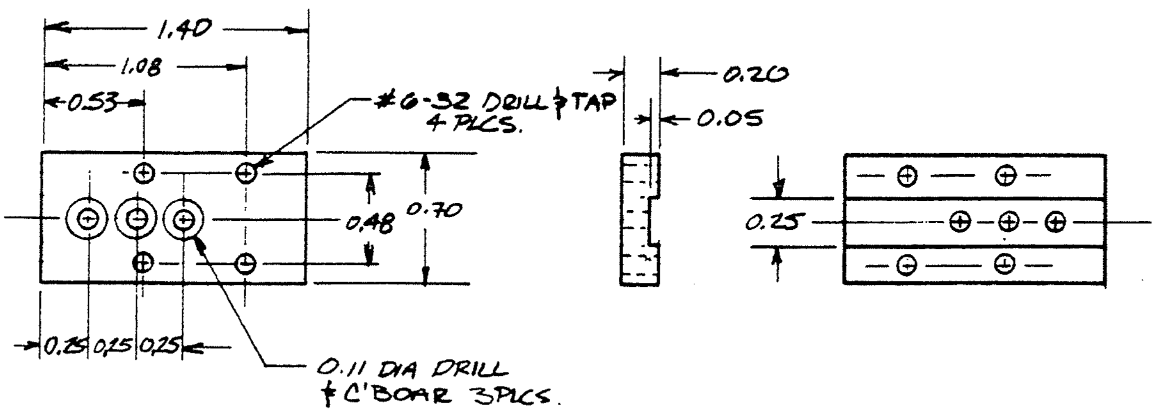
JAW
SCALE 1:1 MTL: 6061-T6 Al
(2) REQD.

UNIVERSITY of CALIFORNIA at DAVIS UCD ENGINEERING	DATE: 6/27/92
TITLE: GRIPPER : JAW	SCALE FULL
DESIGN B. COBENE	REV 1
DR B.C.	PAGE 2 of 3



BASE

SCALE 1:1 MTL: 6061-T6 Al.



JAW BASE

SCALE 1:1 MTL 6061-T6

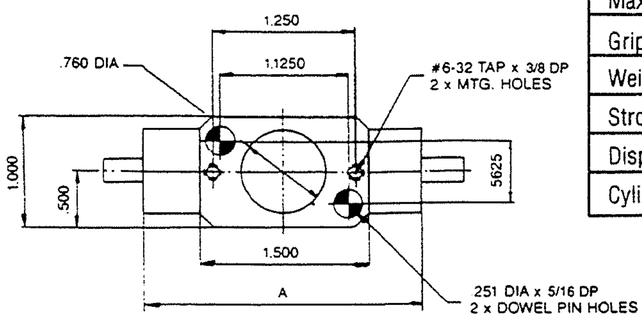
(2) REQ'D

UNIVERSITY of CALIFORNIA at DAVIS UCD ENGINEERING	DATE: 6/30/92
TITLE: GRIPPER: BASE/JAW BASE	SCALE FULL
DESIGN B. COBENE	REV 1
DR. B.C.	PAGE 3 of 3

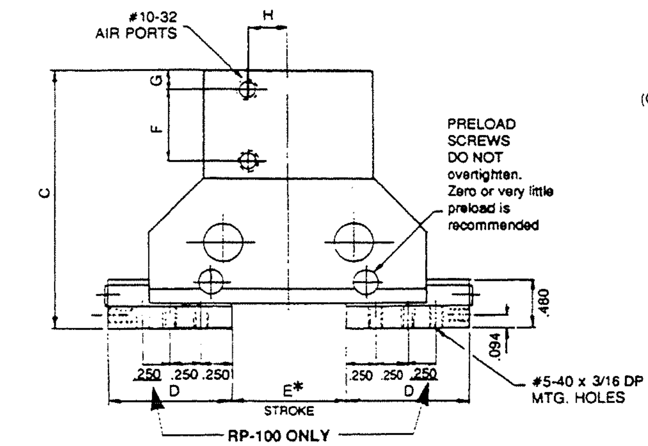
PARALLEL GRIPPER RP-50P RP-100P

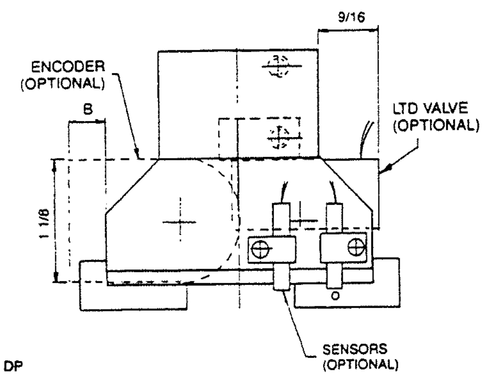
SPECIFICATIONS	RP-50P	RP-100P
Maximum Air Pressure	100 psi	100 psi
Gripping Force @ 80 psi	20 lbs.	30 lbs.
Weight	5 oz.	8 oz.
Stroke	1/2	1"
Displacement	.10 cu. in.	.22 cu. in.
Cylinder Bore	5/8"	3/4"

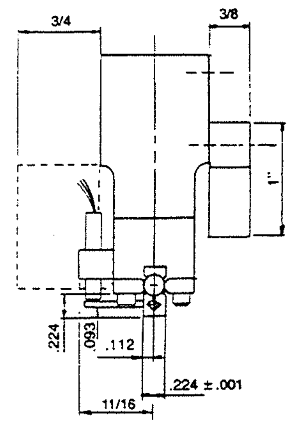
DIMENSIONS	RP-50P	RP-100P
A	2.000	2.750
B	3/8	5/16
C	1.84	2.34
D	.875	1.125
E	1/2	1"
F	.375	.600
G	.220	.197
H	.300	.422



* All dowel hole diameters are S.F. Locations are held to ± .0005.





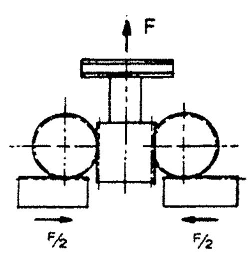


* FINGERS SHOWN OPEN

To Order, Please See Other Side of This Page

GRIPPING FORCE

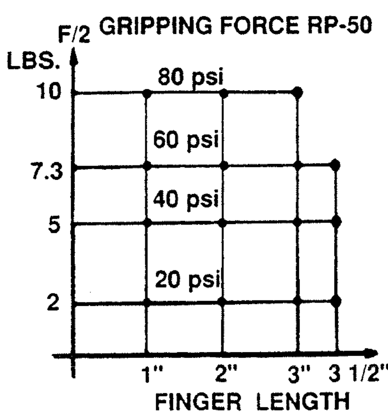
Gripping force is proportional to air pressure.

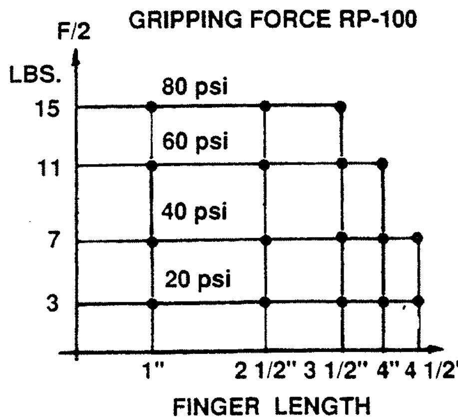


At 80 PSI air pressure

RP-50P F = 20 lbs. F/2 = 10 lbs.

RP-100P F = 30 lbs. F/2 = 15 lbs.





MAXIMUM ALLOWABLE FINGER LENGTH

WARNING WARNING
WARNING WARNING

DO NOT EXCEED MAXIMUM ALLOWABLE FINGER LENGTH AS SPECIFIED ON CHARTS AT LEFT. EXCEEDING THE VALUES CAN CAUSE PRE-MATURE WEAR.

UNIVERSITY of CALIFORNIA at DAVIS — UCD ENGINEERING

TITLE: CLIPPER ASM.

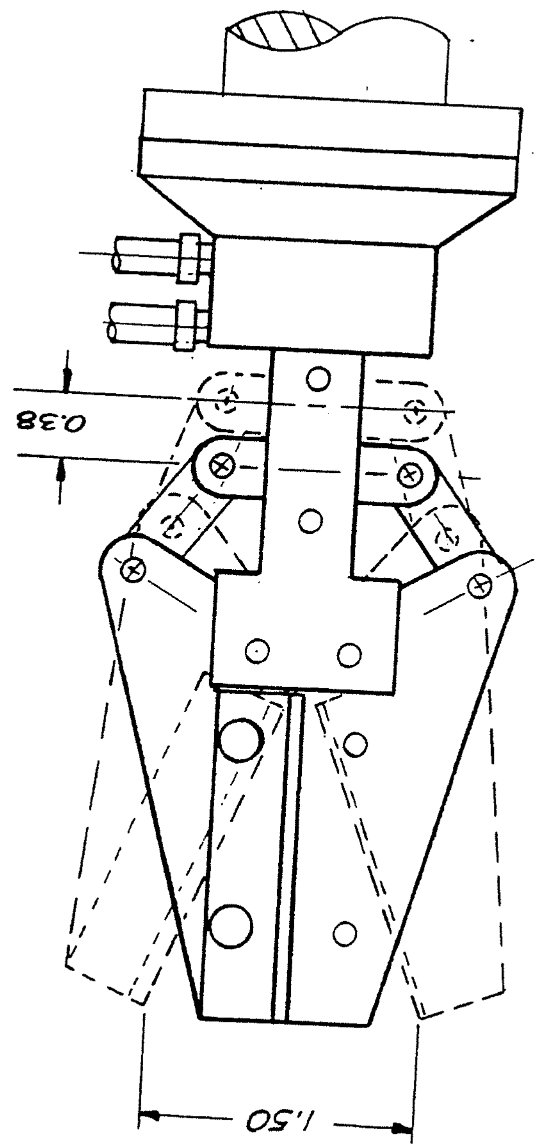
DESIGN: Bob Cogene ; UCD Robotics Lab

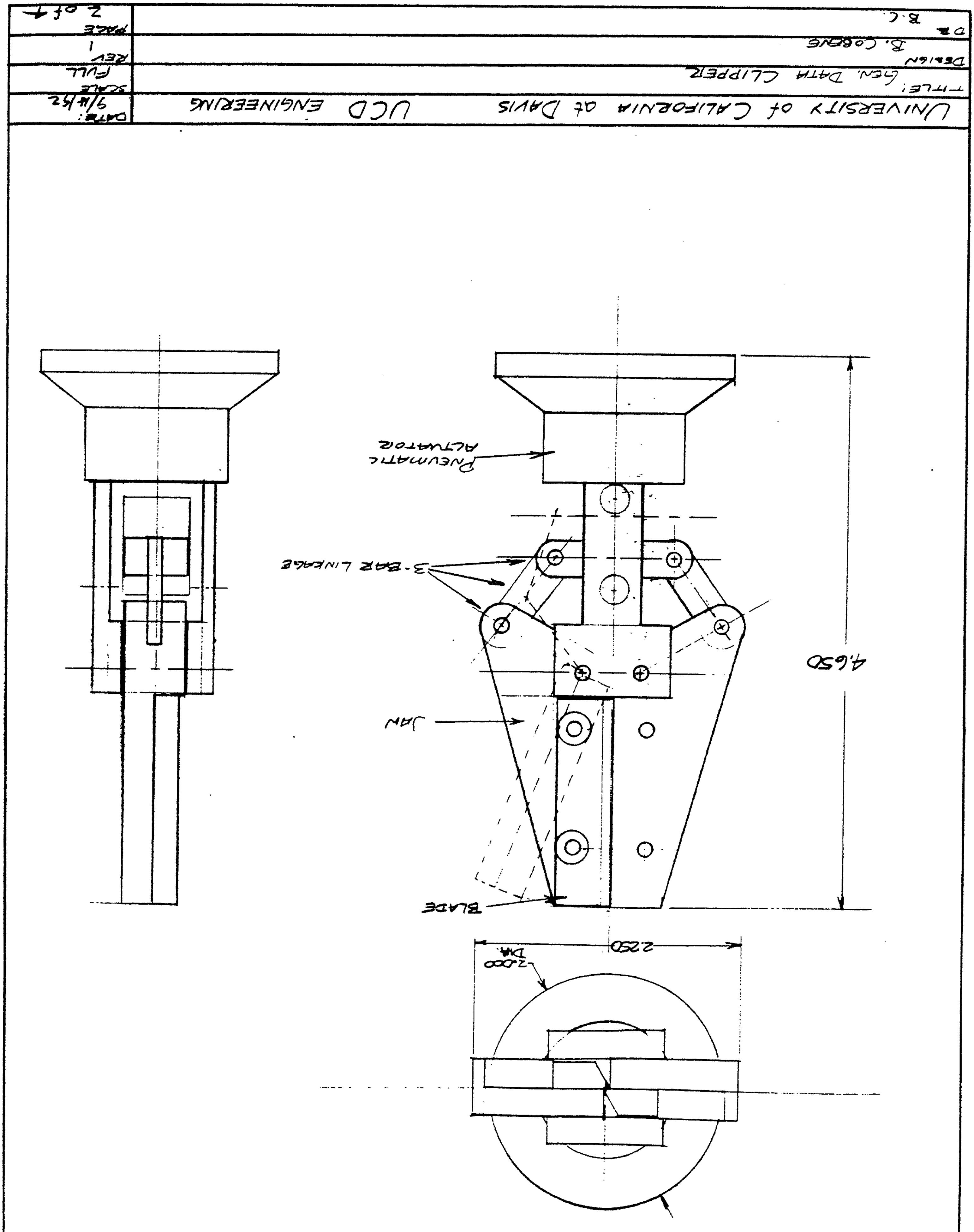
DR. B.C.

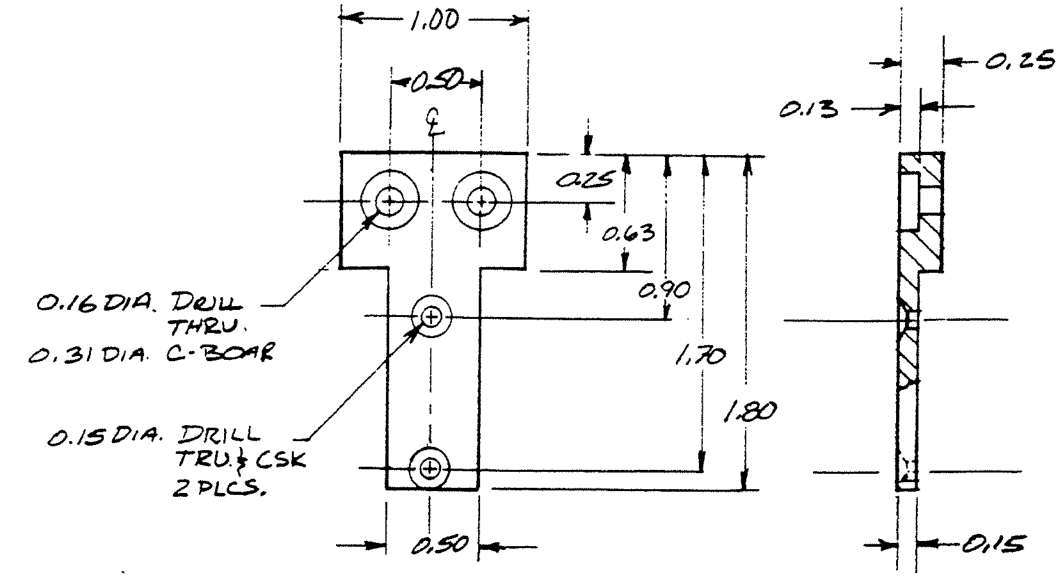
DATE: 6/18/92

SCALE: FULL

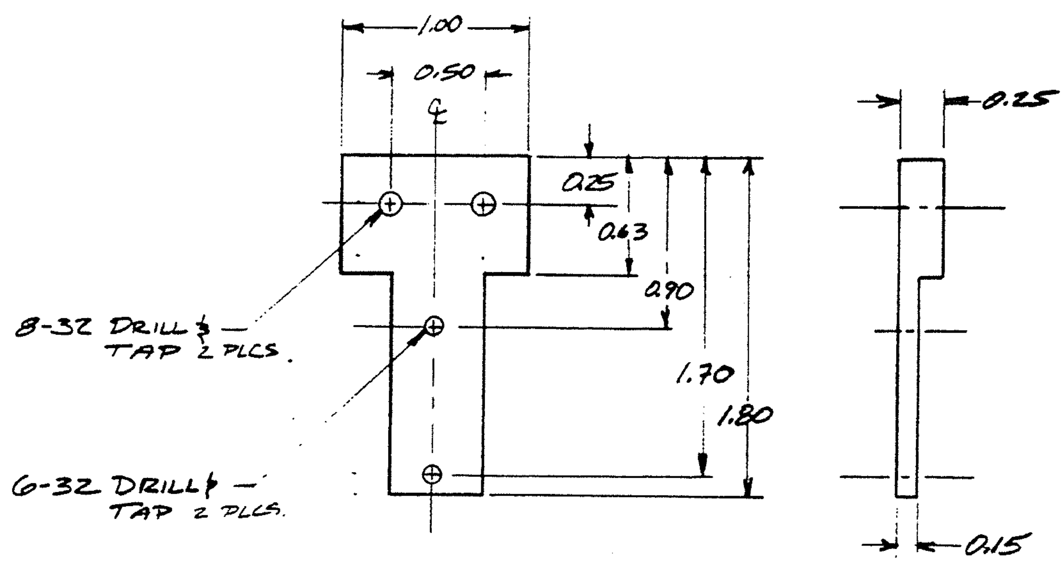
REV: 1




UNIVERSITY of CALIFORNIA at DAVIS
UCD ENGINEERING
DATE: 9/11/92
TITLE: GEN. DATA CLIPPER
SCALE FULL
DESIGN
REV 1
DR. B.C.
PAGE 2 of 7
PNEUMATIC ACTUATOR
3-BAR LINKAGE
JAW
BLADE
4.650
2.250
2.000 DIA.

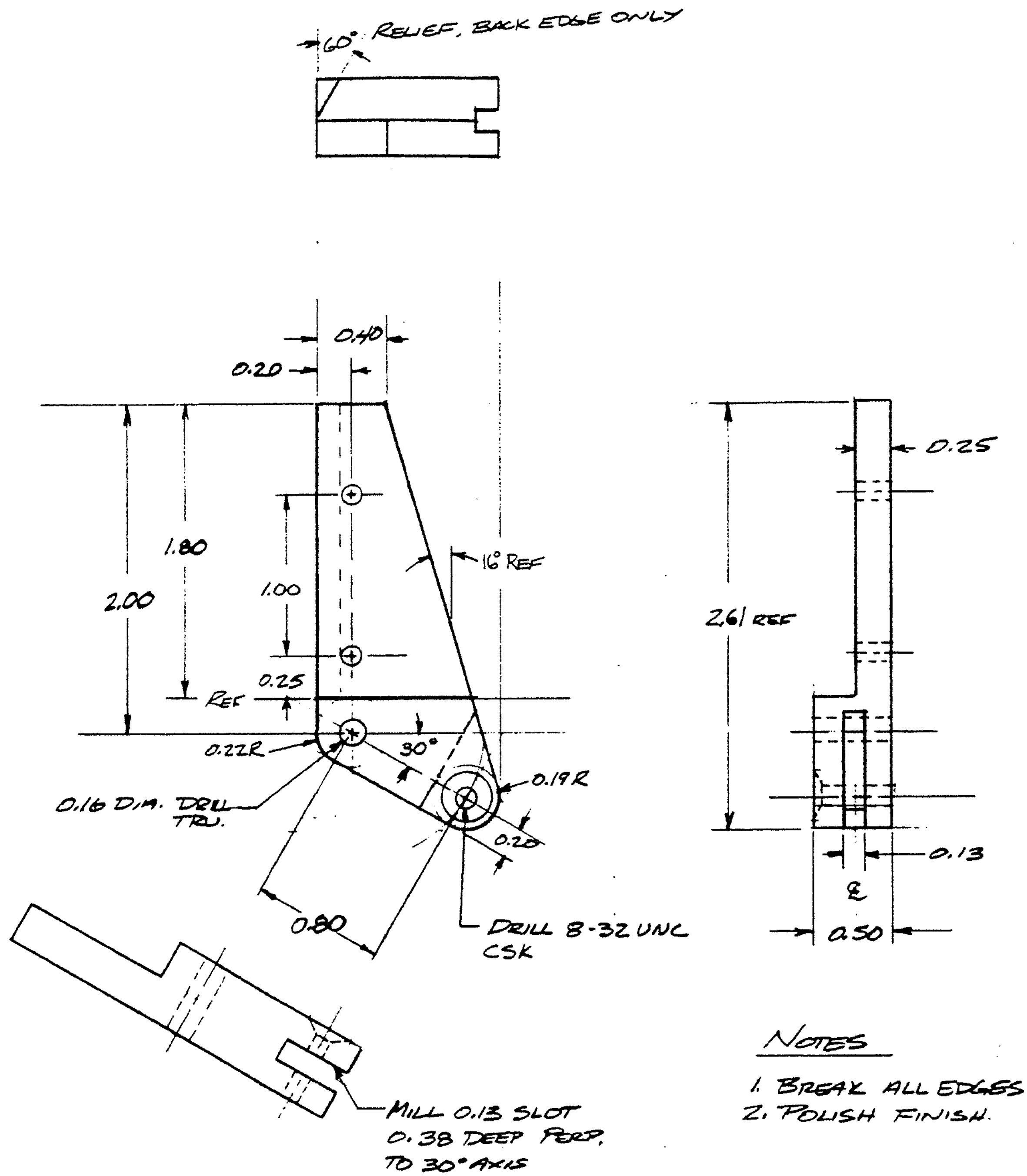


BRKT.
SCALE: 1:1 MTL: 6061-T6 Al.
(1) REQD



BRKT (MATING PIECE)
SCALE 1:1 MTL: 6061-T6 Al
(1) REQD

UNIVERSITY of CALIFORNIA at DAVIS UCD ENGINEERING	DATE: 9/20/92
TITLE: CLIPPER: BASE MT. BRKT.	SCALE FULL
DESIGN B. COBENE	REV 1
DR B.C.	PAGE 3 of 4



NOTES

1. BREAK ALL EDGES
2. POLISH FINISH.

JAW

SCALE: 1:1   MTL: 6061-T6 Al.

(2) REQD

UNIVERSITY of CALIFORNIA at DAVIS UCD ENGINEERING	DATE: 9/20/92
TITLE: CLIPPER : JAW	SCALE FULL
DESIGN B. COBENE	REV 1
DR B.C.	PAGE 4 of 4

Appendix C Source Code

```
plist usercd
.PROGRAM usercd
 ;
 ;
 ;       **************************************************
 ;       *       "USERCD" : zpoke Program                 *
 ;       * This program is used process receive /send     *
 ;       * interrupt . While Initialize or Zero , to exe_ *
 ;       * cute it. It is written in zpoke instructions , *
 ;       * so be care .                                   *
 ;       *                                                *
 ;       * March 20 , 1992 . by Wang  Xiaoxi              *
 ;       **************************************************
 ;
 ;
 ;Initialization program of poking the receive/send interrupt code
 ;into memory and setting handware addresses , vectors , buffers for
 ; rec/snd , and etc .
 ;You can use the following communication ports for sending or
 ;receiving between VAL__II and supervisor.
 ;       -------------------------------------------------
 ;       | port addr | interrupt vector addr|   port      |
 ;       |-----------|----------------------|-------------|
 ;       |  176520   |         320          | ACCESSOTY#1 |
 ;       |-----------|----------------------|-------------|
 ;       |  176600   |         340          |  DIGIMIG    |
 ;       |-----------|----------------------|-------------|
 ;       |  176610   |         350          |  SUPERVISOR |
 ;       |-----------|----------------------|-------------|
 ;       |  176620   |         360          |   ALTER     |
 ;       |-----------|----------------------|-------------|
 ;       |  176630   |         370          | ACCESSORY#2 |
 ;       -------------------------------------------------
 ;
 ;HARDWARE addresses:
         hw = ^176520;ACCESSORY #1
         recsta = hw;Receive status port
         rechw = hw+2;Port for receiving data
         sndsta = hw+4;Send status port
         sndhw = hw+6;Port for sending data
 ;
 ;HARDWARE interrupt vector addresses
         intvector = ^320;ALTER port
         recintvec = intvector;Receive vector addr
         sndintvec = intvector+4;Send vector addr
 ;SYMBOLS:
         del = ^377
         dle = ^220
         stx = ^202
         etx = ^203
 ;
 ;RAM addresses:
         start = ^67000;start address
 ;
 ;Variables and buffers of receiving / sending data
         var = start+450;begin address
         sndptr = var;Pointer to current sndbuf
         recptr = var+2;Pointer to current recbuf
         dleflag = var+4;Byte to control sending second DLE
         cntrlbt = var+6;Used as control byte during receiv
         jmpwd = var+8;Jmp to proper entrance
         char = var+10;Temporary unit for datum receive
         checkin = var+12;Checksum for receiving
         temp = var+14;Save R0
         reg0 = var+16;Save R0
         realend = var+18;Real send end
         sndbuf = var+20;Send buffer , 128 bytes
         sndend = var+150;End of send buffer
         recbuf = var+152;Receive buffer , 128 bytes
         recend = var+280;End of receive buffer
 ;
 ;PROGRAM addresses:
         entrnc = start;Table of jmp to proper task
 ;during receiving data
         recv = start+10;Entrance of receive interrupt
         sndv = start+300;Entrance of send interrupt
 ;
 ;Poke in the receive interrupt service routine
 ;
 ;Set entrances for respective task
         ZPOKE entrnc = recv+18;Entrance for handing DEL DLE
         ZPOKE entrnc+2 = recv+50;Entrance for receiving STX
         ZPOKE entrnc+4 = recv+100;Entrance for processing <data>
         ZPOKE entrnc+6 = recv+150;Entrance for checking second DLE
         ZPOKE entrnc+8 = recv+202;Entrance for checking checksum
 ;
 ;Beginning of receive interrupt : recv
         ZPOKE recv = ^113737
         ZPOKE recv+2 = rechw;Get datum from ACCESSORY #1
         ZPOKE recv+4 = char
         ZPOKE recv+6 = ^10037;Preserve R0
         ZPOKE recv+8 = reg0
         ZPOKE recv+10 = ^13700
         ZPOKE recv+12 = jmpwd
         ZPOKE recv+14 = ^170;Jump to the proper entrance
         ZPOKE recv+16 = entrnc
 ;
 ;Entrance for receiving DEL DLE : recv+18
         ZPOKE recv+18 = ^123727;DLE ?
         ZPOKE recv+20 = char
         ZPOKE recv+22 = dle
         ZPOKE recv+24 = ^1410;Yes , to recv+42
         ZPOKE recv+26 = ^123727
         ZPOKE recv+28 = char;Then if DEL ?
         ZPOKE recv+30 = del
         ZPOKE recv+32 = ^1403;Yes , to recv+40
 ;
         ZPOKE recv+34 = ^152737
         ZPOKE recv+36 = ^2;Set format error
         ZPOKE recv+38 = cntrlbt
 ;
         ZPOKE recv+40 = ^556;Exit to recv+262
 ;
         ZPOKE recv+42 = ^12737;Set next entrance for receive STX
         ZPOKE recv+44 = ^2
         ZPOKE recv+46 = jmpwd
         ZPOKE recv+48 = ^552;Exit to recv+262
 ;
 ;Entrance for receiving STX  : recv+50
         ZPOKE recv+50 = ^123727
         ZPOKE recv+52 = char;STX ?
         ZPOKE recv+54 = stx
         ZPOKE recv+56 = ^1364;No ,jmp to recv+34 set err
         ZPOKE recv+58 = ^12737
         ZPOKE recv+60 = recbuf;Set recptr
         ZPOKE recv+62 = recptr
         ZPOKE recv+64 = ^5037
         ZPOKE recv+66 = recbuf+4
```

```
        ZPOKE recv+68 = ^5037
        ZPOKE recv+70 = recbuf+6
        ZPOKE recv+72 = ^5037
        ZPOKE recv+74 = recbuf+8
        ZPOKE recv+76 = ^5037
        ZPOKE recv+78 = recbuf+10
        ZPOKE recv+80 = ^5037
        ZPOKE recv+82 = recbuf+12
        ZPOKE recv+84 = ^5037
        ZPOKE recv+86 = recbuf+14
        ZPOKE recv+88 = ^5037
        ZPOKE recv+90 = checkin;Zero checksum
;
        ZPOKE recv+92 = ^12737
        ZPOKE recv+94 = ^4
        ZPOKE recv+96 = jmpwd;Set next entrance
        ZPOKE recv+98 = ^521;Exit to recv+262
;
;Entrance for processing <data field> :recv+100
        ZPOKE recv+100 = ^123727
        ZPOKE recv+102 = char;DLE in the data field ?
        ZPOKE recv+104 = dle
        ZPOKE recv+106 = ^1421;Yes jmp recv+142 set next entrance
;
        ZPOKE recv+108 = ^113777;No ,then reserve this datum
        ZPOKE recv+110 = char
        ZPOKE recv+112 = ^510;(recptr)
        ZPOKE recv+114 = ^63737
        ZPOKE recv+116 = char;Calculate checksum
        ZPOKE recv+118 = checkin
        ZPOKE recv+120 = ^23727;Data over ?
        ZPOKE recv+122 = recptr
        ZPOKE recv+124 = recbuf+16
        ZPOKE recv+126 = ^2404;No to recv+136
        ZPOKE recv+128 = ^152737;Yes set data overrun error
        ZPOKE recv+130 = ^3
        ZPOKE recv+132 = cntrlbt
        ZPOKE recv+134 = ^477;Exit to recv+262
;
        ZPOKE recv+136 = ^5237
        ZPOKE recv+138 = recptr;Inc recptr
        ZPOKE recv+140 = ^474
;
        ZPOKE recv+142 = ^12737
        ZPOKE recv+144 = ^6
        ZPOKE recv+146 = jmpwd
        ZPOKE recv+148 = ^470;Exit to recv+262
;
;Entrance for checking if there is second DLE in the data
; field : recv +150
        ZPOKE recv+150 = ^123727
        ZPOKE recv+152 = char;DLE ?
        ZPOKE recv+154 = dle
        ZPOKE recv+156 = ^1467;Yes ,to recv+268
        ZPOKE recv+158 = ^123727
        ZPOKE recv+160 = char;ETX ?
        ZPOKE recv+162 = etx
        ZPOKE recv+164 = ^1004;No, to recv+174 to set err
        ZPOKE recv+166 = ^12737
        ZPOKE recv+168 = ^10
        ZPOKE recv+170 = jmpwd;Set next for receiving checksum
        ZPOKE recv+172 = ^454;Exit to recv+262
;
;
        ZPOKE recv+174 = ^152737
        ZPOKE recv+176 = ^5;Set PROTOCOL err
        ZPOKE recv+178 = cntrlbt
        ZPOKE recv+180 = ^123727
        ZPOKE recv+182 = char
        ZPOKE recv+184 = stx;STX ?
        ZPOKE recv+186 = ^1320;No, to recv+92 to set err
        ZPOKE recv+188 = ^12737
        ZPOKE recv+190 = recbuf;Yes,maybe another package coming
        ZPOKE recv+192 = recptr;Set recptr
        ZPOKE recv+194 = ^152737
        ZPOKE recv+196 = ^4
        ZPOKE recv+198 = cntrlbt;Too many messages err
        ZPOKE recv+200 = ^711;Goto recv+92 receive data field
;
;Entrance for checking checksum :rec+202
        ZPOKE recv+202 = ^123737
        ZPOKE recv+204 = char;Checksum error ?
        ZPOKE recv+206 = checkin
        ZPOKE recv+208 = ^1403;No  , to recv+216
        ZPOKE recv+210 = ^152737;Yes , set checksum err
        ZPOKE recv+212 = ^1
        ZPOKE recv+214 = cntrlbt
;
        ZPOKE recv+216 = ^12737;Reset entrance for receive
;DEL DLE
        ZPOKE recv+218 = ^0
        ZPOKE recv+220 = jmpwd
        ZPOKE recv+222 = ^133727
        ZPOKE recv+224 = recbuf;control byte = 0 ?
        ZPOKE recv+226 = ^377
        ZPOKE recv+228 = ^1004;No , to recv+238
        ZPOKE recv+230 = ^133727
        ZPOKE recv+232 = cntrlbt;Any error ?
        ZPOKE recv+234 = ^7
        ZPOKE recv+236 = ^1414
;
        ZPOKE recv+238 = ^413
        ZPOKE recv+240 = recbuf+4; then zero alter data
        ZPOKE recv+242 = ^5037
        ZPOKE recv+244 = recbuf+6
        ZPOKE recv+246 = ^5037
        ZPOKE recv+248 = recbuf+8
        ZPOKE recv+250 = ^5037
        ZPOKE recv+252 = recbuf+10
        ZPOKE recv+254 = ^5037
        ZPOKE recv+256 = recbuf+12
        ZPOKE recv+258 = ^5037
        ZPOKE recv+260 = recbuf+14
;
        ZPOKE recv+262 = ^13700;Recover R0
        ZPOKE recv+264 = reg0
        ZPOKE recv+266 = ^2;RTI
;
        ZPOKE recv+268 = ^12737
        ZPOKE recv+270 = ^4
        ZPOKE recv+272 = jmpwd
        ZPOKE recv+274 = ^654
;Poke in the send interrupt service routine
;
;Begining of send interrupt :sdnv
        ZPOKE sndv = ^23727;DLE in the field ?
        ZPOKE sndv+2 = dleflag
        ZPOKE sndv+4 = ^1
```

```
        ZPOKE sndv+6 = ^1006;No to sndv+20
        ZPOKE sndv+8 = ^12737;Yes ,send second DLE
        ZPOKE sndv+10 = dle
        ZPOKE sndv+12 = sndhw
        ZPOKE sndv+14 = ^5037;Clear dleflag
        ZPOKE sndv+16 = dleflag
        ZPOKE sndv+18 = ^2;RTI
;
        ZPOKE sndv+20 = ^117737;Send this datum
        ZPOKE sndv+22 = ^176;(sndptr)
        ZPOKE sndv+24 = sndhw
        ZPOKE sndv+26 = ^10037;Preserve R0
        ZPOKE sndv+28 = temp
        ZPOKE sndv+30 = ^23727;< Position of data field ?
        ZPOKE sndv+32 = sndptr
        ZPOKE sndv+34 = sndbuf+4
        ZPOKE sndv+36 = ^2420;No ,to sndv+70
        ZPOKE sndv+38 = ^13700;> Position of data field ?
        ZPOKE sndv+40 = realend
        ZPOKE sndv+42 = ^162700
        ZPOKE sndv+44 = ^3
        ZPOKE sndv+46 = ^23700
        ZPOKE sndv+48 = sndptr
        ZPOKE sndv+50 = ^3011;to sndv+70
        ZPOKE sndv+52 = ^117700
        ZPOKE sndv+54 = ^136;(sndptr):current byte =>R0
        ZPOKE sndv+56 = ^60077;Calculate checksum
        ZPOKE sndv+58 = ^154;(realend)
        ZPOKE sndv+60 = ^120027;DLE ?
        ZPOKE sndv+62 = dle
        ZPOKE sndv+64 = ^1002;No , to sndv+70
        ZPOKE sndv+66 = ^5237
        ZPOKE sndv+68 = dleflag;Set dleflag
;
        ZPOKE sndv+70 = ^13700;Recover R0
        ZPOKE sndv+72 = temp
        ZPOKE sndv+74 = ^5237;Inc sndptr
        ZPOKE sndv+76 = sndptr
        ZPOKE sndv+78 = ^123727;Having send DEL ?
        ZPOKE sndv+80 = sndptr
        ZPOKE sndv+82 = sndbuf+2
        ZPOKE sndv+84 = ^1006;No , to sndv+98
        ZPOKE sndv+86 = ^153737;Yes ,set control byte
        ZPOKE sndv+88 = cntrlbt
        ZPOKE sndv+90 = sndbuf+5
        ZPOKE sndv+92 = ^142737;Clear control byte
        ZPOKE sndv+94 = ^7
        ZPOKE sndv+96 = cntrlbt
;
        ZPOKE sndv+98 = ^23737;End of send data ?
        ZPOKE sndv+100 = sndptr
        ZPOKE sndv+102 = realend
        ZPOKE sndv+104 = ^3413;No ,to sndv+128,RTI
        ZPOKE sndv+106 = ^12737
        ZPOKE sndv+108 = sndbuf+1;Reset sndptr
        ZPOKE sndv+110 = sndptr
        ZPOKE sndv+112 = ^12737;Reset sndend
        ZPOKE sndv+114 = sndbuf+22
        ZPOKE sndv+116 = realend
        ZPOKE sndv+118 = ^105037;Clear error_byte
        ZPOKE sndv+120 = sndbuf+5
        ZPOKE sndv+122 = ^42737
        ZPOKE sndv+124 = ^100;Close send interrupt
        ZPOKE sndv+126 = sndsta
;
        ZPOKE sndv+128 = ^2;RTI
;
;
;HARDWARE interrupt vector addresses :
        intvector = ^320
        recintvec = intvector;Receive vector address
        sndintvec = intvector+4;Send vector address
;
;Set up the vectors:
        ZPOKE recintvec = recv;Start addresses of receive
        ZPOKE recintvec+2 = ^30340;KERNAL mode
        ZPOKE sndintvec = sndv;Start addresses of send routine
        ZPOKE sndintvec+2 = ^30340;KERNAL mode
.END
.
```

```
;
;The Process Control Program 'pcg' handles the communications
;       and ALTOUT alter data to the Robot Control Program 'main'
        DO
;           ALTOUT 0, dx, dy, dz, rx, ry, rz
;
;
;           ZPOKE sndbuf+4 = rexcptn
            tmpv = ((rexcptn BAND ^377)*256) BOR HAND
            ZPOKE sndbuf+6 = tmpv
;
;       Send the current (X Y Z O A T ) or 6 joint value to HOST
            ZPOKE sndbuf+8 = x[0]*32
            ZPOKE sndbuf+10 = x[1]*32
            ZPOKE sndbuf+12 = x[2]*32
            ZPOKE sndbuf+14 = x[3]*32
            ZPOKE sndbuf+16 = x[4]*32
            ZPOKE sndbuf+18 = x[5]*32
            ZPOKE sndbuf+22 = 0;Zero checksum
;
            IF (ZPEEK(recptr) == recbuf+16) AND (mready == 0) THEN
                ZPOKE recptr = recbuf
                excptn = ZPEEK(recbuf) BAND ^377
                oxmess = ZPEEK(recbuf+2)
                handst = oxmess BAND ^377
                oxmess = (oxmess/256) BAND ^377
                dx = ZPEEK(recbuf+4)
                dy = ZPEEK(recbuf+6)
                dz = ZPEEK(recbuf+8)
                rx = ZPEEK(recbuf+10)
                ry = ZPEEK(recbuf+12)
                rz = ZPEEK(recbuf+14)
                mready = -1
                vready = 1
            END
            ZPOKE sndbuf+4 = vready
            ZPOKE sndsta = ^100;Turn on send interrupt
            WAIT
            IF STATE(1) <> 7 THEN
                extstate = TRUE
            END
        UNTIL extstate
```

```
;
;
; Communication Data
;
        dx = 0
        dy = 0
        dz = 0
        rx = 0
        ry = 0
        rz = 0
        excptn = 0
        oxmess = 0
        handst = 0
;
; Communication Data (Real)
; used in MAIN control loop
;
        rdx = 0
        rdy = 0
        rdz = 0
        rrx = 0
        rry = 0
        rrz = 0
        rexcptn = 0
        roxmess = 0
        rhandst = 0
;
; locks and flags
;
        mready = 0
        vready = 0;
        trmode = FALSE
;
        i = 0;
        FOR i = 0 TO 5
            trp[i] = 0
        END
;
; Joint limit values
;
        llimit[0] = -158
        llimit[1] = -226
        llimit[2] = -51
        llimit[3] = -108
        llimit[4] = -99
        llimit[5] = -265
;
        ulimit[0] = 158
        ulimit[1] = 42
        ulimit[2] = 231
        ulimit[3] = 168
        ulimit[4] = 99
        ulimit[5] = 265
;
;The normal format of the communication between VAL_II and the
;supervisor is as following:
;
;                  --------------------------------
;sndptr-------->0|      DEL      |                |
;                |---------------|----------------|
;               2|      STX      |      DLE       |
;                |---------------|----------------|
;               4|  error byte   |  cntrl byte    | 0<--recbuf
;                |---------------|----------------|
;               6| OX     byte   |  hand  byte    | 2
;                |--------------------------------|
;               8|j_1 (or X )   or X offset       | 4
;                |--------------------------------|
;              10|j_2 (or Y )   or Y offset       | 6
;                |--------------------------------|
;              12|j_3 (or Z )   or Z offset       | 8
;                |--------------------------------|
;              14|j_4 (or O )   or X rotation     |10
;                |--------------------------------|
;              16|j_5 (or A )   or Y rotation     |12
;                |--------------------------------|
;              18|j_6 (or T )   or Z rotation     |14<---end
;                |--------------------------------|
;              20|     ETX       |      DLE       |
;                |---------------|----------------|
;        end-->22|     DLE       | checksum       |
;                 --------------------------------
;Control byte:
;       From VAL_II: bit0=1/0: Cumulative/Non_cumulative
;                    bit1=1/0: World / Tool mode
;                    bit2=1/0: Include location data / Not
;                    bit3=1/0: Transformation/joint angle
;                    bit4=1/0: Enable /Disable path_modify
;             Initial :Cumulative(1)+World(2)+Enable(16)=19
;       To  VAL_II : = 0: No change
;                    = 1: Switch to TOOL mode
;                    = 2: Switch to WORLD mode
;                    =-1: Exit
;Error byte:
;       From VAL_II: 0: Noerror
;               bit7 = 1:
;                    bit0=1/0: 1st joint error / no
;                    bit1=1/0: 2nd joint error / no
;                    bit2=1/0: 3rd joint error / no
;                    bit3=1/0: 4th joint error / no
;                    bit4=1/0: 5th joint error / no
;                    bit5=1/0: 6th joint error / no
;               bit7 = 0:
;                    bit0--2:error of the communication as to VAL_II
;               bit6 = 1:
;                    both .
;       To VAL_II:   =0 : no error
;                     1 : checksum error
;                     2 : framing or format error
;                     3 : data overrun
;                     4 : tool many messages
;                     5 : protocol error
;                     6 : timeout error
;                     7 : location out of range
;hand byte:
;               from VAL_II:current status , to VAL_II: reset
;                       0000a000  a=0: close ; =1: open
;OX byte:
;               current status(from VAL_II) or reset (to VAL_II)
;               a8 a7 a6 a5 a4 a3 a2 a1 =0:off;  1:on
;
;
;Initilize communications
        ZPOKE sndptr = sndbuf+1;Set sending pointer
        ZPOKE realend = sndbuf+22
;Set up the vectors:
        ZPOKE recintvec = recv;Start addr of receive routine
        ZPOKE recintvec+2 = ^30340;KERNAL mode
```

```
        ZPOKE sndintvec = sndv;Start addr of send routine
        ZPOKE sndintvec+2 = ^30340;KERNAL mode
        ZPOKE jmpwd = 0;Set entrance of receive
        ZPOKE recptr = recbuf
        ZPOKE recsta = ^100; Turn on receive interrupt
;
;Initialize send protocol data and other control information
;
        ZPOKE sndbuf = ^177400;DEL
        ZPOKE sndbuf+2 = ^101220;DLE STX
        ZPOKE sndbuf+20 = ^101620;ETX , DLE
        ZPOKE dleflag = 0;Control whether send second
;
;DLE during sending setpoint data
;start communication
        getpoint = TRUE;Control DECOMPOSE be done only
;once every 28 ms
        rexcptn = 0;Switch mode
;0: no change   1: TOOL
;2: WORLD       3: open hand
;4: close hand -1: exit
        altrmode = 19;cumulative(1)+WORLD(2)
; +enable(16)
        extstate = FALSE;WHEN ext_state = TRUE
;When extstate = TRUE ,
;the Robot Control Program 'main'
;and Process Control Program 'pcg'
;end .
;Read the current location into x[]
        HERE #jnt
        DECOMPOSE x[] = #jnt
;Read the HAND status into open_close
        IF HAND == 0 THEN
            openclose = 0
        ELSE
            openclose = 8
        END
;Read the OX 1--8 into ox_state
        oxstate = BITS(1, 8)
        mode = ^200
        oxmess = oxstate
;
;Start the Process Control Program "pcg" and internal ALTER
        PCEXECUTE pcg, 0, 0
;       ALTER (-1, altrmode)
;       MOVES HERE
;
;Processing change corresponding to control_byte and hand_byte
;from coordinator , and obtain current setpoint value , hand_status
; and OX_signals .
        DO
            IF mready THEN
                rexcptn = excptn
                roxmess = oxmess
                rhandst = handst
                rdx = dx
                rdy = dy
                rdz = dz
                rrx = rx
                rry = ry
                rrz = rz
                mready = 0
            END
            IF (rexcptn <> 0) AND (rexcptn <> 2) THEN
                trmode = FALSE
            END
            CASE rexcptn OF
              VALUE 0: ; Do nothing
                vready = 0;
              VALUE 1: ; Command Mode
                NOALTER
              VALUE 2: ; Tjoint mode (Teleopertion)
                IF NOT trmode THEN
                    HERE #jnt
                    DECOMPOSE trp[] = #jnt
                    TYPE "getting original point"
                    trmode = TRUE
                END
                trp[0] = trp[0]+rdx/32
                trp[1] = trp[1]+rdy/32
                trp[2] = trp[2]+rdz/32
                trp[3] = trp[3]+rrx/32
                trp[4] = trp[4]+rry/32
                trp[5] = trp[5]+rrz/32
                FOR i = 0 TO 5
                    IF trp[i] < llimit[i] THEN
                        trp[i] = llimit[i]
                    END
                    IF trp[i] > ulimit[i] THEN
                        trp[i] = ulimit[i]
                    END
                END
                MOVE #PPOINT(trp[0], trp[1], trp[2], trp[3], trp[4], trp[5])
;               vready = 0;
              VALUE 10:
                DRIVE rdx, rdy/64, rdz/64
              VALUE 11:
                MOVE #PPOINT(rdx/32, rdy/32, rdz/32, rrx/32, rry/32, rrz/32)
              VALUE 12:
                MOVES #PPOINT(rdx/32, rdy/32, rdz/32, rrx/32, rry/32, rrz/32)
              VALUE 13:
                MOVE TRANS(rdx/32, rdy/32, rdz/32, rrx/32, rry/32, rrz/32)
              VALUE 14:
                MOVES TRANS(rdx/32, rdy/32, rdz/32, rrx/32, rry/32, rrz/32)
              VALUE 15:
                TYPE "Exiting Main"
                extstate = TRUE
            END
            CASE rhandst OF
              VALUE 0:   ; Do nothing
              VALUE 1:   ; Close Gripper
                CLOSE
              VALUE 2:   ; Open Gripper
                OPEN
            END
            IF rexcptn <> 0 THEN
                rexcptn = 0
                rhandst = 0
                rdx = 0
                rdy = 0
                rdz = 0
                rrx = 0
                rry = 0
                rrz = 0
            END
            HERE #jnt
            DECOMPOSE x[] = #jnt; Ger current joint values
        UNTIL extstate  ;Loop until extstate
```

```
PCEND
```

```c
/*
** file: ddi.h
** date: 6/29/92
**   by: Gregory D. Benson
** desc: TSR device driver interface
**
*/

/* DDI signiture */
#define DDI     0x00

/* User functions */

#define DDI_INIT                0x00
#define DDI_QUIT                0x01

#define DDI_QUIT_ALL            0x10
#define DDI_CHECKDRIVERS        0x11

/* Error codes */

#define DDI_ERR_UNDEF   1       /* device driver number undefined */
#define DDI_ERR_REINIT  0x10
#define DDI_ERR_REQUIT  0x11

/* Interface INT */

#define DDI_INT_USER 0x61       /* INT 0x61 (Software INT) */


/* Data Structures */

struct ddilist_elm_tag {
        char *name;
        char *desc;
        int (*interface)(int, unsigned, unsigned, unsigned);
        int (*init)(void);
        int (*quit)(void);
        };

typedef struct ddilist_elm_tag DDILIST_ELM;



/* Typedefs */

typedef unsigned char byte;


/* Function prototypes */

int DDIcheck_drivers(void);
int DDIsend_message(int driver, int function, void far *packet);
```

```
/*
** file: ddi.c
** date: 6/29/92
**   by: Gregory D. Benson
** desc: TSR device driver interface
**
*/

#include"stdio.h"
#include"dos.h"
#include"ddi.h"

/*
** DDIcheck_driver - determine if the drivers have been installed
*/
int DDIcheck_drivers()
{
	unsigned far *intptrseg;
	unsigned far *intptroff;
	void far *isrpointer;

	intptroff = (unsigned far *) MK_FP(0x0000,0x0184);
	intptrseg = (unsigned far *) MK_FP(0x0000,0x0186);
	/*
	printf("DDI: intptrseg = %X, intptroff = %X\n", intptrseg, intptroff);
	*/
	isrpointer = MK_FP(*intptrseg, *intptroff);

	// printf("isrpointer = %X\n", isrpointer);

	if (isrpointer != NULL)
		return(DDIsend_message(DDI, DDI_CHECKDRIVERS, NULL));
	else
		return(0);
}

/*
** DDIsend_message - send a message to a driver
*/
int DDIsend_message(driver, function, packet)
int driver;
int function;
void far *packet;
{
	int rv;
	union REGS regs;

	regs.h.al = (byte) driver;	/* ADC driver */
	regs.h.ah = (byte) function;	/* ADC read function */
	regs.x.bx = FP_SEG(packet);
	regs.x.cx = FP_OFF(packet);

	rv = int86(DDI_INT_USER, &regs, &regs);
	return(rv);
}
```

```
/*
** file: ddiinit.h
** date: 6/29/92
**   by: Gregory D. Benson
** desc: TSR device driver routines - data structures and function prototypes
**
** note:
**
** Do not compile with Stack Overflow turned on.
*/

/* Definitions */

#ifdef __cplusplus
    #define __CPPARGS ...
#else
    #define __CPPARGS
#endif


/* Function Prototypes */

int DDIint_user(void);
int DDIinterface(int, unsigned, unsigned, unsigned);
int DDIinit(void);
int DDIquit(void);
int DDIinit_int(void);
int DDIinit_all(void);
int DDIquit_all(void);
```

```c
/*
** file: ddiinit.c
** date: 6/29/92
**   by: Gregory D. Benson
** desc: TSR device driver routines (initialization)
**
** note:
**
** Do not compile with Stack Overflow turned on.
*/

char *progname = "ddiinit";

#include <stdio.h>
#include <stdlib.h>
#include <dos.h>
#include <conio.h>
#include <alloc.h>
#include <math.h>

#include "ddi.h"        /* device driver interface definitions */
#include "ddiinit.h"    /* definitions specific to ddiinit.c */
#include "drivers.h"    /* the device driver list */


/* Comiler Data */

/* reduce heaplength and stacklength to make a smaller program in memory */

//extern unsigned _heaplen = 1024;
extern unsigned _stklen  = 2048;


/* External Functions */

void interrupt DDIintentry(void);


/* Global Data */

void interrupt ( *DDIint_user_old)(__CPPARGS);
byte DDIinit_flag = 0;
byte DDIquit_flag = 0;

/*
** DDI support routines
*/

/*
** DDIint_user - application interface software INT (DDI_INT_USER)
*/
int DDIint_user(void)
{
        unsigned dev, func, rv;
        unsigned regAX, regBX, regCX, regDX;

        regAX = _AX;
        regBX = _BX;
        regCX = _CX;
        regDX = _DX;

        dev  = regAX & 0x00FF;
        func = (regAX >> 8) & 0x00FF;
```



```c
        /* make sure that the driver request is in range */
        if (dev >= driver_count) {
                rv = DDI_ERR_UNDEF;
        }
        else {
                /* execute the appropriate driver interface */
                rv = (driver_list[dev].interface)(func, regBX, regCX, regDX);

        }
        return(rv);
}

/*
** DDIinterface - DDI interface routine
*/
int DDIinterface(int operation, unsigned regBX,
                unsigned regCX, unsigned regDX)
{
        switch(operation) {

                /* current no DDI support routines */
                case DDI_INIT:
                        return(DDIinit());
                case DDI_QUIT:
                        return(DDIquit());
                case DDI_QUIT_ALL:
                        return(DDIquit_all());
                case DDI_CHECKDRIVERS:
                        return(TRUE);   /* return true */
                default:
                        return(0);
        }
}

/*
** DDIinit - intialize DDI
*/
int DDIinit(void)
{
        /* make sure we haven't already called DDIinit() */
        if (DDIinit_flag) {
                return(DDI_ERR_REINIT);
        } else {
                DDIinit_flag = 1;
        }

        /* currently no initialization here */

        return(0);
}

/*
** DDIquit - uninitialize DDI
*/
int DDIquit(void)
{
        /* make sure we haven't already called DDIquit() */
        if (DDIquit_flag) {
                return(DDI_ERR_REQUIT);
        } else {
                DDIquit_flag = 1;
        }

        /* reset old interrupt vector */
```

```
        setvect(DDI_INT_USER, DDIint_user_old);

        /* currently no uninitialization here */

        return(0);
}

/*
** DDIinit_int - initialize the software INT (DDI_INT_USER)
*/
int DDIinit_int(void)
{
        /* set up interrupt vector */

        DDIint_user_old = getvect(DDI_INT_USER);
        setvect(DDI_INT_USER, DDIintentry);

        return(TRUE);
}

/*
** DDIinit_all - initialize all drivers in the 'driver_list'
*/
int DDIinit_all(void)
{
        int i;

        for (i = 0; i < driver_count; i++) {
                if ((driver_list[i].init)()) {
                        printf("DDI: Cannot initailize %s\n",
                                driver_list[i].name);
                        return(FALSE);
                }
                else {
                        printf("DDI: %s [%s] initialized\n",
                                driver_list[i].name,
                                driver_list[i].desc);
                }
        }
        return(TRUE);
}

/*
** DDIquit_all - uninitialize all drivers in the 'driver_list'
*/
int DDIquit_all(void)
{
        int i, rv = TRUE;

        for (i = 0; i < driver_count; i++) {
                if ((driver_list[i].quit)()) {
                        printf("DDI: Cannot quit %s\n",
                                driver_list[i].name);
                        rv = FALSE;
                }
                else {
                        printf("DDI: %s [%s] uninitialized\n",
                                driver_list[i].name,
                                driver_list[i].desc);
                }
        }
        return(rv);
}
```



```
/*
** M A I N
*/
int main(void)
{
        unsigned progsize;

        /* Make sure that the DDI is not already installed */
        if (DDIcheck_drivers()) {
                printf("DDI: Drivers already installed\n");
                exit(0);
        }

        /* install the interface INT vector */
        DDIinit_int();

        /* initialize all the drivers */
        DDIinit_all();

        /*
        printf("_psp    = %.4X:%.4X\n", _psp, 0);
        printf("_SS     = %.4X:%.4X\n", _SS, 0);
        printf("_stklen = %X\n", _stklen);
        */

        progsize = (_SS - _psp) + (_stklen >> 4) + 1;
        if (progsize <= 0) {
                printf("\n%s: Error invalid program size (_SS - _psp)\n",
                        progname);
                exit(0);
        }

        /* Terminate and Stay Resident */
        printf("DDI: Drivers are now resident\n");

        keep(0, progsize);
        return(0);
}
```

```
;
; file: intentry.asm
; date: 6/29/92
;   by: Gregory D. Benson
; desc: DDI INT interface entry point
;


        .MODEL large

        EXTRN  _DDIint_user:PROC

        .DATA

_DDIaxhold      dw      0
_DDIbxhold      dw      0
_DDIcxhold      dw      0
_DDIdxhold      dw      0

_sstemp         dw      0
_sptemp         dw      0
_DDIsshold      dw      0
_DDIsphold      dw      0
_newstack       db      1000h dup (?)
_newstacktop    equ     $-2


        .CODE

        PUBLIC _DDIintentry
        PUBLIC _DDInewstack

;
; _DDInewstack - switch to a new stack
;
; Input:  BX = new stack segment (ss)
;         CX = new stack pointer (sp)
;
; Output: BX = old stack segment (ss)
;         CX = old stack pointer (sp)
;
; Global Data: _sptemp, _sstemp
;

_DDInewstack    PROC
                cli                     ; disable interrupts to change stack
                pop     si              ; get return address for stack
                pop     di
                mov     DGROUP:_sstemp,ss       ; save original stack segment
                mov     DGROUP:_sptemp,sp       ; save original stack offset
                mov     ax,bx                   ; set new stack segment
                mov     ss,ax
                ;lea    ax,cx                   ; set new stack offset
                mov     ax,cx
                mov     sp,ax

                mov     bx,DGROUP:_sstemp       ; return old stack segment
                mov     cx,DGROUP:_sptemp       ; return old stack offset

                push    di              ; put return address on stack
                push    si
                sti                     ; enable interrupts
                ret
_DDInewstack    ENDP
```



```
_DDIintentry    PROC
        jmp     procstart
        db      'ddiint'
procstart:
        push    si
        push    di
        push    es
        push    ds
        push    bp
        mov     bp,DGROUP
        mov     ds,bp
        mov     bp,sp

        ; save current ?x regs
        mov     DGROUP:_DDIaxhold, ax
        mov     DGROUP:_DDIbxhold, bx
        mov     DGROUP:_DDIcxhold, cx
        mov     DGROUP:_DDIdxhold, dx

        ; switch to a new stack
        mov     bx,DGROUP
        lea     cx,_newstacktop
        call    _DDInewstack

        ; save old stack segment and pointer
        mov     DGROUP:_DDIsshold,bx
        mov     DGROUP:_DDIsphold,cx

        ; restore ?x given at entry
        mov     ax, DGROUP:_DDIaxhold
        mov     bx, DGROUP:_DDIbxhold
        mov     cx, DGROUP:_DDIcxhold
        mov     dx, DGROUP:_DDIdxhold

        ; call the DDI interface routine
        call    _DDIint_user

        ; save current ?x regs
        mov     DGROUP:_DDIaxhold, ax
        mov     DGROUP:_DDIbxhold, bx
        mov     DGROUP:_DDIcxhold, cx
        mov     DGROUP:_DDIdxhold, dx

        ; switch to the old stack
        mov     bx, DGROUP:_DDIsshold
        mov     cx, DGROUP:_DDIsphold
        call    _DDInewstack

        ; restore the ?x regs returned from _DDIint_user
        mov     ax, DGROUP:_DDIaxhold
        mov     bx, DGROUP:_DDIbxhold
        mov     cx, DGROUP:_DDIcxhold
        mov     dx, DGROUP:_DDIdxhold

        pop     bp
        pop     ds
        pop     es
        pop     di
        pop     si
        iret
_DDIintentry    ENDP
```

```
        END
```

```
/*
** file: drivers.h
** date: 6/29/92
**   by: Gregory D. Benson
** desc: list of available device drivers
*/


#include "adcdrv.h"
#include "kindrv.h"
#include "comdrv.h"

DDILIST_ELM driver_list[] = {
        { "DDI", "Device Driver Interface", DDIinterface, DDIinit, DDIquit },
        { "ADC", "ADC Driver", ADinterface, ADinit, ADquit },
        { "KIN", "KIN Driver", KINinterface, KINinit, KINquit },
        { "COM", "COM Driver", COMinterface, COMinit, COMquit }
        };

int driver_count = 4;
```

```
/*
** file: adcdrv.h
** date: 6/28/92
**   by: Gregory D. Benson
** desc: ADC driver - header file
**
*/

/* User functions */

#define ADC_START        0x10
#define ADC_STOP         0x11
#define ADC_GETCOUNTS    0x12
#define ADC_GETDEGREES   0x13
#define ADC_GETRADIANS   0x14
#define ADC_SETOFFSETS   0x15
#define ADC_SETCONFIG    0x16
#define ADC_GETDIG       0x20
#define ADC_GETTOGGLE    0x21
#define ADC_SETTOGGLE    0x22
#define ADC_GETCOUNTER   0x30
#define ADC_SETCOUNTER   0x31

/* Error codes */

#define ADC_ERR_UNKNOWNFUNC     0x10
#define ADC_ERR_REINIT          0x11
#define ADC_ERR_REQUIT          0x12

/* Definitions */

#ifndef TRUE
#define TRUE    -1
#define FALSE   0
#endif

#define AD_BASE         0x330

#define AD_LSBCH        AD_BASE
#define AD_START        AD_BASE
#define AD_MSB          AD_BASE+1
#define AD_MUX          AD_BASE+2
#define AD_DIG          AD_BASE+3
#define AD_STATUS       AD_BASE+8
#define AD_DMAINT       AD_BASE+9
#define AD_PACER        AD_BASE+10
#define AD_GAIN         AD_BASE+11
#define AD_CNT0         AD_BASE+12
#define AD_CTR1         AD_BASE+13
#define AD_CTR2         AD_BASE+14
#define AD_8254         AD_BASE+15

#define AD_FULL_RANGE   4096
#define AD_ORIGIN       AD_FULL_RANGE/2
#define AD_INT_ISR      0x0F            /* INT 0x0A (Hardware INT 0x02) */


#define CHANNELS        6
#define MAXCHANNELS     6

#define ADC_BUTTON1     0x02
#define ADC_BUTTON2     0x04
#define ADC_BUTTON3     0x08
```



```
/* Data Structures */

struct ad_counts_tag {
        unsigned int data[CHANNELS];
};
typedef struct ad_counts_tag ADCOUNTS;

struct ad_degrees_tag {
        double data[CHANNELS];
};
typedef struct ad_degrees_tag ADDEGREES;

struct ad_radians_tag {
        double data[CHANNELS];
};
typedef struct ad_radians_tag ADRADIANS;


/*
struct adpacket_tag {
        int type;
        union {
                ADCOUNTS counts;
                ADDEGREES degrees;
        } pos;
};
typedef struct adpacket_tag ADPACKET;
*/


/* Function Prototypes */

int  ADinterface(int, unsigned, unsigned, unsigned);
int  ADinit(void);
int  ADstart(void);
int  ADstop(void);
int  ADquit(void);
void interrupt ADint_isr(void);
int  ADsend_counts(unsigned, unsigned);
int  ADsend_degrees(unsigned, unsigned);
int  ADsend_radians(unsigned, unsigned);
int  ADset_offset(unsigned, unsigned);
int  ADset_config(unsigned, unsigned);
int  ADget_digital(unsigned, unsigned);
int  ADget_toggle(unsigned, unsigned);
int  ADset_toggle(unsigned, unsigned);
int  ADget_counter(unsigned, unsigned);
int  ADset_counter(unsigned, unsigned);
```

```
/*
** file: adcdrv.c
** date: 6/28/92
**   by: Gregory D. Benson
** desc: ADC driver
**
*/

#include<stdio.h>
#include<conio.h>
#include<dos.h>
#include<math.h>
#include"ddi.h"
#include"adcdrv.h"

/* Definintions */

#define PI      3.1415926536


/* Global Data */

void interrupt ( *ADint_isr_old)();
void interrupt ( *ADint_user_old)();
byte ADCold_i8259_mask;
byte ADCi8259bit;
byte ADCdmaint_mask;
byte ADCinit_flag       = 0;
byte ADCquit_flag       = 0;
byte ADCtoggle          = 0;
byte ADCtoggle_new      = 0;
byte ADCtoggle_old      = 0;
byte ADCtoggle_mask     = 0;
long ADCcounter         = 0;

ADCOUNTS adincounts, adincounts_busy;
ADDEGREES adindegrees, adindegrees_busy;
ADDEGREES adk, adkold;

ADDEGREES conv_cnt_deg  = {{ 0.079, 0.080, 0.080, 0.084, 0.082, 0.082 }};
ADDEGREES jconfig       = {{-1.0, -1.0, 1.0, -1.0, -1.0, 1.0}};
ADDEGREES OffSchilling  = {{0.0, 0.0, 0.0, 0.0, 0.0, 0.0}};


/* Low Pass Filter constants */

double a_T      = 0.005999;
double a_F      = 10.0;
double a_coeff  = 0;


/*
** ADinterface - INT interface dispatch
*/
int ADinterface(operation, regBX, regCX, regDX)
int operation;
unsigned regBX, regCX, regDX;
{
        switch (operation) {

                case DDI_INIT:
                        return(ADinit());
                case DDI_QUIT:
                        return(ADquit());
```



```
                case ADC_START:
                        return(ADstart());
                case ADC_STOP:
                        return(ADstop());
                case ADC_GETCOUNTS:
                        return(ADsend_counts(regBX, regCX));
                case ADC_GETDEGREES:
                        return(ADsend_degrees(regBX, regCX));
                case ADC_GETRADIANS:
                        return(ADsend_radians(regBX, regCX));
                case ADC_SETOFFSETS:
                        return(ADset_offset(regBX, regCX));
                case ADC_SETCONFIG:
                        return(ADset_config(regBX, regCX));
                case ADC_GETDIG:
                        return(ADget_digital(regBX, regCX));
                case ADC_GETTOGGLE:
                        return(ADget_toggle(regBX, regCX));
                case ADC_SETTOGGLE:
                        return(ADset_toggle(regBX, regCX));
                case ADC_GETCOUNTER:
                        return(ADget_counter(regBX, regCX));
                case ADC_SETCOUNTER:
                        return(ADset_counter(regBX, regCX));
                default:
                        return(ADC_ERR_UNKNOWNFUNC);
        }
}

/*
** ADinit - intialize the ADC board and interrupt vector
*/
int ADinit(void)
{
        int i;
        byte temp;
        unsigned int ch;

        /* Make sure we haven't already initialized */
        if (ADCinit_flag) {
                return(ADC_ERR_REINIT);
        } else {
                ADCinit_flag = 1;
        }

        /* initialize global data structures */

        for (i=0; i < CHANNELS; i++) {
                adincounts.data[i] = 0;
                adincounts_busy.data[i] = 0;
                adindegrees.data[i] = 1.0;
                adindegrees_busy.data[i] = 1.0;
                }

        /* initialize filter */

        a_coeff = exp(-2*PI*a_T*a_F);  /* a=e^(-2*PI*T*F) */

        /* initialize AD16JR */

        outportb(AD_GAIN, 0x05);        /* 0-5V mode */

        ch = (unsigned) CHANNELS - 1;
        ch <<= 4;
```

```
        ch &= 0xF0;
        outportb(AD_MUX, ch);           /* Use channels 0-CHANNELS */

        /* set up interrupt vector */

        ADint_isr_old = getvect(AD_INT_ISR);
        setvect(AD_INT_ISR, ADint_isr);

        /* set i8254 timer on AD16JR */
        /* set for 1KHz */

        outportb(AD_8254, 0x74);
        outportb(AD_CTR1, 0xF4);
        outportb(AD_CTR1, 0x01);
        outportb(AD_8254, 0xB4);
        outportb(AD_CTR2, 0x02);
        outportb(AD_CTR2, 0x00);

        /* set ADCi8259bit for interrupt mask */

        ADCi8259bit = 0x01 << (AD_INT_ISR - 0x08);
        ADCdmaint_mask = ((AD_INT_ISR - 0x08) << 4) | 0x83;

        /* make sure that interrupts are disabled until ADstart */
        ADstop();


        return(0);
}


/*
** ADstart - start the ADC interrupts
*/
int ADstart(void)
{
        byte temp;

        disable();

        /* Now enable the 8259 for ADC interrupts */

        outportb(0x21, inportb(0x21) & ~ADCi8259bit);

        outportb(AD_PACER, 0x01);
        outportb(AD_DMAINT, ADCdmaint_mask);
        outportb(AD_STATUS, 0x00);

        enable();

        return(0);
}

/*
** ADstop - stop the ADC interrupts
*/
int ADstop(void)
{
        byte temp;

        disable();

        outportb(0x21, inportb(0x21) | ADCi8259bit);
        outportb(AD_PACER, 0x00);
```



```
        outportb(AD_DMAINT, 0x00);

        enable();
        return(0);
}

/*
** ADquit - disable ADC interrupts and restore old handler
*/
int ADquit(void)
{
        /* make sure we haven't already quit */

        if (ADCquit_flag) {
                return(ADC_ERR_REQUIT);
        } else {
                ADCquit_flag = 1;
        }

        /* stop the ADC interrupts */

        ADstop();

        /* reset old interrupt vector */

        setvect(AD_INT_ISR, ADint_isr_old);


        return(0);
}

/*
** ADint_isr - ADC interrupt service routine
*/
void interrupt ADint_isr(void)
{
        int i;
        unsigned int msb, lsb, ch;

        lsb = (unsigned)inportb(AD_LSBCH);
        msb = (unsigned)inportb(AD_MSB);

        ch  = lsb & 0x0F;

        lsb &= 0xF0;
        lsb >>= 4;
        msb <<= 4;

        adincounts_busy.data[ch] = msb | lsb;
        adindegrees_busy.data[ch] = conv_cnt_deg.data[ch]
                                    * (double) adincounts_busy.data[ch];

        adk.data[ch] = (1 - a_coeff)*adindegrees_busy.data[ch]
                + a_coeff * adkold.data[ch];

        adkold.data[ch] = adk.data[ch];

        disable();
        if (ch == (CHANNELS - 1)) {

                for (i=0 ; i < CHANNELS; i++) {
                        adincounts.data[i] = adincounts_busy.data[i];
                        adindegrees.data[i] = adk.data[i];
                }
```

```
            /* process button toggle */

            ADCtoggle_new  = (inportb(AD_DIG));
            ADCtoggle_mask = (ADCtoggle_old ^ ADCtoggle_new)
                             & ADCtoggle_new;
            ADCtoggle_old  = ADCtoggle_new;
            ADCtoggle = ADCtoggle_mask ^ ADCtoggle;

        }
        ADCcounter++;
        enable();

        outportb(AD_STATUS,0x00);       /* Start conversion for next sample */
        outportb(0x20, 0x20);           /* Send EOI to 8259 */
}

/*
** ADsend_counts - copy current ADC count values to regBX:regCX
*/
int ADsend_counts(regBX, regCX)
unsigned regBX, regCX;
{
        int i;
        ADCOUNTS far *adoutput;

        adoutput = MK_FP(regBX, regCX);

        disable();
        for (i = 0; i < CHANNELS; i++) {
                adoutput->data[i] = adincounts.data[i];
        }
        enable();

        return(0);
}

/*
** ADsend_degrees - copy current ADC degree values to regBX:regCX
*/
int ADsend_degrees(unsigned regBX, unsigned regCX)
{
        int i;
        ADDEGREES far *adoutput;

        adoutput = MK_FP(regBX, regCX);

        disable();
        for (i = 0; i < CHANNELS; i++) {
                adoutput->data[i] = jconfig.data[i] * (adindegrees.data[i]
                                    + OffSchilling.data[i]);
        }
        enable();
        return(0);
}

/*
** ADsend_radians - copy current ADC radian values to regBX:regCX
*/
int ADsend_radians(unsigned regBX, unsigned regCX)
{
        int i;
        ADRADIANS far *adoutput;
```



```
        adoutput = MK_FP(regBX, regCX);

        disable();
        for (i = 0; i < CHANNELS; i++) {
                adoutput->data[i] = jconfig.data[i] *
                                (adindegrees.data[i] + OffSchilling.data[i])*
                                (M_PI / 180.0);
        }
        enable();
        return(0);
}

/*
** ADset_offset - set offset values from regBX:regCX
*/
int ADset_offset(unsigned regBX, unsigned regCX)
{
        int i;
        ADDEGREES far *adinput;

        adinput = MK_FP(regBX, regCX);

        disable();
        for (i = 0; i < CHANNELS; i++) {
                OffSchilling.data[i] = adinput->data[i];
        }
        enable();
        return(0);
}

/*
** ADset_config - set config values from regBX:regCX
*/
int ADset_config(unsigned regBX, unsigned regCX)
{
        int i;
        ADDEGREES far *adinput;

        adinput = MK_FP(regBX, regCX);

        disable();
        for (i = 0; i < CHANNELS; i++) {
                jconfig.data[i] = adinput->data[i];
        }
        enable();
        return(0);
}

/*
** ADget_digital - get digital input information from ADC board
*/
int ADget_digital(unsigned regBX, unsigned regCX)
{
        unsigned char *dig;

        dig = MK_FP(regBX, regCX);

        *dig = inportb(AD_DIG);

        return(0);
}

/*
** ADget_toggle - get button toggle information from ADC board
```

```
*/
int ADget_toggle(unsigned regBX, unsigned regCX)
{
        unsigned char *dig;

        dig = MK_FP(regBX, regCX);

        disable();
        *dig = ADCtoggle;
        enable();

        return(0);
}

/*
** ADset_toggle - set button toggle information
*/
int ADset_toggle(unsigned regBX, unsigned regCX)
{
        unsigned char *dig;

        dig = (unsigned char *) MK_FP(regBX, regCX);

        disable();
        ADCtoggle = *dig;
        enable();

        return(0);
}

/*
** ADget_counter - get ADC interrupt counter
*/
int ADget_counter(unsigned regBX, unsigned regCX)
{
        long *count;

        count = (long *) MK_FP(regBX, regCX);

        disable();
        //*count = ADCcounter;
        enable();

        return( (int) 321 );
}

/*
** ADset_counter - set ADC interrupt counter
*/
int ADset_counter(unsigned regBX, unsigned regCX)
{
        long *count;

        count = (long *) MK_FP(regBX, regCX);

        disable();
        ADCcounter = *count;
        enable();

        return(regBX);
}
```

```
/*
** file: comdrv.h
** date: 7/9/92
**   by: Gregory D. Benson
** desc: COM driver - header file
**
*/

/* User functions */

#define COM_INSTALL     0x10
#define COM_DEINSTALL   0x11
#define COM_SET_SPEED   0x12
#define COM_SET_PARITY  0x13
#define COM_LOWER_DTR   0x14
#define COM_RAISE_DTR   0x15
#define COM_TX          0x16
#define COM_TX_STRING   0x17
#define COM_RX          0x18
#define COM_TX_READY    0x19
#define COM_TX_EMPTY    0x1A
#define COM_RX_EMPTY    0x1B
#define COM_FLUSH_TX    0x1C
#define COM_FLUSH_RX    0x1D
#define COM_CARRIER     0x1E


/* Error codes */

#define COM_ERR_UNKNOWNFUNC     0x10
#define COM_ERR_REINIT          0x11
#define COM_ERR_REQUIT          0x12

/* Definitions */

#ifndef TRUE
#define TRUE    -1
#define FALSE   0
#endif

/* Data Structures */

typedef struct {
        int parity;
        int stop_bits;
} PARITY;


typedef struct {
        union {
                int portnum;
                int speed;
                unsigned char tx;
                char *txs;
                unsigned char rx;
                PARITY par;
        } arg;  /* arguements */
        int rv;
} COM_PACKET;


/* Function Prototypes */

int COMinterface(int, unsigned, unsigned, unsigned);
int COMinit(void);
int COMquit(void);
```

```c
/*
** file: comdrv.c
** date: 7/9/92
**   by: Gregory D. Benson
** desc: COM driver - serial communications driver
**
*/

#include<stdio.h>
#include<conio.h>
#include<dos.h>
#include<math.h>
#include"ddi.h"
#include"comdrv.h"
#include"ibmcom.h"

/* Definintions */

/* Global Data */

byte COMinit_flag = 0;
byte COMquit_flag = 0;


/*
** COMinterface - INT interface dispatch
*/
int COMinterface(operation, regBX, regCX, regDX)
int operation;
unsigned regBX, regCX, regDX;
{
        COM_PACKET *pkt;

        pkt = (COM_PACKET *) MK_FP(regBX, regCX);


        switch (operation) {

                case DDI_INIT:
                        return(COMinit());

                case DDI_QUIT:
                        return(COMquit());

                case COM_INSTALL:
                        pkt->rv = com_install(pkt->arg.portnum);
                        return(pkt->rv);

                case COM_DEINSTALL:
                        com_deinstall();
                        return(0);

                case COM_SET_SPEED:
                        com_set_speed(pkt->arg.speed);
                        return(0);

                case COM_SET_PARITY:
                        com_set_parity(pkt->arg.par.parity,
                                       pkt->arg.par.stop_bits);
                        return(0);

                case COM_LOWER_DTR:
                        com_lower_dtr();
                        return(0);

                case COM_RAISE_DTR:
                        com_raise_dtr();
                        return(0);

                case COM_TX:
                        com_tx(pkt->arg.tx);
                        return(0);

                case COM_TX_STRING:
                        com_tx_string(pkt->arg.txs);
                        return(0);

                case COM_RX:
                        pkt->rv = (int) com_rx(&(pkt->arg.rx));
                        return(0);

                case COM_TX_READY:
                        pkt->rv = com_tx_ready();
                        return(0);

                case COM_TX_EMPTY:
                        pkt->rv = com_tx_ready();
                        return(0);

                case COM_RX_EMPTY:
                        pkt->rv = com_rx_empty();
                        return(0);

                case COM_FLUSH_TX:
                        com_flush_tx();
                        return(0);

                case COM_FLUSH_RX:
                        com_flush_rx();
                        return(0);

                case COM_CARRIER:
                        pkt->rv = com_carrier();
                        return(0);

                default:
                        return(COM_ERR_UNKNOWNFUNC);
        }
}

/*
** COMinit - intialize the COM driver
*/
int COMinit()
{
        int i;

        /* Make sure we haven't already initialized */
        if (COMinit_flag) {
                return(COM_ERR_REINIT);
        } else {
                COMinit_flag = 1;
        }

        /* initialize global data structures */

        return(0);      /* no errors */
}
```

```
/*
** COMquit - disable the COM driver
*/
int COMquit()
{
        /* make sure we haven't already quit */

        if (COMquit_flag) {
                return(COM_ERR_REQUIT);
        } else {
                COMquit_flag = 1;
        }

        com_deinstall();

        return(0);      /* no errors */
}
```

```
/****************************************************************************
 *                              ibmcom.h                                    *
 ****************************************************************************
 * DESCRIPTION: ANSI C function prototypes and other definitions for the    *
 *              routines in ibmcom.c                                        *
 *                                                                          *
 * REVISIONS:   18 OCT 89 - RAC - Original code.                            *
 ****************************************************************************/


/* Definitions */

#define COM_NONE        0
#define COM_EVEN        1
#define COM_ODD         2
#define COM_ZERO        3
#define COM_ONE         4


/* Function prototypes */

int             com_carrier(void);
void            com_deinstall(void);
void            com_flush_rx(void);
void            com_flush_tx(void);
int             com_install(int portnum);
void interrupt  com_interrupt_driver();
void            com_lower_dtr(void);
void            com_raise_dtr(void);
int             com_rx(unsigned char *);
int             com_rx_empty(void);
void            com_set_parity(int parity, int stop_bits);
void            com_set_speed(unsigned speed);
void            com_tx(unsigned char);
int             com_tx_empty(void);
int             com_tx_ready(void);
void            com_tx_string(char *s);
```

```
/*************************************************************************
 *                              ibmcom.c                                 *
 *************************************************************************
 * DESCRIPTION: This file contains a set of routines for doing low-level *
 *              serial communications on the IBM PC.  It was translated  *
 *              directly from Wayne Conrad's IBMCOM.PAS version 3.1, with *
 *              the goal of near-perfect functional correspondence between *
 *              the Pascal and C versions.                               *
 *                                                                       *
 * REVISIONS:   18 OCT 89 - RAC - Original translation from IBMCOM.PAS, with *
 *                                liberal plagiarism of comments from the *
 *                                Pascal.                                *
 *************************************************************************/

#include        <stdio.h>
#include        <dos.h>
#include        "ibmcom.h"

/*************************************************************************
 *                          8250 Definitions                             *
 *************************************************************************/

/*      Offsets to various 8250 registers.  Taken from IBM Technical     */
/*      Reference Manual, p. 1-225                                       */

#define TXBUFF  0                       /* Transmit buffer register */
#define RXBUFF  0                       /* Receive buffer register */
#define DLLSB   0                       /* Divisor latch LS byte */
#define DLMSB   1                       /* Divisor latch MS byte */
#define IER     1                       /* Interrupt enable register */
#define IIR     2                       /* Interrupt ID register */
#define LCR     3                       /* Line control register */
#define MCR     4                       /* Modem control register */
#define LSR     5                       /* Line status register */
#define MSR     6                       /* Modem status register */

/*      Modem control register bits                                      */

#define DTR     0x01                    /* Data terminal ready */
#define RTS     0x02                    /* Request to send */
#define OUT1    0x04                    /* Output #1 */
#define OUT2    0x08                    /* Output #2 */
#define LPBK    0x10                    /* Loopback mode bit */

/*      Modem status register bits                                       */

#define DCTS    0x01                    /* Delta clear to send */
#define DDSR    0x02                    /* Delta data set ready */
#define TERI    0x04                    /* Trailing edge ring indicator */
#define DRLSD   0x08                    /* Delta Rx line signal detect */
#define CTS     0x10                    /* Clear to send */
#define DSR     0x20                    /* Data set ready */
#define RI      0x40                    /* Ring indicator */
#define RLSD    0x80                    /* Receive line signal detect */

/*      Line control register bits                                       */

#define DATA5   0x00                    /* 5 Data bits */
#define DATA6   0x01                    /* 6 Data bits */
#define DATA7   0x02                    /* 7 Data bits */
#define DATA8   0x03                    /* 8 Data bits */

#define STOP1   0x00                    /* 1 Stop bit */
#define STOP2   0x04                    /* 2 Stop bits */

#define NOPAR   0x00                    /* No parity */
#define ODDPAR  0x08                    /* Odd parity */
#define EVNPAR  0x18                    /* Even parity */
#define STKPAR  0x28                    /* Stick parity */
#define ZROPAR  0x38                    /* Zero parity */

/*      Line status register bits                                        */

#define RDR     0x01                    /* Receive data ready */
#define ERRS    0x1E                    /* All the error bits */
#define TXR     0x20                    /* Transmitter ready */

/*      Interrupt enable register bits                                   */

#define DR      0x01                    /* Data ready */
#define THRE    0x02                    /* Tx buffer empty */
#define RLS     0x04                    /* Receive line status */

/*************************************************************************
 *                          Names for Numbers                            *
 *************************************************************************/

#define MAX_PORT        4

#define TRUE            1
#define FALSE           0

/*************************************************************************
 *                             Global Data                               *
 *************************************************************************/

/*  UART i/o addresses.  Values depend upon which COMM port is selected  */

int     uart_data;              /* Data register */
int     uart_ier;               /* Interrupt enable register */
int     uart_iir;               /* Interrupt identification register */
int     uart_lcr;               /* Line control register */
int     uart_mcr;               /* Modem control register */
int     uart_lsr;               /* Line status register */
int     uart_msr;               /* Modem status register */

char    com_installed;          /* Flag: Communications routines installed */
int     intnum;                 /* Interrupt vector number for chosen port */
char    i8259bit;               /* 8259 bit mask */
char    old_i8259_mask;         /* Copy as it was when we were called */
char    old_ier;                /* Modem register contents saved for */
char    old_mcr;                /*  restoring when we're done */
void interrupt (*old_vector)(); /* Place to save COM1 vector */

/*  Transmit queue.  Characters to be transmitted are held here until the  */
/*  UART is ready to transmit them.  */

#define TX_QUEUE_SIZE   16      /* Transmit queue size.  Change to suit */

char    tx_queue[TX_QUEUE_SIZE];
int     tx_in;                  /* Index of where to store next character */
int     tx_out;                 /* Index of where to retrieve next character */
int     tx_chars;               /* Count of characters in queue */

/*  Receive queue.  Received characters are held here until retrieved by  */
/*  com_rx()  */

#define RX_QUEUE_SIZE   4096    /* Receive queue size.  Change to suit */
```

```
char    rx_queue[RX_QUEUE_SIZE];
int     rx_in;                  /* Index of where to store next character */
int     rx_out;                 /* Index of where to retrieve next character */
int     rx_chars;               /* Count of characters in queue */

/*****************************************************************************
 *                              com_install()                                *
 *****************************************************************************
 * DESCRIPTION: Installs the communications drivers.                         *
 *                                                                           *
 * SYNOPSIS:    status = com_install(int portnum);                           *
 *              int     portnum;        Desired port number                  *
 *              int     status;         0 = Successful installation          *
 *                                      1 = Invalid port number              *
 *                                      2 = No UART for specified port       *
 *                                      3 = Drivers already installed        *
 *                                                                           *
 * REVISIONS:   18 OCT 89 - RAC - Translated from IBMCOM.PAS                 *
 *****************************************************************************/

const int       uart_base[] =   { 0x3F8, 0x2F8, 0x3E8, 0x2E8 };
const char      intnums[] =     { 0x0C,  0x0B,  0x0D,  0x0B };
const char      i8259levels[] = { 4,     3,     5,     4 };

int com_install(int portnum) {

    if (com_installed)                          /* Drivers already installed */
        return 3;
    if ((portnum < 1) || (portnum > MAX_PORT))  /* Port number out of bounds */
        return 1;

    uart_data = uart_base[portnum-1];           /* Set UART I/O addresses */
    uart_ier  = uart_data + IER;                /*  for the selected comm */
    uart_iir  = uart_data + IIR;                /*  port */
    uart_lcr  = uart_data + LCR;
    uart_mcr  = uart_data + MCR;
    uart_lsr  = uart_data + LSR;
    uart_msr  = uart_data + MSR;
    intnum    = intnums[portnum-1];             /* Ditto for interrupt */
    i8259bit  = 1 << i8259levels[portnum-1];    /*  vector and 8259 bit mask */

    old_ier = inportb(uart_ier);                /* Return an error if we */
    outportb(uart_ier, 0);                      /*  can't access the UART */
    if (inportb(uart_ier) != 0)
        return 2;

    disable();                                  /* Save the original 8259 */
    old_i8259_mask = inportb(0x21);             /*  mask, then disable the */
    outportb(0x21, old_i8259_mask | i8259bit);  /*  8259 for this interrupt */
    enable();

    com_flush_tx();                             /* Clear the transmit and */
    com_flush_rx();                             /*  receive queues */

    old_vector = getvect(intnum);               /* Save old COMM vector, */
    setvect(intnum, &com_interrupt_driver);     /*  then install a new one, */
    com_installed = TRUE;                       /*  and note that we did */

    outportb(uart_lcr, DATA8 + NOPAR + STOP1);  /* 8 data, no parity, 1 stop */

    disable();                                  /* Save MCR, then enable */
    old_mcr = inportb(uart_mcr);                /*  interrupts onto the bus, */
    outportb(uart_mcr,                          /*  activate RTS and leave */
             (old_mcr & DTR) | (OUT2 + RTS));   /*  DTR the way it was */
    enable();

    outportb(uart_ier, DR);                     /* Enable receive interrupts */

    disable();                                  /* Now enable the 8259 for */
    outportb(0x21, inportb(0x21) & ~i8259bit);  /*  this interrupt */
    enable();
    return 0;                                   /* Successful installation */
    }                                           /* End com_install() */

/*****************************************************************************
 *                              com_install()                                *
 *****************************************************************************
 * DESCRIPTION: Denstalls the communications drivers completely, without     *
 *              changing the baud rate or DTR.  It tries to leave the        *
 *              interrupt vectors and enables and everything else as they    *
 *              were when the driver was installed.                          *
 *                                                                           *
 * NOTE:        This function MUST be called before returning to DOS, so the *
 *              interrupt vector won't point to our driver anymore, since it *
 *              will surely get overwritten by some other transient program  *
 *              eventually.                                                  *
 *                                                                           *
 * REVISIONS:   18 OCT 89 - RAC - Translated from IBMCOM.PAS                 *
 *****************************************************************************/

void com_deinstall(void) {

    if (com_installed) {                        /* Don't de-install twice! */
        outportb(uart_mcr, old_mcr);            /* Restore the UART */
        outportb(uart_ier, old_ier);            /*  registers ... */
        disable();
        outportb(0x21,                          /*  ... the 8259 interrupt */
                 (inportb(0x21)  & ~i8259bit) | /*  mask ... */
                 (old_i8259_mask &  i8259bit));
        enable();
        setvect(intnum, old_vector);            /*  ... and the comm */
        com_installed = FALSE;                  /*  interrupt vector */
        }                                       /* End com_installed */
    }                                           /* End com_deinstall() */

/*****************************************************************************
 *                              com_set_speed()                              *
 *****************************************************************************
 * DESCRIPTION: Sets the baud rate.                                          *
 *                                                                           *
 * SYNOPSIS:    void com_set_speed(unsigned speed);                          *
 *              unsigned speed;                 Desired baud rate            *
 *                                                                           *
 * NOTES:       The input parameter can be anything between 2 and 65535.     *
 *              However, I (Wayne) am not sure that extremely high speeds    *
 *              (those above 19200) will always work, since the baud rate    *
 *              divisor will be six or less, where a difference of one can   *
 *              represent a difference in baud rate of 3840 bits per second  *
 *              or more.)                                                    *
 *                                                                           *
 * REVISIONS:   18 OCT 89 - RAC - Translated from IBMCOM.PAS                 *
 *****************************************************************************/

void com_set_speed(unsigned speed) {

    unsigned    divisor;                        /* A local temp */
```

```
    if (com_installed) {
        if (speed < 2) speed = 2;               /* Force proper input */
        divisor = 115200L / speed;              /* Recond baud rate divisor */
        disable();                              /* Interrupts off */
        outportb(uart_lcr,                      /* Set up to load baud rate */
                inportb(uart_lcr) | 0x80);      /*  divisor into UART */
        outport(uart_data, divisor);            /* Do so */
        outportb(uart_lcr,                      /* Back to normal UART ops */
                inportb(uart_lcr) & ~0x80);
        enable();                               /* Interrupts back on */
        }                                       /* End "comm installed" */
    }                                           /* End com_set_speed() */

/*****************************************************************************
 *                            com_set_parity()                               *
 *****************************************************************************
 * DESCRIPTION: Sets the parity and stop bits.                               *
 *                                                                           *
 * SYNOPSIS:    void com_set_parity(enum par_code parity, int stop_bits);    *
 *              int     code;           COM_NONE = 8 data bits, no parity    *
 *                                      COM_EVEN = 7 data, even parity       *
 *                                      COM_ODD  = 7 data, odd parity        *
 *                                      COM_ZERO = 7 data, parity bit = zero *
 *                                      COM_ONE  = 7 data, parity bit = one  *
 *              int     stop_bits;      Must be 1 or 2                       *
 *                                                                           *
 * REVISIONS:   18 OCT 89 - RAC - Translated from the Pascal                 *
 *****************************************************************************/

const char      lcr_vals[] = {
                    DATA8 + NOPAR,
                    DATA7 + EVNPAR,
                    DATA7 + ODDPAR,
                    DATA7 + STKPAR,
                    DATA7 + ZROPAR
                    } ;

void com_set_parity(int parity, int stop_bits) {
    disable();
    outportb(uart_lcr, lcr_vals[parity] | ((stop_bits == 2) ? STOP2 : STOP1));
    enable();
    }                                           /* End com_set_parity() */

/*****************************************************************************
 *                            com_raise_dtr()                                *
 *                            com_lower_dtr()                                *
 *****************************************************************************
 * DESCRIPTION: These routines raise and lower the DTR line.  Lowering DTR   *
 *              causes most modems to hang up.                               *
 *                                                                           *
 * REVISIONS:   18 OCT 89 - RAC - Transltated from the Pascal.               *
 *****************************************************************************/

void com_lower_dtr(void) {
    if (com_installed) {
        disable();
        outportb(uart_mcr, inportb(uart_mcr) & ~DTR);
        enable();
        }                                       /* End 'comm installed' */
    }                                           /* End com_raise_dtr() */

void com_raise_dtr(void) {
    if (com_installed) {
        disable();
```



```
        outportb(uart_mcr, inportb(uart_mcr) | DTR);
        enable();
        }                                       /* End 'comm installed' */
    }                                           /* End com_lower_dtr() */

/*****************************************************************************
 *                               com_tx()                                    *
 *                            com_tx_string()                                *
 *****************************************************************************
 * DESCRIPTION: Transmit routines.  com_tx() sends a single character by     *
 *              waiting until the transmit buffer isn't full, then putting   *
 *              the character into it.  The interrupt driver will then send  *
 *              the character once it is at the head of the transmit queue   *
 *              and a transmit interrupt occurs.  com_tx_string() sends a    *
 *              string by repeatedly calling com_tx().                       *
 *                                                                           *
 * SYNOPSES:    void    com_tx(char c);         Send the character c         *
 *              void    com_tx_string(char *s); Send the string s            *
 *                                                                           *
 * REVISIONS:   18 OCT 89 - RAC - Translated from the Pascal                 *
 *****************************************************************************/

void com_tx(unsigned char c) {
    if (com_installed) {
        while (!com_tx_ready()) ;               /* Wait for non-full buffer */
        disable();                              /* Interrupts off */
        tx_queue[tx_in++] = c;                  /* Stuff character in queue */
        if (tx_in == TX_QUEUE_SIZE) tx_in = 0;  /* Wrap index if needed */
        tx_chars++;                             /* Number of char's in queue */
        outportb(uart_ier,                      /* Enable UART tx interrupt */
                inportb(uart_ier) | THRE);
        enable();                               /* Interrupts back on */
        }                                       /* End 'comm installed' */
    }                                           /* End com_tx() */

void com_tx_string(char *s) {
    while (*s) com_tx(*s++);                    /* Send the string! */
    }                                           /* End com_tx_string() */

/*****************************************************************************
 *                               com_rx()                                    *
 *****************************************************************************
 * DESCRIPTION: Returns the next character from the receive buffer, or a     *
 *              NULL character ('\0') if the buffer is empty.                *
 *                                                                           *
 * SYNOPSIS:    c = com_rx();                                                *
 *              char    c;                      The returned character       *
 *                                                                           *
 * REVISIONS:   18 OCT 89 - RAC - Translated from the Pascal.                *
 *****************************************************************************/
/*
** com_rx - modified to return '\0' as input.
*/
int     com_rx(unsigned char *c) {

    int         rv;                             /* Local temp */

    if (!rx_chars || !com_installed) {            /* Return NULL if receive */
        *c = '\0';
        return 0;
    }                                           /*  buffer is empty */
    disable();                                  /* Interrupts off */
    *c = rx_queue[rx_out++];                    /* Grab char from queue */
    if (rx_out == RX_QUEUE_SIZE)                /* Wrap index if needed */
```

```
        rx_out = 0;
    rx_chars--;                                 /* One less char in queue */
    enable();                                   /* Interrupts back on */
    return 1;                                   /* The answer! */
    }                                           /* End com_rx() */

/*************************************************************************
 *                       Queue Status Routines                          *
 *************************************************************************
 * DESCRIPTION: Small routines to return status of the transmit and receive *
 *              queues.                                                 *
 *                                                                      *
 * REVISIONS:   18 OCT 89 - RAC - Translated from the Pascal.           *
 *************************************************************************/

int com_tx_ready(void) {                        /* Return TRUE if the */
    return ((tx_chars < TX_QUEUE_SIZE) ||       /*  transmit queue can */
            (!com_installed));                  /*  accept a character */
    }                                           /* End com_tx_ready() */

int com_tx_empty(void) {                        /* Return TRUE if the */
    return (!tx_chars || (!com_installed));     /*  transmit queue is empty */
    }                                           /* End com_tx_empty() */

int com_rx_empty(void) {                        /* Return TRUE if the */
    return (!rx_chars || (!com_installed));     /*  receive queue is empty */
    }                                           /* End com_tx_empty() */

/*************************************************************************
 *                          com_flush_tx()                              *
 *                          com_flush_rx()                              *
 *************************************************************************
 * DESCRIPTION: Buffer flushers!  These guys just initialize the transmit *
 *              and receive queues (respectively) to their empty state.  *
 *                                                                      *
 * REVISIONS:   18 OCT 89 - RAC - Translated from the Pascal            *
 *************************************************************************/

void com_flush_tx(void) { disable(); tx_chars = tx_in = tx_out = 0; enable(); }
void com_flush_rx(void) { disable(); rx_chars = rx_in = rx_out = 0; enable(); }

/*************************************************************************
 *                          com_carrier()                               *
 *************************************************************************
 * DESCRIPTION: Returns TRUE if a carrier is present.                   *
 *                                                                      *
 * REVISIONS:   18 OCT 89 - RAC - Translated from the Pascal.           *
 *************************************************************************/

int com_carrier(void) {
    return com_installed && (inportb(uart_msr) & RLSD);
    }                                           /* End com_carrier() */

/*************************************************************************
 *                      com_interrupt_driver()                          *
 *************************************************************************
 * DESCRIPTION: Handles communications interrupts.  The UART will interrupt *
 *              whenever a character has been received or when it is ready *
 *              to transmit another character.  This routine responds by *
 *              sticking received characters into the receive queue and *
 *              yanking characters to be transmitted from the transmit queue *
 *                                                                      *
 * REVISIOSN:   18 OCT 89 - RAC - Translated from the Pascal.           *
 *************************************************************************/
```



```
void interrupt com_interrupt_driver() {

    char        iir;                            /* Local copy if IIR */
    char        c;                              /* Local character variable */

/*  While bit 0 of the IIR is 0, there remains an interrupt to process  */

    while (!((iir = inportb(uart_iir)) & 1)) {  /* While there is an int ... */
        switch (iir) {                          /* Branch on interrupt type */

            case 0:                             /* Modem status interrupt */
                inportb(uart_msr);              /* Just clear the interrupt */
                break;

            case 2:                             /* Transmit register empty */

/*************************************************************************
 *  NOTE:  The test of the line status register is to see if the transmit *
 *         holding register is truly empty.  Some UARTS seem to cause    *
 *         transmit interrupts when the holding register isn't empty,   *
 *         causing transmitted characters to be lost.                   *
 *************************************************************************/

                if (tx_chars <= 0)              /* If tx buffer empty, turn */
                    outportb(uart_ier,          /*  off transmit interrupts */
                             inportb(uart_ier) & ~2);
                else {                          /* Tx buffer not empty */
                    if (inportb(uart_lsr) & TXR) {
                        outportb(uart_data, tx_queue[tx_out++]);
                        if (tx_out == TX_QUEUE_SIZE)
                            tx_out = 0;
                        tx_chars--;
                        }
                    }                           /* End 'tx buffer not empty */
                break;

            case 4:                             /* Received data interrupt */
                c = inportb(uart_data);         /* Grab received character */
                if (rx_chars < RX_QUEUE_SIZE) { /* If queue not full, save */
                    rx_queue[rx_in++] = c;      /*  the new character */
                    if (rx_in == RX_QUEUE_SIZE) /* Wrap index if needed */
                        rx_in = 0;
                    rx_chars++;                 /* Count the new character */
                    }                           /* End queue not full */
                //else { /* overwrite if the buffer is full */
                //    rx_queue[rx_in++] = c;
                //    if (rx_in == RX_QUEUE_SIZE)
                //      rx_in = 0;
                //    if (++rx_out == RX_QUEUE_SIZE)
                //      rx_out =0;
                //}
                break;

            case 6:                             /* Line status interrupt */
                inportb(uart_lsr);              /* Just clear the interrupt */
                break;

            }                                   /* End switch */
        }                                       /* End 'is an interrupt' */
    outportb(0x20, 0x20);                       /* Send EOI to 8259 */
    }                                           /* End com_interrupt_driver() */
```

```
/*
** file: ddicheck.c
** date: 6/29/92
**   by: Gregory D. Benson
** desc: TSR device driver routines (allow DOS to determine if drivers exist)
*/

char *progname = "ddicheck";

#include <stdio.h>
#include <stdlib.h>
#include <dos.h>
#include <conio.h>
#include <alloc.h>

#include "ddi.h"        /* device driver interface definitions */

int main()
{
        if (!DDIcheck_drivers()) {
                printf("%s: device drivers not found\n", progname);
                return(0);
        }
        else {
                printf("%s: device drivers found\n", progname);
                return(1);
        }
}
```

```
/*
** file: ddiquit.c
** date: 6/29/92
**   by: Gregory D. Benson
** desc: TSR device driver routines (quit all drivers)
*/

char *progname = "ddiquit";

#include <stdio.h>
#include <stdlib.h>
#include <dos.h>
#include <conio.h>
#include <alloc.h>

#include "ddi.h"        /* device driver interface definitions */

int main()
{
        if (!DDIcheck_drivers()) {
                printf("%s: device drivers not found\n", progname);
                exit(0);
        }

        if (!DDIsend_message(DDI, DDI_QUIT_ALL, NULL)) {
                printf("%s: unable to quit all drivers\n", progname);
        }
        else {
                printf("%s: quit all drivers\n", progname);
        }

        return(0);
}
```

```
#
# file: makefile
# data: 6/30/92
#   by: Gregory D. Benson
# desc: DDI (device driver interface)
#

# Compiler Info

BCPATH=\pkg\bc
LIB=$(BCPATH)\lib\fp87 $(BCPATH)\lib\mathl $(BCPATH)\lib\cl
STARTUP=$(BCPATH)\lib\c0l

# Object files

DDIIOBJ= ddiinit.obj intentry.obj ddi.obj adcdrv.obj kindrv.obj comdrv.obj \
        ibmcom.obj
DDIQOBJ= ddiquit.obj ddi.obj
DDICOBJ= ddicheck.obj ddi.obj
TSTOBJ=  drvtst.obj ddi.obj
ADCTSTOBJ = adctst.obj ddi.obj

# Rules

.c.obj:
  bcc -ml -c $<

.asm.obj:
  tasm $<

# Dependencies

all: ddiinit.exe ddiquit.exe ddicheck.exe drvtst.exe adctst.exe

ddiinit.exe: $(DDIIOBJ)
  tlink @ddiobjs, $*, ,$(LIB)

ddiquit.exe: $(DDIQOBJ)
  tlink $(STARTUP) $(DDIQOBJ), $*, ,$(LIB)

ddicheck.exe: $(DDICOBJ)
  tlink $(STARTUP) $(DDICOBJ), $*, ,$(LIB)

drvtst.exe: $(TSTOBJ)
  tlink $(STARTUP) $(TSTOBJ), $*, ,$(LIB)

adctst.exe: $(ADCTSTOBJ)
  tlink $(STARTUP) $(ADCTSTOBJ), $*, ,$(LIB)


ddiinit.obj:    ddiinit.c ddiinit.h ddi.h
intentry.obj:   intentry.asm
ddi.obj:        ddi.c ddi.h
ddiquit.obj:    ddiquit.c ddi.h
ddicheck.obj:   ddicheck.c ddi.h
adcdrv.obj:     adcdrv.c adcdrv.h
kindrv.obj:     kindrv.c kindrv.h
comdrv.obj:     comdrv.c comdrv.h
ibmcom.obj:     ibmcom.c ibmcom.h
drvtst.obj:     drvtst.c
adctst.obj:     adctst.c
```

```
/*
** file: robot.h
** date: 7/14/92
**   by: Gregory D. Benson
** desc: simple robot data structures
*/



#define NJOINTS 6

typedef struct {
        double j[NJOINTS];
} JOINTS;

typedef struct {
        double j[NJOINTS];
} JOINTSDEG;

typedef struct {
        double p[3];
        double r[3];
} LOCATION;

typedef double IMATRIX[4][4];

typedef IMATRIX *PIMATRIX;

typedef struct {
        IMATRIX m;
} TMATRIX;

typedef struct {
        double a[NJOINTS];
        double d[NJOINTS];
} DHPARAMS;


/* Function Prototypes */

void TMatrixPrint(TMATRIX *);
void DHParamsPrint(DHPARAMS *);
void fkin(JOINTS *, TMATRIX *, DHPARAMS *);
```

```
/*
** file: kingen.c
** date: 7/14/92
**   by: Gregory D. Benson
** desc: forward kinematics for the Schilling Miniature Manipulator
*/

#include<stdio.h>
#include<math.h>
#include"robot.h"



/*
** TMatrixPrint - print a transformation matrix
*/
void TMatrixPrint(TMATRIX *t)
{
        int i, j;

        for (i = 0; i < 4; i++) {
                printf("%7.2lf %7.2lf %7.2lf %7.2lf\n",
                        t->m[i][0], t->m[i][1], t->m[i][2], t->m[i][3]);
        }
}


/*
** DHParamsPrint - prints a set of D-H parameters
*/
void DHParamsPrint(DHPARAMS *s)
{
        int i;

        for (i=0; i<NJOINTS; i++) {
                printf("a[%d] = %7.2lf  d[%d] = %7.2lf\n",
                        i, s->a[i], i, s->d[i]);
        }
}

void fkin(JOINTS *jt, TMATRIX *t, DHPARAMS *p)
{
        double a2, a3, d2, d4, d6;
        double C1, C2, C3, C4, C5, C6;
        double S1, S2, S3, S4, S5, S6;
        double C23, S23;
        double te0, te1, te2, te3;

        /* set D-H parameters */

        a2 = p->a[1];
        a3 = p->a[2];

        d2 = p->d[1];
        d4 = p->d[3];
        d6 = p->d[5];

        /* set C and S terms */

        C1 = cos(jt->j[0]);
        S1 = sin(jt->j[0]);
        C2 = cos(jt->j[1]);
        S2 = sin(jt->j[1]);
```



```
        C23 = cos(jt->j[1]+jt->j[2]);
        S23 = sin(jt->j[1]+jt->j[2]);

        C4 = cos(jt->j[3]);
        S4 = sin(jt->j[3]);

        C5 = cos(jt->j[4]);
        S5 = sin(jt->j[4]);

        C6 = cos(jt->j[5]);
        S6 = sin(jt->j[5]);

        /* compute Nx, Ny, Nz */

        te0 = C4 * C5 * C6 - S4 * S6;
        te1 = S23 * S5 * C6;
        te2 = S4 * C5 * C6 + C4 * S6;
        te3 = C23 * te0 - te1;

        t->m[0][0] = C1 * te3 - S1 * te2;
        t->m[1][0] = S1 * te3 + C1 * te2;
        t->m[2][0] = -S23 * te0 - C23*S5*C6;
        t->m[3][0] = 0;

        /* compute Sx, Sy, Sz */

        te0 = C4 * C5 * S6 + S4 * C6;
        te1 = S23 * S5 * S6;
        te2 = -S4 * C5 * S6 + C4 * C6;
        te3 = -C23 * te0 + te1;

        t->m[0][1] = C1 * te3 - S1 * te2;
        t->m[1][1] = S1 * te3 + C1 * te2;
        t->m[2][1] = S23 * te0 + C23*S5*S6;
        t->m[3][1] = 0;

        /* compute Ax, Ay, Az */

        te0 = C23*C4*S5+ S23*C5;

        t->m[0][2] = C1 * te0 - S1*S4*S5;
        t->m[1][2] = S1 * te0 + C1*S4*S5;
        t->m[2][2] = -S23*C4*S5 + C23*C5;
        t->m[3][2] = 0;

        /* compute Px, Py, Pz */

        te0 = d6*(C23*C4*S5 + S23*C5) + S23*d4 + a3*C23 + a2*C2;
        te1 = d6*S4*S5 + d2;

        t->m[0][3] = C1*te0 - S1*te1;
        t->m[1][3] = S1*te0 + C1*te1;
        t->m[2][3] = d6*(C23*C5 - S23*C4*S5) + C23*d4 - a3*S23 - a2*S2;
        t->m[3][3] = 1;
}
```

```c
/*
** file: trinput.c
** date: 7/14/92
**   by: Gregory D. Benson
** desc: Telerobotics input interface program - using ADJR16 board
**
** note:
**
** Basic telerobotic interface with digital output (buttons)
**
*/

char *progname = "master";

/* stack size */

extern unsigned _stacklen = 16386U;


#include<stdio.h>
#include<stdlib.h>
#include<conio.h>
#include<ctype.h>
#include<math.h>
#include<bios.h>
#include"ddi.h"
#include"adcdrv.h"
#include"comdrv.h"
#include"ibmcom.h"

#include"robot.h"

/* ADC signiture */
#define ADC     0x01
#define KIN     0x02
#define COM     0x03

/* definitions */

#define DEG_TO_RAD      M_PI / 180.0
#define RAD_TO_DEG      180.0 / M_PI

#define MAXDIFF 2.0

/* Character Codes */

#define CODE_STX        0x82
#define CODE_ETX        0x83
#define CODE_DLE        0x90
#define CODE_DEL        0xFF


/* Protocol States */

#define VAL_NUM_TRIES   100

#define VAL_GET_DEL     0
#define VAL_GET_DLE1    1
#define VAL_GET_STX     2
#define VAL_GET_DATA    3
#define VAL_GET_DATADLE 4
#define VAL_GET_DLE2    5
#define VAL_GET_ETX     6
#define VAL_GET_CHKSUM  7
```



```c
/* Trajectory definitions */

#define MAX_PATH_DIFFS  500
#define VAL_NO_RECORD   0
#define VAL_RECORD      1
#define VAL_SAFE_OFF    0
#define VAL_SAFE_ON     1

/* Data Structures */

typedef struct {
        byte control;
        byte error;
        byte hand;
        byte oxbyte;
        int  data[6];
} VALMESSAGE;


typedef struct {
        JOINTS  config;                 /* configuration signs */
        JOINTS  mag;                    /* magnification values */
        JOINTS  maxdiff;                /* maximum joint differences */
        JOINTS  start;                  /* start of path */
        JOINTS  diff[MAX_PATH_DIFFS];   /* joint differences */
        char    hand[MAX_PATH_DIFFS];   /* hand status */
        long    count;                  /* number of joint differences */
} VALJDPATH;

typedef struct {
        double max[3];
        double min[3];
} VALBOX;


/* Global Data */

DHPARAMS DHPuma560      = {{ 0.0, 431.8, -20.32, 0.0, 0.0, 0.0 },
                           { 0.0, 149.09, 0.0, 433.07, 0.0, 56.25 } };

DHPARAMS DHSchilling    = {{ 0.0, 180.0, 0.0, 0.0, 0.0 },
                           { 0.0, 0.0, 0.0, 180.0, 0.0, 0.0 } };

JOINTSDEG OffZero       = {{ 0.0, 0.0, 0.0, 0.0, 0.0, 0.0 }};
JOINTSDEG OffSchilling  = {{-158.0, -85.0, -75.0, -168.0, -168.0, -168.0}};

JOINTSDEG VALzero_pos   = {{0.0, 0.0, 0.0, 0.0, 0.0, 0.0 }};
JOINTSDEG VALready_pos  = {{0.0, -90.0, 90.0, 0.0, 0.0, 0.0 }};

//double jfactor[2][6]  = {{-1.0, 1.0, -1.0, -1.0, 1.0, 1.0},  /* forward */
//                         {-1.0, -1.0, 1.0, -1.0, -1.0, 1.0}};/* backward */

JOINTS    zero          = {{ 0.0, 0.0, 0.0, 0.0, 0.0, 0.0 }};
//JOINTS          jconfig       = {{-1.0, -1.0, 1.0, -1.0, -1.0, 1.0}};/* backward */
JOINTS    jconfig       = {{-1.0, 1.0, -1.0, -1.0, 1.0, 1.0}};/* backward */
JOINTS    nconfig       = {{ 1.0, 1.0, 1.0, 1.0, 1.0, 1.0 }};
JOINTSDEG jmaxdiff      = {{ 4.0, 4.0, 4.0, 8.0, 8.0, 8.0 }};
JOINTS    jmag          = {{ 1.0, 1.0, 1.0, 1.5, 1.5, 2.0 }};

LOCATION  lmaxdiff      = {{ 10.0, 10.0, 10.0}, {1.0, 1.0, 1.0 }};
LOCATION  lmag          = {{ 0.5, 0.5, 0.5}, {1.0, 1.0, 1.0 }};

VALMESSAGE val;
```

```c
JOINTSDEG  deg;
JOINTS     rad, new, old, diff;
JOINTS     rjmaxdiff;

LOCATION   lnew, lold, ldiff;
TMATRIX    pos;

VALMESSAGE snd, rec;
char val_checksum;
char val_hand;
int engage_stat = 0;

VALJDPATH val_path;

char *valmenu[] = {     "Robot Control Menu:",
                        "-------------------",
                        "[1] Puma: Get Current Position",
                        "[2] Puma: PPmove",
                        "[3] Puma: Move",
                        "[4] Puma: Moves",
                        "[5] Puma: Ready Position",
                        "[6] Puma: Zero Position",
                        "[7] Schilling: Get Current Position",
                        "[8] Teleoperation - Joint-to-Joint",
                        "[9] Teleoperation - Record Joint-to-Joint",
                        "[A] Teleoperation - Playback Joint-to-Joint",
                        "[B] Teleoperation - Joint-to-Joint Safe",
                        "[?] This menu",
                        "[Q] Quit",
                        "[R] Quit - But keep drivers running"
                  };
int valmenucount = 16;


/* Function prototypes */

int master_exit(void);
int ADgetdig(int);
int ADgettog(int);

void sleep(int);

int  comtx(unsigned char);
int  comrx(unsigned char *);

int  GetHandStat(void);
void SetEngageStat(int);
int  GetEngageStat(void);

void JointsSet(JOINTS *, double);
void JointsCopy(JOINTS *, JOINTS *);
void JointsGetDiff(JOINTS *, JOINTS *, JOINTS *);
void JointsMaxDiff(JOINTS *, JOINTS *);
void JointsAddTo(JOINTS *, JOINTS *);
void JointsMulTo(JOINTS *, JOINTS *);
void JointsMulTo(JOINTS *, JOINTS *);
void JointsScalerMulTo(JOINTS *, double);
void JointsPrint(JOINTS *);

void PrintDegrees(JOINTS *);
void PrintRadians(JOINTS *);
void DegreesToRadians(JOINTSDEG *, JOINTS *);
void RadiansToDegrees(JOINTS *, JOINTSDEG *);
```



```c
void LocationCopy(LOCATION *, LOCATION *);
void LocationGetDiff(LOCATION *, LOCATION *, LOCATION *);
void LocationMaxDiff(LOCATION *, LOCATION *);
void LocationAddTo(LOCATION *, LOCATION *);
void LocationMulTo(LOCATION *, LOCATION *);
void LocationPrint(LOCATION *);

void ValPrintMessage(VALMESSAGE *);
void ValCopyMessage(VALMESSAGE *, VALMESSAGE *);
void ValClearPort(void);
void Valcomtx(unsigned char);
int  ValGetMessage(VALMESSAGE *);
void ValSendMessage(VALMESSAGE *);
void ValSetHand(int);
int  ValIsReady(void);
void ValDecodeTransData(int *, JOINTS *);
void ValDecodeJointData(int *, JOINTS *);
void ValEncodeTransData(JOINTS *, int *);
void ValEncodeJointData(JOINTS *, int *);
void ValEncodeLocationData(LOCATION *, int *);
void ValDecodeMessage(VALMESSAGE *, JOINTS *);

void ValQuit(void);

void ValSetCommandMode(void);
void ValSetAlterMode(void);
void ValJDMove(JOINTS *);
void ValPPMove(JOINTS *);
void ValPPMoves(JOINTS *);
void ValJCMove(LOCATION *);
void ValMove(LOCATION *);
void ValMoves(LOCATION *);
void ValGetLocation(JOINTS *, LOCATION *);
void ValSetBox(JOINTS *, VALBOX *, double);
int  ValIsInBox(JOINTS *, VALBOX *);

void ValPrintMenu(char **, int);
void ValGetCommand(void);

void ValSaveJDPath(void);
void ValLoadJDPath(void);

void ValTRJMode(int, VALJDPATH *, int);
int  ValTRJPlay(VALJDPATH *);
void ValTRCMode(int);

/*
** M A I N
*/
void main()
{
        COM_PACKET com;
        unsigned char toggle;

        /* determine if the drivers are installed */

        if (!DDIcheck_drivers()) {
                printf("%s: device drivers not found\n", progname);
                exit(0);
        }

        /* initialize Serial Port */
```

```
        com.arg.portnum = 1;    /* set com port number */
        DDIsend_message(COM, COM_INSTALL, &com);

        if (com.rv == 3) {
                printf("COM: com port already installed.\n");
        } else if (com.rv) {
                printf("com_install() error: %d\n", com.rv);
                exit(0);
        } else {
                DDIsend_message(COM, COM_RAISE_DTR, NULL);
                com.arg.speed = 19200;
                DDIsend_message(COM, COM_SET_SPEED, &com);
                com.arg.par.parity = COM_NONE;
                com.arg.par.stop_bits = 1;
                DDIsend_message(COM, COM_SET_PARITY, &com);
                DDIsend_message(COM, COM_FLUSH_RX, &com);
        }

        /* initialize ADC driver */

        DDIsend_message(ADC, ADC_START, NULL);
        //DDIsend_message(ADC, ADC_SETOFFSETS, &OffSchilling);

        /* set up screen */

        printf("UCD Telerobotics Testbed - Schilling Controller\n");
        printf("\n");

        /* test button toggles */

        while(!kbhit()) {
                DDIsend_message(ADC, ADC_GETTOGGLE, &toggle);
                printf("button1 = %d  button2 = %d button3 = %d\n",
                        (int) toggle & ADC_BUTTON1,
                        (int) toggle & ADC_BUTTON2,
                        (int) toggle & ADC_BUTTON3 );
        }
        getch();


        /* Process Val Commands */

        ValGetCommand();

        /* exit on return */
}

/*
** master_exit - perform all necessary exit code
*/
int master_exit()
{
        DDIsend_message(ADC, ADC_STOP, NULL);
        DDIsend_message(COM, COM_DEINSTALL, NULL);
        exit(0);
}

/*
** ADgetdig - get digital information for the AD board
*/
int ADgetdig(button)
int button;
{
        unsigned char dig;
```



```
        DDIsend_message(ADC, ADC_GETDIG, &dig);

        if (dig & ((unsigned char) 0x01 << (button)))
                return(1);
        else
                return(0);
}

/*
** ADgettog - get toggle information for the AD board
*/
int ADgettog(button)
int button;
{
        unsigned char dig;

        DDIsend_message(ADC, ADC_GETTOGGLE, &dig);

        if (dig & ((unsigned char) 0x01 << (button)))
                return(1);
        else
                return(0);
}


/*
** sleep - emulate unix sleep function - pause for x seconds
*/
void sleep(int x)
{
        long next;


        next = biostime(0, 0);
        next += (long) (18.2 * x);


        while ( biostime(0,0) < next) {
                /* printf("%lu  %lu\n", biostime(0,0), next); */
                if (kbhit()) {
                        getch();
                        printf("%s: aborting sleep(%d)\n", progname, x);
                        break;
                }
        }
}


/*
** comtx - send a character to COM driver
*/
int comtx(unsigned char c)
{
        COM_PACKET pkt;

        pkt.arg.tx = c;
        DDIsend_message(COM, COM_TX, &pkt);
        return(pkt.rv);
}

/*
** comtrx - receive a character from COM driver
*/
```

```
int comrx(unsigned char *c)
{
        COM_PACKET pkt;

        DDIsend_message(COM, COM_RX, &pkt);
        *c = pkt.arg.rx;
        return(pkt.rv);
}

/*
** GetHandStat - determine if 3nd button has been pressed
*/
int GetHandStat(void)
{
        static int stateold = 0;
        static int toggle = 0;
        int statenew;
        int rv = 0;

        statenew = ADgettog(3);

        if (stateold != statenew) {
                toggle = !toggle;
                if (toggle) {
                        rv = 1;
                }
                else {
                        rv = 2;
                }
        }
        stateold = statenew;

        return(rv);
}

/*
** SetEngageStat - set engage status
*/
void SetEngageStat(int x)
{
        engage_stat = x;
}

/*
** GetEngageStat - determine if 2nd button has been pressed
*/
int GetEngageStat(void)
{
        static int stateold = 0;
        int statenew;

        statenew = ADgettog(1);

        if (stateold != statenew) {
                engage_stat = !engage_stat;
        }
        stateold = statenew;

        return(engage_stat);
}


/*
** JointsSet - set each j1 value to s
*/
void JointsSet(JOINTS *j1, double s)
{
        int i;

        for (i = 0; i < NJOINTS; i++)
                j1->j[i] = s;
}

/*
** JointsCopy - copy values in j1 to j2
*/
void JointsCopy(JOINTS *j1, JOINTS *j2)
{
        int i;

        for (i = 0; i < NJOINTS; i++) {
                j2->j[i] = j1->j[i];
        }
}

/*
** JointsGetDiff - set d equal to j1 - j2
*/
void JointsGetDiff(JOINTS *d, JOINTS *j1, JOINTS *j2)
{
        int i;

        for (i = 0; i < NJOINTS; i++) {
                d->j[i] = j1->j[i] - j2->j[i];
        }
}

/*
** JointsMaxDiff - if abs(d) > max then limit d to max (both pos and neg)
*/
void JointsMaxDiff(JOINTS *d, JOINTS *max)
{
        int i;

        for (i = 0; i < NJOINTS; i++) {
                if ( fabs(d->j[i]) > max->j[i]) {
                        if (d->j[i] < 0) d->j[i] = -max->j[i];
                        else d->j[i] = max->j[i];
                }
        }
}

/*
** JointsAddTo - set j1 = j1 + j2
*/
void JointsAddTo(JOINTS *j1, JOINTS *j2)
{
        int i;

        for (i = 0; i < NJOINTS; i++)
                j1->j[i] += j2->j[i];
}

/*
** JointsMulTo - set j1 = j1 * j2
*/
void JointsMulTo(JOINTS *j1, JOINTS *j2)
{
```

```
        int i;

        for (i = 0; i < NJOINTS; i++)
                j1->j[i] *= j2->j[i];
}

/*
** JointsScalerMulTo - set j1 = s * j1
*/
void JointsScalerMulTo(JOINTS *j1, double s)
{
        int i;

        for (i = 0; i < NJOINTS; i++)
                j1->j[i] *= s;
}

/*
** JointsPrint - print a set of joint values
*/
void JointsPrint(JOINTS *jt)
{
        int i;

        for (i=0; i<NJOINTS; i++) {
                printf("j[%d] = %7.2lf\n", i, jt->j[i]);
        }
        printf("\n");
}

/*
** PrintDegrees - print an JOINTSDEG structure
*/
void PrintDegrees(JOINTS *a)
{
        int i;
        JOINTSDEG deg;

        RadiansToDegrees(a, &deg);
        for (i = 0; i < CHANNELS; i++) {
                printf("%7.3lf ", deg.j[i]);
        }
        printf("\n");
}

/*
** PrintRadians - print an JOINTS structure
*/
void PrintRadians(JOINTS *a)
{
        int i;

        for (i = 0; i < CHANNELS; i++) {
                printf("%7.3lf ", a->j[i]);
        }
        printf("\n");
}


/*
** DegreesToRadians - convert JOINTSDEG to JOINTS
*/
void DegreesToRadians(JOINTSDEG *deg, JOINTS *rad)
{
```



```
        int i;

        for (i=0; i < 6; i++) {
                rad->j[i] = deg->j[i] * DEG_TO_RAD;
        }
}

/*
** RadiansToDegrees - convert JOINTS to JOINTSDEG
*/
void RadiansToDegrees(JOINTS *rad, JOINTSDEG *deg)
{
        int i;

        for (i=0; i < 6; i++) {
                deg->j[i] = rad->j[i] * RAD_TO_DEG;
        }
}

/*
** LocationCopy - copy values in l1 to l2
*/
void LocationCopy(LOCATION *l1, LOCATION *l2)
{
        int i;

        for (i = 0; i < 3; i++) {
                l2->p[i] = l1->p[i];
                l2->r[i] = l1->r[i];
        }
}

/*
** LocationGetDiff - set d equal to l1 - l2
*/
void LocationGetDiff(LOCATION *d, LOCATION *l1, LOCATION *l2)
{
        int i;

        for (i = 0; i < 3; i++) {
                d->p[i] = l1->p[i] - l2->p[i];
                d->r[i] = l1->r[i] - l2->r[i];
        }
}

/*
** LocationMaxDiff - if abs(d) > max then limit d to max (both pos and neg)
*/
void LocationMaxDiff(LOCATION *d, LOCATION *max)
{
        int i;

        for (i = 0; i < 3; i++) {
                if ( fabs(d->p[i]) > max->p[i]) {
                        if (d->p[i] < 0) d->p[i] = -max->p[i];
                        else d->p[i] = max->p[i];
                }
                if ( fabs(d->r[i]) > max->r[i]) {
                        if (d->r[i] < 0) d->r[i] = -max->r[i];
                        else d->r[i] = max->r[i];
                }
        }
}
```

```
/*
** LocationAddTo - set l1 = l1 + l2
*/
void LocationAddTo(LOCATION *l1, LOCATION *l2)
{
        int i;

        for (i = 0; i < 3; i++) {
                l1->p[i] += l2->p[i];
                l1->r[i] += l2->r[i];
        }
}

/*
** LocationMulTo - set l1 = l1 * l2
*/
void LocationMulTo(LOCATION *l1, LOCATION *l2)
{
        int i;

        for (i = 0; i < 3; i++) {
                l1->p[i] *= l2->p[i];
                l1->r[i] *= l2->r[i];
        }
}

/*
** LocationPrint - print a location variable
*/
void LocationPrint(LOCATION *l)
{
        int i;

        printf("%7.3lf %7.3lf %7.3lf ", l->p[0], l->p[1], l->p[2]);
        printf("%7.3lf %7.3lf %7.3lf\n", l->r[0], l->r[1], l->r[2]);
}

/*
** ValPrintMessage - Print the contents of a VAL communication message
*/
void ValPrintMessage(VALMESSAGE *v)
{
        int i;

        printf("VALMESSAGE:\n");
        printf("  control =   %2X\n", (unsigned) v->control);
        printf("    error =   %2X\n", (unsigned) v->error);
        printf("   oxbyte =   %2X\n", (unsigned) v->oxbyte);
        printf("     hand =   %2X\n", (unsigned) v->hand);

        for (i = 0; i < 6; i++) {
                printf("  data[%d] = %4X\n", i, v->data[i]);
        }
        printf("\n");
}

/*
** ValCopyMessage - Copy the contents of message v1 into v2
*/
void ValCopyMessage(VALMESSAGE *v1, VALMESSAGE *v2)
{
        int i;

        v2->control = v1->control;
        v2->error   = v1->error;
        v2->oxbyte  = v1->oxbyte;
        v2->hand    = v1->hand;

        for (i = 0; i < 6; i++) {
                v2->data[i] = v1->data[i];
        }
}

/*
** ValClearPort - Clear the serial connection between IBM and VAL
*/
void ValClearPort(void)
{
        int i;

        for (i = 0; i < 50; i++) {
                comtx(CODE_DEL);
        }
}

/*
** Valcomtx - send a character to COM driver, and send second DLE
*/
void Valcomtx(unsigned char c)
{
        COM_PACKET pkt;

        comtx(c);
        if (c == CODE_DLE) comtx(CODE_DLE);
}


/*
** ValGetMessage - Get a VALMESSAGE from VAL via the COM driver
**
** State driven to accept message packets
*/
int ValGetMessage(VALMESSAGE *v)
{
        int rv = TRUE;
        int done = FALSE;
        int state;
        int tries;
        int count;
        int limit;
        unsigned char incom;
        char *buffer;

        state = VAL_GET_DEL;
        tries = 0;
        count = 0;
        limit = sizeof(VALMESSAGE);
        buffer = (char *) v;

        while (!done) {

                /* only try over VAL_NUM_TRIES bytes */

                if (tries >= VAL_NUM_TRIES) {
                        rv = FALSE;
                        done = TRUE;
                        break;
                }
```

```
        /* get the next character */

        while(comrx(&incom) == 0) {
//              if (kbhit()) {
//                      getch();
//                      rv = FALSE;
//                      break;
//              }
        }


        /* print out state information */

        //printf("state = %d, incom = %2X\n",
        //      state, (unsigned) incom);

        switch(state) {


        case VAL_GET_DEL:       /* start state, get first DEL */

        if (incom == CODE_DEL)  {
                state = VAL_GET_DLE1;
        } else {
                tries++;
                state = VAL_GET_DEL;
        }
        break;

        case VAL_GET_DLE1:      /* DLE1 state */

        if (incom == CODE_DLE) {
                state = VAL_GET_STX;
        } else {
                tries++;
                state = VAL_GET_DEL;
        }
        break;

        case VAL_GET_STX:       /* STX state */

        if (incom == CODE_STX) {
                state = VAL_GET_DATA;
        } else {
                tries++;
                state = VAL_GET_DEL;
        }
        break;

        case VAL_GET_DATA:      /* DATA state */

        if (incom == CODE_DLE) {
                state = VAL_GET_DATADLE;
                break;
        }

        buffer[count++] = incom;
        if (count >= limit) {
                state = VAL_GET_DLE2;
        }
        break;

        case VAL_GET_DATADLE:   /* DATADLE state */
```



```
                        if (incom == CODE_DLE) {
                                state = VAL_GET_DATA;
                        } else {
                                state = VAL_GET_ETX;
                        }
                        break;

                        case VAL_GET_DLE2:      /* DLE2 state */

                        if (incom == CODE_DLE) {
                                state = VAL_GET_ETX;
                        } else {
                                tries++;
                                state = VAL_GET_DEL;
                        }
                        break;

                        case VAL_GET_ETX:       /* ETX state */

                        if (incom == CODE_ETX) {
                                state = VAL_GET_CHKSUM;
                        } else {
                                tries++;
                                state = VAL_GET_DEL;
                        }
                        break;

                        case VAL_GET_CHKSUM:    /* Checksum state */

                        val_checksum = incom;
                        done = TRUE;
                        rv = TRUE;
                        break;

                        default:
                                tries++;
                                state = VAL_GET_DEL;
                                break;
                        }
                }
                /* ValPrintMessage(v); */
                return(rv);

}

/*
** ValSendMessage - Send a VALMESSAGE to VAL via the COM driver
*/
void ValSendMessage(VALMESSAGE *v)
{
        int i, limit;
        char *buffer;
        char checksum = 0;

        buffer = (char *) v;
        limit = sizeof(VALMESSAGE);

        comtx(CODE_DEL);
        comtx(CODE_DLE);
        comtx(CODE_STX);

        Valcomtx(v->control);
        Valcomtx(v->error);
```

```
        Valcomtx(v->hand);
        Valcomtx(v->oxbyte);

        for (i = 0; i < 6; i++) {
                Valcomtx((unsigned char) v->data[i] & 0xFF);
                Valcomtx((unsigned char) (v->data[i] >> 8) & 0xFF);
        }

        for (i = 0; i < limit; i++) {
                checksum += buffer[i];
                //Valcomtx(buffer[i]);
        }
        comtx(CODE_DLE);
        comtx(CODE_ETX);
        comtx(checksum);

        /* printf("checksum = %d\n", (int) checksum); */

}

/*
** ValSetHand - set the val_hand variable to open or close the hand on next
**              move
*/
void ValSetHand(int x)
{
        val_hand = (char) x;
}

/*
** ValIsReady - determine if VAL can receive a new message
*/
int ValIsReady(void)
{
        //VALMESSAGE val;

        //DDIsend_message(COM, COM_FLUSH_RX, NULL);
        ValGetMessage(&val);

        if (val.oxbyte == 0) {
                return(TRUE);
        }
        return(FALSE);
}

/*
** ValDecodeTransData - Decode a transformation from VAL
*/
void ValDecodeTransData(int *in, JOINTS *out)
{
        int i, j;

        for (i=0,j=3 ; i<3 ; i++, j++) {
                out->j[i] = (double) in[i] / 32.0;
                out->j[j] = (double) in[i] * DEG_TO_RAD / 32.0;
        }
}

/*
** ValDecodeJointData - Decode joint values from VAL
*/
void ValDecodeJointData(int *in, JOINTS *out)
{
        int i;
```



```
        for (i = 0; i < 6; i++) {
                out->j[i] = (double) in[i] * DEG_TO_RAD / 32.0;
        }
}

/*
** ValEncodeTransData - Encode a tranformation for VAL
**
** For X, Y, Z:
**
** double A = A * 32.0
**
** For Rx, Ry, Rz:
**
** double A = A/PI*32768
**
*/
void  ValEncodeTransData(JOINTS *in, int *out)
{
        int i, j;

        for (i=0,j=3 ; i<3 ; i++, j++) {
                out[i] = (int) in->j[i] * 32.0;
                out[j] = (int) in->j[i] * RAD_TO_DEG * 32.0;
        }

}


/*
** ValEncodeJointData - Encode joint values for VAL
*/
void  ValEncodeJointData(JOINTS *in, int *out)
{
        int i;

        for (i = 0; i < 6; i++) {
                out[i] = (int) (in->j[i] * RAD_TO_DEG * 32.0) ;
        }
}

/*
** ValEncodeLocationData - Encode a location for VAL
*/
void ValEncodeLocationData(LOCATION *in, int *out)
{
        int i;

        for (i = 0; i < 3; i++) {
                out[i] = (int) (in->p[i] * 32.0);
                out[i+3] = (int) (in->r[i] * 182.0444);
        }
}

/*
** ValDecodeMessage - Decode a VAL message to get ARM data
*/
void ValDecodeMessage(VALMESSAGE *v, JOINTS *a)
{

        if (v->control & 0x08) {        /* Get Transformation Data */
                ValDecodeTransData(v->data, a);
        } else {                        /* Get Joint Values */
```

```
                ValDecodeJointData(v->data, a);
        }
}


/*
** ValQuit - Send a quit message to VAL
*/
void ValQuit(void)
{
        int i;
        VALMESSAGE snd;

        snd.control     = 15;   /* control byte for quit */
        snd.error       = 0;
        snd.hand        = 0;    /* close hand */
        snd.oxbyte      = 0;

        for (i = 0; i < 6; i++) {
                snd.data[i] = 0;
        }
        ValPrintMessage(&snd);
        ValSendMessage(&snd);
        sleep(1);               /* make sure VAL get the message */
                                /* before COM interrupts are disabled */
}

/*
** ValSetCommandMode - set VAL to command mode
*/
void ValSetCommandMode(void)
{
        int i;
        VALMESSAGE snd;

        snd.control     = 1;    /* control byte for move jdmove */
        snd.error       = 0;
        snd.hand        = 0;
        snd.oxbyte      = 0;

        for (i = 0; i < 6; i++) {
                snd.data[i] = 0;
        }

        ValSendMessage(&snd);
}

/*
** ValSetAlterMode - set VAL to alter mode
*/
void ValSetAlterMode(void)
{
        int i;
        VALMESSAGE snd;

        snd.control     = 3;    /* control byte for alter mode */
        snd.error       = 0;
        snd.hand        = 0;
        snd.oxbyte      = 0;

        for (i = 0; i < 6; i++) {
                snd.data[i] = 0;
        }
```



```
        ValSendMessage(&snd);
}

/*
** ValJDMove - Send a Joint Difference Move command to VAL
*/
void ValJDMove(JOINTS *a)
{
        VALMESSAGE snd;

        snd.control     = 2;    /* control byte for move jdmove */
        snd.error       = 0;
        snd.hand        = val_hand;
        snd.oxbyte      = 0;

        ValEncodeJointData(a, snd.data);
        /* ValPrintMessage(&snd); */
        ValSendMessage(&snd);
}


/*
** ValPPMove - Send a Precision Point Move command to VAL
*/
void ValPPMove(JOINTS *a)
{
        VALMESSAGE snd;

        snd.control     = 11;   /* control byte for move #ppoint() */
        snd.error       = 0;
        snd.hand        = val_hand;
        snd.oxbyte      = 0;

        ValEncodeJointData(a, snd.data);
        ValPrintMessage(&snd);
        ValSendMessage(&snd);
}

/*
** ValPPmoves - Send a Straight Line Precision Point Move command to VAL
*/
void ValPPmoves(JOINTS *a)
{
        VALMESSAGE snd;
        JOINTS  rad;

        snd.control     = 13;   /* control byte for moves #ppoint() */
        snd.error       = 0;
        snd.hand        = val_hand;
        snd.oxbyte      = 0;

        ValEncodeJointData(a, snd.data);
        ValSendMessage(&snd);
}

/*
** ValCDMove - Send a Cartesian Difference Data to VAL (for alter)
*/
void ValCDMove(LOCATION *a)
{
        VALMESSAGE snd;

        snd.control     = 0;    /* control byte for alter */
        snd.error       = 0;
```

```
        snd.hand        = val_hand;
        snd.oxbyte      = 0;

        ValEncodeLocationData(a, snd.data);
        //ValPrintMessage(&snd);
        ValSendMessage(&snd);
}


/*
** ValMove - Send a Move command to VAL
*/
void ValMove(LOCATION *a)
{
        VALMESSAGE snd;

        snd.control     = 12;   /* control byte for move */
        snd.error       = 0;
        snd.hand        = val_hand;
        snd.oxbyte      = 0;

        ValEncodeLocationData(a, snd.data);
        ValSendMessage(&snd);
}

/*
** ValMoves - Send a Moves command to VAL
*/
void ValMoves(LOCATION *a)
{
        VALMESSAGE snd;

        snd.control     = 14;   /* control byte for moves */
        snd.error       = 0;
        snd.hand        = val_hand;
        snd.oxbyte      = 0;

        ValEncodeLocationData(a, snd.data);
        ValSendMessage(&snd);
}

/*
** ValGetLocation - given a set of joint angles return a LOCATION
*/
void ValGetLocation(JOINTS *j, LOCATION *l)
{
        TMATRIX T;

        fkin(j, &T, &DHSchilling);

        l->p[0] = T.m[0][3];
        l->p[1] = T.m[1][3];
        l->p[2] = T.m[2][3];
        l->r[0] = 0.0;
        l->r[1] = 0.0;
        l->r[2] = 0.0;
}

/*
** ValSetBox - setup a safe region
*/
void ValSetBox(JOINTS *j, VALBOX *b, double radius)
{
        int i;
```



```
        TMATRIX T;

        fkin(j, &T, &DHPuma560);

        for (i = 0; i < 3; i++) {
                b->max[i] = T.m[i][3] + radius;
                b->min[i] = T.m[i][3] - radius;
        }
}

/*
** ValIsInBox - check to see if the joint values j are in box b
*/
int  ValIsInBox(JOINTS *j, VALBOX *b)
{
        int rv = TRUE;
        int i;
        TMATRIX T;

        fkin(j, &T, &DHPuma560);

        for (i = 0; i < 3; i++) {
                if (T.m[i][3] > b->max[i]) {
                        rv = FALSE;
                        break;
                }
                if (T.m[i][3] < b->min[i]) {
                        rv = FALSE;
                        break;
                }
        }
        return(rv);
}


/*
** ValPrintMenu - print the robot control menu
*/
void ValPrintMenu(char **menu, int count)
{
        int i;

        printf("\n");

        for (i = 0; i < count; i++) {
                printf("%s\n", menu[i]);
        }

        printf("\n");
}

/*
** ValGetCommand - process user input
*/
void ValGetCommand()
{
        int i, done = FALSE;
        long trcount;
        char command[40];
        char first;
        JOINTS  radtmp;


        for (i = 0; i < NJOINTS; i++) {
```

```
        diff.j[i] = 0.0;
    }
    DegreesToRadians(&jmaxdiff, &rjmaxdiff);

    ValPrintMenu(valmenu, valmenucount);

    while(!done) {

        printf(".");

        scanf("%s", &command);

        first = command[0];

        switch(tolower(first)) {

        case '1': /* Print Current Position */

            DDIsend_message(COM, COM_FLUSH_RX, NULL);
            ValGetMessage(&val);
            ValDecodeMessage(&val, &rad);
            printf("Puma: Current Position\n");
            printf("Radians:\n");
            PrintRadians(&rad);
            printf("Degrees:\n");
            PrintDegrees(&rad);
            fkin(&rad, &pos, &DHPuma560);

            printf("TMatrix:\n");
            TMatrixPrint(&pos);
            break;

        case '2': /* move to ppoint */

            printf("Input 6 Joint Angles (in degrees)\n");
            printf(":");
            scanf("%lf %lf %lf %lf %lf %lf",
                &(deg.j[0]), &(deg.j[1]), &(deg.j[2]),
                &(deg.j[3]), &(deg.j[4]), &(deg.j[5]));
            DegreesToRadians(&deg, &rad);
            printf("PPmove: ");
            PrintDegrees(&rad);

            printf("PPmove: ");
            PrintRadians(&rad);
            ValPPMove(&rad);

        break;

        //case '3': /* move to transformation */
        //break;

        //case '4': /* move to transformation straight-line */
        //break;

        case '5': /* move to ready position */

            printf("Puma: Moving to ready position\n");
            DegreesToRadians(&VALready_pos, &radtmp);
            ValPPMove(&radtmp);
            break;

        case '6': /* move to zero position */
```



```
            printf("Puma: Moving to zero position\n");
            DegreesToRadians(&VALzero_pos, &radtmp);
            ValPPMove(&radtmp);
            break;

        //case '7':
        //break;

        case '8': /* teleoperation: joint-to-joint */

            ValTRJMode(VAL_NO_RECORD, NULL, VAL_SAFE_OFF);
            break;

        case '9': /* teleoperation: joint-to-joint record */

            ValTRJMode(VAL_RECORD, &val_path, VAL_SAFE_OFF);
            break;

        case 'a': /* teleoperation: joint-to-joint playback */

            ValTRJPlay(&val_path);
            break;

        case 'b': /* teleoperation: joint-to-joint safe */

            ValTRJMode(VAL_NO_RECORD, NULL, VAL_SAFE_ON);
            break;

        case '?': /* print the menu */

            ValPrintMenu(valmenu, valmenucount);
            break;

        case 'q': /* quit the program */

            ValQuit();
            DDIsend_message(ADC, ADC_STOP, NULL);
            DDIsend_message(COM, COM_DEINSTALL, NULL);

            done = TRUE;
            break;

        case 'r': /* quit, but leave drivers running */

            done = TRUE;
            break;


        default:  /* undefined command */

            printf("Undefined command\n");
            break;

        }
    }
}

/*
** ValTRJMode - Teleoperaction joint-to-joint
*/
void ValTRJMode(int record, VALJDPATH *jdpath, int safe)
{
    int done = FALSE;
    char hand = 0;
```

```c
	long trcount = 0;
	JOINTS radtmp;
	VALBOX saferegion;

	/* set zero offsets for the ADC driver */

	DDIsend_message(ADC, ADC_SETOFFSETS, &OffZero);
	DDIsend_message(ADC, ADC_SETCONFIG, &jconfig);

	/* set up safe region */

	if (safe) {
		ValGetMessage(&val);
		ValDecodeJointData(val.data, &radtmp);
		ValSetBox(&radtmp, &saferegion, 150.0);
	}

	/* get start position if in record mode */

	if (record) {
		DDIsend_message(COM, COM_FLUSH_RX, NULL);
		ValGetMessage(&val);
		ValDecodeMessage(&val, &radtmp);
		JointsCopy(&radtmp, &(jdpath->start));
	}

	/* set old and new joint values */

	DDIsend_message(ADC, ADC_GETRADIANS, &new);
	DDIsend_message(ADC, ADC_GETRADIANS, &old);

	/* flush the COM port */

	DDIsend_message(COM, COM_FLUSH_RX, NULL);

	/* make sure we are disengaged */

	SetEngageStat(0);

	while(!kbhit()) {
		while (!GetEngageStat()) {
			if (kbhit()) {
				done = TRUE;
				break;
			}
			printf("Paused.\n");
			ValSetCommandMode();
			DDIsend_message(ADC, ADC_GETRADIANS, &new);
			DDIsend_message(ADC, ADC_GETRADIANS, &old);
			DDIsend_message(COM, COM_FLUSH_RX, NULL);
		}
		if (done) break;

		//while(!ValIsReady());// {printf("trying\n");}

		while(1) {
			DDIsend_message(COM, COM_FLUSH_RX, NULL);
			ValGetMessage(&val);
			if (val.oxbyte == 0) break;
		}

		DDIsend_message(ADC, ADC_GETRADIANS, (ADRADIANS *) &new);
		JointsGetDiff(&diff, &new, &old);
```



```c
			JointsMaxDiff(&diff, &rjmaxdiff);
			JointsAddTo(&old, &diff);
			JointsMulTo(&diff, &jmag);

			//PrintDegrees(&diff);

			hand = GetHandStat();
			ValSetHand(hand);

			if (safe) {
				ValDecodeJointData(val.data, &radtmp);
				JointsAddTo(&radtmp, &diff);
				if (ValIsInBox(&radtmp, &saferegion)) {
					ValJDMove(&diff);
				}
				else {
					ValJDMove(&zero);
				}
			} else {
				ValJDMove(&diff);
			}

			ValSetHand(0);

			if (record) {
				JointsCopy(&diff, &(jdpath->diff[trcount]));
				jdpath->hand[trcount] = hand;
				if (trcount++ >= MAX_PATH_DIFFS) {
					printf("Path too long to record\n");
					break;
				}
			} else {
				trcount++;
			}
		}
		if (kbhit()) getch();
		ValSetCommandMode();
		printf("trcount = %ld\n", trcount);
		if (record) {
			jdpath->count = trcount;
			printf("Trajectory Recorded\n");
			printf("Start Position:\n");
			PrintDegrees(&(jdpath->start));
			printf("Number of via points: %ld\n", jdpath->count);
		}
}


/*
** ValTRJPlay - play back a recorded path
*/
int ValTRJPlay(VALJDPATH *jdpath)
{
	int done = FALSE;
	char inchar;
	long i;

	if (jdpath == NULL) {
		printf("Not a valid trajectory.\n");
		return(FALSE);
	}

	if (jdpath->count <= 0) {
		printf("No points in trajectory.\n");
```

```
		return(FALSE);
	}

	printf("Start position:\n");
	PrintDegrees(&(jdpath->start));
	printf("Press return to go to the start position:\n");
	getch();

	ValPPMove(&(jdpath->start));

	printf("Press return start the trajectory:\n");
	getch();

	printf("Press return to abort the trajectory:\n");

	/* set old and new joint values */

	DDIsend_message(ADC, ADC_GETRADIANS, &new);
	DDIsend_message(ADC, ADC_GETRADIANS, &old);

	/* flush the COM port */

	DDIsend_message(COM, COM_FLUSH_RX, NULL);


	for (i = 0; i < jdpath->count; i++) {
		if (kbhit()) {
			inchar = tolower((char) getch());
			if (inchar == 'p') {
				printf("Paused: press return to continue:\n");
				while(!kbhit());
				ValSetCommandMode();
				DDIsend_message(ADC, ADC_GETRADIANS, &new);
				DDIsend_message(ADC, ADC_GETRADIANS, &old);
				DDIsend_message(COM, COM_FLUSH_RX, NULL);
			} else {
				printf("Trajectory aborted\n");
				done = TRUE;
				break;
			}
		}
		if (done) break;

		//while(!ValIsReady()); // {printf("trying\n");}

		while(1) {

			//DDIsend_message(COM, COM_FLUSH_RX, NULL);

			ValGetMessage(&val);
			if (val.error != 0) {
				printf("Communications Error: %d\n",
					(int) val.error);
			}
			if (val.oxbyte == 0) break;
			//printf("waiting for VAL\n");
		}


		//PrintDegrees(&(jdpath->diff[i]));

		ValSetHand((int) jdpath->hand[i]);
		ValJDMove(&(jdpath->diff[i]));
		ValSetHand(0);
```



```
	}
	ValSetCommandMode();
	return(TRUE);
}

/*
** ValTRCMode - Teleoperation cartesian
*/
void ValTRCMode(int record)
{
	int done = FALSE;
	char hand = 0;
	long trcount = 0;
	JOINTS radtmp;

	printf("Teleoperation: Cartesian\n");

	/* set Schilling offsets for cartesian mode */

	DDIsend_message(ADC, ADC_SETOFFSETS, &OffSchilling);
	DDIsend_message(ADC, ADC_SETCONFIG, &jconfig);

	/* get start position if in record mode */

	/* set old and new joint values */

	DDIsend_message(ADC, ADC_GETRADIANS, &new);
	//DDIsend_message(ADC, ADC_GETRADIANS, &old);

	/* flush the COM port */

	DDIsend_message(COM, COM_FLUSH_RX, NULL);

	/* make sure we are disengaged */

	SetEngageStat(0);

	while(!kbhit()) {
		if (!GetEngageStat()) {
			ValSetCommandMode();

			while (!GetEngageStat()) {
				if (kbhit()) {
					done = TRUE;
					break;
				} else {
					printf("Paused.\n");
				}
			}
			if (!done) {
				DDIsend_message(ADC, ADC_GETRADIANS, &new);
				ValGetLocation(&new, &lnew);
				LocationCopy(&lnew, &lold);
				DDIsend_message(COM, COM_FLUSH_RX, NULL);
				ValSetAlterMode();
			}
		}
		if (done) break;

		while(!ValIsReady()); /* {printf("trying\n");} */

		//while(1) {
```

```
            //DDIsend_message(COM, COM_FLUSH_RX, NULL);

            //ValGetMessage(&val);
            //if (val.oxbyte == 0) break;
            //}

            DDIsend_message(ADC, ADC_GETRADIANS,
                    (ADRADIANS *) &new);


            //PrintDegrees(&new);

            ValGetLocation(&new, &lnew);
            //LocationPrint(&lnew);


            LocationGetDiff(&ldiff, &lnew, &lold);
            LocationMaxDiff(&ldiff, &lmaxdiff);
            LocationAddTo(&lold, &ldiff);
            LocationMulTo(&ldiff, &lmag);

            LocationPrint(&ldiff);

            hand = GetHandStat();
            ValSetHand(hand);

            ValCDMove(&ldiff);

            ValSetHand(0);

            if (record) {
                    //JointsCopy(&diff, &(jdpath->diff[trcount]));
                    //jdpath->hand[trcount] = hand;
                    //if (trcount++ >= MAX_PATH_DIFFS) {
                    //      printf("Path too long to record\n");
                    //      break;
                    //}
            } else {
                    trcount++;
            }
        }
        if (kbhit()) getch();

        ValSetCommandMode();
        printf("trcount = %ld\n", trcount);

        if (record) {
                //jdpath->count = trcount;
                //printf("Trajectory Recorded\n");
                //printf("Start Position:\n");
                //PrintDegrees(&(jdpath->start));
                //printf("Number of via points: %ld\n", jdpath->count);
        }
}
```

```
BCPATH=\pkg\bc
LIB=$(BCPATH)\lib\fp87 $(BCPATH)\lib\mathl $(BCPATH)\lib\cl
OBJ= trinput.obj ddi.obj kingen.obj

trinput.exe: $(OBJ)
  tlink $(BCPATH)\lib\c0l $(OBJ),trinput.exe,,$(LIB)

.c.obj:
  bcc -ml -c $<

trinput.obj:     trinput.c robot.h
kingen.obj:      kingen.c robot.h
```