California AHMCT Program
University of California at Davis
California Department of Transportation

TELEOPERATED AND AUTOMATED MAINTENANCE EQUIPMENT ROBOTICS PHASE III

Volume I

Andrew McKee, Ross Lamm, Sonja Sun
Andrew Frank, Paul Chen

AHMCT Research Report
UCD-ARR-98-08-31-01

Report of Contract
65X875 T.O. 95-8

August 31, 1998

This work was supported by the California Department of Transportation (Caltrans) Advanced Highways Maintenance and Construction Technology Program (AHMCT) at UC-Davis

ABSTRACT

The research of this phase at UC Davis was to further improve performance, productivity and operator comfort of the existing teleoperated front-end loader that has been developed in Phase I and Phase II. The enhancement of the existing teleoperated front-end loader includes development of teachable programs and improvement of the 3D video feedback system.

Three teachable programs (time-based, position-based and hybrid) were developed during this phase. These programs, with necessary sensory feedback incorporated, enable the loader to follow the main operation sequences demonstrated by an operator to complete various tasks. The objective is to automatically execute repetitive tasks in order to reduce the operator's fatigue and increase productivity. The time-based program records operator control inputs, duration and sequences, stores them in memory and plays the sequence back with the stored control values for the amount of time recorded. The position-based program records data from position sensors on the loader's bucket and arm, and plays back accordingly. This program offers man-machine interactive feedback. The hybrid program takes characteristics from both the time-based and position-based programs. It controls all loader functions and also incorporates man-machine interactive feedback.

The three semi-automatic programs were compared along with manual operation and normal teleoperation in field testing. All modes were monitored for accuracy, productivity, and operator fatigue. The results have shown that among three programs, the position-base program is most feasible for implementation.

An improved video/audio feedback system was integrated into the teleoperated front-end loader. The system is compact and low cost compare to the previous system developed in Phase II. The system employed an ALCS stereo display system developed by StereoGraphics Corp. for the front view and two LCD monitors by Sharp for side and rear views.

Field test evaluations demonstrated usefulness of incorporating both side and rear monoscopic peripheral views in addition to the stereoscopic front view. Peripheral visions enable operators to identify obstacles to the sides and the rear of the vehicle which could be dangerous to both the machine and the operator. The flicker-free stereoscopic front view proved to allow for depth perception which led to more precise control of the vehicle and bucket operations compared to the monoscopic system. Much energy and effort was placed into designing and constructing an ergonomic and effective telepresence display unit. The system which Allows for both multiple telepresence views and line-of-sight has greatly enhanced the telepresence performance and increased operators' confidence of teleoperating a heavy duty equipment.

A relationship between optimal parallel stereo camera separation (Inter-Viewpoint Distance, IVD) and object distance was determined. For the range of from 10 - 30 feet an IVD of 5.0 inches was found to present large enough retinal disparity with minimal distortion to allow for optimal depth perception cues as opposed to the previous standard of 2.5 inch IVD. The (IVD) investigation concludes that an optimum spacing versus range can be determined. Future research may involve the design and construction of a system which can be automatically or perhaps manually adjusted to optimize IVD for the task at hand.

EXECUTIVE SUMMARY

Significant hazards exist for which the enclosed cab devices and other preventive measures offer inadequate operator protection. Casualties and injuries occur every year across the nation, despite preventive efforts. The need to remove the operator from the vehicle cab to a safe operating site under hazardous operating conditions has been made evident by such hazards inherent in highway maintenance.

In an attempt to eliminate the operator's exposure to the hazards associated with highway maintenance and to improve productivity, University of California at Davis initiated and has been granted from California Department of Transportation, a project entitled Teleoperated and Automatic Maintenance Equipment Robotics (TAMER) to developed a teleoperation system (remote control package) for a front-end loader. The development of the TAMER project has three phases.

Previous phases (phase I and phase II):

During Phase I of the TAMER project, a teleoperation system with human engineered controls which can be adapted to general purpose heavy equipment was developed. A 116 hp CASE 621 front-end loader with a 2.75 yard bucket capacity was used as a model. The teleoperation system allows the operator to normally operate the loader in the cab, as well as to leave the vehicle cab and continue the operation remotely at a safe site if hazardous conditions are encountered. The system has built-in safety features, such as emergency stop, automatic failure stop and, communication safeguard to assure reliable communication and safety.

In Phase II, a three dimensional color video/audio feedback system was developed and integrated into the teleoperated front-end loader. The video/audio feedback system enables the operator to regain the major senses, as to the status of the vehicle and position of the vehicle relative to the points of operation, which are lost during remote operation from a distance greater than 200 ft. The three dimensional and color images enable operators to better identify objects in unfamiliar environments.

Tests and evaluations have been conducted at various sites including Caltrans District 11 (San Diego), District 4 (Oakland), District 3 (Sacramento) and a land slide site along Highway 101 (District 1), where the teleoperated front-end loader was operated by more than 30 Caltrans maintenance workers. The entire system has proven to be practical, reliable, safe and sufficiently easy to operate. The overall performance of the teleoperated

front-end loader has been very satisfactory according to surveys and questionnaires filled out by the users.

The feasibility of teleoperating unmanned highway maintenance equipment has thus been well demonstrated. Demonstrations of the remote controlled front-end loader at various Caltrans districts have evoked a strong desire of using such equipment for high risk operations, as commented by the manager of District 1: "Two to four such teleoperated units are highly desired."

Phase III:

There are two objectives of the Phase III. The first objective was to further improve performance, productivity and operator comfort of the existing teleoperated front-end loader that has been developed in Phase I and Phase II. The second objective was to equip two in-service front-end loaders with teleoperation systems (through subcontract) for deployment to Caltrans districts for field evaluation and feedback. This report only covers the enhancement of the teleoperated loader developed at UC Davis. The report on the development of the additional two teleoperation units was delivered by the subcontractor, the Unmanned Solutions, Inc. (USI) and included as an attachment of this main report.

The enhancement of the existing teleoperated front-end loader includes development of teachable programs and improvement of the 3D video feedback system.

Teachable Programs:

Three teachable programs (time-based, position-based and hybrid) were developed during this phase. These programs, with necessary sensory feedback incorporated, enable the loader to follow the main operation sequences demonstrated by an operator to complete various tasks. The objective is to automatically execute repetitive tasks in order to reduce the operator's fatigue and increase productivity. The time-based program records operator control inputs, duration and sequences, stores them in memory and plays the sequence back with the stored control values for the amount of time recorded. The position-based program records data from position sensors on the loader's bucket and arm, and plays back accordingly. This program allows the operator to "trim" the operation during playback. The hybrid program combines characteristics from both the time-based and position-based programs.

Field tests have shown that among three programs, the position-base program is most feasible for implementation. Excellent accuracy and speed were achieved when performing repetitive tasks of bucket operations. The feasibility of incorporating teachable programs to the teleoperation system to reduce operator's fatigue and improve productivity was demonstrated. The position-based program can be upgraded to include vehicle navigation if desired. However the system will be more complicated and costly than the existing one.

Video/audio Feedback system:

An improved video/audio feedback system was integrated into the teleoperated front-end loader. The system is compact and low cost compare to the previous system developed in Phase II. The system employed an Alternating Liquid Crystal stereo display system (ALCS) developed by StereoGraphics Corp. for the front view and two LCD monitors by Sharp for side and rear views.

Field test evaluations demonstrated usefulness of incorporating both side and rear monoscopic peripheral views in addition to the stereoscopic front view. Peripheral visions enable operators to identify obstacles to the sides and the rear of the vehicle which could be dangerous to both the machine and the operator. The flicker-free stereoscopic front view proved to allow for depth perception which led to more precise control of the vehicle and bucket operations compared to the monoscopic system. Much energy and effort was placed into designing and constructing an ergonomic and effective telepresence display unit. The system which Allows for both multiple telepresence views and line-of-sight has greatly enhanced the telepresence performance and increased operators' confidence of teleoperating a heavy duty equipment.

A relationship between optimal parallel stereo camera separation (Inter-Viewpoint Distance, IVD) and object distance was determined. For the range of from 10 - 30 feet an IVD of 5.0 inches was found to present large enough retinal disparity with minimal distortion to allow for optimal depth perception cues as opposed to the previous standard of 2.5 inch IVD. The (IVD) investigation concludes that an optimum spacing versus range can be determined. Future research may involve the design and construction of a system which can be automatically or perhaps manually adjusted to optimize IVD for the task at hand.

Two in-service teleoperated front-end loaders:

Two commercial grade TAMER systems were developed by the subcontractor USI based on the technologies developed at University of California at Davis. The systems were installed on two Caltrans Case 721B front-end loaders and delivered to Caltrans District 1 and District 5.

The two systems are ruggedized, weather-proof and vibration tolerant. The stationary remote operating unit (SROU) is enclosed in a cab from Case Co. to better protect the control system and provide operator's comfort. Instead of ON/OFF controls used in the previous TAMER system, the newly developed systems employed proportional controls to implement functions of loader arm, buck, clamshell, steering, throttle and brake to allow speed control to nearly all of the loader's functionality. A radio modem that communicates at 38,400 baud rate and has its' own ID name was used to enhance communication reliability. The system's multiple radio frequency capability was well demonstrated during field tests with both remote controlled loaders operating in the same area without communication failure due to interference. Other safety features include a second modem only for emergency stop and a tilt switch on the PROU to automatically shut down the system if the operator accidentally falls down.

Recommendations for future improvement include significant weight reduction of the portable remote operating unit (PROU) by cutting down the power consumption and selecting light weight joystick and other components, refinement of hydraulic control subsystem for improved performance, and improvement of battery life.

For detail report on the two TAMER systems, refer to the attachment "Design and Installation of A Teleoperated and Automated Maintenance Equipment Robotics (TAMER) System" by USI.

TABLE OF CONTENTS

LIST OF FIGURES

DISCLAIMER / DISCLOSURE

"The research reported herein was performed as part of the Advanced Highway Maintenance and Construction Technology (AHMCT) Center, within the Department of Mechanical and Aeronautical 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 author(s) who is (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."

SECTION I

SEMI-AUTOMATIC CONTROL (TEACHABLE PROGRAM) OF THE TELEOPERATED FRONT-END LOADER

Chapter 1

Introduction

The Teleoperated and Automated Maintenance Equipment Robotics (TAMER) project was created with the intent of improving the safety of the operators of heavy earth moving equipment. The project was developed by the University of California, Davis' Advanced Highway Maintenance and Construction Technology (AHMCT) center with the support and funding of the California Department of Transportation (Caltrans). Caltrans' interest in teleoperation was piqued when it realized that injuries and casualties to operators in hazardous environments can be reduced or eliminated. Teleoperation would allow the equipment operator to be safely distant from the site of danger, while retaining full control. With this goal in mind, a Case 621 front end loader with a 2 yard bucket was converted to be capable of both normal operation and teleoperation.

The TAMER loader system has five basic: computers, machine interfaces, communications, operator interfaces, and operator. A block diagram of the system follows:

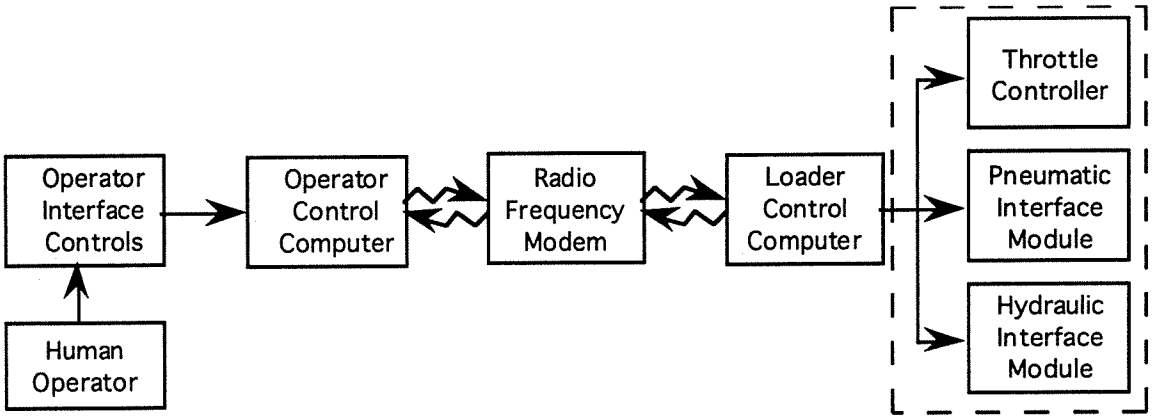


Figure 1.1 Tamer System Block Diagram

The computers consist of the Loader Control Computer (LCC) and the Operator Control Computer (OCC). The OCC acts as a data acquisition device by interpreting the human operator's control movements and sending that information to the LCC. The OCC also receives loader status messages from the LCC and displays them to the operator. The LCC is the heart and brain of the whole system. It decodes the OCC data and translates it into machine movements. It also controls the mode, e.g., remote, standby, emergency stop, etc., the loader is in.

There are three machine interface modules for directly controlling the loader: throttle, pneumatic, and hydraulic. The LCC controls the pneumatic flow, which controls the loader's brakes, and the hydraulic flow, which controls steering, bucket, and arm operations. Because of the importance of precise throttle control, it has its own controller and actuator.

Communication between the OCC and LCC is accomplished through a secure Spread Spectrum radio frequency modem link operating in the 900 MHz range. With the addition of a three dimensional (3-D) video system, the loader can be safely operated from as far as a quarter of a mile away. This operational range is limited only by the radio link.

For flexibility, two different operator interfaces have been created. One is a sit-down station that duplicates the layout of controls on the loader, and allows long distance operation as well as long periods of operation. It further provides operator familiarity with the controls. The second is a portable unit which is used for rough terrain, where closer supervision and mobility is needed. It is a portable back pack and chest-mounted unit that enables the operator to control all loader functions with just his hands, including throttle and brakes.

The most important subsystem of TAMER is the operator. In teleoperation, the human is always in control of the machine and is responsible for its movements. While some operations may be automatic, the operator has the power to interrupt and take control at will. The result is a safer system that is less dependent on sophisticated sensors and controls.

Adding computer control to the loader system creates more flexibility, resulting in greater productivity and convenience. The computer enables multiple tasks at the push of a button and even "teachable functions". Current industrial methods for teaching a robot include guiding (either by pendant or manually), off-line programming, and on-line programming [1]. The TAMER project is unique in that it integrates all three methods. The guiding method is used initially to watch and memorize as the operator works through a sequence of movements. During subsequent playbacks of the recorded information, the operator has the option of modifying the motion in the recorded program, an aspect of on-line real-time programming. Off-line programming is also available, where preset motions are pulled from libraries or programmed mathematically step by step. Certain preset sequences include lifting a load of dirt to dump height, dumping a load, moving fore and aft, etc. Recorded functions can be played back multiple times, increasing productivity and reducing operator fatigue.

This section of the report involves the incorporation of "teachable functions" into the TAMER front end loader's control programs. The goal is to have a "record-and-play" capability that will perform as accurately and in the same amount of time, or less, as normal manual operation. In order to do this, the program will be constructed with both open loop and feedback control techniques. Feedback will be introduced in both conventional and unique forms. Conventional feedback will involve a concept known as a "finite state machine," where each state is defined as the position of all controls, components, etc. at a given moment in time [12]. The unique method of feedback will be "human controlled" automatic controls or an "interactive automated system". The human will have the option to correct or alter automated

functions at every instant in time so that he is in continuous control. He does not, however, have to control all functions completely since he is only performing a trim function. In this thesis it will be shown that this control scheme, with its combination of automatic and interactive feedback control, greatly reduces the human task load and is both repeatable and accurate.

Chapter 2
An Overview of Teachable Programming

From the introduction of industrial robots in the 1960's, a primary factor determining their usefulness has been successfully teaching them their task(s). The first robots, "UNIMATES," were constructed by Unimation, Inc. in 1961. UNIMATE operators used a method called "teach by showing," wherein an operator guided the robot through the task to be learned with a hand-held controller. Other early robots were taught by direct human manipulation, wherein an operator physically guided the robot through its motions. Both of these methods were sufficient for simple point-to-point movement, such as spot welding, or continuous path, such as spray painting operations.

With the advent of more powerful computers and sophisticated programming languages, off- and on-line programming of robots became possible. Robot programming languages (RPL's) were developed having special features that apply specifically to the manipulation of robots. Current robot teaching research is focusing on task-level programming languages. These are languages that allow the user to command desired sub-goals of the task directly, rather than to specify the details of every action the robot is to take [1,2].

To best present teachable programming in robotics, a description of some background concepts and definitions is necessary. The most basic concept is that of the *point*. The point gives a description of the robot manipulator configuration at a given instant. Most commonly it is represented in one of two different coordinate systems. In the *joint-level representation* (JLR), the position of all the links of a manipulator of n degrees of freedom can be specified with a set of n joint variables, often referred to as the joint vector. The space of all joint vectors makes up joint space [2]. The JLR of a simple two-link manipulator is as follows:

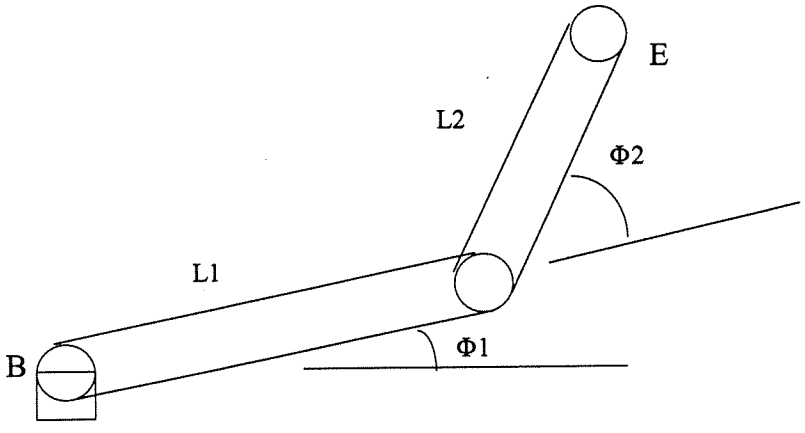


Figure 2.1 Simple 2-Link Manipulator

In order to know the location of the end effector E with respect to the manipulator base B, only the joint angles and arm lengths need be known.

However, in many instances it is desirable to represent points in *Cartesian* or *World Coordinate Representation* (WCR). In this representation the point is specified by the vector of positions and orientation of a representative coordinate frame on the robot with respect to the room coordinate frame [1]. In the case of the simple mechanism shown in Fig. 2 the joint angles and lengths can be used to find the end effector point with respect to the base through trigonometry:

$$X_{end} = L_1 \sin\Phi_1 + L_2 \sin(\Phi_1+\Phi_2)$$

$$Y_{end} = L_1 \cos\Phi_1 + L_2 \cos(\Phi_1+\Phi_2)$$

In more complex robots the trigonometric relationships become very complex, so homogeneous transformations are used. A matrix is constructed to allow the transformation between a coordinate frame located in the end effector and a coordinate frame located in the base.

A combination or continuum of points is known as a *path*. However, point data alone is not necessarily sufficient to define the path, as different robot motions can be characterized by the same path. Thus, the path needs to be parameterized by a time variable, introducing *velocities*. When velocity data is introduced, the path is called a *trajectory*.

In simple robots there are several classes of tasks. The most basic is the *single chain point-to-point*. In these tasks the exact velocity or trajectory of the end effector is not important. Some examples are pick-and-place, or machine loading/unloading operations where the robot can take alternate paths dependent on acquired data (i.e., discard, or keep according to quality control). Another class of robot task is that of the *continuous path*. In this type of task the robot end effector is constrained to move along a specific path in either joint or world coordinates. An example might be straight line motion with obstacle avoidance, where the path is important but velocity is not. However, in some continuous path operations, such as painting or welding, velocity is also important.

Robot teaching means providing complete point and velocity profile data to a computer program that controls the performance of tasks. Teaching is the user- controlled part of the task control program. There are many different methods in use today that allow the user to impart task data to the robot.

The most elementary is manual guiding of the robot. In this method, the robot is actually physically led through an operational sequence by a human teacher. This method is sometimes preferred because of its low-technology simplicity, but it has many obvious limitations. First, it is constrained by human speeds and accuracy. Second, it is not useful in

hazardous environments where the human teacher would be in danger. Finally, a skilled human teacher who is familiar with the dynamics of the machine is necessary.

A more common teaching method is *guiding*, using a teaching pendant. The pendant is a hand-held device, such as a keyboard, that the human teacher can use as a remote controller. Teaching with a pendant involves two fundamental steps. The first step is to move the robot manipulator to a desired task point. The second step is to store that point in memory, usually with a button press. These two steps are repeated until the entire desired task is learned [4].

Pendants can usually be used to manipulate the robot in both joint-level and Cartesian motion. In joint-level mode, each joint of the robot is controlled individually. This method tends to be inefficient as it is not very intuitive to a human operator. In Cartesian coordinate mode, the reference frame's origin is usually the robot base. This method is more convenient to and easily understood by the average operator. Also, it allows the use of additional controls such as joysticks to facilitate operation.

The main advantage to teaching a robot through guiding with a pendant is the method's simplicity. It is relatively inexpensive and does not require the operator to have extraordinary skills. However, guiding with a pendant requires an interruption in normal operation while the robot is being taught, resulting in a loss in productivity. Some applications, such as batch-production for instance, require the robot to be reprogrammed regularly. In addition, guiding with a pendant is very time consuming and inflexible, which also decreases productivity. In fact, the time overhead in teaching a robot with a pendant may make it uneconomical [3].

With the development of more powerful and versatile computer control technology, *off-line programming* became possible. Off-line programming allows the points and paths of the robot task(s) to be specified through programming. No pendant is necessary. A successful off-line programming system will include the following characteristics [3,6]:

- knowledge of the process or task to be programmed
- a three-dimensional world model
- kinematic and dynamic models of the robot
- a graphical or textual robot programming system
- methods for verifying the correctness of the programs
- interfaces to the robots for downline loading of programs and upline transfer of process data
- a user-friendly programming environment

The development of off-line programming systems has encountered a few problems. One problem is that it is difficult to create a modeling and programming package that is independent of both the robot and the application [3]. Also, there is a lack of a standard interface between robots, sensors, and computers. Finally, is it very difficult, if not impossible, to model the uncertainties involved in real-world operations.

The potential benefits of an off-line programming system however, are enormous. First, the robot is free for use while programming is in progress, so the robot's downtime is greatly decreased. Second, off-line programming can be accomplished in a safe, remote environment, reducing risk to the operator. Third, if many robots are being employed with the same system, programs can be portable from one robot to another.

Probably the only disadvantage to off-line programming is that it requires a skillful programmer who has excellent knowledge of the robot and its dynamics. However, an advanced interface system with very user-friendly characteristics can greatly minimize this problem.

The possibility of a direct computer interface with the robot while it is in operation brings up the subject of *on-line programming*. On-line programming allows a programmer to modify an existing taught program as the process occurs, to account for deviations or irregularities. For example, if in a pick-and-place operation the manipulator begins to damage the product, the programmer might reduce the gripping force on-line. This implies an operator is monitoring the action of the robot continuously and able to correct misbehavior on-line without stopping it.

Guiding with a pendant, off-line programming, and on-line programming are integrated into the TAMER project. In this project, teaching will be done not by a specialized pendant, but rather the actual vehicle remote controller while the vehicle is performing a task. The sequence of motions taught will be stored in the LCC computer's memory and can be replayed at the push of a button. The important innovation is that the operator can "adjust" or "correct" the program as it plays back. This allows productivity to continue while small corrections are made to the memorized routine. Current teachable systems do not have this level of interactive control. However, since the TAMER is operating in a dynamic and undefined environment, this method of teaching appears to be necessary.

In addition, the TAMER system is designed so that the human operator can interactively control the machine at every instant. This makes the operator responsible for the actions of the machine and relieves the system designer liability in the unlikely event of a system failure. The operator can always resort to the Emergency Stop switch, which brings the machine to a safe condition.

Chapter 3

System Description

3.1 HARDWARE

In order to present how teachable functions were incorporated into the TAMER system, a discussion of the various components is necessary. In designing the TAMER system, the goal was to minimize cost and complexity while maximizing performance and safety. To accomplish this, off-the-shelf equipment was selected that would be powerful enough to achieve the desired performance, with leftover capabilities to leave room for flexibility and future developments. The following hardware was deemed necessary:

- Case 621 front end loader with a two yard bucket
- Loader Control Computer (LCC)
- Operator Control Computer (OCC)
- hydraulic interface
- pneumatic interface
- RF communications link
- throttle controller
- remote controller units (stationary and portable)

The 621 is a diesel engine powered, hydraulically operated, four-wheel drive articulated front-end loader that has built-in options for automatic travel and dump height control, and automatic return to dig [7]. It is used extensively by Caltrans and others in the road maintenance and construction businesses.

3.1.1 LCC Computer

The most important component of the TAMER system is the LCC computer. This computer makes automatic decisions, realizes operator control movements, and provides loader status feedback to the operator. The engine chosen for this task was the A-Drive™ analog controller from TERN Inc. It was designed for industrial process control, automatic test equipment, and data acquisition applications that require many channels of analog signal I/Os [9]. It features 22 channels of 12-bit analog to digital conversion (ADC) inputs and 8 channels

of digital to analog conversion (DAC) outputs. Onboard is a V104™ that features a 16-bit 8MHz V25 CPU, 48 user-programmable I/O lines, three 16-bit timers, 3 serial ports, up to 512K battery backed-up SRAM, and a supervisor chip providing detection of power failure. Integrated with the A-Drive™ is an interface board with 24 solid state relays, to switch various loader functions. Figure 3.1 lists the I/O lines for the LCC.

	Description	Type	Legend
1	Throttle	AO	AO = Analog Output AI = Analog Input DI = Digital Input DO = Digital Output *go to transmission multiplexer, all else are outputs to solid state relays (SSR's) **RS-232 port 9600 baud rate
2	Steering Angle	AI	
3	Bucket Position	AI	
4	Arm Position	AI	
5	Neutral Sensor	DI	
6	Oil Pressure	DI	
7	AOK	DI	
8	Return to Dig Sensor	DI	
9	Travel Height Sensor	DI	
10	Dump Height Sensor	DI	
11	First Gear	DI	
12	Direction Neutral	DO*	
13	Remote Enable	DO*	
14	Clutch Engage	DO*	
15	Clutch Disengage	DO*	
16	Brake Level 0	DO	
17	Brake Level 1	DO	
18	Brake Level 2	DO	
19	Brake Level 3	DO	
20	Arm Up	DO	
21	Arm Down	DO	
22	Arm Stop	DO	
23	Arm Float	DO	
24	No Float	DO	
25	Roll Bucket Back	DO	
26	Dump Bucket Forward	DO	
27	Throttle Power	DO	
28	Steer Right	DO	
29	Steer Left	DO	
30	Steer 0	DO	
31	Emergency Stop	DO	
32	Transmission Enable	DO	
33	Engine Start	DO	
34	Communications	D I/O**	

Figure 3.1 LCC I/O Lines

It should be noted that the LCC computer provides many more analog and digital channels than are actually employed. This provides the system with the capability of being upgraded with more sophisticated or numerous sensor inputs. Also, extra outputs could be used to add on new components or adapt the computer control system to an entirely different machine.

3.1.2 OCC Computer

Interfacing with the LCC through a radio frequency (RF) modem link is the Operator Control Computer (OCC). The OCC has two major functions. The first is to take all the operator control manipulations and transmit them to the LCC. The second is to display to the operator the loader status on an LCD screen. The computer chosen to complete these tasks was a Sensor Watch™ by TERN, Inc.. The Sensor Watch™ is designed for data-acquisition and control applications. It features an analog signal conditioning circuit, eight channels of 12-bit ADC, seven channels of comparator inputs, three serial ports (RS-232, RS-485), one channel of 10-bit DAC, seven solenoid drivers, three 16-bit high speed counters, seven digital inputs, eight digital outputs, one power relay, a PDC (Portable Data Carrier) interface, 4x4 keypad, character or graphic LCD interface, beeper, LEDs, and on-board power supplies. The CPU is a 16-bit NEC V25 which runs at an 8MHz clock speed and is Intel 80x86 compatible [10]. The only output from the OCC is a data stream to the LCC through the RF modems. Figure 3.2 lists the I/O lines for the OCC. Like the LCC, the OCC also has spare digital and analog channels for future applications and advancements.

	Function	Type
1	Enable/Disable	Keyboard - DI
2	Engine Start	Keyboard - DI
3	Asterisk	Keyboard - DI
4	Record	Keyboard - DI
5	Play	Keyboard - DI
6	Travel Height	Keyboard - DI
7	Dump Height	Keyboard - DI
8	Rattle	Keyboard - DI
9	Float	Keyboard - DI
10	Return to Dig	Keyboard - DI
11	Gear 1	Switch - Keyboard - DI
12	Gear 2	Switch - Keyboard - DI
13	Forward - Neutral - Reverse	Switch - Keyboard - DI
14	Emergency Stop Switch	Switch - DI
15	Brake	Analog Input
16	Throttle	Analog Input
17	Steering	Analog Input
18	Joystick - Arm Control	Analog Input
19	Joystick - Bucket Control	Analog Input
20	Communications	RS-232 9600 Baud

Figure 3.2 OCC I/O Lines

3.1.3 RF Communications

As mentioned before, communications between the OCC and LCC is accomplished through an RF modem communication link connected to the respective computers' RS-232 ports. Safety concerns make secure communications of paramount importance teleoperating heavy earth moving equipment such as a front end loader. With this in mind, spread spectrum

technology RF modems (STI SpectraData Model 5500) were chosen. These modems are both transceivers, meaning they can both transmit as well as receive data. In order to eliminate the chance of interference or spurious signals, the modems talk to each other over a range of frequencies. Information is transmitted redundantly on different frequencies in an apparently random pattern, which only the modems can decrypt. In the rare case of interference with the signal, unexpected control of the loader will not result. Instead a non-compatible signal will be generated, causing the machine to be automatically stopped. Each modem communicates to its corresponding computer (LCC or OCC) through an RS-232 port at a 9600 baud rate.

3.1.4 Machine Interfaces

In order for the LCC to directly control the front end loader's various systems, two different interfaces were developed. For the loader's brakes, which are air pressure over hydraulic, a pneumatic interface was constructed. In normal, non-remote, operation the brakes are fully proportional with a theoretically infinite number of possible control levels. However, electronic duplication of this is impractical as well as unnecessary. Instead, the pneumatic interface has solenoid valves which allow the brakes to be applied at three preset levels (low, medium, full). These solenoid valves are switched by solid state relays (SSR's) in the LCC.

To control the loader's hydraulic system, another interface was created. The hydraulic system controls the arm and bucket motions, as well as the steering angle of the loader. Again, in normal operation the arm and bucket motion controls are fully proportional. However, by switching solenoid valves in the interface from SSR's in the LCC, these controls become fixed rate. This does not seem to reduce the level of control, requiring only an adjustment by the operator. Also, the rate can still be modified through the use of the throttle by keeping the engine revolutions up and thus, the hydraulic flow rate. Control of the steering angle is slightly more complicated. Solenoid valves are used to dictate in which direction the vehicle turns, and a potentiometer (measuring machine angle) provides feedback to the LCC to tell it when the appropriate direction is reached.

3.1.5 Throttle Controller

The most important single control component on the loader is the throttle. Besides determining the vehicle's speed, it also effects all the hydraulically controlled systems. Thus, it was decided that maintaining full proportional throttle control was necessary during teleoperation. For this purpose a separate controller and actuator was selected. The throttle controller takes an analog signal from the LCC and manipulates the actuator on the throttle linkage to the desired level.

3.1.6 Remote Operating Units

Two different remote operating units were created for teleoperation of the front end loader. The first is a sit down unit (referred as SROU in the previous report) that features a keyboard interface to the OCC, automotive steering wheel, steering column mounted shift

lever, joystick for bucket and arm control, pedals for brake and throttle control, and a 3-D video system.

Figure 3.3 Sit-Down Workstation.

The three major design goals for the sit down station were to increase operator familiarity, reduce fatigue, and integrate a 3-D video system that would not hinder direct line of sight operation. Operator familiarity is enhanced by the fact that the layout of the controls on the station accurately duplicates the front end loader's actual controls. Thus, an operator can quickly switch from manual to remote manipulation with minimal readjustment. Fatigue is reduced on the sit down station because the operator is no longer subjected to the noise, dust, and heavy jostling of the loader in manual operation.

In previous research it was determined that at increased operational distances (100'+) a video system was necessary as depth perception and field of view were compromised [13]. The current video system consists of a 15" main monitor that displays a 3-D image through the use of a Stereographics™ system. In addition, three 2-D views are provided on 6" active matrix LCD screens for peripheral and rear vision.

For close-in teleoperation and portability there is a second remote controller. This unit connects to a backpack and hangs comfortably over the operator's shoulders in front of the stomach. The challenge in designing this controller was that pedals were no longer an option.

Thus a unique throttle, brake, and steering control mechanism was constructed [7]. The portable unit also has a keyboard interface to the OCC, a joystick for bucket and arm operation, and a direction selector.

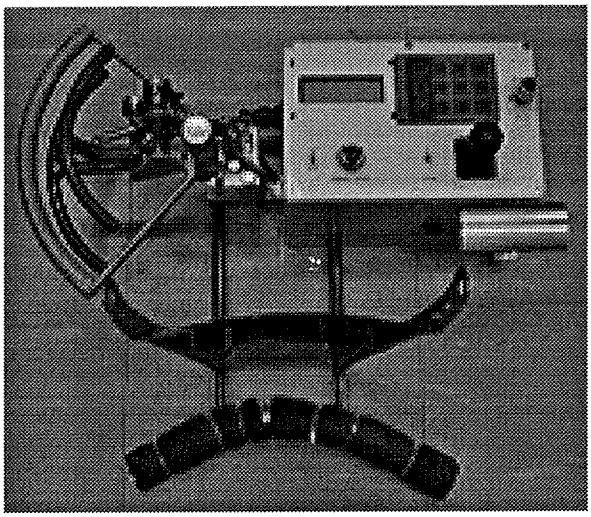


Figure 3.4 Portable Control Unit.

3.2 Software

In order to fully exploit the capabilities of an electronic system, the most advanced software available must be used. The computer industry has a long history of software development lagging behind hardware development. Since the TAMER system is a prototype, a flexible computer language that could handle unforeseen developments was needed. The A-Drive™ (LCC) and Sensorwatch™ (OCC) were chosen in Phase III of the TAMER project partially because they are C/C++ programmable microcontrollers. C is a middle level computer language that is flexible, easy to use, and widely known by many programmers. Typical robot programming languages could not be used on the TAMER system because they are not oriented towards a dedicated teleoperated mechanism. Also, RPL's are usually written with specific robots or classes of robots in mind. TAMER's high level of machine-man interface and variable operating environment are unique, and require a language such as C, which is versatile enough to grow with the system.

Three software packages are necessary for programming the LCC and OCC computers:

1. Borland C/C++ 4.0/3.1 (or Microsoft Visual C/C++)
2. Paradigm LOCATE, PDREMOTE/ROM, DEBUG/RT
3. TERN C libraries

The following block diagram illustrates how they interact [11]:

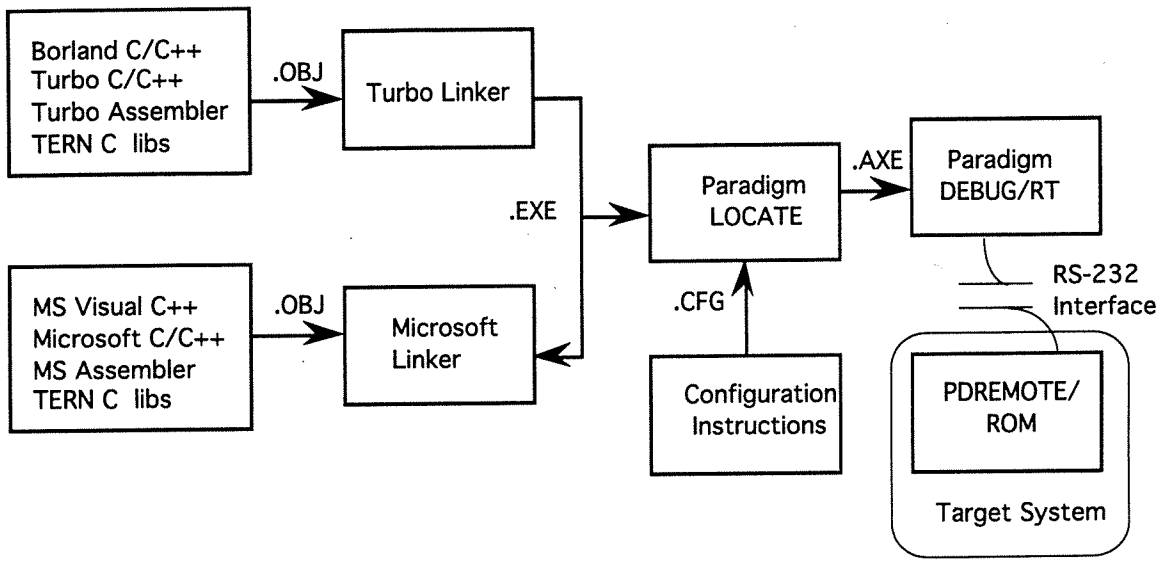


Figure 3.5 Software Utilities Interaction

(Borland C ver 3.1 was used to develop the program on a PC computer. The TERN C libraries allow the programmer to easily access the capabilities of the microcontroller, such as data acquisition functions or the switching of digital I/O lines. Paradigm's software utilities LOCATE, DEBUG/RT, and PDREMOTE/ROM allow the developer to transfer C code over to the target microcontroller. LOCATE is designed to interact with Borland and Microsoft language products, and is necessary to allow the use of DEBUG/RT. DEBUG/RT and PDREMOTE/ROM work together as a state-of-the-art, source-level debugger system designed to support embedded system applications designed around the popular 80x86 compatible microprocessor families from Intel, AMD, and NEC [11].

3.2.1 LCC Program

The main control program of the TAMER system is the LCC program. It performs a variety of important functions in determining the status of the loader, and realizing control inputs. The LCC keeps the loader in one of two modes: Standby and Remote. In Standby, the loader's brakes are held on and the transmission in neutral while the LCC awaits a signal from the OCC to go into Remote mode. In Remote mode, the teleoperator has active control of the loader. In addition to these general modes, the LCC also monitors communications and checks for valid command codes. If the communications check fails, the LCC program automatically puts the loader back into Standby.

Most importantly, the LCC program sends back status information to the OCC so that it can be displayed to the operator. For instance, when the loader is in Standby, the message "Loader in Standby" is shown, while "OCC in Control" is displayed during Remote mode. In the case where the operator suspects that the loader is behaving incorrectly, there is an

Emergency Stop switch on the OCC. When this is activated, the LCC puts the loader in Standby, and kills the engine for safety. A general block diagram of the LCC program is as follows:

3.2.2 OCC Program

While the OCC program does not make the important automatic decisions that the LCC program does, it performs several significant tasks. First, it takes all the control position information from the operator and converts it into a data string to be sent to the LCC. In the case of the analog inputs from the brakes, throttle, and steering potentiometers, the OCC program must also scale the data appropriately. The raw data from these sensors is converted to an 8-bit (0-255) scale. The slope and intercept necessary to scale each sensor is determined in a calibration mode. Thus, if a new sensor is installed or a sensor is adjusted, the new output voltage range can be handled by software, and there is no need for re-downloading the entire program. Since the RF modems transmit 4-bit numbers, the OCC program also breaks up all the 8-bit numbers into their 4-bit components prior to transmission.

The second major task of the OCC program is to display information to the operator on an LCD screen. These messages fall into three major categories: loader control status, teachable function status, and error messages. Loader control status can be either STANDBY or REMOTE. Teachable function status is either Normal (teachable function not being used), Record (session being stored in memory), or Playback (recorded operations being repeated). Error messages warn of communications problems or improper control settings.

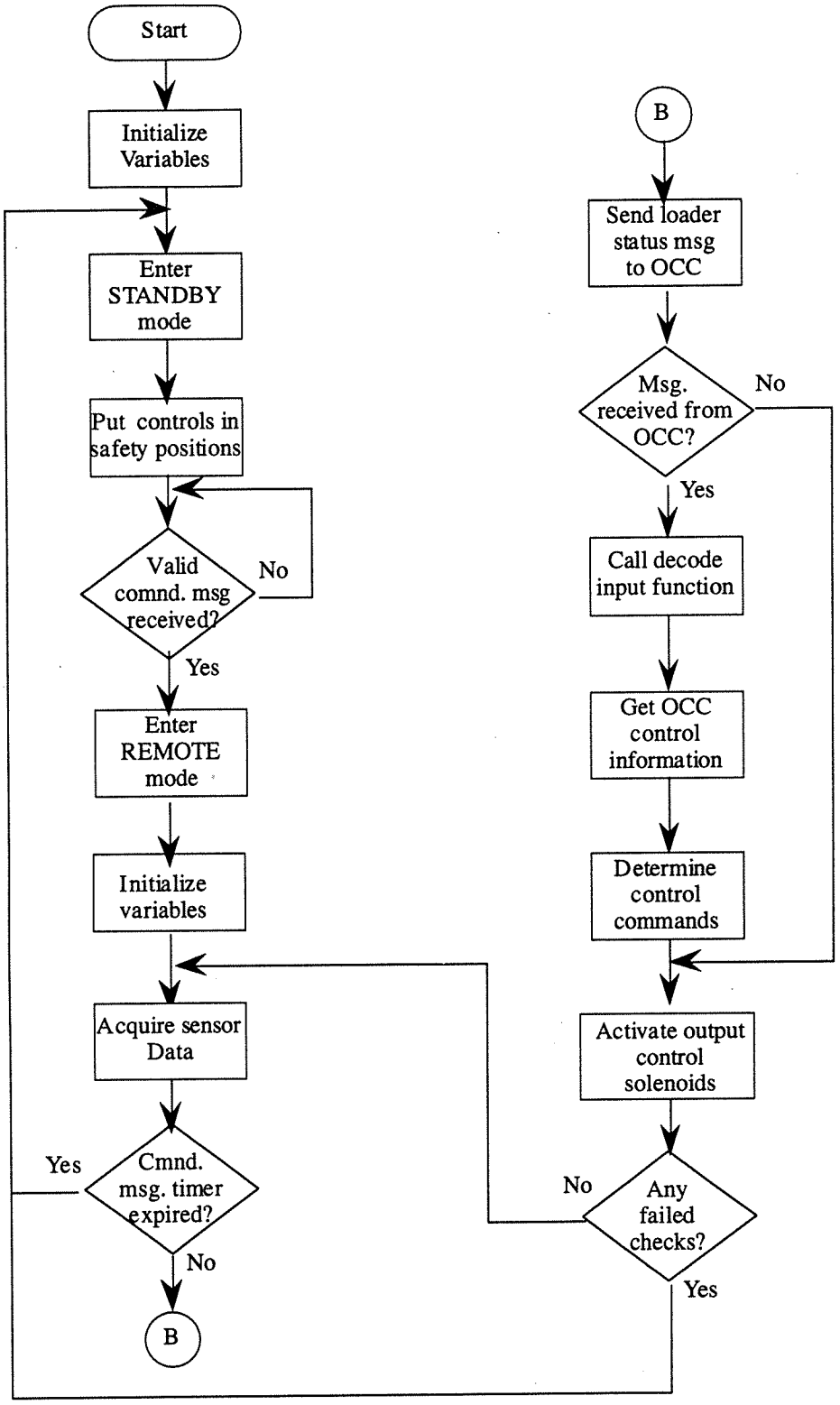


Figure 3.6 LCC Program Block Diagram

Chapter 4

Program Design

As mentioned in Chapter 1, this phase of the project involves the incorporation of "teachable functions" into the TAMER front-end loader's control programs. These functions were developed to improve productivity by faster operation and reduce operator fatigue. Important factors in evaluating any teachable program include accuracy of playback and ease of use. In order to fully explore all possibilities for the Record And Play (RAP) program, three different control strategies were attempted. The first strategy featured a *time-based* program, which was basically synchronous open loop control. The second strategy featured a *position-based* program that integrated sensor data feedback and asynchronous control. The third and final strategy was developed after both the first two programs had been created and tested; taking the best qualities of both.

4.1 TIME-BASED PROGRAM

The first "teachable function" program developed was a time-based one. The basic philosophy was that the program would watch and record how long in duration each control input was operated. Then, during synchronized playback, the program would execute the same control sequence for the same time durations. This program would not allow any feedback, either automatic or from the operator. The control method is open loop where the loader navigates by dead reckoning.

4.1.1 Time-based Recording

The record sequence for the time-based program is relatively simple. The operator initiates recording by merely pressing a button on the OCC keypad. The LCC enters record mode and starts a counter. It then sends a message back to the OCC indicating the change in status, so that the OCC can display "Recording!" on the LCD screen. The operator proceeds to manipulate the loader in the desired sequence, and terminates with a push of the same button to start recording.

The most difficult task for the time-based program is determining what information to store and when. Ideally, just enough information will be stored in order to complete the task, so that memory space can be conserved. Thus, the program watches the OCC controls for operator changes, and only stores new information when changes are observed. Since many of the controls are different, detection ranges had to be set for each control. When a change is detected, all the current positions of the OCC controls are assembled into a state vector along with the current counter value. These state vectors together make up a recording array. Each state vector is composed of eight parts, representing individual control positions.

1. Steering
2. Brakes
3. Throttle
4. Arm
5. Bucket
6. Automatic function keys
7. Automatic function keys
8. Counter value

The recording can be terminated three different ways. When the Start/Stop key is pressed. When the counter reaches a preset limit which, for the purpose of this project, was set at 90 seconds. (However, this limit is constrained only by the amount of memory the computer has.) And when the number of state vectors reaches a preset limit. (This limit was introduced, again, with memory usage in mind.) A flowchart of how the record mode works in the time-based program is as shown in Figure 4.1.

4.1.2 Time-based Playback

The playback sequence is also keypress initiated. When this is done, the LCC enters playback mode and sends a message to the OCC, causing a "Playback" message to be displayed to the operator. The program then starts a counter and begins substituting in the recorded control values from the state vector array. The state vector is incremented when the counter value exceeds the recorded counter value. Thus, the loader operates according to the state vector for the same amount of time that was recorded. The playback is terminated when the program successfully reaches the recorded maximum counter value, or when the Start/Stop key is pressed.

An obvious drawback to the time-based program is that the operator has no way of adjusting the playback if it becomes inaccurate. Also, the program itself does not have any position feedback to tell it if the loader is playing back correctly. The only recourse the operator has is to terminate or discontinue the playback entirely, and assume normal teleoperated control. A flowchart of how the playback works in the time based program is as shown in Figure 4.2.

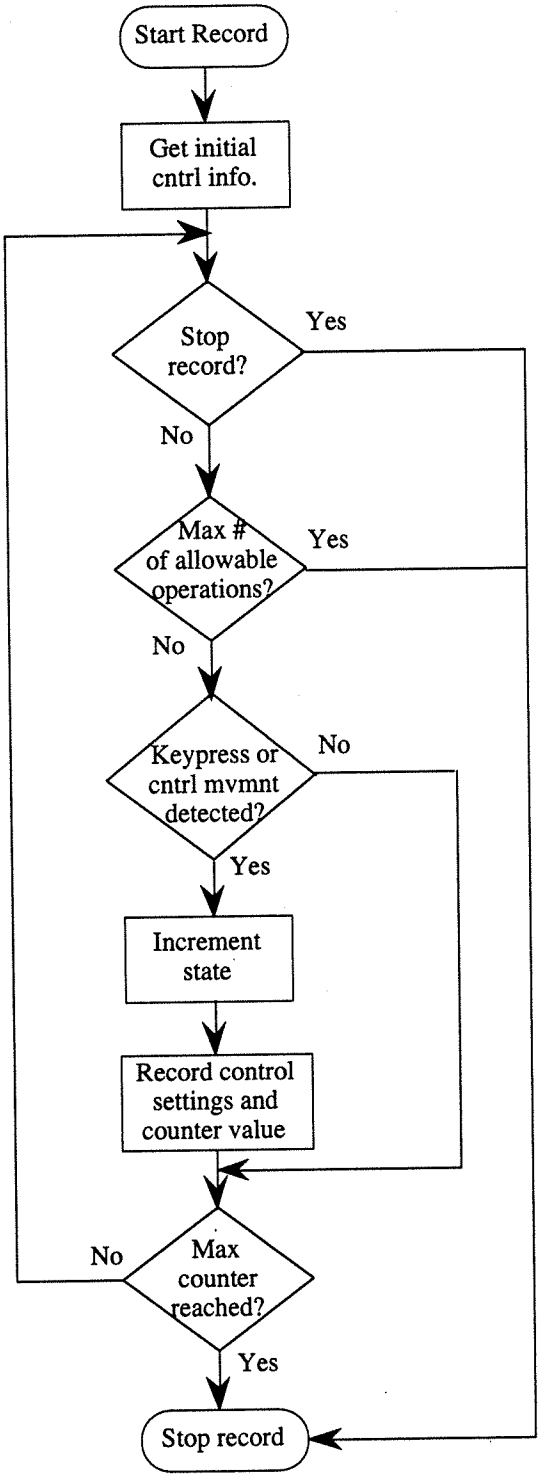


Figure 4.1 Time Based Flowchart - Record.

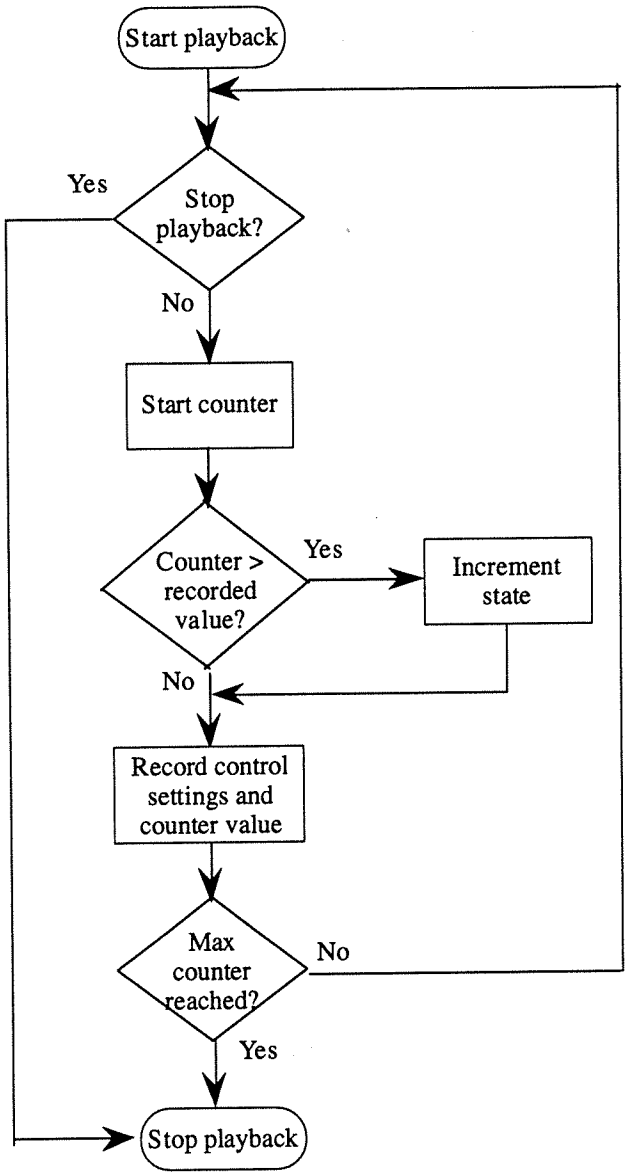


Figure 4.2 Time Based Flowchart - Playback.

4.1.3 Preliminary Testing and Evaluation

Initial testing of the time-based program revealed that it performed as intended: the operator was able to record and playback relatively complex sequences involving all the different loader functions. The playback was shown to accurately repeat functions for the same amounts of time, and in a smooth manner indistinguishable from an actual operator. A sample sequence is as follows:

1. Move the loader forward (performed w/ steering changes)
2. Lift the bucket to dump height
3. Dump the bucket
4. Return the bucket and arm to travel conditions
5. Back the loader up (performed w/ steering changes)

However, the limitations of open-loop control soon became evident. In order to reproduce the playback accurately, it was necessary to re-position the loader in exactly the configuration it was in at the start of the recording. Also, different weights of dirt would cause subsequent playbacks to differ, due to the variation in load. So, even though the program was performing the playback in the correct sequence and amount of time, the result was not necessarily what the operator intended.

4.2 POSITION-BASED PROGRAM

A position-based program was developed as an alternate approach to the time-based. The goal of this program was to introduce feedback control in the form of position sensors on the loader. To accomplish this, a rotary potentiometer was mounted on the bucket, and a linear potentiometer was mounted on the arm of the loader. Together with the existing steering sensor, the LCC program could now monitor steering angle, bucket angle, and arm position. Now, instead of monitoring OCC control positions as was done in the time-based program, the program looks at the actual positions of the components on the loader. Then, during playback, the program can manipulate the loader to move between those positions according to an asynchronous control algorithm known as a *finite state machine* [12]. A finite state machine runs according to the guiding principle that the system can only advance to the next future state when all conditions for the current state are met, independent of the time it takes.

The position-based program also introduced the concept of *trimming* the playback. Trimming allows the operator to make small modifications to the recorded sequence while it is playing back. This may be necessary when boundary conditions change. If a recorded sequence was lifting and dumping into a truck, trimming would allow the operator to change the playback so that different truck side heights could be accommodated, without completely

re-recording. Since the operator is now only required to make minor corrections, his task difficulty is greatly reduced.

However, the position-based program was designed to only record bucket, arm, and steering movements. All of the other controls, such as brakes and throttle, did not lend themselves to simple position recording. Also, the position of the loader would require a sophisticated Global Positioning System (GPS), or wheel speed sensors (not accurate). Ideally, knowledge of the loader and the position all of its components would allow fully automatic control. That was determined to be beyond the scope of this project, which has as its real goal to develop a human-machine interactive teachable function.

4.2.1 Position-based Recording

Record mode begins, as in the time-based program, when the operator pressed the Start/Stop button. Once in record mode, the LCC starts a counter and begins storing information from the position sensors in state vectors. Each state vector is comprised of data from the steering, bucket, and arm potentiometers. A new state is stored after a specified time interval of one second. (The one-second interval was chosen after testing of shorter periods resulted in jerky playbacks.) All of the state vectors together comprise the record array. Record is terminated when the operator presses the Start/Stop button or by a specified time limit of 90 seconds. A flowchart of the recording process is as shown in Figure 4.3.

4.2.2 Position-based Playback

The first task the LCC performs upon entering the playback mode is to examine the current positions of the loader components (steering, bucket, arm). Then the program looks at the first state vector from the recorded array and determines what control inputs are necessary to move from the current loader state to the recorded one. Those inputs are carried out while the program monitors whether the next state has been reached or not. When all the components have satisfied the next state's requirements, the process is repeated. Note that the various components move from state to state at their own individual rates, depending on loading and environmental conditions. For example, steering might be reaching each state more quickly than other components, resulting in a delay as the steering waits for the other components to catch up. The result is jerkiness in the playback. This can be minimized by choosing an appropriate time interval to store the information.

Although the position-based program was originally intended to move from state to state based strictly upon satisfying the position criteria, time proved to be an important factor. If a recorded sequence of operations had a period where none of the recorded components were changing, it might not be desirable to neglect that period. For instance, if the loader is recorded moving forward in a straight line with the arm and bucket in constant positions, the recorded states do not change. Thus, when the playback reaches a state it automatically looks ahead to the next state to see if changes occur. If not, the playback is paused for an equal amount of time to the recorded time interval between states. In effect, the position-based playback utilizes

asynchronous control for micro-motions (bucket and arm movements) and synchronous control for macro-movements (navigation under direct teleoperation).

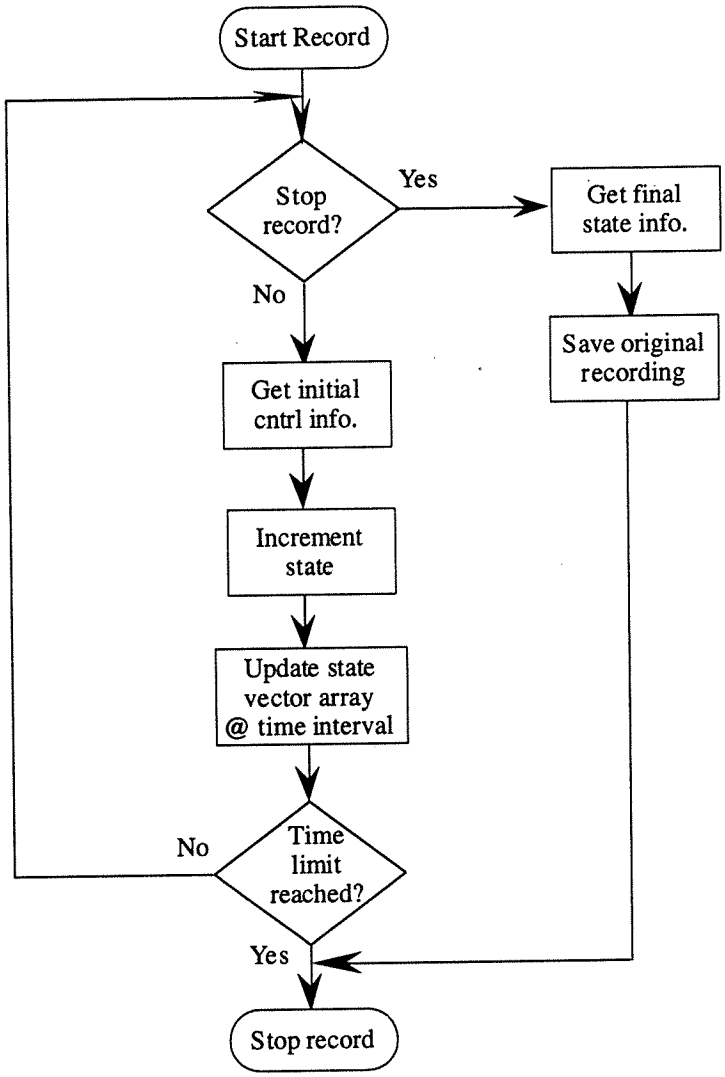


Figure 4.3 Position Based Flowchart - Record.

If the operator observes that the playback needs to be adjusted due to changing boundary conditions, etc. , he has the option of trimming. Trimming is accomplished by manipulating the OCC control for the desired component to be modified. For instance, if the operator wanted the arm of the machine to be a little higher, he would use the bucket/arm joystick control. The program detects the operator's control movement and pauses the

playback while the trimming operation takes place. When the control is returned to its neutral position, the program assumes the trimming operation is complete and calculates the amount the component was moved. This *trim value* is then applied to all the following states. If the downstream state data was not modified, then the machine would merely return to the original playback instantly and negate the purpose of trimming in the first place. State component values are only modified up to their range limits so that incorrect operation does not occur. For example, if the arm is trimmed upward 100 points on a 0-255 scale, all subsequent states are also increased by 100. If the resultant sum is greater than 255, then it is set to 255 (and vice versa for 0).

One concern with trimming was that if the operator needed to make a lot of corrections, the recording would become too corrupted. To compensate, it was decided to restore the original recording, without the trimming corrections, at the end of any playback. This control policy makes each playback independent of previous actions. Thus, if subsequent playbacks require the same trim at the same time in the playback sequence, the operator may decide to re-record the states to minimize the needed trim. Of course, he can continue operation and trimming if he wishes. Each playback is terminated when the Start/Stop button is pressed or when the final recorded state is reached. The following flowchart depicts the operation of playback for the position-based program:

4.2.3 Preliminary Testing and Evaluation

Initial testing of the position-based program showed that it performed as intended: the machine moved through the recorded states and allowed trimming operations. Also, planned periods where none of the recorded components were moving were reproduced accurately. The testing did, however, reveal the importance of the choice of recording time interval. Short intervals (.5 seconds) were tried with the idea of reducing discrete error, but resulted in very jerky playbacks due to the closeness of states. Long intervals (2+ seconds) were also tried, but resulted in unacceptable discrete error and problems due to the varying movement rates of the different components.

The main limitation of the position-based program is that it does not record all the loader components (for reasons mentioned in sec. 4.2). However, the simultaneous meshing of active human control and recorded sequences may be the most viable option for real field use.

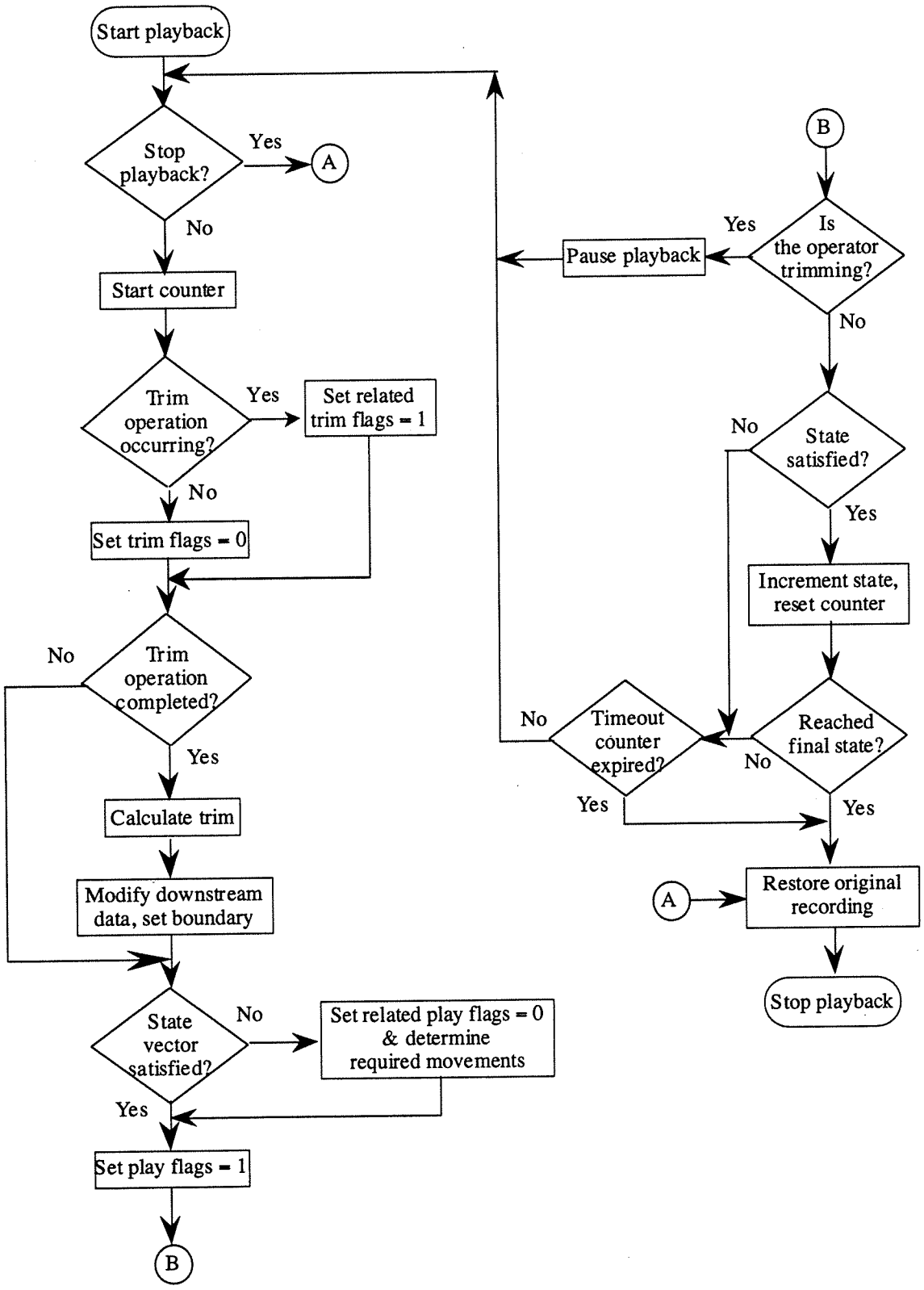


Figure 4.4 Position Based Flowchart - Playback.

4.3 Hybrid Program

The third program developed for teachable functions incorporates characteristics from both the time- and position-based programs. The goal was to create a control program that would take the most successful methods from previous attempts and meld them together. Essentially, the "hybrid" program copies the time-based control structure, but with two important additions from the position-based program. First, the hybrid program records initial position data from the sensors on the machine so the machine's components can be properly arranged at the start of a playback session. Second, the hybrid program allows feedback correction (trimming) by the human operator for all recorded controls. The hybrid program was based more upon the time-based program than the position-based program mainly because of the time-based program's ability to control all loader functions.

The initial positioning capability was added for several reasons. It added greatly to the playback's convenience by freeing the operator from having to accurately position all the loader components (bucket, arm, steering angle) at the start of the playback. Also, initial positioning was added with the intent to improve the accuracy of playback. Preliminary testing of the time-based program revealed that the replication of the starting position of the machine was crucial if the recorded sequence was to be duplicated. The starting point in any open-loop motion controlled process is important, since any initial errors will be propagated as the process is executed.

Preliminary testing also revealed the benefits of a method of feedback. However, because the time-based control structure does not allow for the correction of position as the position-based program does, a new method of trimming was developed. In the new method, the operator can override and correct the recorded controls individually. The program substitutes the desired control corrections in for the recorded ones, and does not modify future control state data.

An automatic position control system was rejected because of the manner in which teleoperation of the hydraulic controls works. As mentioned in Ch. 3, the bucket, arm, and steering operations of the loader are fixed rate, except when engine speed is changed. Thus, if the program detected that the component was falling behind in position during playback, it would be unable to make it move faster to compensate unless throttle was increased. However, the throttle change would impact all other loader components as well. It should be noted that this problem is only a factor during a playback where control states are determined by time (e.g., time-based). The "finite state machine" concept employed in the position-based program, by its very nature, obviates the need for position correction since each state vector must be completed before continuing.

4.3.1 Hybrid Program Recording

As mentioned in Sec. 4.3, the hybrid program borrows heavily from the time-based program's methods. This is especially true in the record mode. Again, the process is initiated

by pressing the Start/Stop button on the OCC, and the operator moving the loader through the desired sequence. While the movements are occurring, the LCC stores the OCC control positions in a state vector array, with each vector consisting of all the recorded controls and a timer counter value. New state vectors are stored only when changes in the operator's movement of controls are detected in order to minimize unnecessary use of memory. The hybrid program additionally records the data from the arm position, bucket position, and steering angle sensors at the start of the recording. The record process is terminated by either pressing the Start/Stop button, time limit, or state vector number limit. A flowchart of the hybrid program's record mode is as in Figure 4.5.

4.3.2 Hybrid Program Playback

Like the other programs, the hybrid's playback mode is started by pressing a key on the OCC. Before starting a counter and progressing with the playback, however, the program enters an initial positioning mode. In this mode, the program first looks at the position data for the bucket, arm, and steering that was stored at the start of the recording. This data is then compared to the current component positions so that the required movements to get to the initial position can be determined. Once the movements are executed and the initial position state of all three components is satisfied, the program enters normal playback mode.

In the normal playback mode, the hybrid program acts like the time-based and substitutes the recorded control position states for actual movements of controls made by an operator. Each state is kept until the time duration with which it was recorded elapses. However, the hybrid program also allows the operator to trim each control individually, so that the human provides continuous position feedback. This is accomplished in different manners for each control.

The arm and bucket controls are fixed rate and not proportional. They are either on or off. Thus, if the program detects movement on the OCC controls by the operator (i.e. a trimming operation is occurring) it uses the current OCC control value instead of the recorded one. Once the control is put back in its neutral position, the recorded values for that control are used again.

The throttle and brakes are more proportional controls in that they have many possible levels. Again, the program watches the OCC controls during playback to see if trimming is desired. However, the current OCC control value is used only if it exceeds the recorded brake or throttle values.

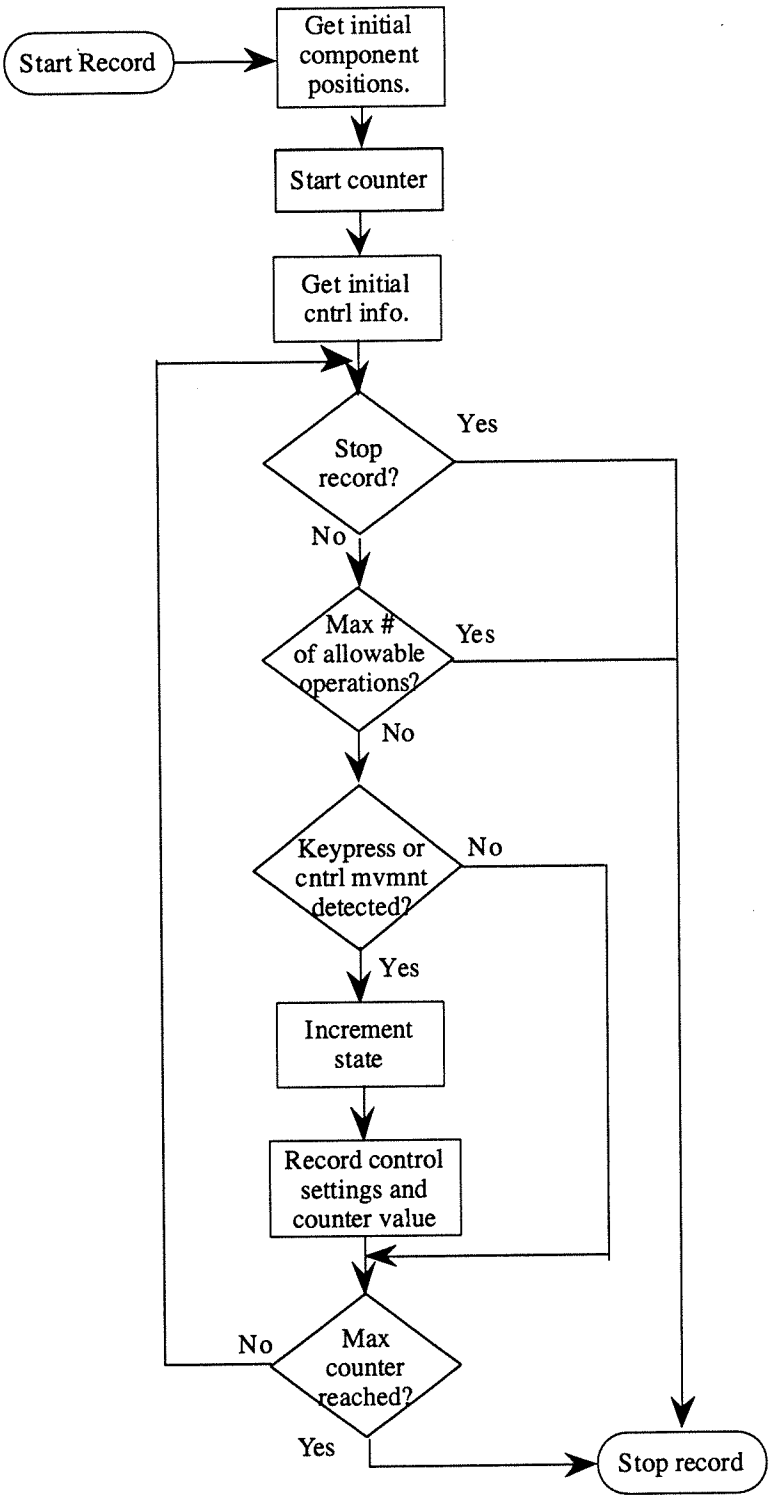


Figure 4.5 Hybrid RAP Program - Record.

The last control that allows trimming is steering. Steering is very different in that it is fixed rate and based upon position. Input from the OCC's steering potentiometer determines what angle the loader will adopt. So, in order to adjust the angle of the machine during playback, the range of the steering control on the OCC is modified. Instead of the normal full range of values, the steering sensor data is set into 3 regions; left, right and neutral. If no corrections to the steering are desired, the control is left in neutral. However, if an adjustment is needed, the operator moves the control in the desired direction, left or right. When the program detects the operator's movement of the control, it interprets it as either a full left or full right turn and starts moving the machine. When enough correction is accomplished, the operator moves the steering control back into the neutral position and the program restores the machine to the recorded steering angle. While the recorded control data has not been changed, the path of the machine has.

Playback for the hybrid program terminates when the final counter value is reached, or when the operator presses the playback button a second time. A flowchart of how the hybrid program's playback works is as shown in Figure 4.6.

4.3.3 Preliminary Testing and Evaluation

Initial testing of the hybrid program showed that it performed all functions as intended. It properly executed the initial positioning phase and proceeded into normal playback. During normal playback, all the loader functions were able to be trimmed as they were designed to be. Variable recorded sequences were all repeated properly in the correct amount of time.

In testing it was immediately apparent that initial positioning is a great benefit. The operator no longer had to worry as much about how the loader was positioned, because this was done automatically. Also, because the automatic initial positioning was more accurate than initial positioning dependent upon the operator's memory, the resultant playback was more consistent than the time-based program's. The addition of trimming capability aided playback accuracy as well. This form of feedback by the operator is essential to the concept of semi-automatic teleoperation, where the human operator remains an integral part of the control loop. However, the operator's tasks are relieved so that less concentration and action is required.

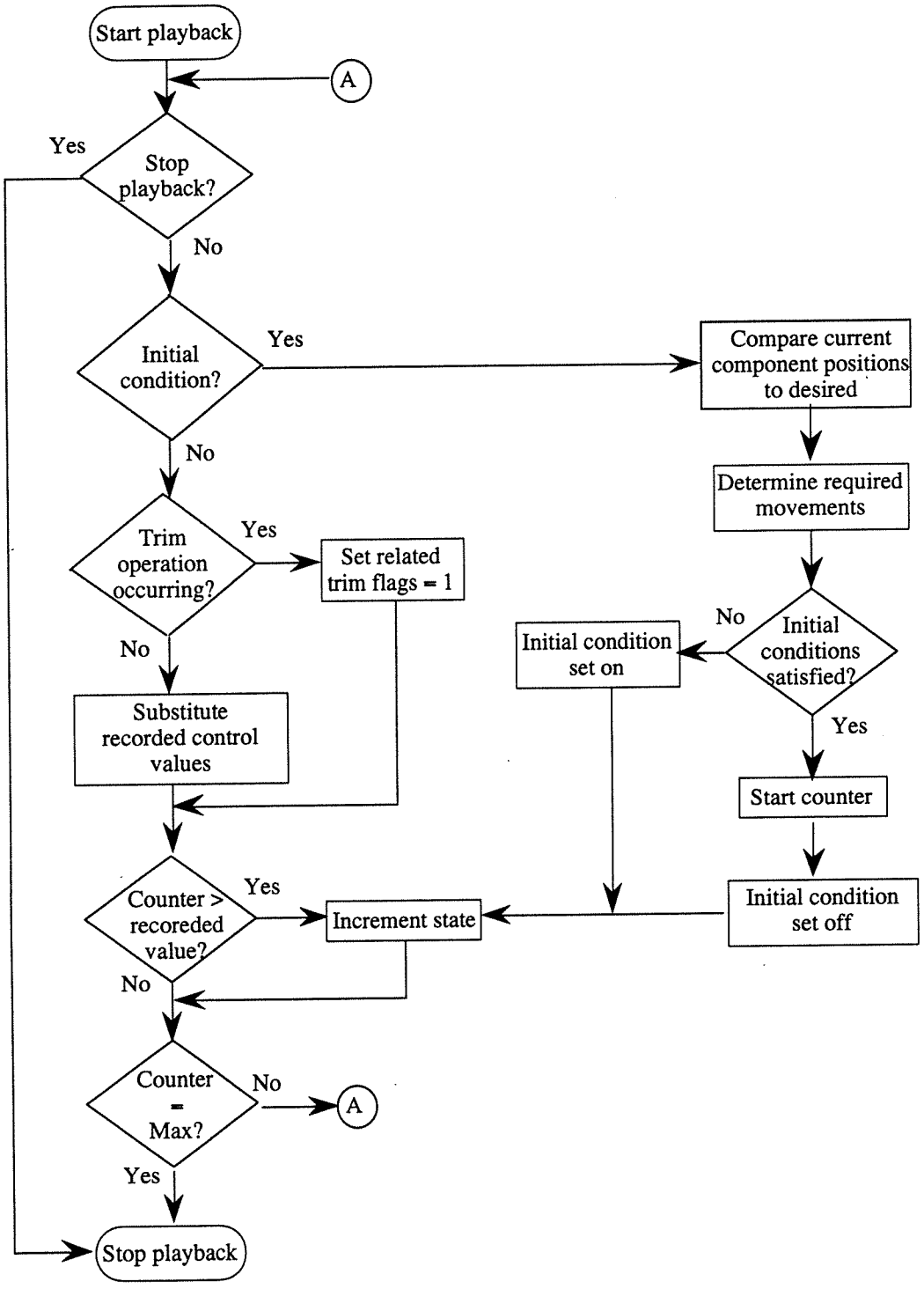


Figure 4.6 Hybrid RAP Flowchart - Playback.

Chapter 5

Testing and Results

The motivating goal of the TAMER project is to remove the operator of a front end loader from a possible hazardous environment. This was accomplished through the construction of a sophisticated teleoperation system. Because teleoperation imposes different tasks and demands on an operator, the teachable functions were incorporated into the loader's control programs to increase productivity and decrease the operator's fatigue. Three different control programs were developed to realize these goals.

In order to compare the different control programs, a test was developed that would be sufficiently difficult to reveal limitations, but easy enough to generate consistent data. The test was to meet the following objectives:

- Demonstrate the possibility of teachable functions within the framework of the TAMER system.
- Evaluate the most accurate and easiest to use control method.
- Demonstrate the benefit of having a man-machine interactive feedback in semi-automatic functions.
- Identify which (if not all) loader components should be involved in teachable functions.

5.1 TEST LAYOUT AND PROCEDURE

The test was designed to require the use of all loader components, movements, and possible operational sequences, so that real-world application could be seen. The site chosen for the test was a field on UC Davis grounds, with the following layout.

In order to involve all the loader functions in a continuous loop test that could be repeated easily, the following sequence was selected. (Steps correspond to numbers on Fig 5.1.):

1. Position the loader at the start line; facing the pile of dirt and square to the line between the double cones.
2. Move forward into the pile of dirt and attempt to acquire a full load in the bucket.
3. Back up, holding the load at a safe travel height, and turn to face left towards dump area.
4. Pull forward to dump area; lift, and dump load.

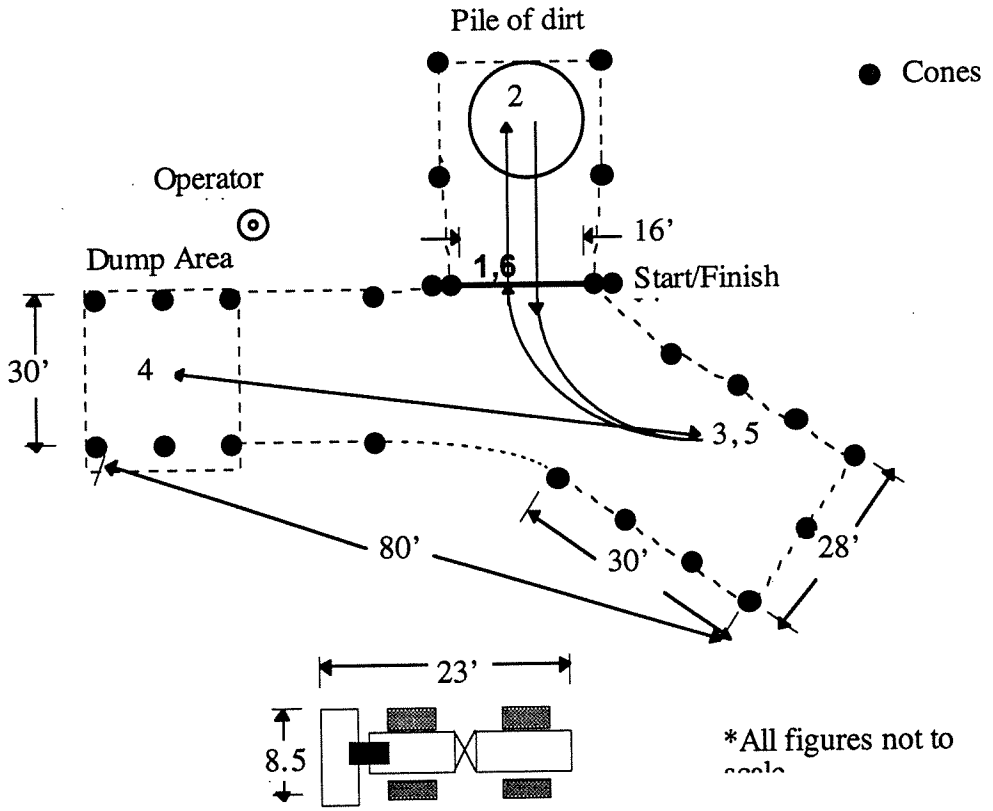


Figure 5.1 Test Layout

6. Move forward and turn so that loader has returned to start position and orientation; stop.

To complete a single test, the loader performed this sequence twice. The loader was restrained to first gear only, in order to keep the data consistent and promote safety. Also, five different modes of operation were tried:

1. Manual
2. Normal teleoperation
3. Time-based
4. Position-based
5. Hybrid

In manual mode the operator is physically on the machine, with no teleoperation or teachable function capability. In normal teleoperation mode the operator manipulating the

loader via remote control and also with no teachable function capability. All teleoperation during testing was done with the backpack portable unit.

In the case of time-based, position-based, and hybrid modes, there were a few more considerations. In these modes the operator would first record half of a complete test (one complete loop of starting and finishing at the same line). Then the actual test would be performed with the previously made recording for both loops. This meant that at the halfway point of the test, the operator had to position (if necessary) and restart a new playback. The time-based mode (3) used the time-based program with open-loop control and the ability to record and play all functions of the loader. The position-based mode (4) used the position-based program that used feedback from the sensors. However, this mode only operated on the bucket and arm functions. All other operations (throttle, direction, steering, etc.) had to be performed by the operator just as in normal teleoperation mode. The hybrid mode (5) used the hybrid program, which was essentially the time-based program with the added initial position and trimming capabilities.

A large variety of data was collected during each test, so that the different control methods could be analyzed for their benefits and limitations. The following variables were recorded:

1. Total time from start to finish of the test.
2. Amount of dirt transported in percentage of full loads.
3. Amount of cones knocked down or contacted.
4. Accuracy of dumping.
5. Final position accuracy.
6. Record time (if applicable).
7. Number of trimming operations (if applicable).

The total time for the loader to complete two loops was recorded in seconds. The clock was started when the operator first moved the loader and stopped when a satisfactory final position was reached at the start line. The amount of dirt transported was recorded in percentages of full loads. (For one test, the maximum amount of dirt transported that could be achieved would be 200%). The amount of cones knocked down or contacted was recorded for the complete test. If the same cone was hit more than once because the test is two loops, then it was counted each time.

The accuracy of dumping the second load on top of the first was determined by a 1-5 scale where:

1. Way off. Second load not in vicinity of first.
2. Two loads are in vicinity of each other (within 10').
3. The loads are at least partially overlapping.
4. The second load is at least half on the first (50%+ overlap).
5. Near perfect dumping of the second load on top of the first (90%+ overlap).

It should be noted that this scale does not take into account whether the loads are dumped within the dump area or not; it merely indicates how well the test is reproduced.

The final position accuracy was applicable only to those modes that used teachable functions (3-5). It refers to the ability of the playback to place the loader back at the start position and is also ranked on a 1-5 scale where:

1. Way off. Loader is not even close to starting position, and lots of correction was necessary between playbacks.
2. Orientation off (not square to start line) and within 5' of start line.
3. Within 5' of start line, and orientation is on.
4. Within 1' of start line, but orientation is off.
5. Within 1' of start line, orientation is on.

This quantity was recorded twice; once for each loop during the test.

The record time was kept for the amount of time in seconds that the operator took when completing the pre-test recording (one loop). This involved transporting one load of dirt and making sure the loader finished in a satisfactory final position (5 on the final position scale).

The last value to be recorded was the number of trims during the test. This quantity was only relevant for the hybrid and position-based programs as they are the ones that have trimming capability. It reflects the level of feedback necessary from the operator to keep the playback accurate.

The terrain the test was layed out on was an uneven dirt field with some slight elevation changes. This site was chosen because it represented fairly the environment a front-end loader often operates in.

The cones used to delineate the area in which the loader was allowed to operate were large orange, 10-pound traffic cones. They were chosen because they were easily visible to the operator and durable enough to be run over by a 14-ton loader. Spacing in between the cones

and the size of the operating environment were determined by considering the loader's extreme size and allowing it room to maneuver.

The size of the dump area was set at approximately 30' x 30' to satisfy two objectives. First, it was desired that the space be large enough for the operator to be able to easily dump there. Too small an area might "guide" the operator to dump in a specific spot, when part of the purpose of the test is to promote variability. Second, too large an area would not define the test well enough.

The pile of dirt constructed for the testing was approximately 20 cubic yards. A smaller pile of dirt would be too susceptible to the loader just pushing through it, because of its insufficient mass. The large pile assured uniformity and consistency for the tests.

As can be seen in Fig. 5.1, the operator for the testing occupied a strategic position that would allow a clear view of the loader during all phases of the test. This also helped contribute to consistent testing. Testing of other systems on the loader demonstrated that the learning curve for operating all this equipment was significant and unquantifiable. When novice operators were used, their performance often changed drastically from one test to another as they became more proficient with the system, and did not necessarily have anything to do with the different conditions they were supposed to be testing. Thus, to help minimizing the effect of the learning curve, all tests were run by the team member who developed the teachable programs and also is an experienced loader operator.

5.2 RESULTS

The results for the tests mostly turned out following expected trends. The time-based (TB) program behaved poorly, but this was not surprising considering it does not involve any correctional feedback. The hybrid program (RAP), with its initial positioning and "human-in-the-loop" feedback, was dramatically better than the TB program. However, the RAP program was also proved to have serious limitations. The position-based (PB) program probably had the most interesting results of all even though it was only operating on bucket and arm controls. The steering control was removed from the PB program for final testing after preliminary testing showed it to be unfeasible.

Figure 5.2 shows the averaged results for each program.

	Normal	TB	RAP	PB	Manual
Total Time (secs)	149.67	189.14	154.6	111.67	120.67
Record Time (secs)	N/A	69.86	62.6	58 .67	N/A
# Cones Hit	0	4.14	.6	0	0
Load Size 1 (%full)	100	92.86	100	98.33	100
Load Size 2 (%full)	100	78.57	93	91.67	100
Dump Accuracy	4.67	2.57	4	5	5
Final Pos. Accuracy 1	5	3.71	3.4	5	5
Final Pos. Accuracy 2	5	2.86	3	5	5
# Trim Operations	N/A	N/A	6	1.33	N/A
Time Differential	N/A	49	29	-5.7	N/A
Composite Score	.943	.376	.627	.949	.97

Figure 5.3 Test Results

The total time figures are somewhat misleading in their significance. However, some gross observations can be made. In comparing them, it should be noted that the order in which the modes are listed above is the order in which they were tested. While using one experienced operator helped eliminate the steep learning curve associated with novices, there is still some impact when the test is repeated many times and the operator becomes more familiar with the procedure, terrain, etc.

Three conclusions can be drawn from the time data. First, the TB and RAP programs take significantly longer to perform the test than other modes. This is due to the fact that the total time for these modes includes maneuvering time between the two playbacks. If the program did not return the loader to the starting line at the end of the first playback, the operator had to take time to position the loader for the start of the second playback. The amount of correction time varied with each of the teachable function programs as seen in Figure 5.4.

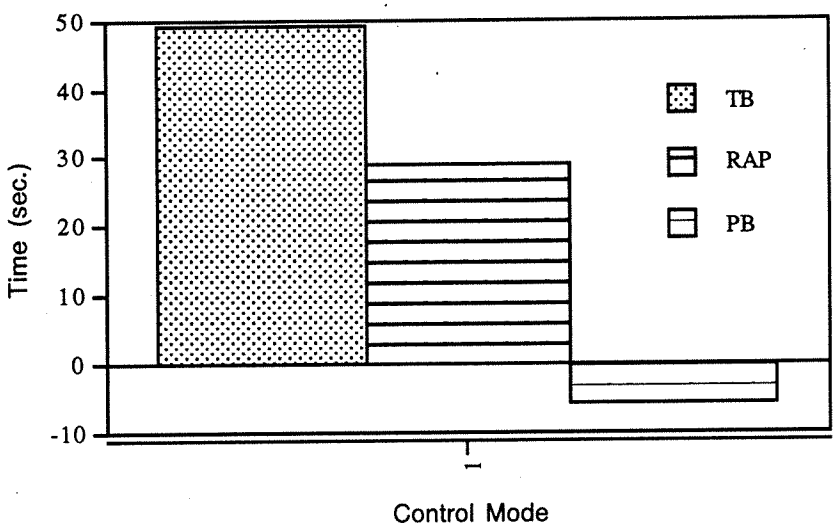


Figure 5.4 Correction Time

As this graph shows, the TB program required the most correction time in between playbacks and to position the loader properly at the end of the test. The RAP program showed significant improvement over the TB, demonstrating the benefits of initial positioning and trimming. The initial positioning was especially crucial to reducing the correction time because, even if the loader's final position accuracy was the same for the TB and RAP, the latter required far less time to put the arm, bucket, and steering angle in acceptable positions. Also, even if the loader's global position was not perfect, the RAP program allowed the operator to adjust during the playback through trimming, whereas with the TB program the operator had to take care that everything was as perfect as possible when initiating a new playback.

Unlike TB and RAP, the PB program consistently saved time during the test. Since the PB program was only controlling bucket and arm operations, the operator had full normal control of navigating the loader. Thus, when the loader was performing the dump sequence, the operator could start moving as soon as the dirt was out of the bucket without concentrating on the bucket and arm movements. In fact, the operator consistently moved the loader to the dump area so fast that he had to wait for the dump sequence to begin. If the load acquisition and dumping had been two different recordings activated by separate keys, even more time might have been saved.

The last conclusion to be drawn from the time data is that the manual operation of the loader appeared to be slower than the PB controlled. The reason for this is twofold. First, when the operator is not physically on the loader he is less subject to the bouncing around from rough terrain and is more likely to go faster. Second, the PB program required less concentration on the part of the operator, because it was handling the bucket and arm

movements. It was easier to move the loader quickly when the operator had fewer objects to pay close attention to.

The number of cones knocked down or contacted during the test really showed the importance of feedback. As expected, all three of the non-automatic navigation modes (Normal, Manual, PB) avoided cones entirely. However, there was a large difference between the TB and RAP programs. The TB program knocked down significantly more cones than the RAP program, because it does not offer any form of feedback correction. With the RAP program the operator could observe the impending cone contact and adjust the loader's path to avoid them. This again illustrates the benefit of having human-interactive feedback control.

As mentioned in Section 5.1, the dirt pile was sized such that acquiring a full load of dirt should have been easy. However, as Fig. 5.5 shows, this is not necessarily the case.

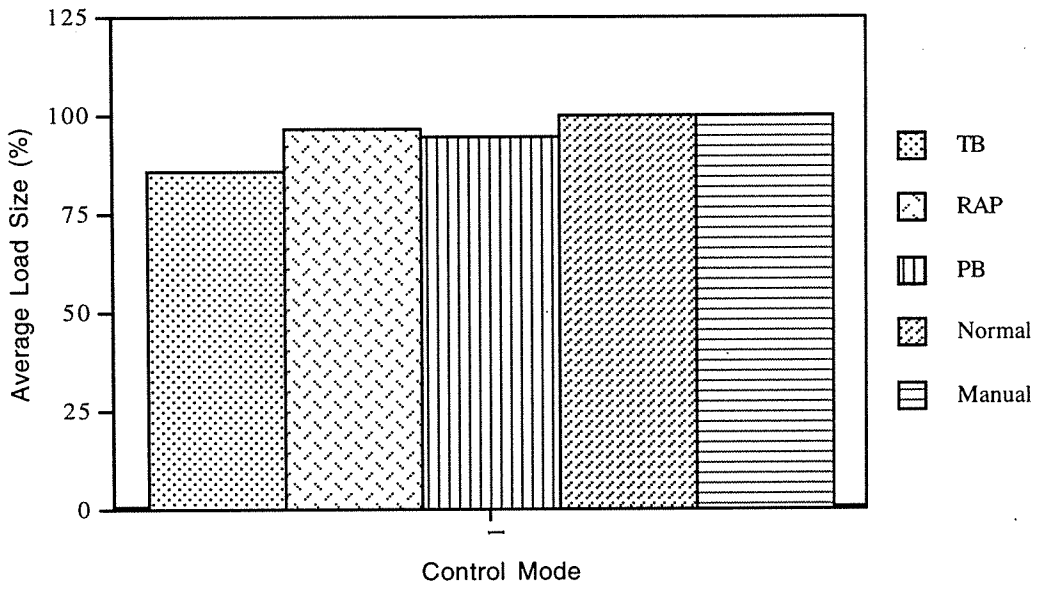


Figure 5.5 Load Acquisition Data

Again, the normal and manual modes performed perfectly as expected. The three teachable function modes did encounter problems. The primary problem was that the second load size consistently decreased. This was due to the fact that the original recording and the first playback were performed with a similar pile, while the second playback had to deal with a pile modified by one load of dirt already removed. The TB program, with no feedback, was affected the most by this. Also, during at least one test, the TB program procured a full load, but then did not dump it out completely at the dump site. The RAP program performed better because the operator was able to trim the arm and bucket while acquiring the load. While the PB program appeared to also suffer in performance, on the day it was tested the pile was

smaller (courtesy of UC Davis Yard Services), making it more difficult to get a full load. If the pile had been full size, the results likely would have been close to the normal teleoperation's.

The accuracy of the dump measured the ability of the control mode to accurately place the second load of dirt on top of the first. Figure 5.6 shows the results.

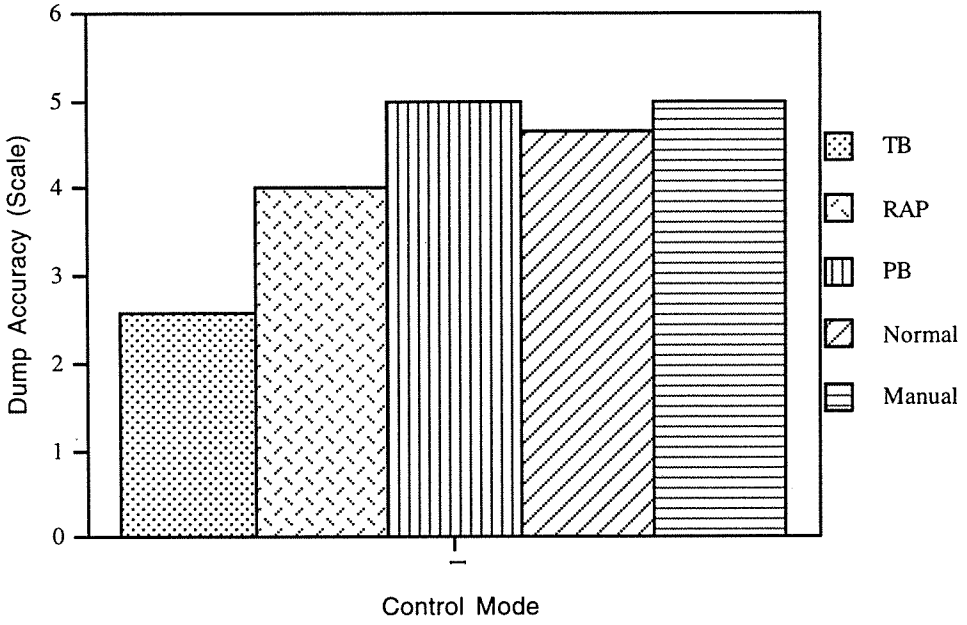


Figure 5.6 Accuracy of Dump Data

This figure clearly shows a trend of increasing accuracy with increased levels of feedback. Also, it is clear that all the modes that do not involve automation of navigation and steering have perfect or near perfect accuracy, while the other modes do not. This indicates that the man-interactive feedback for navigation and steering, though somewhat effective, falls short of expected performance. To emphasize how poorly TB performed in this test, the accuracy was only measured by the two piles' proximity to each other and did not take into account where the two piles were dumped. In more than one TB test, the loads were dumped completely outside the dump area because minor course corrections could not be made during the test.

The accuracy of the final position was measured twice for each test. For the normal, manual, and PB modes the accuracy was assumed to be perfect as the operator was in full control of navigating the machine. Somewhat surprisingly, however, the RAP program performed as poorly as the TB. The first reason for this is that since the RAP program allows trimming, this might induce error due to the effect on hydraulic pressure and hence, engine RPM. For instance, if the operator corrects the steering angle slightly, the hydraulics load the

engine more and effect anything that depends on engine RPM. And the second reason is that if the playback is going to stop short of the finish line, the operator has no way of knowing beforehand and cannot act to correct it. The human brain is not smart enough to compute how well the playback has been progressing and determine if corrections need to be made in the present to avoid future problems. This is not necessarily true for all recorded sessions, but was a problem here because of the complex nature of the test.

The number of trimming operations made per test revealed a few things. It made it obvious that, in the modes where it was available, the operator needed the trimming capability. However, the RAP program used far more corrections than the PB. While this might be assumed to be because the RAP is recording all functions, most of the RAP corrections were steering corrections. This indicates that the automatic navigation of the loader requires the highest level of feedback. In fact, 75% of all trimming operations were steering.

A final piece of data that was more subjective was fatigue level. Of the four control modes, TB was the easiest. After a recording was made, the operator merely had to position the loader properly and push a button. However, the inaccuracy of this method makes the low fatigue level a moot point. The next easiest method to use was the PB mode. The operator had only to concentrate on moving the machine from the pick-up to the dump area, and supervise the arm and bucket operations. Behind PB came normal teleoperation. In this mode the operator felt relaxed by being in control of all of the loader's functions at all times. However, stress levels were increased when multiple motions were being performed simultaneously. The worst method for fatigue was the RAP program. While the trimming concept proved a good one, when the operator had to worry about correcting all the loader functions, especially steering, it was mentally very stressful. Ranking the manual mode is more difficult. Because of the high level of feedback of actually being on the machine, mental stress while operating the loader manually is minimal. However, the operator is physically stressed by being subjected to far more noise, vibrations, and harshness (NVH) than in the teleoperated modes, to say nothing about dust and heat. Severe jostling by the loader will often cause the operator to slow down when it is not actually necessary, and tire him out sooner.

In order to directly compare all the control modes, a composite scoring was developed. The following five categories were all weighted equally:

1. Number of cones contacted
2. Loads transported
3. Dump Accuracy
4. Final Position Accuracy
5. Time difference (see Fig. 5.4)

The raw data for each individual category was combined into an array for all modes. Each data point was then given a percentage statistical rank based upon its position in that array. Figure 5.7 shows the results.

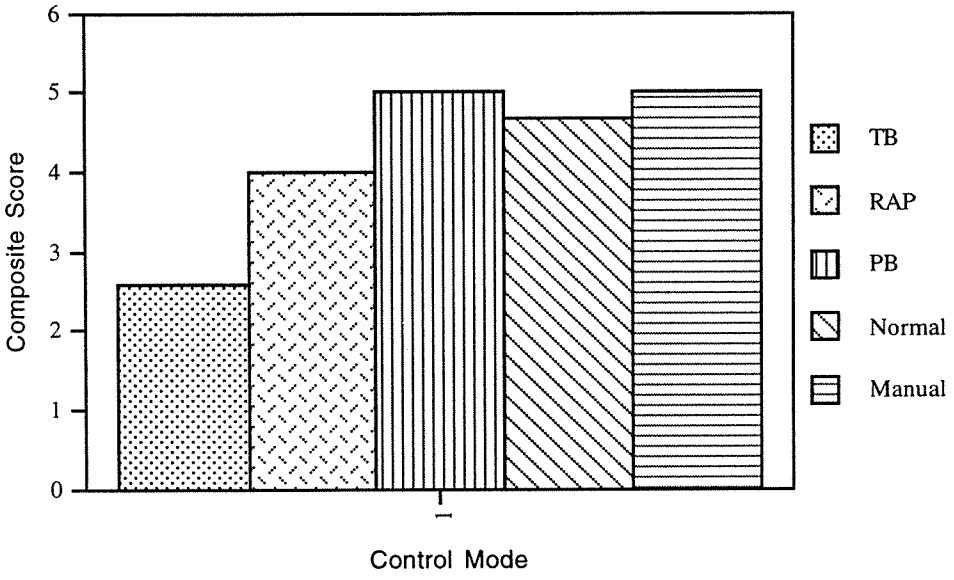


Figure 5.7 Composite Testing Scores

From this chart, it can easily be seen that the PB mode far outperforms the other two teachable function programs and even approaches manual operation. While normal teleoperation is ranked nearly as high as PB, the PB mode is better because of the reduced operator fatigue level.

5.3 Summary

From the testing and results, some important conclusions can be drawn. First, while the time-based program (TB) performed synchronous playback of all loader functions as intended, the lack of feedback rendered it inaccurate and unfeasible. Second, the hybrid program (RAP) substantiated the concept of man-machine interactive semi-automatic control. It demonstrated a dramatic improvement in accuracy over the TB program. However, RAP still did not perform accurately enough to be considered as a production possibility. In addition, the RAP program showed that with the man as the feedback controller in navigation maneuvers, operator stress levels were too elevated to make it a viable option. Third, the position-based program (PB), with control over only bucket and arm functions, proved to be the highest performing teachable function program as far as accuracy, productivity, and fatigue are concerned. In fact, if the factor of fatigue is taken into account, the PB program outperforms both normal teleoperation and manual operation.

Chapter 6

Conclusion and Recommendations

The TAMER project was conceived with the idea of improving heavy earth moving equipment operator safety in hazardous environments. This goal was accomplished by removing the operator from the machine and making it teleoperated.

This phase of the TAMER project introduced "teachable functions" to the TAMER system. These functions allow an operator to "record" and "play back" sequences of movements on the loader, with the goal of reducing operator fatigue and increasing productivity. In addition to the existing manual operation and normal teleoperation modes, three new control programs incorporating "teachable functions" were developed. All five modes were field tested and evaluated for accuracy, productivity, and operator fatigue level. The new control programs are time-based (synchronous), position-based (asynchronous), and hybrid.

The time-based program records operator control inputs, durations, and sequences and stores them in memory. To play a sequence back, the program substitutes in the stored control values for the same amount of time that they were recorded. This program does not allow the operator any form of feedback during playback. Testing showed that while this method repeated recorded sequences, the lack of feedback rendered it unacceptably inaccurate.

The position-based program records data from position sensors on the loader's bucket and arm. This program only controls the bucket and arm components, so navigation of the machine is still completely controlled by the operator. During playback, the program moves the bucket and arm between positions according to the concept of the "finite state machine". Testing of this program showed it to be very promising. Operator fatigue was reduced since he did not have to manipulate the loader's arm and bucket continuously. Also, accuracy and speed of this method were excellent. One reason for the improved accuracy of this method is that it allowed feedback from the operator during playback. If the operator sensed the playback needed to be changed he could make corrections.

The hybrid program takes characteristics from both the time-based and position-based programs. From the time-based it takes the ability to record and play all loader functions in a synchronous manner. From the position-based it takes the ability to record position sensor data from the machine components. It also incorporates the feature of man-machine interactive feedback during playback. Testing of this mode demonstrated it to be much more accurate than the time-based program, but still short of the other modes. Also, while the man-machine interactive feedback was shown to be effective, the mental stresses it place upon the operator were deemed unacceptable.

Of the three teachable function programs, the position-based should be the one implemented in any production capacity. While the other programs controlled more loader functions, they would require a much greater level of feedback and control than exists now in order to become feasible. A simple upgrade of the position-based program to include multiple recorded sequences. It was noted during testing that if the program had had this capability it would have been even faster and more convenient. If there is a great desire to have recorded navigation of the machine, a much more sophisticated sensor network with a very accurate Global Positioning System (GPS) would be needed. However, a system like this would be drastically more complicated (and costly) than the existing one.

The TAMER system has the capability to improve safety and reduce injuries by removing the operator from the machine. The addition of teachable function programming to a teleoperated system would help improve productivity and reduce operator fatigue. In fact, with teachable functions, a teleoperator should be able to just as productive as an operator on the machine, but with a much reduced fatigue level.

SECTION II

TELEPRESENCE AND AUDIO/VIDEO SYSTEMS FOR REMOTE OPERATION

Chapter 7
Introduction

For the proper operation of a remotely operated vehicle where the human is part of the closed loop control of the machine, the operator needs to be supplied with necessary information on the operation being performed. The attempt to artificially supply these senses is referred to as telepresence. During phase II of the TAMER project, a three dimensional color video/audio system developed at Metron Optics was integrated into the teleoperated front-end loader. The 3D video/audio feedback system consists of a camera subsystem on the loader and a viewing system that is mounted on the remote operating station. The video/audio feedback system enables the operator to regain the major senses, as to the status of the vehicle and position of the vehicle relative to the points of operation, which are lost during remote operation from a distance greater than 200 ft. Field test evaluations showed the advantages of using three dimensional video display system over two dimensional systems. It is clear that three dimensional color systems offer better object recognition such as identification of bumps and ruts that could potentially cause the rollover of a loaded front-end loader, and give the remote driver an accurate driving and manipulating capability.

However, the system has several disadvantages. First, its relatively large size (66 cm x 81 cm x 64 cm) and high cost made it impractical for Caltrans field use. Furthermore, the head movement of the operator has to be limited in order to maintain the three dimensional image. Base on the feedback collected from Caltrans operators during the field tests, side mounted two 2D cameras with corresponding secondary monitors mounted at the workstation is recommended to allow for a duplicated peripheral view for the remote operator. Since the front-end loader is frequently engaged in driving in reverse, a back up camera view is also suggested.

The purpose of this phase was to further investigate the feasibility of using a telepresence system for the teleoperated front-end loader. This section of the report presents the research resulting in an improved operation of the teleoperated front-end loader. The telepresence system that was created utilizes a stereoscopic front view with monoscopic side and rear views. Another major effort of this research involved the enhancement of the stereoscopic system by empirically evaluating the relationship between stereo camera spacing (interocular viewing distance) for a parallel camera configuration and object distance.

Chapter 8
System Design

The telepresence feedback system requires three main subsystems which include: an acquisition system, transmittal system, and finally, a display unit. See figure 8.1.

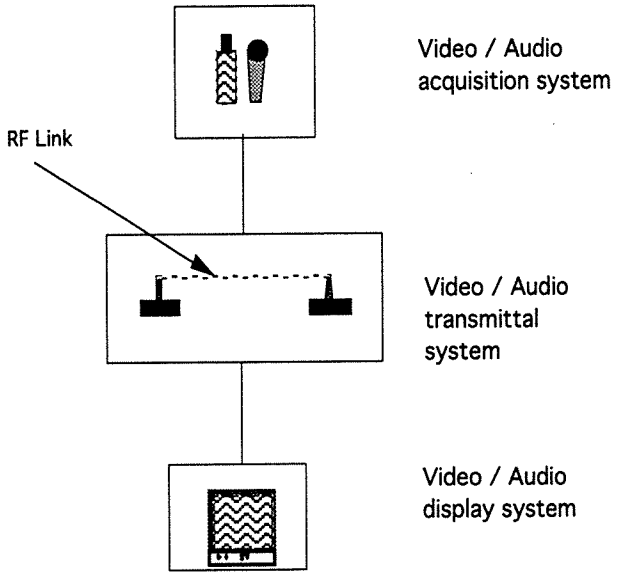


Figure 8.1 Telepresence System Diagram.

There exists several standard options in the selection of display units. The transmittal system can either be hard wired or RF link system available off the shelf. The video audio acquisition system requires attention to stereo camera configuration as well as horizontal field of view.

8.1 VIDEO AUDIO ACQUISITION SYSTEM

8.1.1 Design Consideration

It may be possible to acquire a full 360 degree stereoscopic image to supply an immersive telepresence environment for the operator. A full 360 degree stereoscopic image would be substantially more expensive and more difficult to maintain than a simpler system while supplying depth cues in views that may not be necessary. For practical remote operation, only the most important and useful locations of views should be acquired for operator feedback.

Investigation on audio feedback during Phase II revealed that stereo audio is not necessary for teleoperation of heavy earthmoving equipment. Initial trials prove mono audio to enhance teleoperation control and will therefore mono audio feedback was be included in this design.

During remote operation of a front loader, where object manipulation as well as driving navigation takes place in the front of the machine, visual depth cues are necessary in a front view. On the sides and rear of the machine where no object manipulation but vehicle navigation is required, simple monoscopic images are necessary and sufficient. In hazardous terrain locations such as mountain slopes where mud slides have occurred, it may be necessary to navigate the vehicle along cliff sides in both forward and reverse directions. These maneuverability requirements necessitate video object recognition in the form of monoscopic side and rear views. See figure 8.2.

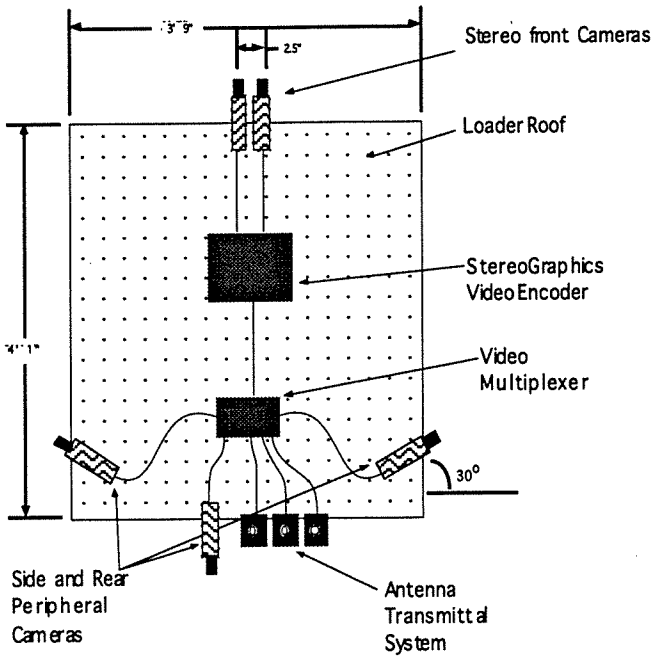


Figure 8.2 Vehicle Camera Positioning (Loader Roof).

As previously mentioned, it may be possible to increase the HFOV by incorporating a single camera system on a slaved rotating mechanism. Previous attempts at creating such systems have had problems with lag time for rotation (Kruetzfeldt, 1995). Kruetzfeldt reports that operators became frustrated with latency and simply stopped using the slaving mechanism. Another detriment to the slaved system is that it may necessitate the operator to toggle a switch for the rotation, thereby, requiring attention by the operator, taking attention away from the other controls of the machine. Due to these detriments, the multiple camera system was selected over the slaving system for effectively increasing the HFOV.

8.1.2 Camera location and configuration

After choosing the multiple camera system for increasing the effective HFOV, care and consideration had to be given for optimal camera placement. Originally, it was believed that supplying the view from within the cab where the normal manual operators head was located would be favorable. It was soon determined that bucket operations block the normal operators view and that a better view could be obtained from a higher location up on top of the cab. A second benefit of placing the cameras up on top of the cab was realized. The original configuration (cameras in the same place as the normal operators head) required that the cameras physically be located inside the cab. In the case that the vehicle was to be operated manually, the driver had to move the camera system out of the cab so as to allow room for the operator to work. The benefit with the second camera

configuration was that the loader could be operated manually without moving the cameras in and out of position, effectively leaving them in a semi- permanent position. See figure 8.2.

In the literature, there is some debate over the optimal stereoscopic camera configuration for various viewing conditions. Shields et al., and Pepper et al., suggest orthostereoscopy while Diner et al., and Yamanoue argue that optimal camera configuration changes depending on the viewing conditions.

A somewhat orthodox configuration was selected for the initial trials of the telepresence system (orthostereoscopy). Human eyes are spaced at 2.5 inches, therefore, the standard camera inter-viewpoint distance (IVD) was selected as 2.5 inches. In order to minimize the problems of reverse parallax and concave planes of equal depth, a parallel camera configuration was selected at this IVD. See figure 8.3.

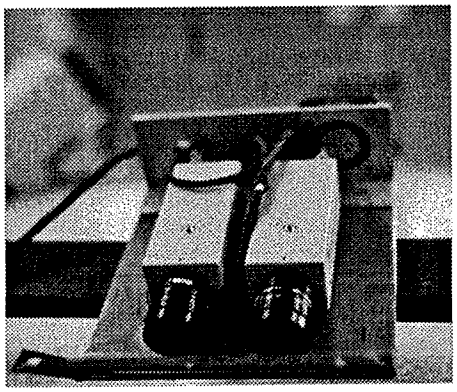


Figure 8.3 Stereoscopic cameras

8.1.3 Description of The Telepresence Acquisition System

The stereoscopic camera system was placed in a semi weather proof housing up on top of the cab as shown in figure 3.2. The two stereo front cameras (Sony SSC-C370) were configured with 6 mm lenses. The signal from the two stereo cameras was sent to the StereoGraphics video encoder which compresses the two images vertically into one video signal. The single video signal was then wired to the RF video antenna.

The two side cameras (Cohu 1300) with 6 mm lenses were mounted in Pelco weather proof camera housings and attached to the loader roof as in figure 8.2. The cameras were angled (roughly 30 degrees) so that during a full turn, the respective side camera would be facing forward. The video signal from the left side camera was led into the video multiplexer while the signal from the right camera was led directly to the RF antenna.

The rear camera (Sony SSC-C370) with a 6 mm auto iris lens was mounted on the roof of the loader facing directly to the rear of the machine (figure 8.2). The rear camera video signal was wired into the video multiplexer just as the left camera signal.

The video multiplexer accepted the left and rear camera video signal as well as a digital signal from the Loader Control Computer (LCC). An integrated circuit in the video multiplexer output the left camera signal when the LCC signal was high and the rear camera signal when the LCC signal was low. The LCC signal was toggled when the machine switched from forward to reverse gear. The output from the video multiplexer was then sent to the RF antenna.

8.2 VIDEO AUDIO DISPLAY UNIT

8.2.1 Stereo display selection

Three stereo display systems were examined during the investigation.

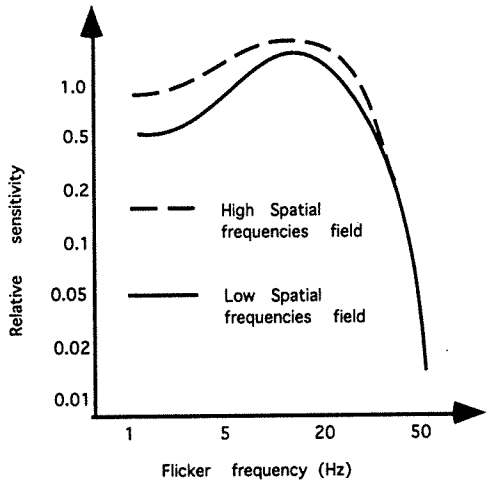
The Concave Mirror Hologram (CMH) system developed by Metron Optics was tested during phase I and was found to give adequate depth perception. However, several problems became apparent. One problem was that the CMH afforded a very narrow HFOV. Secondly, the mirror required for the CMH system was extremely expensive and delicate to handle. The most noticeable disadvantage of the CMH system was the large and bulky size of the implement. Two monitors and a concave mirror are required to be placed in specific locations in order for the image to be clear and produce retinal disparity.

The Head Mounted Display (HMD), an I-glasses Personal Display system from Virtual IO corp. was also tested in field by Caltrans operators. Depth perception was afforded, however, several problems soon became apparent. While the operator is wearing the HMD, his vision is restricted to only the view displayed on the HMD and he/she can not look at the teleoperation control unit, other display monitors, or the direct line of sight to the machine. Due to the fact that the multiple camera system was selected, multiple display units would have to be utilized. An HMD would restrict the viewer so as not to see any views other than that on the HMD.

An Alternating Liquid Crystal (ALCS) system. The ALCS system allows for depth perception in the form of retinal disparity while not limiting the system to only one view as is the case with the HMD alternative. The viewing area is only limited by the size of the monitors used and is not restricted by a bulky mirror system as is found with the CMH option.

The first ALCS system tried in the telepresence unit utilized a standard interlaced television monitor for the display. The left camera image was displayed as the even interlaced field and the right camera image as the odd interlaced field. In this fashion, only 30 full stereoscopic images per second could be displayed on a standard television

monitor. Studies have shown that the human visual system is sensitive to flicker at rate less than 50 Hz (see Figure xxx). Therefore, The human observer can clearly detect flicker at this rate of 30 frames per second, subsequently, the benefit of the depth perception in the stereo system was overridden by the bothersome flickering of the slow alternating frequency.



Reproduced from Jain 1989.

Figure 8.4 Threshold of Human Flicker Recognition.

The CrystalEyes system developed at Stereo Graphics Corporation takes in the standard left and right camera images and holds them in video memory. By holding the left and right images in video memory the two distinct images can be displayed multiple times before the next set of images is supplied by the cameras. With the use of a high speed computer monitor, 120 distinct non interlaced pictures can be displayed per second. The effective full stereoscopic images are then displayed at 60 frames per second which is well above the human flicker perception rate. This ALCS system does not limit the user to only one view. Because the ALCS glasses flicker at such a high rate, when the observer views another monitor or direct line of sight, he/she feels as though they are looking through normal sun glasses. The ALCS glasses can also be designed to be light and compact in order to minimize user fatigue.

8.2.2 Peripheral display selection

The peripheral camera system includes two monoscopic side views and one monoscopic rear view. See figure 3.2. The two main factors in selecting the peripheral view display system was size and human engineering. The monitors had to be small enough so that they would not interfere with the stereoscopic and direct line of sight views yet large enough to supply adequate object recognition on the sides and rear of the machine. The first monitors selected where Sony 4" LCD displays. The two LCD monitors were mounted on the remote driving station on either side of the stereoscopic monitor. See figure 8.5.

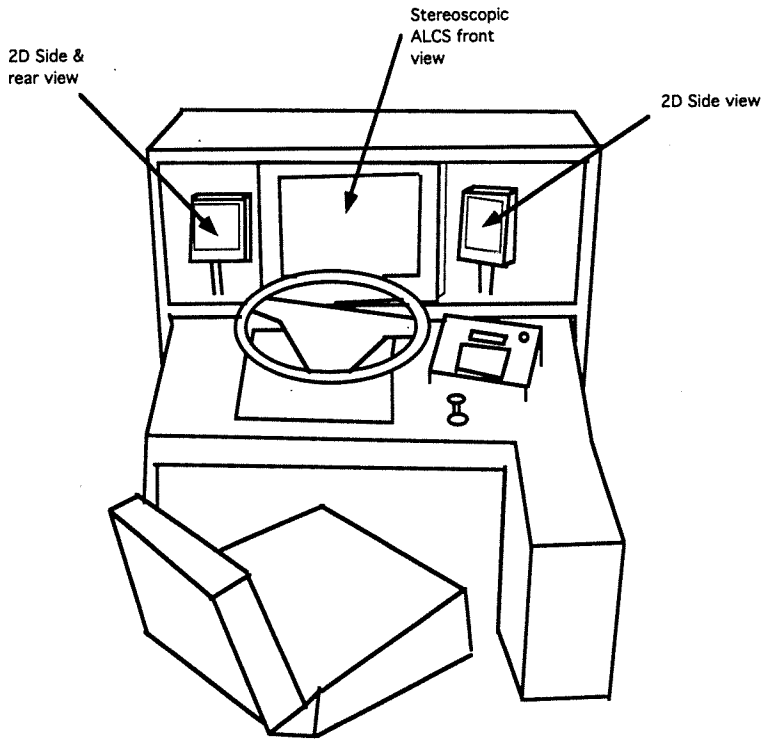


Figure 8.5 Telepresence display configuration

The left and right peripheral views were displayed on the respective LCD monitors. When the machine switched into reverse gear, a video switch on board the loader switched the left camera view to the rear camera view. By using this switching method, all three peripheral views could be displayed on only two LCD monitors. The use of only two peripheral display monitors reduced the total size of the telepresence display system in addition to cost and complexity.

Early in preliminary testing, the 4" LCD monitors proved to be too small for their location. In order to see the monitors properly, the operator had to lean in close to the displays. An effective ergonomic telepresence system should not increase operator work load but decrease it. In order to display a useful image to the operator, the monitors could either be moved closer to the user or be replaced by larger monitors. It was decided that if the monitors were moved any closer to the operator, they would interfere with the controls of the remote driving station, therefore, larger 6" LCD monitors were selected. The bigger LCD monitors displayed a large enough image to be useful to the user.

8.2.3 Description of Telepresence Display Unit

Three monitors were utilized in the Telepresence display unit. The stereoscopic image was displayed on a 15 inch View Sonic non-interlaced monitor with a vertical refresh rate of 120 frames per second. The two peripheral monitors were 6 inch Sharp LCD color monitors.

The stereo image sent via RF link was received at the telepresence display unit as two distinct images squeezed vertically into a single NTSC video signal. A StereoGraphics video decoder takes the single signal and decompresses the two images and outputs the two distinct images time sequentially for display on a high speed monitor at 120 frames per second.

CrystalEyes ALCS glasses are required for viewing the stereo image on the high speed monitor. The CrystalEyes glasses are light and compact so as to reduce user fatigue. The glasses are synchronized to the stereo monitor by an infrared signal sent out by the StereoGraphics decoder. The glasses are designed to flicker only while receiving this signal. If the user looks up away from the viewing center, the glasses stop receiving the signal and go clear. By allowing the glasses to go clear, the direct line of sight can be viewed completely unobstructed.

A special telepresence mount device was constructed in order to house all of the video monitors on the remote driving station. The display mount was designed so that the telepresence unit would not interfere with the normal operation of the controls of the driving unit. The telepresence mount was also designed so as not to interfere with the drivers ability to look up at the actual machine and obtain a direct line of sight to the loader. See figure 3.6 for an actual photograph of the final telepresence display unit.

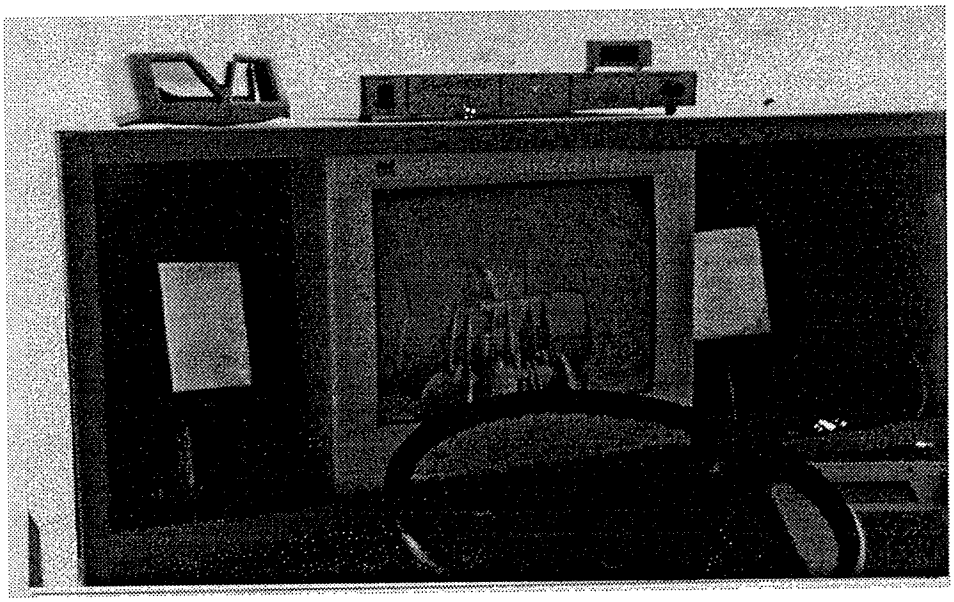


Figure 8.6 Final Telepresence display unit

The LCD monitors were rotated 90 degrees in the final configuration to eliminate the conflict of polarization between CrystalEyes glasses and the LCD monitors. The glasses for the ALCS system use liquid crystal panels for the electro shuttering of each eye. LCD monitors produce polarized light on a liquid crystal panel. The polarized light from the LCD monitors can only pass through the ALCS glasses if the polarization of the light and that of the glasses are in phase. When the LCD monitors are in their normal upright position and the glasses are worn normally on a persons head, the polarization of the ALCS glasses are out of phase with the polarized light from the LCD monitors. A person wearing the ALCS glasses looking at an upright LCD monitor sees a blank screen because all of the polarized light from the monitor is blocked out. When the LCD monitor is turned 90 degrees, the polarized light is in phase with that of the glasses and the image can be viewed normally. Therefore the LCD monitors were mounted sideways. Subsequently, the cameras on board the loader had to be turned 90 degrees so that the final image on the LCD monitors would be viewed vertical to the telepresence operator.

8.3 TELEPRESENCE TRANSMITTAL SYSTEM

At any one time, four distinct images (3D front, left, right and rear views) were collected by cameras. Only three images were sent in parallel to the remote telepresence unit in real time. As previously mentioned, the left and rear images share the same monitor as well as the transmittal unit. Therefore the video transmittal system consists of three

Pelco wireless video transmittal units which operate at 2414, 2450, 2475 MHz respectively. Pelco reports their wireless video transmission system to have a direct line of sight range of 1000 feet. This range is acceptable as it exceeds the present range of the remote communication system.

During initial trials, the three transmitters were mounted side by side up on top of the roof of the machine. See figure 8.7. It was soon determined that the transmitters truly needed a direct line of sight to the receivers in order to have a clear signal. Due to the high frequency at which the transmitters operate, even the smallest physical object between the transmitter and receiver can cause signal distortion, including the antennas of the other two transmitters.

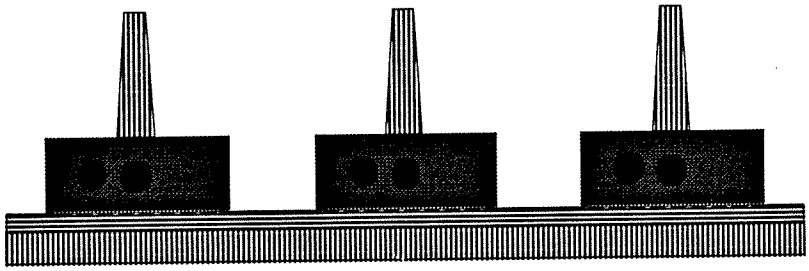


Figure 8.7 Initial (horizontal) configuration of the video transmitters.

In response to the signal distortion problem, the transmitters were mounted vertically, one on top of the other. This tree mount configuration allows for the least amount of obstruction to the two lower transmitters and no obstruction to the top transmitter. See figure 8.6.

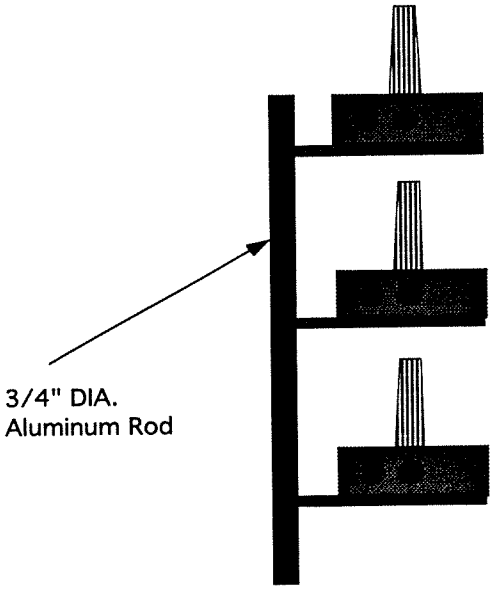


Figure 8.8 Vertical tree mount transmitter configuration.

Chapter 9
Testing And Results

The objectives of the tests are to:

1. Test effectiveness of incorporating side peripheral views and rear view to the stereoscopic front view for the teleoperated front-end loader.

2. Study the relationship of inter-viewpoint distance (IVD) of a pair of parallel cameras and the object distance, and to optimize the camera configuration for best 3D images.

3. Evaluate overall performance of the telepresence feedback system developed in this phase (TFS v.2).

The teleoperated front-end loader with the telepresence feedback system (refers to TFS V2 hereafter) was tested in the open fields at UC Davis campus as well as at District 3, operated by skilled heavy equipment operators and Caltrans maintenance crews, to evaluate the overall performance of TFS V2. In order to reduce the glare and ambient light wash out, an enclosure with full roof and three side walls were used to host the remote operating station. Feedback and comments collected through written and verbal questionnaires have indicated that, since the system provided multiple telepresence views (front, left, right and rear) while allowing for line-of-sight, operators felt relative comfortable and confident to remotely operate the front-end loader. The CrystalEyes glasses was proved to be easy to use and "care-free" - the infrared emitter automatically toggles the glasses between "stereoscopic" mode and "clear" mode when the operator made his/her head movement as needed. When the glasses are in the "clear" mode, the operator can see through the glasses for the peripheral views or line-of-sight.

In addition to the field tests, test procedures were developed to conduct two sets of testing with better controlled parameters. The first set of tests, the telepresence operation tests conducted in an open field at UC Davis campus was to determine the usefulness of the telepresence system. The second set of tests, optimal inter-viewpoint distance (IVPD) tests were designed to empirically evaluate the optimal inter-viewpoint distance as a function of object distance for parallel camera configuration in the range of from ten to thirty feet as is required for teleoperation of a front-end loader.

9.1 Telepresence operation Testing

9.1.1 Testing Setup

In order to verify the true usefulness of the telepresence system and all of the components implemented, the testing of the system was created in a way that would simulate a real world application of the loader. Precision bucket maneuvers as well as gross vehicle navigation are required during actual loader operations and were included in the design of the vehicle testing layout.

The telepresence testing was broken up into two distinct tasks. The first task required that the operator attempt to make precision bucket placements by placing an extension of the bucket in a specified location. The second task required the operator to navigate the machine through gross vehicle movements in the attempt to move the largest amount of dirt possible in two scooping operations while navigating through a specified course marked with construction cones. All tests were done with the remote driving station placed at a distance of 250 feet from the center of the test course. See figure 9.1 for the course setup. The remote operating station was placed inside of a large trailer with its back door open to reduce the glare and wash out of the monitors due to bright Sun light.

9.1.2 Testing Procedures

Task 1. Precision Bucket Placement

A vertical pole, one foot tall and two inches in diameter, was attached to the end of a three foot long bar extending from the loader bucket. The objective of the first task was to drive the machine from the start location over to two vertical poles standing six feet tall and place the small vertical pole extending from the loader bucket directly in-between the two vertical poles. The distance in inches between the true center of the poles and the actual position of the operator positioned pole was measured. The two vertical poles were of the same diameter as the small pole extending from the bucket and were spaced six feet apart. A two inch orange stripe was painted at the top of each pole in order to enhance visual cues.

Task 2. Gross Navigation & Dirt Movement

The second task required that the operator navigate the loader from the start location around to the front of the dirt pile, pick up two loads of dirt and dump them into a specified dump area, and return back to the start location. The course was lined with construction cones that the operator was instructed to avoid by staying within the specified course. Based on the constant angle of repose of soil, the height of the dirt pile created by the two loads of dirt is used as an indicator of both the volume of dirt moved and the accuracy to which it was placed. If two full loads were picked up but not placed one on top of the other, or if the operator were not able to obtain full loads but did place them accurately, the recorded pile height would not be as large as if two full loads were obtained and placed perfectly one on top of the other. During the test, the number of cones hit, total time and the pile height were recorded.

In order to satisfy the two objectives listed above, the testing was broken down into five treatments (3D+, 3D-, 2D+, 2D-, 0D)*. Due to the nature of the test, and the human interaction, a repeated measures design was utilized. The order of the five treatments were randomly assigned to one operator at a time and nine total operators were run through the testing procedure over three days. Each operator was given about 25 minutes to become familiar with the system before testing began.

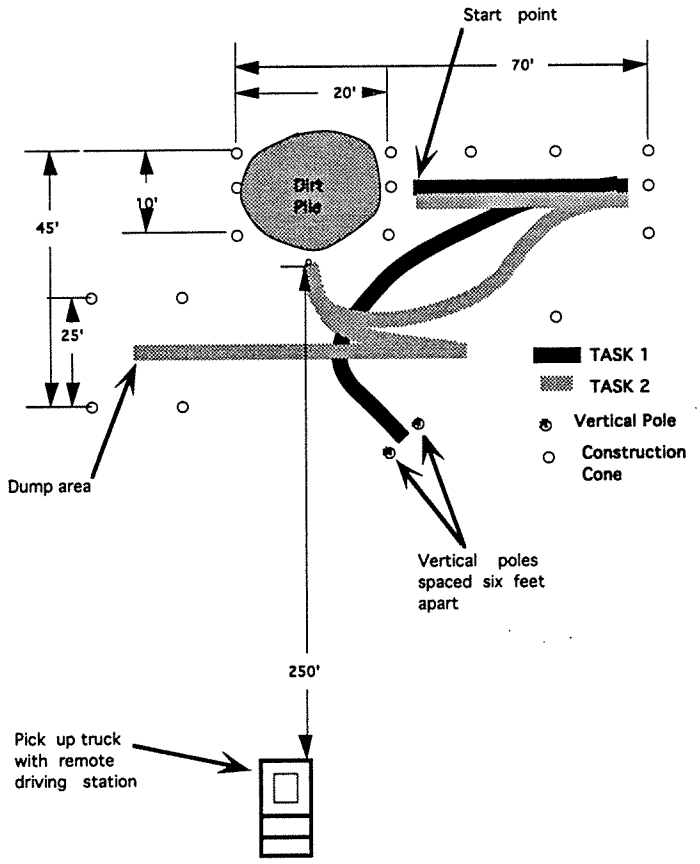


Figure 9.1 Telepresence Test Course.

* (3D) refers to stereoscopic, (2D) refers to monoscopic, (+) refers to the addition of peripheral views, (-) refers to the lack of peripherals, (0D) refers to no video system, using simply direct line of sight (LOS).

9.1.3 Telepresence Operating Testing Results

The raw data from the three telepresence tasks can be found in the appendix section. The data from the three vehicle telepresence tasks were compared and the results are as follows. The 3D vs. 2D vs. 0D distinction was most clear in the precision bucket placement task. The peripheral view comparison became clear by comparing the number of cones hit for each treatment throughout both task one and task two. The pile height indicator in task two showed no difference between 3D and 2D but did show a statistical difference between tests done with no video system (0D) and with video systems (2D/3D).

1 Precision Bucket Placement

The precision bucket placement (task 1) proved to be a good indicator for the comparison of the 3D, 2D and 0D feedback front view. A graphical representation of the absolute error for each of the feedback systems is depicted in figure 9.2.

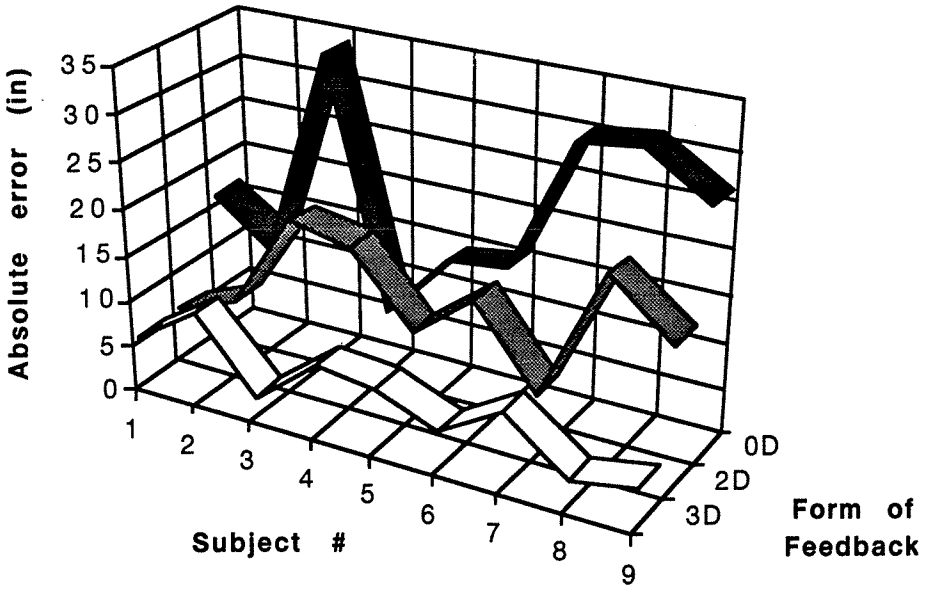


Figure 9.2 Precision Bucket Placement With Various Video Feedback.

An analysis of variance using SAS statistical software was conducted on the data displayed above. All three of the feedback systems were proven to be statistically different by the Duncan test. The normalcy and F value assumptions for ANOVA were valid.

2 Gross Vehicle Navigation

The Gross Vehicle Navigation test (Task 2) proved to be a good indicator of the comparison between the telepresence system with peripheral views and the system without peripheral views. The second task did not show much of a distinction in stereoscopic versus monoscopic front views. The utility of the peripheral views can be seen when comparing the number of cones hit with the peripherals as to the number of cones hit without the peripherals. Table 9.1 depicts the number of cones hit under the two differing components for all nine drivers. Nine cones total were hit without the peripheral views while only one cones were hit with the peripheral views.

Table 9.1 Number of cones hit with and without the peripheral views.

Subject #	1	2	3	4	5	6	7	8	9
# of Cones hit w/ Peripherals	0	0	0	0	0	0	1	0	0
# of Cones hit w/o Peripherals	1	0	2	1	1	1	1	1	1

3. Pile height

The pile height indicator during task two had very little difference between 3D and 2D but did show a difference between the treatments with video feedback (Stereoscopic and Monoscopic) and no video feedback (0D). See Figure 9.3.

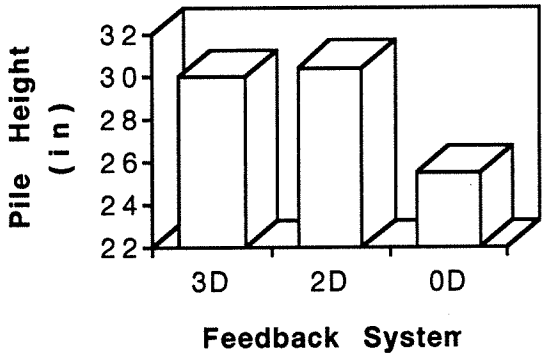


Figure 9.3 Pile height under the three front view systems.

Both the Duncan and Tuckey tests in an ANOVA procedure showed statistical difference between the tests run with the video feedback systems (Stereoscopic and Monoscopic) and without (0D). The normalcy and F value assumptions for ANOVA were valid.

During field tests, all of the test subjects indicated that both peripheral camera views and the stereoscopic front view afforded the most useful telepresence information. The test

subjects indicated that they felt reasonably comfortable operating the large machine with such a system.

9.2 Optimal Inter-Viewpoint Testing

The second set of tests were designed to empirically evaluate the theory that optimal inter-viewpoint distance (IVD) is a function of the object distance for parallel camera configuration. If a relationship exists then the clear objective would be to design a model for the IVD spacing in the range of from ten to thirty feet such as is required for teleoperation of a front loader.

9.2.1 IVD Test Set Up

A test site was constructed in order to test the theory for a relationship of IVD and object distance. The driving station used in the telepresence testing was used for ease of test set up. The Stereoscopic cameras were placed on a specially designed mount so that the IVD could be altered at will. Precision machined spacing bars were created so that the cameras could be spaced consistently at 2.5, 5.0, and 7.5 inches. In addition to the three stereoscopic distances, the spacing of 0" was also included by turning off one of the cameras in the stereo system, effectively creating a monoscopic system.

Three poles, each one foot tall and two inches in diameter were used. Two of the poles spaced four feet apart were placed at any of five distances 10, 15, 20, 25, or 30 feet from the stereoscopic cameras. The third pole was placed on a sliding car mechanism that could be slid along a track in-between the two stationary poles. Figure 9.4 shows the test site with the viewing station placed behind a barrier perpendicularly with respect to the stereoscopic cameras and pole placement track.

A barrier was constructed so as to hide the base of each pole in order to eliminate depth cues by shadows. In addition, the stereoscopic camera assembly was placed at the same elevation (one foot high) as the poles so as to restrict any plan view from supplying depth cues.

9.2.2 Procedures

In order to eliminate the variance between people, a split plot design was used with the subject (person) blocked. For each of six subjects, the sub-effect of object distance (10', 15', 20', 25' or 30') was randomly assigned within the main-effect of IVD (0", 2.5", 5.0", 7.5") . Each subject repeated the entire set of tests three times for a total of 18 replicates.

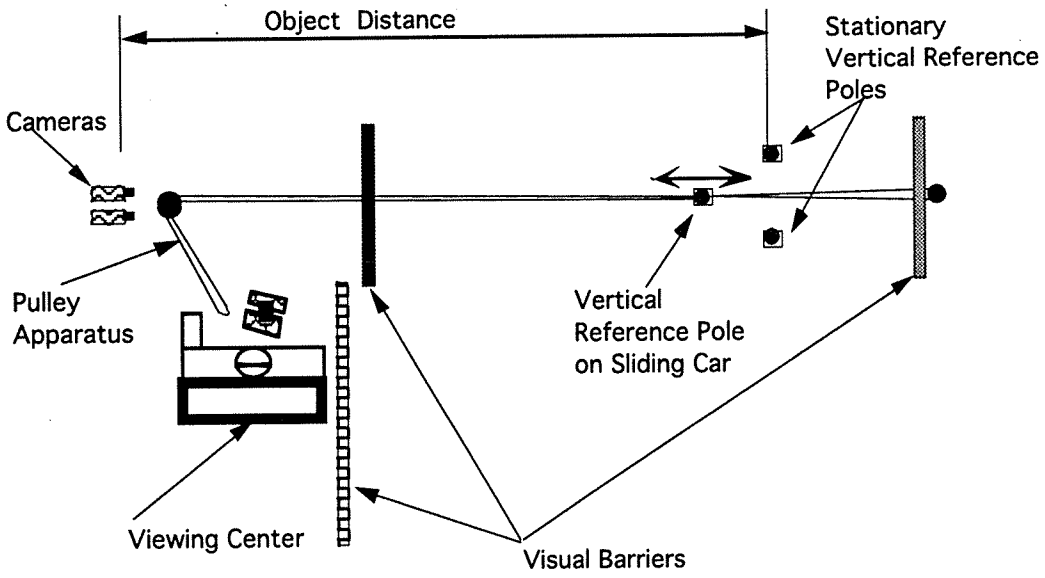


Figure 9.4 IVD Test Set Up.

During each testing replicate, the subject was informed how to maneuver the center pole by pulling on the line connected through pulley system to the center pole sliding car. Each person was given several minutes to become familiar with maneuvering the pole under each of the camera IVD's. The center pole was placed at random starting positions roughly five feet either in front or behind the stationary poles for each test. The operator was informed to place the center pole directly in line with (in-between) the two outer reference poles. Once the operator felt comfortable with the system, testing began. Each treatment was applied and the error between the perceived center and the actual center of the two stationary poles was recorded.

9.2.3 Optimal Inter-Viewpoint Results

A strong relationship between optimal camera inter-viewpoint distance and object distance was found. The results also showed a strong absolute difference between monoscopic and stereoscopic viewing.

1. vision error for each IVD setting varies with the object distance.

At the 2.5 inch IVD a dramatic and constant increase in error was observed from ten feet object distance up to the maximum error occurring at the furthest object distance of thirty feet. Figure 9.5 shows the increase in error as object distance increased.

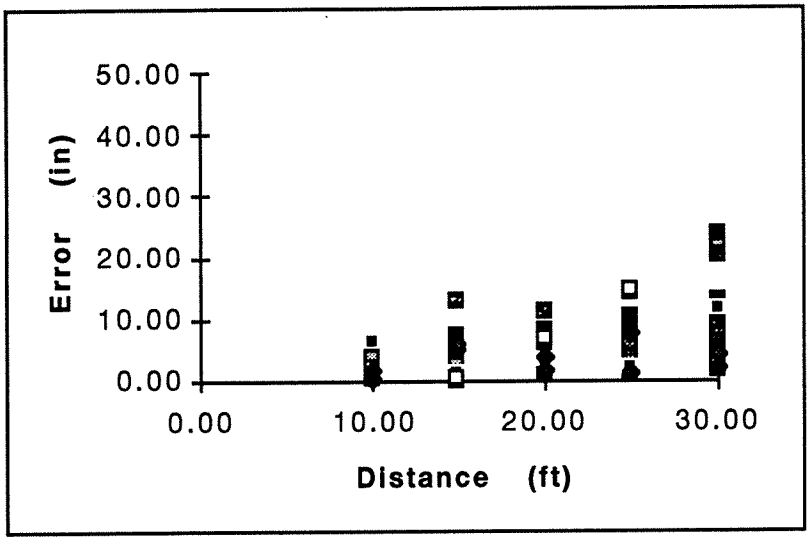


Figure 9.5 2.5 inch IVD Error.

At the 5.0 inch IVD a constant error was observed from ten feet object distance up to the furthest object distance of thirty feet. Figure 9.6 shows the relative constant error as object distance increased.

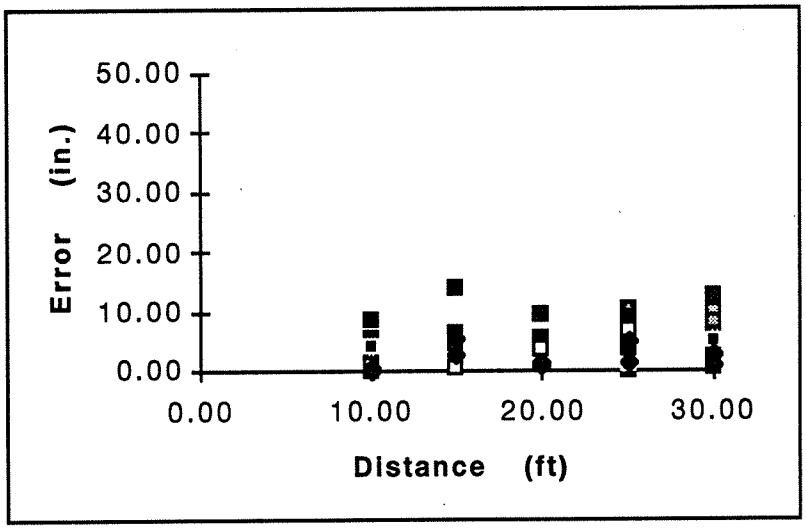


Figure 9.6 5.0 inch IVD Error.

It should be noted here that a large majority of the test subjects complained of some eye strain and general discomfort while viewing at the near object distances with both the 5.0 and 7.5 inch IVD. No discomfort was reported at object distances of greater than 15 feet with either the 5.0 or 7.5 IVD.

The Tuckey test in an ANOVA procedure showed no statistically differing groups. The 10', 15', 20', 25' and 30' object distances were classified into one group. This grouping was found only after one outlier was removed from the data set. The normalcy and F value assumptions for ANOVA were valid.

The 7.5 inch IVD showed an interesting anomaly. Other than the ten foot object distance, a dramatic and constant decrease in error from the 15 foot object distance down to the minimum error occurring at the furthest object distance of thirty feet was observed. Figure 9.7 shows the decrease in error as object distance increased from 15' up to 30'.

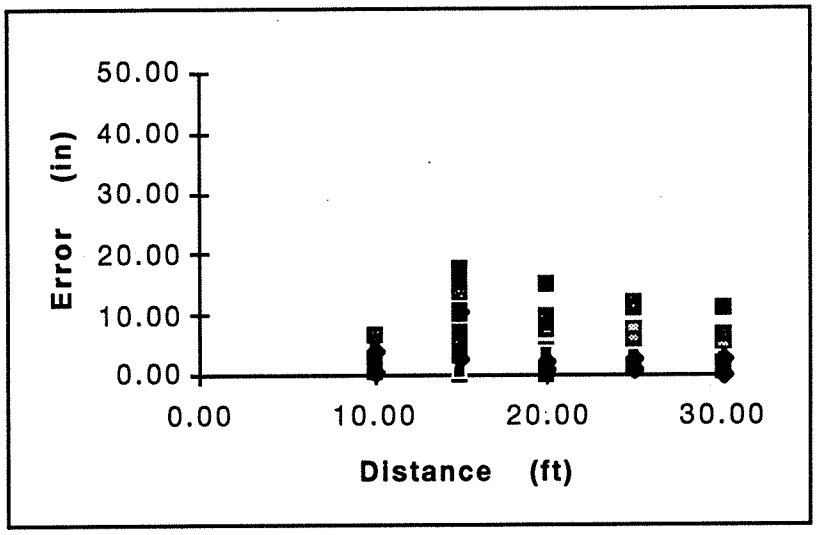


Figure 9.7 7.5 inch IVD Error.

In comparison with the three IVD's selected during stereoscopic viewing, zero camera spacing or monoscopic viewing was a dramatic change. As has been shown by several previous researchers, the benefit of stereoscopy over simple monoscopy can be seen in figure 9.8. The error variance for 0D is much greater at all distances than for any of the stereoscopic systems and the error generally increases with greater object distances.

The statistical data have indicated that there is some relationship between inter-viewpoint distance and accuracy of perceived depth, it is obvious that the next step should be to somehow quantify this relationship so as to be able to predict an "optimal" inter-viewpoint distance for any particular object distance.

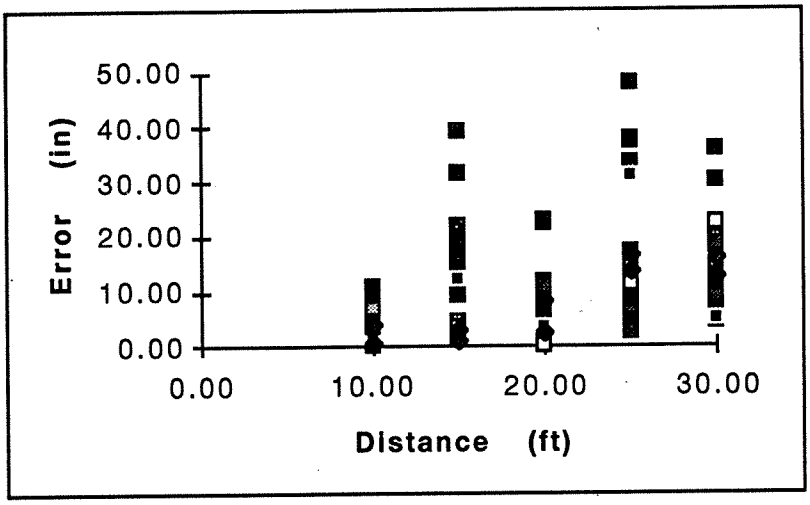


Figure 9.8 Monoscopic (0D) Error.

2. The optimum IVD for various object distances

Figure 9.9 depicts the average error for each of the IVD's versus the object distance. It can been seen that the average error for all three of the IVD's at small object distances shows small error. As mentioned previously, however, both the 5.0 and 7.5" IVD's caused discomfort to the viewers for short object distances. For this reason of "Human Factors" both of the larger spacings were considered non-optimal for the short viewing distances. The IVD of 2.5 inches yielded good perceived depth accuracy without the user fatigue and nausea. The 2.5 inch IVD was chosen as the optimal IVD for the short object distances.

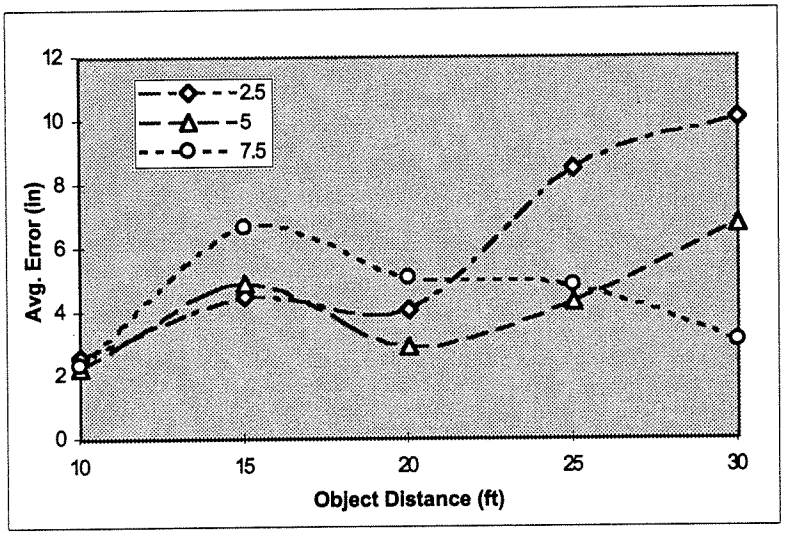


Figure 9.9 Avg. error of each IVD vs. Object Distance

As the object distances increased, the 2.5 inch IVD error had a constant trend towards greater error. The 5.0 inch IVD error seemed to start with a high error, dip to a minimum, and then start an upward trend as the object distance increased. This minimum error for the 5.0

inch IVD (which is lower error than the other IVD's at 20 ft) indicates that the optimal distance for the 5.0 IVD was at the 20 foot object distance.

For the higher object distances, both the 2.5 and 5.0 inch IVD's display trends toward increased error. The 7.5 inch IVD has a consistent trend towards decreased error with increasing object distance. At the highest object distance of thirty feet, the 7.5 inch IVD had a minimum value that was less than the 2.5 and 7.5 IVD's. The 7.5 inch IVD actually yielded errors at 30 feet comparable to the other IVD's at their respective optimal object distances.

3. Linear relationship between optimized IVD and object distance

By extracting the optimized IVD vs. object distance for each of the individual IVD's from figure 9.9, one can determine the optimal IVD as a function of average object distance (Figure 9.10). If this linear relationship can be verified by further tests, it can be used as a guideline for determining an optimum camera setting for a specified viewing (focusing) distance.

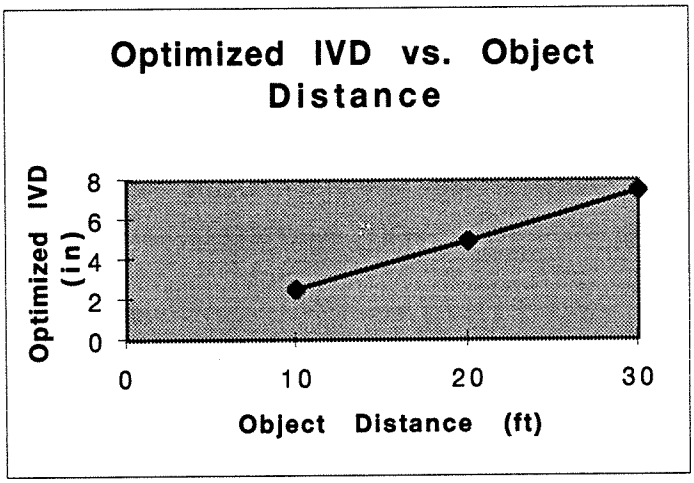


Figure 9.10 Optimal IVD As A Function Of Object Distance.

Chapter 10

Conclusion - Telepresence

This phase of research project has further identified the benefits of incorporating and optimizing a telepresence feedback system in order to artificially supply the operator necessary information for a teleoperated Case 621 front end loader.

An improved video/audio feedback system was integrated into the teleoperated front-end loader. The system is compact and low cost compare to the previous system developed in Phase II. The system employed an ALCS stereo display system developed by StereoGraphics Corp. for the front view and two LCD monitors by Sharp for side and rear views.

Field test evaluations demonstrated usefulness of incorporating both side and rear monoscopic peripheral views in addition to the stereoscopic front view. Peripheral visions enable operators to identify obstacles to the sides and the rear of the vehicle which could be dangerous to both the machine and the operator. The flicker-free stereoscopic front view proved to allow for depth perception in the form of retinal disparity which led to more precise control of the vehicle and bucket operations when compared to the monoscopic system. Much energy and effort was placed into designing and constructing an ergonomic and effective telepresence display unit. Consideration had to be given to allow for a direct line of sight to the machine while still presenting multiple telepresence views to the operator which would not interfere with the standard controls of the driving station. The system which Allows for both multiple telepresence views and line-of-sight has greatly enhanced the telepresence performance and increased operators' confidence of teleoperating a heavy duty equipment.

A relationship between optimal parallel stereo camera separation (Inter-Viewpoint Distance, IVD) and object distance was determined. For the range of from 10 - 30 feet an IVD of 5.0 inches was found to present large enough retinal disparity with minimal distortion to allow for optimal depth perception cues as opposed to the previous standard of 2.5 inch IVD. The (IVD) investigation concludes that an optimum spacing versus range can be determined. Future research may involve the design and construction of a system which may be automatically or perhaps manually adjusted to optimize IVD for the task at hand.

The use of telepresence feedback systems has proven to allow for improved teleoperation of highway maintenance vehicles. Not only can such systems improve safety to the operator but potentially could improve productivity by reducing the fatiguing nature of the on board operation environment.

REFERENCES

Section I

1 Craig, John J., 1989. "Introduction to Robotics, Mechanics and Control." 2nd. Ed., Addison-Wesley Pub. Comp.

2 Dorf, Richard C., 1990. "Concise International Encyclopedia of Robotics, Applications and Automation." John Wiley & Sons, Inc.

3 Frank, Andrew A., "Automatic Control Systems for Legged Locomotion Machines", Electronics Sciences Laboratory, USC, 1968.

4 Kruetzfeldt, Keith J., "Remote Vehicle Telepresence & Three-Dimensional Video Systems for Use in Heavy Earth Moving Equipment Operations". M.S. Thesis Paper, Univ. of CA, Davis, 1995.

5 McKerrow, Philip J., 1991. " Introduction to Robotics." Addison-Wesley Pub. Comp.

6 Nakamura, Yoshihiko, 1991. "Advanced Robotics, Redundancy and Optimization." Addison-Wesley Pub. Comp.

7 Nieminen, Turo J., Sampo, Mikko, 1993. "Unmanned Vehicles for Agricultural and Off-Highway Applications", SAE Paper 932475.

8 Paul, Richard P., 1981. " Robot Manipulators, Mathematics, Programming, and Control." MIT Press, 1981.

9 Paradigm Systems, "Paradigm Reference Manual", Paradigm Systems, 1995.

10 Sun, S., Mehlscau, J., Smith, N., Kruetzfeldt, K., Chen, P., Frank, A., Chang, T., 1994. "Teleoperated and Automated Maintenance Equipment Robotics (Phase I)." AHMCT Research Report, UCD-SEP-94-09-30-01, 1994.

11 TERN, Inc. "A-Drive Technical Manual", TERN, Inc., 1995.

12 TERN, Inc. "Sensor Watch Technical Manual", TERN, Inc., 1994.

13 Wolovich, William A., 1987. "Robotics: Basic Analysis and Design." Holt, Reinhardt and Winston.

Section II

14. Blais, C., R. D. Lyons, 1988. "Telepresence: Enough is enough." Teleoperation and Control Proceedings of the International Symposium, IFS Publications, Bedford, UK. pp. 217-226

15. Crooks, H., L. Freedman, P. Coan, 1975. "Television Systems For Remote Manipulation." Human Factors Society, ProceedingsHuman Factors Society 19th Annual Meeting pp. 428-435.

16. Diner, B., D. Fender, 1993. "Human Engineering in Stereoscopic Viewing Devices." Plenum Press, New York.

17. Diner, D. B., M. Sydow, 1988. " Stereo Depth Distortions in Teleoperation." Jet Propulsion Laboratory, NASA-CR-180242.

18. Dumbreck, A., E. Abel, 1988. " A 3-D television system for remote handling." Teleoperation and Control Proceedings of the International Symposium, IFS Publications, Bedford, UK. pp. 197-208

19. He, D., T. Xu, Y. Wang, H. Hua, Y. Hu, 1996. "A head mounted display system for virtual reality." SPIE Proceedings Display Devices and Systems. vol. 2892 pp. 126-128.

20. Hermann A., Vu-Han, V., Pirschel, H., 1985. "Occupant Protection in Earth-Moving Machines", Institute Automotive Engineering, Berlin, SAE #840202.

21. Jain, A. K., 1989. "Fundamentals of Digital Image Processing." Prentice Hall, Englewood Cliffs, NJ.

22. Jensen, J. F., J. Hill, 1996. "Advanced Telepresence Surgery System Development." Proceedings of Medicine Meets Virtual Reality 4 pp. 107-117.

23. Kruetzfeldt, K., 1995. "Remote Vehicle Telepresence & Three Dimensional Video Systems for Use in Heavy Earth Moving Equipment Operations." Master's Thesis. University of California Davis.

24. Lessard, J., J. Robert, P. Rondot, 1994. "Evaluation of working techniques using teleoperation for power line maintenance." SPIE vol. 2351 pp. 88-98.

25. Lin, Q., C. Kuo, 1997. "Virtual Teleoperation of Underwater Robots." IEEE 25 International Conference on Robotics and Automation vol. 2 pp. 1022-1027.

26. Lipton,L., 1989. "Selection Devices for Field-Sequential Stereoptic Displays: a Brief History." Stereographics Corporation, SPIE vol. 1457 pp. 274-281.

27. Lipton,L., 1990. "Stereo Vision on Your Workstation." Mechanical Engineering March, 1990 pp. 36-39.

28. Lipton, L., 1992. "The Future of Autostereoscopic Electronic Displays." StereoGraphics Corporation, SPIE vol. 1083 pp. 53-58.

29. Miller, D. P., D. E. McGovern, 1988. "A Laboratory simulation approach to the evaluation of vision systems for teleoperated vehicles." Teleoperation and Control Proceedings of the International Symposium, IFS Publications, Bedford, UK. pp. 85-91

30. Pepper, R. L., 1986. "Human Factors in Remote Vehicle Control." Naval Oceans Systems Center, Human Factors Society, pp. 417-421.

31. Pepper, R. L., R. E. Cole, 1983. " The Influence of Camera Separation and Head Movement on Perceptual Performance Under Direct and TV-Displayed Conditions." Naval Oceans Systems Center, Proceedings of the SID, vol. 24 pp. 73-80.

32. Pepper, R. L., P. K. Kaomea, 1988. "Teleoperation: Telepresence and performance assessment." ." Teleoperation and Control Proceedings of the International Symposium, IFS Publications, Bedford, UK. pp. 227-234.

33. Pichler, C., K. Radermacher, W. Boeckmann, G. Rau, G. Jakse, 1996. "Three Dimensional versus Two-Dimensional Video Endoscopy." Proceedings of Medicine ` Meets Virtual Reality 4 pp. 667-674.

34. Robinson, M., P. Shuttleworth, 1988 "The development of stereoscopic vision systems for use in hazardous environments." Teleoperation and Control Proceedings of the International Symposium, IFS Publications, Bedford, UK. pp. 191-195

35. Sharkey, P., D. Murray, 1997. "Feasibility of Using Eye Tracking to Increase Resolution for Visual Telepresence." 1997 IEEE International Conference on Systems, Man and Cybernetics. v2. pp. 1078-1083.

36. Sheridan, T. B. , 1988 "Teleoperation Scorecard: Deja vu and really new." Teleoperation and Control Proceedings of the International Symposium, IFS Publications, Bedford, UK. pp. 3-9

37. Shields, N., M. Kirkpatric, T. Malone, C. Huggins, G. Marshall, 1975. "Design Parameters for a Stereoptic System Based on Direct Vision Depth Perception Cues." ProceedingsHuman Factors Society 19th Annual Meeting pp. 423-435.

38. Schweiwiller, P., V. Reading, A. Dumbreck, E. Abel, 1988. "The effects of display flicker on task performance." Teleoperation and Control Proceedings of the International Symposium, IFS Publications, Bedford, UK. pp. 249-260.

39. Yamanoue, H., 1997. "The Relation Between Size Distortion and Shooting Conditions for Stereoscopic Images" SMPTE Journal, April. pp225-232.

40. Yeh, Y., L. Silverstein, 1992. "Spatial Judgments with Monoscopic and Stereoscopic Presentation of Perspective Displays." University of Wisconsin, Human Factors Society, pp. 583-599.

Appendix A. : LCC Time Based Program

```
/*          LCC.C  LOADER CONTROL PROGRAM WITH RATTLE FEATURE          */


#include        "stdio.h"
#include        "conio.h"
#include        "dos.h"
#include        "string.h"
#include       "stdlib.h"
#include        "c:\stebc31\ve.h"
#include        "c:\stebc31\ad.h"
#include        "c:\stebc31\ser1.h"

#define         OUTPUT_INTERVAL 300
#define         INPUT_INTERVAL  1000


#define         SSR             0xc101  /*High current digital outputs*/
#define         SSR2            0xc100  /*More high current digital outputs*/
#define         THROTTLE_DAC            0    // D/A channel for the throttle
#define        POWER_FAIL      10

/*      Machine input bit values                                        */

#define         LBRAKE            2
#define         LNEU              1
#define         OIL_PRESSURE      4
#define         AOK               8
#define         LCC_RTD           0x10
#define         LCC_TH            0x40
#define         LCC_DH            0x20
#define         FIRST_GEAR        0x10

//      OCC Input Block bit values

#define         OCC_ENG_START     1
#define         OCC_RTD           8
#define         OCC_TH            4
#define         OCC_DH            4
#define         OCC_FLOAT         0x40
#define         RATTLE            0x80
//#define         OCC_VUP           8
//#define         OCC_VDOWN         4
#define        PLAYBACK        8
#define        REC_STFIN       4
#define         OCC_VRIGHT        2
#define         OCC_VLEFT         1
#define         GEAR_MASK         0x30
#define         GEAR_1            0x10
#define         DIRECTION_MASK    0xc0


/*      Transmission output bits                                        */

#define         DIR_NEUTRAL       8
#define         REMOTE_ENABLE     0x40
#define         CLUTCH_ENGAGE     0x20
#define         CLUTCH_DISENGAGE 0xdf
#define        DIR_REV         4
#define        DIRECTION       0x0e

/*      SSR output bits                                                 */

#define         BRAKE_0           0xf8
#define         BRAKE_1           1
#define         BRAKE_2           3
#define         BRAKE_3           5
#define         ARM_UP            8
#define         ARM_DOWN          0x10
#define         ARM_STOP          0xe7
```

```
#define         ARM_FLOAT       0x30
#define         NO_FLOAT        0xdf
#define         ROLL_BACK       0x40
#define         DUMP            0x80
#define         BUCKET_STOP     0x3f

/*      SSR2 output bits                                                */

#define         THROTTLE_POWER  4
#define         STEER_RIGHT     1
#define         STEER_LEFT      2
#define         STEER_0         0xfc
#define         EMERG_STOP      0x20
#define         XMISSION_EN     0x40
#define         ENG_START       0x80
#define        VIDEO_SWITCH   8
#define        NO_VIDEO_SWITCH 0xf7


/*      Switching parameters for analog functions                       */

#define         UP_SWITCH_0     200
#define         DOWN_SWITCH_0   50
#define         RB_SWITCH_0     200
#define         DUMP_SWITCH_0   50
#define         BRAKE_1_SWITCH  80
#define         BRAKE_2_SWITCH  140
#define         BRAKE_3_SWITCH  200

       // serial port 1 variables
extern COM ser1_com;
extern COM *c1=&ser1_com;
unsigned char ser1_in_buf[1024],ser1_out_buf[1024],mode,baud;


/*      GOLBAL VARIABLES                                                */

int desired_steering,steering_angle,desired_brake,desired_throttle;
int arm_angle,bucket_angle,pf;
int desired_arm,desired_bucket,desired_gear,desired_direction;
int status_output,t,v;
float steering_slope=-.2623,steering_offset=607.44;
unsigned int i,input_timer,output_timer,aux_timer,rattle_timer,rtd;
unsigned int up_switch=UP_SWITCH_0,down_switch=DOWN_SWITCH_0;
char input_buffer[30],input_block_buffer[31],output_block_buffer[30];
char *output_msg="s0172\n\0";
char *out_buf_ptr_;
char *block_char_1;
unsigned char pt;
unsigned int input_count,block_count,loop_timer,max_loop_time;
unsigned int ssr,ssr2,xmission;
unsigned int rb_switch=RB_SWITCH_0,dump_switch=DUMP_SWITCH_0,brake_level=3;
unsigned int arm_height,desired_height,auto_arm,float_flag,rattle_flag;
int playkey,recordkey,inc,change,keychange1,keychange2,changeflag,incmax;
unsigned record,play,stoprecord,stopplay;
unsigned long int rec_counter,rec_counter_max;
unsigned rec[200][7];
unsigned long state_counter[200];


void interrupt far tb_isr(void);
void power_fail(void);
int ser_inp(void);
void ser_out(void);
int decode_input(void);

main()
{
int s;
unsigned arm_sensor;

/*      INITIALIZE VARIABLES    */

       ve_init();
       ad_init();
       outportb(0xc103,0x80);
```

```
        outportb(0xc100,0xff);
        outportb(0xc101,0xff);
        for(i=1;i<8;i++)ad_hv(i,0);
        portt_wr(8);
        mode=0xc9;
        baud=8;          //9600 baud
        s1_init(mode,baud,ser1_in_buf,1024,ser1_out_buf,1024,c1);
        time_base_init(1);
        time_base_interrupt(1,tb_isr);
        ssr=0;
        block_count=0;
        block_char_1=&input_block_buffer[0];
        input_timer=INPUT_INTERVAL;
        output_timer=OUTPUT_INTERVAL;
        aux_timer=5000;
        rattle_timer=0;
        rattle_flag=0;
        out_buf_ptr_=output_msg;
        pt=portt_rd();
/*
/
/       BEGIN PROGRAM FUNCTIONS
/


        /* Wait for receipt of 'Idle' message while continuing to
        monitor Remote switch and send 'Standby' message.  */

        /*Set full brakes, transmission neutral, remote mode  */
STANDBY:
                outportb(SSR,~BRAKE_3); //Apply full brakes
                outportb(SSR2,~(XMISSION_EN));//Send power to xmission mux
                xmission=(DIR_NEUTRAL|REMOTE_ENABLE); //Set xmission to neutral
                for(i=1;i<8;i++)ad_hv(i,(xmission>>i)&1);
        /* Wait for receipt of 'command' message and send 'system OK' message.  */
                v=0;
                while((v!=2)||(portt_rd()&LNEU)){
                /* return value of 2 indicates valid 'command' msg
                        Wait for 'command msg and loader in neutral  */
                        //if loader not in neutral, send 'n'
                          pt=portt_rd();
                         if(portt_rd()&LNEU)strcpy(output_block_buffer,"n0n");
                         // if loader is in neutral, send 's'
                         else strcpy(output_block_buffer,"s0s");
                         if(serhit1(c1))ser_inp();
                         if(block_count){
                         v=decode_input();
                         block_count=0;
                         }
                         if(output_timer==0){
                                ser_out();
                                output_timer=OUTPUT_INTERVAL;
                                }
                         pokeb(0xfff0,0x11,0x40); //set up to read the A/D
                         pokeb(0xfff0,0x10,0xf7);
                         //if lcc power has failed, execute power fail routine
                         ad_ad12(POWER_FAIL);
                         /*if(ad_ad12(POWER_FAIL)<800){
                                power_fail();
                                goto STANDBY;
                                }*/
                          ad_init();
                         }

        WAITING_FOR_OCC:
        strcpy(output_block_buffer,"r0r");
        ser_out();
        output_timer=OUTPUT_INTERVAL;
        while(v!=3){
        //  Waiting for OCC to take control
                if(portt_rd()&LNEU)goto STANDBY;
                if(serhit1(c1))ser_inp();
                if(block_count){
                        if((v=decode_input())==1)goto STANDBY;
                        block_count=0;
                        }
```

```
                if(output_timer==0){
                        ser_out();
                        output_timer=OUTPUT_INTERVAL;
                        }
                }


/*      BEGINNING OF REMOTE MODE        */


        REMOTE_ENTRY_POINT:
        max_loop_time=0;
        input_timer=INPUT_INTERVAL;
        rtd=0;
        arm_height=0;
        auto_arm=0;
        float_flag=0;
        strcpy(output_block_buffer,"r0r");
        output_block_buffer[3]=0x0d;
        output_block_buffer[4]=0;
        t=20;
/*Stay in remote mode until vehicle transmission lever is moved or
  OCC commands a return to standby */
        while(!(portt_rd()&LNEU)){
                if(loop_timer>max_loop_time)max_loop_time=loop_timer;
                loop_timer=0;
                // set up to read a/d converter
                pokeb(0xfff0,0x11,0x40);
                pokeb(0xfff0,0x10,0xf7);
                ad_ad12(6);
                steering_angle=ad_ad12(7);  //read ch 6, convert ch 7
                arm_angle=ad_ad12(8);  //read ch 7, convert ch 8
                bucket_angle=ad_ad12(POWER_FAIL);  //read ch 8, convert ch 10
                /*if((ad_ad12(POWER_FAIL)<800)&&(!ssr2&ENG_START)){
                        power_fail();
                        goto STANDBY;
                          }*/
                // return to original configuration
                ad_init();
                steering_angle=steering_angle*steering_slope+steering_offset;
                if(steering_angle<0)steering_angle=0;
                if(steering_angle>255)steering_angle=255;



/* If valid 'command' block is not received within 500 msec, goto STANDBY */
                if(input_timer==0)goto STANDBY;

/* Send 'system OK' message every 400 msec  */
                if(output_timer==0){
                        if(record==2) strcpy(output_block_buffer,"t0t"); //recording
mode
                        else if(play==2) strcpy(output_block_buffer,"l0l");
//playback mode
                          else strcpy(output_block_buffer,"r0r"); //standard remote
mode
                        output_timer=OUTPUT_INTERVAL;
                        ser_out();
                        }
/* when a new message is received from the field unit, call decode_input */
                if(serhit1(c1))ser_inp();
                if(block_count){
                        /*if it is a valid 'command' message, reset input timer */
                        if((v=decode_input())==3)input_timer=INPUT_INTERVAL;
                        if(v==10){
                                record=0;
                                play=0;
                                rec_counter=0;
                                inc=0;
                                  stopplay=0;
                                goto STANDBY; //Emergency Stop
                          }
                        block_count=0;
                }
```

```
                if(v==3) outportb(0xc102,1);
                 else outportb(0xc102,2);
                /* set gear and direction  */
                xmission=desired_gear+desired_direction|REMOTE_ENABLE;
                /* disengage clutch if brake level > 1, otherwise engage it  */
                if(brake_level>1)xmission&=CLUTCH_DISENGAGE;
                else xmission|=CLUTCH_ENGAGE;
                for(i=1;i<8;i++)ad_hv(i,(xmission>>i)&1); /* activate desired
transmission bits */

                /* check to see if steering angle is within 8 unit of correct
value.
                   If so, disable both steering solenoids, otherwise do nothing.
*/
                s=steering_angle-desired_steering;
                if((s<8)&&(s>-8))ssr2&=STEER_0;

                // Disengage starter if engine is running
                if(!(portt_rd()&OIL_PRESSURE))ssr2&=(~ENG_START);

 //  This section checks to see if the loader's arm should be moving
 //  automatically.

                if(auto_arm==2){
                        if(arm_height<desired_height)ssr=(ssr&ARM_STOP)|ARM_UP;
                        else
if(arm_height>desired_height)ssr=(ssr&ARM_STOP)|ARM_DOWN;
                        else ssr&=ARM_STOP,auto_arm=0;
                        }
                arm_sensor=(portt_rd()&(LCC_TH|LCC_DH));
                if(arm_sensor==LCC_TH)arm_height=1;
                else if(arm_sensor==LCC_DH)arm_height=3;
                else{
                        if(arm_height==1){
                                if(ssr&ARM_UP)arm_height=2;
                                else if(ssr&ARM_DOWN)arm_height=0;
                                }
                        else if(arm_height==3){
                                if(ssr&ARM_UP)arm_height=4;
                                else if(ssr&ARM_DOWN)arm_height=2;
                                }
                        }
                if((float_flag)&&!(auto_arm))ssr=(ssr&ARM_STOP)|ARM_FLOAT;
                if(!(float_flag))ssr&=NO_FLOAT;

                if((xmission&DIRECTION)==DIR_REV)ssr2|=VIDEO_SWITCH;
                 else ssr2&=NO_VIDEO_SWITCH;

                outportb(SSR,~ssr); /* activate SSR bits */
                outportb(SSR2,~ssr2);
                for(i=1;i<8;i++)ad_hv(i,(xmission>>i)&1);

                /* send desired throttle position to DAC channel 0 */
                ad_da12(0,desired_throttle);
                 outportb(0xc102,0);
                }
        /* When one of the conditions necessary for remote opperation is not
        met, return to the STANDBY mode.  */

        goto STANDBY;
        }


/*  This is the timebase counter interrupt service routine.
    Each of the timer variables is decremented every millisecond until
    a value of 0 is reached. */

void interrupt far tb_isr(void)
{
        if(input_timer)input_timer--;
        if(output_timer)output_timer--;
        if(aux_timer)aux_timer--;
        if(rattle_timer)rattle_timer--;
        if(v==3)loop_timer++;
         if(rec_counter)rec_counter++;
        fint();
```

```
        }
void power_fail(void)
{
        int pwr_fail=0;
        char ob_save[20];
        outportb(SSR2,~EMERG_STOP); //apply full brakes and kill engine
        outportb(SSR,0xff);
        strcpy(ob_save,output_block_buffer);//save last output block
         strcpy(output_block_buffer,"p0p");//signal LCC power failure
        ser_out();//send power fail message to occ
        pokeb(0xfff0,0x11,0x40);//set up to read A/D
        pokeb(0xfff0,0x10,0xf7);
        while(pwr_fail<1600){
                while(output_timer);//wait for output interval to expire
                output_timer=OUTPUT_INTERVAL;
                ser_out();
                ad_ad12(POWER_FAIL);
                pwr_fail=ad_ad12(POWER_FAIL);
                }
        strcpy(output_block_buffer,ob_save); //restore output_block_buffer
        ad_init();
         }


/*   This is the serial input service routine.  When a character is received,
     it is placed in the input buffer.  When the received character is a
     new line character, or the input buffer is full, the characters received up
     to that point are transferred to the block buffer and the block count is
     set to show the number of characters in the buffer.  The input count is
     then set to 0 and the process starts over.  */

int ser_inp(void)
{
     while(serhit1(c1)){
             if((input_buffer[input_count]=getser1(c1))==13){
                     block_count=input_count+1;
                     input_count=0;
                     input_buffer[block_count]=0;
                     strcpy(input_block_buffer,input_buffer);
                     }
             else input_count++;
             }
        return block_count;
        }

void ser_out(void)
{
        for(i=0;i<3;i++)while(!(putser1(output_block_buffer[i],c1)));
        while(!(putser1(0x0d,c1)));
        }


/*  The decode_input() function evaluates the received input block and returns
    a value as follows:  1=valid 'keep alive' message, 2=OCC asking for control,
    3=valid control message received, -1=incorrect character count, -2=checksum
    error, -3=first character was not a legal command, 10=Emergency Stop.
    If the input block is a valid command message, the function decodes it and
    sets the appropriate bits in the system control variables .     */

int decode_input(void)
{
         unsigned return_val=0,checksum=0,i,ary[20];

        if((block_count!=3)&&(block_count!=21))return -1;
        for(i=0;i<(block_count-2);i++)checksum+=input_block_buffer[i];
        if((checksum&0xf)!=(input_block_buffer[i]&0xf))return -2;
        switch(input_block_buffer[0]){
        case 'e': //Emergency stop
                return_val=10;
                break;
        case 'i': //Keep Alive message
                if(block_count==3)return_val=1;
                else return_val=-3;
                break;
        case 'p': // OCC asking for control
                if(block_count==3)return_val=2;
```

```
                else return_val=-3;
                break;
        case 'c': //Command message
                if(block_count!=21)return_val=-3;
                else{
                        return_val=3;
                        for(i=1;i<19;i+=2)ary[(i-
1)/2]=((input_block_buffer[i]&0xf)<<4)|(input_block_buffer[i+1]&0xf);


                        /* Playback section - to repeat recorded operator
information */

                        switch(play){
                                case 0:  /* No play action -- wait for button press
*/
                                        if(ary[7]&0xf&PLAYBACK) play=1;
                                        break;
                                case 1:  /* Play button pressed, wait for release */
                                        if(ary[7]&0xf&PLAYBACK) break;
                                        else{
                                                play=2;
                                                rec_counter=1;
                                        }
                                        break;
                                case 2:  /* Enter playback mode */
                                        /* Terminate playback if playback button
presses */
                                        switch(stopplay){
                                                case 0:
                                                        if(ary[7]&0xf&PLAYBACK)
stopplay=1;
                                                        break;
                                                case 1:
                                                        if(ary[7]&0xf&PLAYBACK) break;
                                                        else{
                                                                play=0;
                                                                rec_counter=0;
                                                                inc=0;
                                                                stopplay=0;
                                                        }
                                                        break;
                                                default:
                                                        stopplay=0;
                                        }

                                        /*Check to see if it is time to step forward in the
recorded
                                          array of operartions */

if((inc<incmax)&&(state_counter[inc+1]<=rec_counter)) inc++;

                                        /*Substitute recorded control values*/
                                        for(i=0;i<=6;i++) ary[i]=rec[inc][i];

                                        /*If counter has reached the maximum value from the
                                        record action, then stop playback */
                                        if(rec_counter>=rec_counter_max){
                                                play=0;
                                                rec_counter=0;
                                                inc=0;
                                        }

                                        break;
                                default:
                                        play=0;
                                  }

                                /* set desired control actions */
                                desired_steering=ary[0];
                                desired_brake=ary[1];
                                desired_throttle=ary[2];
                                if(desired_throttle>180)desired_throttle=180;
                                desired_arm=ary[3];
                                desired_bucket=ary[4];
```

```
                        /*Record section - OCC movements are monitored and stored in
                          an array */
                        switch(record){
                                case 0:  /*No record action -- wait for button press
*/
                                        if(ary[7]&0xf&REC_STFIN) record=1;
                                        break;
                                case 1:  /*record button pressed, wait for release
*/
                                        if(ary[7]&0xf&REC_STFIN) break;
                                        else{
                                                rec_counter=1;
                                                record=2;
                                                inc=0;

                                                //get initial values
                                                   for(i=0;i<=6;i++)
rec[inc][i]=ary[i];
                                                   state_counter[inc]=rec_counter;
                                        }
                                        break;
                                case 2:  /*enter record mode */
                                        /*see if record start/finish button has been
                                          pressed a second time */
                                        switch(stoprecord){
                                                case 0:
                                                        if(ary[7]&0xf&REC_STFIN)
stoprecord=1;
                                                        break;
                                                case 1:
                                                        if(ary[7]&0xf&REC_STFIN)
break;
                                                        else{
                                                                incmax=inc;
                                                                record=0;

        rec_counter_max=rec_counter;
                                                                rec_counter=0;
                                                                inc=0;
                                                                stoprecord=0;
                                                        }
                                                        break;
                                                default:
                                                        stoprecord=0;
                                        }

                                        //initialize flag
                                        changeflag=0;

                                        /*if the maximum number of allowable
operation changes
                                          has been reached, terminate recording */
                                        if(inc>200){
                                                record=0;
                                                incmax=inc;
                                                rec_counter_max=rec_counter;
                                                rec_counter=0;
                                                inc=0;
                                        }
                                        else if(rec_counter!=0){
                                                /*check to see if operator is
changing anything,
                                                  and if so, update the rec[][]
array*/

                                                keychange1=abs(rec[inc][5]-ary[5]);
                                                keychange2=abs(rec[inc][6]-ary[6]);
                                                if((keychange1>=4)||(keychange2>=4))
changeflag=1;
                                                for(i=0;i<=6;i++){
                                                        change=abs(rec[inc][i]-
ary[i]);
                                                        if(change>=20) changeflag=1;
                                                }
                                                if(changeflag==1){
                                                        inc++;
```

```
                                                    for(i=0;i<=6;i++)
rec[inc][i]=ary[i];

state_counter[inc]=rec_counter;
                                                }
                                                /*check to see if rec_counter has
reached max
                                                  allowable value, terminate if so */
                                                   if(rec_counter>=90000){
                                                        record=0;
                                                        incmax=inc;
                                                        rec_counter_max=rec_counter;
                                                        rec_counter=0;
                                                        inc=0;
                                                }
                                        }
                                        break;
                                default:
                                        record=0;
                         }

                        if((ary[5]&GEAR_MASK)==GEAR_1)desired_gear=FIRST_GEAR;
                        else desired_gear=0;

//  Check desired direction
                        desired_direction=(ary[5]&DIRECTION_MASK)>>5;

/* if neither FWD or REV selected, set direction NEUTRAL */
                if(desired_direction==0)desired_direction=8;


/*Joystick function selection.
If the command value is between the switch points, the solenoids
are disabled.  If the value exceeds one of the base switch points,
the switch point is moved 2 units toward the center to provide
hysteresis and the appropriate solenoid is activated. */

                if(desired_arm>up_switch){
                        up_switch=UP_SWITCH_0-10;
                        down_switch=DOWN_SWITCH_0;
                        ssr=(ssr&ARM_STOP)|ARM_UP;
                        auto_arm=0;
                        float_flag=0;
                        }
                else if(desired_arm<down_switch){
                        up_switch=UP_SWITCH_0;
                        down_switch=DOWN_SWITCH_0+10;
                        ssr=(ssr&ARM_STOP)|ARM_DOWN;
                        auto_arm=0;
                        float_flag=0;
                        }
                else{
                        up_switch=UP_SWITCH_0;
                        down_switch=DOWN_SWITCH_0;
                        ssr&=ARM_STOP;
                        }

                 if(ary[6]&RATTLE){
                        if(!(rattle_timer)){
                                if(rattle_flag)ssr=ssr^0xc0;
                                else{
                                        rattle_flag=1;
                                        ssr=(ssr&BUCKET_STOP)|ROLL_BACK;
                                        }
                                rattle_timer=200;
                                }
                        }
                 else{
                        rattle_flag=0;
                        if(desired_bucket>rb_switch){
                                rb_switch=RB_SWITCH_0-2;
                                dump_switch=DUMP_SWITCH_0;
                                ssr=(ssr&BUCKET_STOP)|ROLL_BACK;
                                rtd=0;  /*terminate return to dig if enabled */
                                }
                        else if(desired_bucket<dump_switch){
```

```
                        rb_switch=RB_SWITCH_0;
                        dump_switch=DUMP_SWITCH_0+2;
                        ssr=(ssr&BUCKET_STOP)|DUMP;
                        rtd=0;  /*terminate return to dig if enabled */
                        }
                else{
                        rb_switch=RB_SWITCH_0;
                        dump_switch=DUMP_SWITCH_0;
                        ssr&=BUCKET_STOP;
                        }
                }
        switch(rtd){
        case 0:  /*No return to dig action--wait for button press */
                if(ary[5]&OCC_RTD)rtd=1;
                break;
        case 1:  /*rtd button pressed, wait for release  */
                if(ary[5]&OCC_RTD)break;
                /*determine if bucket should roll back or dump */
                if((portt_rd()&LCC_RTD))rtd=2; /*dump */
                else rtd=3;  /*roll back */
        case 2:  /*bucket must first dump to clear sensor and then roll
back */
                if((portt_rd()&LCC_RTD))ssr=(ssr&BUCKET_STOP)|DUMP;
                else rtd=3;
                break;
        case 3:  /*bucket must roll back until sensor encountered */
                if((portt_rd()&LCC_RTD)){
                        ssr&=BUCKET_STOP;
                        rtd=0; /* bucket in dig position */
                        }
                else ssr=(ssr&BUCKET_STOP)|ROLL_BACK;
                break;
        default:
                rtd=0;
                ssr&=BUCKET_STOP;
        }

        switch(auto_arm){
        case 0:  /*No movement--wait for TH or DH to be pressed */
                if((ary[5]&OCC_TH)|(ary[6]&OCC_DH)){
                        auto_arm=1;
                        if(ary[5]&OCC_TH){
                                if(arm_height)desired_height=0;
                                else desired_height=1;
                                }
                        else if(arm_height<3)desired_height=3;
                        else desired_height=2;
                        }
                break;
        case 1:  /*TH or DH pressed, wait for release */
                if(!((ary[5]&OCC_TH)|(ary[6]&OCC_DH)))auto_arm=2;
                break;
        case 2: /*in motion */
                break;
        default:
                auto_arm=0;
        }
    if((ary[6]&OCC_FLOAT)&&!(float_flag)){
            float_flag=1;
            if(arm_height){
                    desired_height=0;
                    auto_arm=2;
                    }
            }

            /*Brake Level Selection
              The variable brake_level determines the current braking level.
The
              switching points of that level are adjusted so as to widen
              the current level to provide hysteresis.  If the value of
              desired_brake is outside of one of these switch points, the
              value of brake_level is incremented or decremented by 1.*/


            switch(brake_level){
            case 3: /* Highest level, check lower switch point only.  */
```

```
                        if(desired_brake<(BRAKE_3_SWITCH-10)){
                                brake_level=2; /*move to brake level 2 */
                                ssr=(ssr&BRAKE_0)|BRAKE_2;
                                }
                        /* otherwise, remain in level 3 */
                        else ssr=(ssr&BRAKE_0)|BRAKE_3;
                        break;
                case 2:
                        if(desired_brake<(BRAKE_2_SWITCH-10)){
                                brake_level=1;/* move down to level 1 */
                                ssr=(ssr&BRAKE_0)|BRAKE_1;
                                }
                        else if(desired_brake>=(BRAKE_3_SWITCH+5)){
                                brake_level=3; /* move up to level 3 */
                                ssr=(ssr&BRAKE_0)|BRAKE_3;
                                }
                        /* otherwise, remain in level 2 */
                        else ssr=(ssr&BRAKE_0)|BRAKE_2;
                        break;
                case 1:
                        if(desired_brake<(BRAKE_1_SWITCH-10)){
                                brake_level=0; /*move down to level 0 */
                                ssr=ssr&BRAKE_0;
                                }
                        else if(desired_brake>=(BRAKE_2_SWITCH+5)){
                                brake_level=2;/* move up to level 2 */
                                ssr=(ssr&BRAKE_0)|BRAKE_2;
                                }
                        /* otherwise, remain in level 1 */
                        else ssr=(ssr&BRAKE_0)|BRAKE_1;
                        break;
                case 0: /* no braking action at all */
                        if(desired_brake>(BRAKE_1_SWITCH+5)){
                                brake_level=1; /*move up to level 1 */
                                ssr=(ssr&BRAKE_0)|BRAKE_1;
                                }
                        /* otherwise, remain in level 0 */
                        else ssr=ssr&BRAKE_0;
                        break;
                default:brake_level=3;
                }

                /*  Compute throttle position value */
                desired_throttle=4095-(desired_throttle*6);
        // send power to throttle actuator and transmission mux
                ssr2|=THROTTLE_POWER|XMISSION_EN;
                /* Check engine start switch (transmission must be in neutral) */
                if
((ary[6]&OCC_ENG_START)&&(desired_direction==8)&&(portt_rd()&OIL_PRESSURE))ssr2|=E
NG_START;
                else ssr2&=(~ENG_START);

                /* if desired steering differs from machine angle by 20 units
                   or more, activate appropriate steering solenoid.  */
                if((desired_steering-steering_angle)>20){
                        ssr2=(ssr2&STEER_0)|STEER_RIGHT;
                        }
                else if((desired_steering-steering_angle)<-20){
                        ssr2=(ssr2&STEER_0)|STEER_LEFT;
                        }
                }
                break;
        default:return_val=-3;}
        return return_val;
        }
```

Appendix B: LCC Position Based Program

```
/*          LCC.C  LOADER CONTROL PROGRAM WITH RATTLE FEATURE                */


#include        "stdio.h"
#include        "conio.h"
#include        "dos.h"
#include        "string.h"
#include      "stdlib.h"
#include        "c:\stebc31\ve.h"
#include        "c:\stebc31\ad.h"
#include        "c:\stebc31\ser1.h"

#define         OUTPUT_INTERVAL 300
#define         INPUT_INTERVAL  1000


#define         SSR         0xc101  /*High current digital outputs*/
#define         SSR2        0xc100  /*More high current digital outputs*/
#define         THROTTLE_DAC  0     // D/A channel for throttle control
#define         POWER_FAIL   10     // A/D channel for power fail detect

/*      Machine input bit values                                             */

#define         LNEU            1
#define         LBRAKE          2
#define         OIL_PRESSURE    4
#define         AOK             8
#define         LCC_RTD         0x10
#define         LCC_TH          0x40
#define         LCC_DH          0x20
#define         FIRST_GEAR      0x10

//      OCC Input Block bit values

#define         OCC_ENG_START   1
#define         OCC_RTD         8
#define         OCC_TH          4
#define         OCC_DH          4
#define         OCC_FLOAT       0x40
#define         RATTLE          0x80
#define        REC_STFIN      4
#define        PLAYBACK       8
#define         GEAR_MASK       0x30
#define         GEAR_1          0x10
#define         DIRECTION_MASK  0xc0


/*      Transmission output bits                                             */

#define         DIR_REV         4
#define         DIR_NEUTRAL     8
#define         DIRECTION       0x0e
#define         REMOTE_ENABLE   0x40
#define         CLUTCH_ENGAGE   0x20
#define         CLUTCH_DISENGAGE 0xdf

/*      SSR output bits                                                      */

#define         BRAKE_0         0xf8
#define         BRAKE_1         1
#define         BRAKE_2         3
#define         BRAKE_3         5
#define         ARM_UP          8
#define         ARM_DOWN        0x10
#define         ARM_STOP        0xe7
#define         ARM_FLOAT       0x30
#define         NO_FLOAT        0xdf
#define         ROLL_BACK       0x40
```

```
#define         DUMP            0x80
#define         BUCKET_STOP     0x3f

/*      SSR2 output bits                                                */

#define         STEER_RIGHT     1
#define         STEER_LEFT      2
#define         STEER_0         0xfc
#define         THROTTLE_POWER  4
#define         VIDEO_SWITCH    8
#define         NO_VIDEO_SWITCH  0xf7
#define         EMERG_STOP      0x20
#define         XMISSION_EN     0x40
#define         ENG_START       0x80


/*      Switching parameters for analog functions                       */

#define         UP_SWITCH_0     200
#define         DOWN_SWITCH_0   50
#define         RB_SWITCH_0     200
#define         DUMP_SWITCH_0   50
#define         BRAKE_1_SWITCH  80
#define         BRAKE_2_SWITCH  140
#define         BRAKE_3_SWITCH  200

       // serial port 1 variables
extern COM ser1_com;
extern COM *c1=&ser1_com;
unsigned char ser1_in_buf[1024],ser1_out_buf[1024],mode,baud;


/*      GOLBAL VARIABLES                                                */

int desired_steering,steering_angle,desired_brake,desired_throttle;
int arm_position,bucket_position,pf;
int desired_arm,desired_bucket,desired_gear,desired_direction;
int status_output,t,v;
float steering_slope=-.2623,steering_offset=607.44;
unsigned int i,input_timer,output_timer,aux_timer,rattle_timer,rtd;
unsigned int up_switch=UP_SWITCH_0,down_switch=DOWN_SWITCH_0;
char input_buffer[30],input_block_buffer[31],output_block_buffer[30];
char *output_msg="s0172\n\0";
char *out_buf_ptr_;
char *block_char_1;
unsigned char pt;
unsigned int input_count,block_count,loop_timer,max_loop_time;
unsigned int ssr,ssr2,xmission;
unsigned int rb_switch=RB_SWITCH_0,dump_switch=DUMP_SWITCH_0,brake_level=3;
unsigned int arm_height,desired_height,auto_arm,float_flag,rattle_flag;
int
check_steering,check_bucket,check_arm,arm_flag,bucket_flag,steering_flag,pause_fla
g;
int steering_trim_check,arm_trim_check,bucket_trim_check;
int steering_trim_flag,arm_trim_flag,bucket_trim_flag,trim_flag;
int steering_trim,bucket_trim,arm_trim,temp1,temp2,temp3;
int pause_checker_1,pause_checker_2,pause_checker_3;
unsigned record,stoprecord,play,stopplay,controls[20];
unsigned long rec_counter,check_counter,play_counter,pause_counter;
int state,final_state;
float bucket_slope=.0711,bucket_offset=33.137,arm_slope=.2037,arm_offset=284.74;
int rec[200][6],rec_original[200][6];



void interrupt far tb_isr(void);
void power_fail(void);
int ser_inp(void);
void ser_out(void);
int decode_input(void);

main()
{
int s;
unsigned arm_sensor;
```

```
/*      INITIALIZE VARIABLES    */

        ve_init();
        ad_init();
        outportb(0xc103,0x80);
        outportb(0xc100,0xff);
        outportb(0xc101,0xff);
        for(i=1;i<8;i++)ad_hv(i,0);
        portt_wr(8);
        mode=0xc9;
        baud=8;         //9600 baud
        s1_init(mode,baud,ser1_in_buf,1024,ser1_out_buf,1024,c1);
        time_base_init(1);
        time_base_interrupt(1,tb_isr);
        ssr=0;
        block_count=0;
        block_char_1=&input_block_buffer[0];
        input_timer=INPUT_INTERVAL;
        output_timer=OUTPUT_INTERVAL;
        aux_timer=5000;
        rattle_timer=0;
        rattle_flag=0;
        out_buf_ptr_=output_msg;
        steering_trim_flag=0;
        arm_trim_flag=0;
        bucket_trim_flag=0;
        trim_flag=0;
        pause_flag=0;
        pause_counter=0;
        pt=portt_rd();
/*
/
/       BEGIN PROGRAM FUNCTIONS
/


        /* Wait for receipt of 'Idle' message while continuing to
        monitor Remote switch and send 'Standby' message.  */

        /*Set full brakes, transmission neutral, remote mode  */
STANDBY:
                outportb(SSR,~BRAKE_3); //Apply full brakes
                outportb(SSR2,~(XMISSION_EN));//Send power to xmission mux
                xmission=(DIR_NEUTRAL|REMOTE_ENABLE); //Set xmission to neutral
                for(i=1;i<8;i++)ad_hv(i,(xmission>>i)&1);
        /* Wait for receipt of 'command' message and send 'system OK' message.  */
                v=0;
                while((v!=2)||(portt_rd()&LNEU)){
                // return value of 2 indicates valid 'command' msg
                        // if loader not in neutral, send 'n'
                        pt=portt_rd();
                        if(portt_rd()&LNEU)strcpy(output_block_buffer,"n0n");
                        // if loader is in neutral, send 's'
                        else strcpy(output_block_buffer,"s0s");
                        if(serhit1(c1))ser_inp();
                        if(block_count){
                               v=decode_input();
                               block_count=0;
                               }
                        if(output_timer==0){
                               ser_out();
                               output_timer=OUTPUT_INTERVAL;
                               }
                        pokeb(0xfff0,0x11,0x40); // set up to read A/D
                        pokeb(0xfff0,0x10,0xf7);
                // if lcc power has failed, execute power fail routine
                        ad_ad12(POWER_FAIL);
                /*          if(ad_ad12(POWER_FAIL)<800){
                               power_fail();
                               goto STANDBY;
                                 }                  */
                        ad_init();
                        }
        WAITING_FOR_OCC:
        strcpy(output_block_buffer,"r0r");
        ser_out();
```

```
        output_timer=OUTPUT_INTERVAL;
        while(v!=3){
        //  Waiting for OCC to take control
                if(portt_rd()&LNEU)goto STANDBY;
                if(serhit1(c1))ser_inp();
                if(block_count){
                        if((v=decode_input())==1)goto STANDBY;
                        block_count=0;
                        }
                if(output_timer==0){
                        ser_out();
                        output_timer=OUTPUT_INTERVAL;
                        }
                }


/*      BEGINNING OF REMOTE MODE        */


        REMOTE_ENTRY_POINT:
        max_loop_time=0;
        input_timer=INPUT_INTERVAL;
        rtd=0;
        arm_height=0;
        auto_arm=0;
        float_flag=0;
        strcpy(output_block_buffer,"r0r");
        output_block_buffer[3]=0x0d;
        output_block_buffer[4]=0;
        t=20;
/*Stay in remote mode until vehicle transmission lever is moved or
  OCC commands a return to standby */
        while(!(portt_rd()&LNEU)){
                if(loop_timer>max_loop_time)max_loop_time=loop_timer;
                loop_timer=0;
                // set up to read a/d converter
                pokeb(0xfff0,0x11,0x40);
                pokeb(0xfff0,0x10,0xf7);
                ad_ad12(6);
                steering_angle=ad_ad12(7);  //read ch 6, convert ch 7
                arm_position=ad_ad12(8);  //read ch 7, convert ch 8
                bucket_position=ad_ad12(POWER_FAIL);  //read ch 8, convert ch 10
        /*          if((ad_ad12(POWER_FAIL)<800)&&(!ssr2&ENG_START)){
                        power_fail();
                        goto STANDBY;
                          }                     */
                // return to original configuration
                ad_init();

                 //convert raw steering data to 0-255 scale
                steering_angle=steering_angle*steering_slope+steering_offset;
                if(steering_angle<0)steering_angle=0;
                if(steering_angle>255)steering_angle=255;
                //convert raw arm position data to 0-255 scale
                arm_position=arm_position*arm_slope-arm_offset;
                if(arm_position<0)arm_position=0;
                if(arm_position>255)arm_position=255;
                //convert raw bucket position data to 0-255 scale
                if(bucket_position<0)bucket_position=0;
                if(bucket_position>255)bucket_position=255;



/* If valid 'command' block is not received within 500 msec, goto STANDBY */
                if(input_timer==0)goto STANDBY;

/* Send 'system OK' message every 400 msec  */
                if(output_timer==0){
                        if(record==2)strcpy(output_block_buffer,"t0t");//recording
mode
                        else if(play==2)strcpy(output_block_buffer,"l0l");//playback
mode
                          else strcpy(output_block_buffer,"r0r");//standard remote
mode
```

```
                        output_timer=OUTPUT_INTERVAL;
                        ser_out();
                        }
/* when a new message is received from the field unit, call decode_input */
                if(serhit1(c1))ser_inp();
                if(block_count){
                        /*if it is a valid 'command' message, reset input timer */
                        if((v=decode_input())==3)input_timer=INPUT_INTERVAL;
                        if(v==10){//Emergency Stop -reinitialize teachable functions
                                record=0;
                                play=0;
                                state=0;
                                stopplay=0;
                                goto STANDBY;
                          }
                        block_count=0;
                        }
                if(v==3)outportb(0xc102,1);
                else outportb(0xc102,2);
                /* set gear and direction  */
                xmission=desired_gear+desired_direction|REMOTE_ENABLE;
                /* disengage clutch if brake level > 1, otherwise engage it  */
                if(brake_level>1)xmission&=CLUTCH_DISENGAGE;
                else xmission|=CLUTCH_ENGAGE;
                for(i=1;i<8;i++)ad_hv(i,(xmission>>i)&1); /* activate desired
transmission bits */

                /* check to see if steering angle is within 8 unit of correct
value.
                   If so, disable both steering solenoids, otherwise do nothing.
*/
                s=steering_angle-desired_steering;
                if((s<8)&&(s>-8))ssr2&=STEER_0;

                // Disengage starter if engine is running
                if(!(portt_rd()&OIL_PRESSURE))ssr2&=(~ENG_START);

 //  This section checks to see if the loader's arm should be moving
 //  automatically.

                if(auto_arm==2){
                        if(arm_height<desired_height)ssr=(ssr&ARM_STOP)|ARM_UP;
                        else
if(arm_height>desired_height)ssr=(ssr&ARM_STOP)|ARM_DOWN;
                        else ssr&=ARM_STOP,auto_arm=0;
                        }
                arm_sensor=(portt_rd()&(LCC_TH|LCC_DH));
                if(arm_sensor==LCC_TH)arm_height=1;
                else if(arm_sensor==LCC_DH)arm_height=3;
                else{
                        if(arm_height==1){
                                if(ssr&ARM_UP)arm_height=2;
                                else if(ssr&ARM_DOWN)arm_height=0;
                                }
                        else if(arm_height==3){
                                if(ssr&ARM_UP)arm_height=4;
                                else if(ssr&ARM_DOWN)arm_height=2;
                                }
                        }
                if((float_flag)&&!(auto_arm))ssr=(ssr&ARM_STOP)|ARM_FLOAT;
                if(!(float_flag))ssr&=NO_FLOAT;

                if((xmission&DIRECTION)==DIR_REV)ssr2|=VIDEO_SWITCH;
                else ssr2&=NO_VIDEO_SWITCH;

                outportb(SSR,~ssr); /* activate SSR bits */
                outportb(SSR2,~ssr2);
                for(i=1;i<8;i++)ad_hv(i,(xmission>>i)&1);

                /* send desired throttle position to DAC channel 0 */
                ad_da12(THROTTLE_DAC,desired_throttle);
                outportb(0xc102,0);
                }
        /* When one of the conditions necessary for remote opperation is not
        met, return to the STANDBY mode.  */
```

```
        goto STANDBY;
        }

/*  This is the timebase counter interrupt service routine.
    Each of the timer variables is decremented every millisecond until
    a value of 0 is reached. */

void interrupt far tb_isr(void)
{
        if(input_timer)input_timer--;
        if(output_timer)output_timer--;
        if(aux_timer)aux_timer--;
        if(rattle_timer)rattle_timer--;
        if(v==3)loop_timer++;
        if(rec_counter)rec_counter++;
        if(play_counter)play_counter++;
        if(pause_counter)pause_counter++;
        fint();
        }
void power_fail(void)
{
        int pwr_fail=0;
        char ob_save[20];
        outportb(SSR2,~EMERG_STOP); //Apply full brakes and kill engine
        outportb(SSR,0xff);
        strcpy(ob_save,output_block_buffer);// save last output_block
        strcpy(output_block_buffer,"p0p"); // Signal LCC power failure
        ser_out(); // Send power fail message to occ
        pokeb(0xfff0,0x11,0x40);// set up to read A/D
        pokeb(0xfff0,0x10,0xf7);
        while(pwr_fail<1600){
                while(output_timer); //wait for output interval to expire
                output_timer=OUTPUT_INTERVAL;
                ser_out();
                ad_ad12(POWER_FAIL);
                pwr_fail=ad_ad12(POWER_FAIL);
                }
        strcpy(output_block_buffer,ob_save); //restore output_block_buffer
        ad_init();
        }


/*   This is the serial input service routine.  When a character is received,
     it is placed in the input buffer.  When the received character is a
     new line character, or the input buffer is full, the characters received up
     to that point are transferred to the block buffer and the block count is
     set to show the number of characters in the buffer.  The input count is
     then set to 0 and the process starts over.  */

int ser_inp(void)
{
        while(serhit1(c1)){
                if((input_buffer[input_count]=getser1(c1))==13){
                        block_count=input_count+1;
                        input_count=0;
                        input_buffer[block_count]=0;
                        strcpy(input_block_buffer,input_buffer);
                        }
                else input_count++;
                }
        return block_count;
        }

void ser_out(void)
{
        for(i=0;i<3;i++)while(!(putser1(output_block_buffer[i],c1)));
        while(!(putser1(0x0d,c1)));
        }


/*  The decode_input() function evaluates the received input block and returns
    a value as follows:  1=valid 'keep alive' message, 2=OCC asking for control,
    3=valid control message received, -1=incorrect character count, -2=checksum
    error, -3=first character was not a legal command, 10=Emergency Stop.
```

```
   If the input block is a valid command message, the function decodes it and
   sets the appropriate bits in the system control variables .    */

int decode_input(void)
{
       unsigned return_val=0,checksum=0,i,j,ary[20];
       if((block_count!=3)&&(block_count!=21))return -1;
       for(i=0;i<(block_count-2);i++)checksum+=input_block_buffer[i];
       if((checksum&0xf)!=(input_block_buffer[i]&0xf))return -2;
       switch(input_block_buffer[0]){
       case 'e': //Emergency stop
              return_val=10;
              break;
       case 'i': //Keep Alive message
              if(block_count==3)return_val=1;
              else return_val=-3;
              break;
       case 'p': // OCC asking for control
              if(block_count==3)return_val=2;
              else return_val=-3;
              break;
       case 'c': //Command message
              if(block_count!=21)return_val=-3;
              else{
                     return_val=3;
                     for(i=1;i<19;i+=2)ary[(i-
1)/2]=((input_block_buffer[i]&0xf)<<4)|(input_block_buffer[i+1]&0xf);
                     desired_steering=ary[0];
                     desired_brake=ary[1];
                     desired_throttle=ary[2];
                     desired_arm=ary[3];
                     desired_bucket=ary[4];

                     /*Record Section - store component position data every
                       set time interval */
                     switch(record){
                            case 0:/*No record action -- wait for button press*/
                                   if(ary[7]&0xf&REC_STFIN)record=1;
                                   break;
                            case 1:/*record button pressed, wait for release*/
                                   if(ary[7]&0xf&REC_STFIN) break;
                                   else{
                                          record=2;
                                          state=0;
                                          //get intial state vector information
                                          rec[state][0]=steering_angle;
                                          rec[state][1]=arm_position;
                                          rec[state][2]=bucket_position;
                                          rec_counter=1;
                                   }
                                   break;
                            case 2: /*enter record mode*/
                                   //see if record start/finish button has been
pressed
                                   //a second time
                                   switch(stoprecord){
                                          case 0:

       if(ary[7]&0xf&REC_STFIN)stoprecord=1;
                                                 break;
                                          case 1:
                                                 if(ary[7]&0xf&REC_STFIN)
break;
                                                 else{
                                                        state++; //get final
state info

       rec[state][0]=steering_angle;

       rec[state][1]=arm_position;

       rec[state][2]=bucket_position;
                                                        final_state=state;
                                                        //save original
recording
```

```
                for(i=0;i<=final_state;i++){
                for(j=0;j<=2;j++){
                rec_original[i][j]=rec[i][j];
                                                                        }
                                                                }

                                                                record=0;
                                                                rec_counter=0;
                                                                check_counter=1000;
                                                                state=0;
                                                                stoprecord=0;
                                                        }
                                                        break;
                                                default:
                                                        stoprecord=0;
                                                }

                                                //update state vector array each second
                                                if(rec_counter>=check_counter){
                                                        check_counter+=1000;
                                                        state++;
                                                        rec[state][0]=steering_angle;
                                                        rec[state][1]=arm_position;
                                                        rec[state][2]=bucket_position;
                                                }
                                                //stop record action if time limit is reached
                                                if(rec_counter>=90000){
                                                        state++; //get final state info
                                                        rec[state][0]=steering_angle;
                                                        rec[state][1]=arm_position;
                                                        rec[state][2]=bucket_position;
                                                        record=0;
                                                        rec_counter=0;
                                                        check_counter=1000;
                                                        final_state=state;
                                                        state=0;
                                                }
                                                break;
                                        default:
                                                record=0;
                                }
                                if((ary[5]&GEAR_MASK)==GEAR_1)desired_gear=FIRST_GEAR;
                                else desired_gear=0;

//  Check desired direction
                                desired_direction=(ary[5]&DIRECTION_MASK)>>5;

/* if neither FWD or REV selected, set direction NEUTRAL */
                if(desired_direction==0)desired_direction=8;

                /*Playback Section - step loader through recorded state position
information*/
                switch(play){
                        case 0://No play action - wait for button press
                                if(ary[7]&0xf&PLAYBACK)play=1;
                                break;
                        case 1://play button pressed, wait for release
                                if(ary[7]&0xf&PLAYBACK) break;
                                else{
                                        play=2;
                                        //get initial position of controls on OCC
                                        controls[0]=128;
                                        controls[3]=128;
                                        controls[4]=128;
                                }
                                break;
                        case 2://enter playback mode
                                //check if terminate playback button has been
pressed
                                switch(stopplay){
                                        case 0:
                                                if(ary[7]&0xf&PLAYBACK)stopplay=1;
                                                break;
```

```
                case 1:
                        if(ary[7]&0xf&PLAYBACK)break;
                        else{
                                play=0;
                                state=0;
                                stopplay=0;
                                //restore original recording
                                for(i=0;i<=final_state;i++){
                                        for(j=0;j<=2;j++){

rec[i][j]=rec_original[i][j];
                                        }
                                }
                        }
                        break;
                default:
                        stopplay=0;
        }
        play_counter=1;
        //determine if operator is attempting trim operation
        steering_trim_check=abs(ary[0]-controls[0]);
        if(steering_trim_check<=20)steering_trim_flag=1;
        else steering_trim_flag=0;

        arm_trim_check=abs(ary[3]-controls[3]);
        if(arm_trim_check>=20)arm_trim_flag=1;
        else arm_trim_flag=0;

        bucket_trim_check=abs(ary[4]-controls[4]);
        if(bucket_trim_check>=20)bucket_trim_flag=1;
        else bucket_trim_flag=0;

        //if operator has completed the trimming operation,
determine
        //what was changed and modify downstream state data
accordingly
        temp1=!steering_trim_flag;
        temp2=!arm_trim_flag;
        temp3=!bucket_trim_flag;

        if(trim_flag&&temp1&&temp2&&temp3){
                trim_flag=0;
                steering_trim=steering_angle-rec[state][0];
                arm_trim=arm_position-rec[state][1];
                bucket_trim=bucket_position-rec[state][2];

                for(i=state;i<=final_state-1;i++){
                        rec[i][0]=rec[i][0]+steering_trim;
                        rec[i][1]=rec[i][1]+arm_trim;
                        rec[i][2]=rec[i][2]+bucket_trim;
                        if(rec[i][0]>255)rec[i][0]=255;
                        if(rec[i][1]>255)rec[i][1]=255;
                        if(rec[i][2]>255)rec[i][2]=255;
                        if(rec[i][0]<0)rec[i][0]=0;
                        if(rec[i][1]<0)rec[i][1]=0;
                        if(rec[i][2]<0)rec[i][2]=0;
                }
                play_counter=1;
        }
        //compare current component positions to desired
        check_steering=rec[state][0]-steering_angle;
        check_arm=rec[state][1]-arm_position;
        check_bucket=rec[state][2]-bucket_position;

        //determine required arm movements, if necessary
        if(check_arm>10){ //arm below desired
                arm_flag=0;
                desired_arm=255;//move arm up
        }
        else if(check_arm<-10){ //arm above desired
                arm_flag=0;
                desired_arm=0; //move arm down
        }
        else{
                arm_flag=1;
                desired_arm=127;//don't move arm
```

```
                                }
                                  //determine required bucket movements, if
necessary
                                if(check_bucket>10){ //bucket below desired
                                        bucket_flag=0;
                                        desired_bucket=255;//move bucket up
                                }
                                else if(check_bucket<-10){ //bucket above desired
                                        bucket_flag=0;
                                        desired_bucket=0; //move bucket down
                                }
                                else{
                                        bucket_flag=1;
                                        desired_bucket=127;//don't move bucket
                                }
                                //check if steering is in proper position
                                if((check_steering>20)||(check_steering<-20)){
                                        steering_flag=0;
                                        desired_steering=rec[state][0];
                                }
                                else{
                                        steering_flag=1;
                                        desired_steering=rec[state][0];
                                }
                                //suspend playback operation if operator is trimming

        if(steering_trim_flag||arm_trim_flag||bucket_trim_flag){
                                        steering_flag=0;
                                        arm_flag=0;
                                        bucket_flag=0;
                                        if(ary[0]>170)desired_steering=255;//trim
steer left
                                        else if(ary[0]<90)desired_steering=0;//trim
steer right
                                        else desired_steering=steering_angle;
                                        desired_arm=ary[3];
                                        desired_bucket=ary[4];
                                        play_counter=0;
                                        trim_flag=1;
                                }
                                  //terminate pause action if counter limit reached
                                if(pause_counter>=1000) pause_flag=0;
                                //pause playback operation if necessary
                                if(pause_flag){
                                        steering_flag=0;
                                        arm_flag=0;
                                        bucket_flag=0;
                                        play_counter=0;
                                }
                                //if all flags are on then the state vector has been
                                //satisfied and can be incremented
                                if(steering_flag&&bucket_flag&&arm_flag){
                                        //check to see if the next state is the same
                                        //as the current state and if so, pause
playback
                                        //for the recorded time increment interval
                                        pause_checker_1=abs(rec[state][0]-
rec[state+1][0]);
                                        pause_checker_2=abs(rec[state][1]-
rec[state+1][1]);
                                        pause_checker_3=abs(rec[state][2]-
rec[state+1][2]);

                                        if((pause_checker_1<=5)&&
                                           (pause_checker_2<=5)&&
                                           (pause_checker_3<=5)){
                                                pause_flag=1;
                                                pause_counter=1;
                                        }
                                        state++;
                                        play_counter=0;
                                        if(state==(final_state+1)){
                                                state=0;
                                                play=0;
                                                    //restore original recording
                                                for(i=0;i<=final_state;i++){
```

```
                                                        for(j=0;j<=2;j++){

                rec[i][j]=rec_original[i][j];
                                                        }
                                                }
                                        }
                                }
                                //if timeout for current step is exceeded, terminate
                                //playback (exception for initial step)
                                if(state==0)play_counter=1;
                                if(play_counter>10000){
                                        state=0;
                                        play_counter=0;
                                        play=0;
                                        //restore original recording
                                        for(i=0;i<=final_state;i++){
                                                for(j=0;j<=2;j++){
                                                        rec[i][j]=rec_original[i][j];
                                                }
                                        }
                                }
                                break;
                        default:
                                play=0;
                }
        /*Joystick function selection.
        If the command value is between the switch points, the solenoids
        are disabled.  If the value exceeds one of the base switch points,
        the switch point is moved 2 units toward the center to provide
        hysteresis and the appropriate solenoid is activated. */

                if(desired_arm>up_switch){
                        up_switch=UP_SWITCH_0-10;
                        down_switch=DOWN_SWITCH_0;
                        ssr=(ssr&ARM_STOP)|ARM_UP;
                        auto_arm=0;
                        float_flag=0;
                        }
                else if(desired_arm<down_switch){
                        up_switch=UP_SWITCH_0;
                        down_switch=DOWN_SWITCH_0+10;
                        ssr=(ssr&ARM_STOP)|ARM_DOWN;
                        auto_arm=0;
                        float_flag=0;
                        }
                else{
                        up_switch=UP_SWITCH_0;
                        down_switch=DOWN_SWITCH_0;
                        ssr&=ARM_STOP;
                        }

                 if(ary[6]&RATTLE){
                        if(!(rattle_timer)){
                                if(rattle_flag)ssr=ssr^0xc0;
                                else{
                                        rattle_flag=1;
                                        ssr=(ssr&BUCKET_STOP)|ROLL_BACK;
                                        }
                                rattle_timer=200;
                                }
                        }
                 else{
                        rattle_flag=0;
                        if(desired_bucket>rb_switch){
                                rb_switch=RB_SWITCH_0-2;
                                dump_switch=DUMP_SWITCH_0;
                                ssr=(ssr&BUCKET_STOP)|ROLL_BACK;
                                rtd=0;  /*terminate return to dig if enabled */
                                }
                        else if(desired_bucket<dump_switch){
                                rb_switch=RB_SWITCH_0;
                                dump_switch=DUMP_SWITCH_0+2;
                                ssr=(ssr&BUCKET_STOP)|DUMP;
                                rtd=0;  /*terminate return to dig if enabled */
                                }
                        else{
```

```
                        rb_switch=RB_SWITCH_0;
                        dump_switch=DUMP_SWITCH_0;
                        ssr&=BUCKET_STOP;
                        }
                }
        switch(rtd){
        case 0:  /*No return to dig action--wait for button press */
                if(ary[5]&OCC_RTD)rtd=1;
                break;
        case 1:  /*rtd button pressed, wait for release  */
                if(ary[5]&OCC_RTD)break;
                /*determine if bucket should roll back or dump */
                if((portt_rd())&LCC_RTD)rtd=2; /*dump */
                else rtd=3;  /*roll back */
        case 2:  /*bucket must first dump to clear sensor and then roll
back */
                if((portt_rd())&LCC_RTD)ssr=(ssr&BUCKET_STOP)|DUMP;
                else rtd=3;
                break;
        case 3:  /*bucket must roll back until sensor encountered */
                if((portt_rd()&LCC_RTD)){
                        ssr&=BUCKET_STOP;
                        rtd=0; /* bucket in dig position */
                        }
                else ssr=(ssr&BUCKET_STOP)|ROLL_BACK;
                break;
        default:
                rtd=0;
                ssr&=BUCKET_STOP;
        }

        switch(auto_arm){
        case 0:  /*No movement--wait for TH or DH to be pressed */
                if((ary[5]&OCC_TH)|(ary[6]&OCC_DH)){
                        auto_arm=1;
                        if(ary[5]&OCC_TH){
                                if(arm_height)desired_height=0;
                                else desired_height=1;
                                }
                        else if(arm_height<3)desired_height=3;
                        else desired_height=2;
                        }
                break;
        case 1:  /*TH or DH pressed, wait for release */
                if(!((ary[5]&OCC_TH)|(ary[6]&OCC_DH)))auto_arm=2;
                break;
        case 2: /*in motion */
                break;
        default:
                auto_arm=0;
        }
if((ary[6]&OCC_FLOAT)&&!(float_flag)){
        float_flag=1;
        if(arm_height){
                desired_height=0;
                auto_arm=2;
                }
        }

        /*Brake Level Selection
          The variable brake_level determines the current braking level.
The
          switching points of that level are adjusted so as to widen
          the current level to provide hysteresis.  If the value of
          desired_brake is outside of one of these switch points, the
          value of brake_level is incremented or decremented by 1.*/


        switch(brake_level){
        case 3: /* Highest level, check lower switch point only.  */
                if(desired_brake<(BRAKE_3_SWITCH-10)){
                        brake_level=2; /*move to brake level 2 */
                        ssr=(ssr&BRAKE_0)|BRAKE_2,
                        }
                /* otherwise, remain in level 3 */
                else ssr=(ssr&BRAKE_0)|BRAKE_3;
```

```
                        break;
                case 2:
                        if(desired_brake<(BRAKE_2_SWITCH-10)){
                                brake_level=1;/* move down to level 1 */
                                ssr=(ssr&BRAKE_0)|BRAKE_1;
                                }
                        else if(desired_brake>=(BRAKE_3_SWITCH+5)){
                                brake_level=3; /* move up to level 3 */
                                ssr=(ssr&BRAKE_0)|BRAKE_3;
                                }
                        /* otherwise, remain in level 2 */
                        else ssr=(ssr&BRAKE_0)|BRAKE_2;
                        break;
                case 1:
                        if(desired_brake<(BRAKE_1_SWITCH-10)){
                                brake_level=0; /*move down to level 0 */
                                ssr=ssr&BRAKE_0;
                                }
                        else if(desired_brake>=(BRAKE_2_SWITCH+5)){
                                brake_level=2;/* move up to level 2 */
                                ssr=(ssr&BRAKE_0)|BRAKE_2;
                                }
                        /* otherwise, remain in level 1 */
                        else ssr=(ssr&BRAKE_0)|BRAKE_1;
                        break;
                case 0: /* no braking action at all */
                        if(desired_brake>(BRAKE_1_SWITCH+5)){
                                brake_level=1; /*move up to level 1 */
                                ssr=(ssr&BRAKE_0)|BRAKE_1;
                                }
                        /* otherwise, remain in level 0 */
                        else ssr=ssr&BRAKE_0;
                        break;
                default:brake_level=3;
                }

                /*  Compute throttle position value */
                desired_throttle=4095-(desired_throttle*6);
        // send power to throttle actuator and transmission mux
                ssr2|=THROTTLE_POWER|XMISSION_EN;
                /* Check engine start switch (transmission must be in neutral) */
                if
((ary[6]&OCC_ENG_START)&&(desired_direction==8)&&(portt_rd()&OIL_PRESSURE))ssr2|=E
NG_START;
                else ssr2&=(~ENG_START);

                /* if desired steering differs from machine angle by 20 units
                   or more, activate appropriate steering solenoid.  */
                if((desired_steering-steering_angle)>20){
                        ssr2=(ssr2&STEER_0)|STEER_RIGHT;
                        }
                else if((desired_steering-steering_angle)<-20){
                        ssr2=(ssr2&STEER_0)|STEER_LEFT;
                        }
                }
                break;
        default:return_val=-3;}
        return return_val;
        }
```

Appendix C : LCC Hybrid Program

```
/*          LCC.C  LOADER CONTROL PROGRAM WITH RATTLE FEATURE          */


#include        "stdio.h"
#include        "conio.h"
#include        "dos.h"
#include        "string.h"
#include        "stdlib.h"
#include        "c:\stebc31\ve.h"
#include        "c:\stebc31\ad.h"
#include        "c:\stebc31\ser1.h"

#define         OUTPUT_INTERVAL 300
#define         INPUT_INTERVAL  1000


#define         SSR             0xc101  /*High current digital outputs*/
#define         SSR2            0xc100  /*More high current digital outputs*/
#define         THROTTLE_DAC    0    // D/A channel for the throttle
#define         POWER_FAIL      10

/*      Machine input bit values                                        */

#define         LBRAKE          2
#define         LNEU            1
#define         OIL_PRESSURE    4
#define         AOK             8
#define         LCC_RTD         0x10
#define         LCC_TH          0x40
#define         LCC_DH          0x20
#define         FIRST_GEAR      0x10

//      OCC Input Block bit values

#define         OCC_ENG_START   1
#define         OCC_RTD         8
#define         OCC_TH          4
#define         OCC_DH          4
#define         OCC_FLOAT       0x40
#define         RATTLE          0x80
//#define         OCC_VUP         8
//#define         OCC_VDOWN       4
#define         PLAYBACK        8
#define         REC_STFIN       4
#define         OCC_VRIGHT      2
#define         OCC_VLEFT       1
#define         GEAR_MASK       0x30
#define         GEAR_1          0x10
#define         DIRECTION_MASK  0xc0


/*      Transmission output bits                                        */

#define         DIR_NEUTRAL     8
#define         REMOTE_ENABLE   0x40
#define         CLUTCH_ENGAGE   0x20
#define         CLUTCH_DISENGAGE 0xdf
#define         DIR_REV         4
#define         DIRECTION       0x0e

/*      SSR output bits                                                 */

#define         BRAKE_0         0xf8
#define         BRAKE_1         1
#define         BRAKE_2         3
#define         BRAKE_3         5
#define         ARM_UP          8
#define         ARM_DOWN        0x10
```

```
#define         ARM_STOP        0xe7
#define         ARM_FLOAT       0x30
#define         NO_FLOAT        0xdf
#define         ROLL_BACK       0x40
#define         DUMP            0x80
#define         BUCKET_STOP     0x3f

/*      SSR2 output bits                                                */

#define         THROTTLE_POWER  4
#define         STEER_RIGHT     1
#define         STEER_LEFT      2
#define         STEER_0         0xfc
#define         EMERG_STOP      0x20
#define         XMISSION_EN     0x40
#define         ENG_START       0x80
#define         VIDEO_SWITCH    8
#define         NO_VIDEO_SWITCH 0xf7


/*      Switching parameters for analog functions                       */

#define         UP_SWITCH_0     200
#define         DOWN_SWITCH_0   50
#define         RB_SWITCH_0     200
#define         DUMP_SWITCH_0   50
#define         BRAKE_1_SWITCH  80
#define         BRAKE_2_SWITCH  140
#define         BRAKE_3_SWITCH  200

       // serial port 1 variables
extern COM ser1_com;
extern COM *c1=&ser1_com;
unsigned char ser1_in_buf[1024],ser1_out_buf[1024],mode,baud;


/*      GOLBAL VARIABLES                                                */

int desired_steering,steering_angle,desired_brake,desired_throttle,pf;
int desired_arm,desired_bucket,desired_gear,desired_direction;
int status_output,t,v;
float steering_slope=-.2623,steering_offset=607.44;
unsigned int i,input_timer,output_timer,aux_timer,rattle_timer,rtd;
unsigned int up_switch=UP_SWITCH_0,down_switch=DOWN_SWITCH_0;
char input_buffer[30],input_block_buffer[31],output_block_buffer[30];
char *output_msg="s0172\n\0";
char *out_buf_ptr_;
char *block_char_1;
unsigned char pt;
unsigned int input_count,block_count,loop_timer,max_loop_time;
unsigned int ssr,ssr2,xmission;
unsigned int rb_switch=RB_SWITCH_0,dump_switch=DUMP_SWITCH_0,brake_level=3;
unsigned int arm_height,desired_height,auto_arm,float_flag,rattle_flag;
int inc,change,keychange1,keychange2,changeflag,incmax;
unsigned record,play,stoprecord,stopplay;
unsigned long int rec_counter,rec_counter_max;
float bucket_slope=.0711,bucket_offset=33.137,arm_slope=.2037,arm_offset=284.74;
int check_bucket,check_arm,check_steering,arm_flag,bucket_flag,steering_flag;
int
initial_condition,arm_position,bucket_position,init_bucket,init_arm,init_steering;
int exit_initial_condition,steering_trim_flag,initial_angle,new_angle;
unsigned rec[200][7];
unsigned long state_counter[200];


void interrupt far tb_isr(void);
void power_fail(void);
int ser_inp(void);
void ser_out(void);
int decode_input(void);

main()
{
int s;
unsigned arm_sensor;
```

```
\


/*      INITIALIZE VARIABLES    */

        ve_init();
        ad_init();
        outportb(0xc103,0x80);
        outportb(0xc100,0xff);
        outportb(0xc101,0xff);
        for(i=1;i<8;i++)ad_hv(i,0);
        portt_wr(8);
        mode=0xc9;
        baud=8;         //9600 baud
        s1_init(mode,baud,ser1_in_buf,1024,ser1_out_buf,1024,c1);
        time_base_init(1);
        time_base_interrupt(1,tb_isr);
        ssr=0;
        block_count=0;
        block_char_1=&input_block_buffer[0];
        input_timer=INPUT_INTERVAL;
        output_timer=OUTPUT_INTERVAL;
        aux_timer=5000;
        rattle_timer=0;
        rattle_flag=0;
        out_buf_ptr_=output_msg;
        initial_condition=0;
        exit_initial_condition=1;
        pt=portt_rd();
/*
/
/       BEGIN PROGRAM FUNCTIONS
/


        /* Wait for receipt of 'Idle' message while continuing to
        monitor Remote switch and send 'Standby' message.  */

        /*Set full brakes, transmission neutral, remote mode  */
STANDBY:
                outportb(SSR,~BRAKE_3); //Apply full brakes
                outportb(SSR2,~(XMISSION_EN));//Send power to xmission mux
                xmission=(DIR_NEUTRAL|REMOTE_ENABLE); //Set xmission to neutral
                for(i=1;i<8;i++)ad_hv(i,(xmission>>i)&1);
        /* Wait for receipt of 'command' message and send 'system OK' message.  */
                v=0;
                while((v!=2)||(portt_rd()&LNEU)){
                /* return value of 2 indicates valid 'command' msg
                        Wait for 'command msg and loader in neutral  */
                        //if loader not in neutral, send 'n'
                         pt=portt_rd();
                         if(portt_rd()&LNEU)strcpy(output_block_buffer,"n0n");
                         // if loader is in neutral, send 's'
                         else strcpy(output_block_buffer,"s0s");
                         if(serhit1(c1))ser_inp();
                         if(block_count){
                         v=decode_input();
                         block_count=0;
                         }
                         if(output_timer==0){
                                ser_out();
                                output_timer=OUTPUT_INTERVAL;
                                }
                         pokeb(0xfff0,0x11,0x40); //set up to read the A/D
                         pokeb(0xfff0,0x10,0xf7);
                         //if lcc power has failed, execute power fail routine
                         ad_ad12(POWER_FAIL);
                         /*if(ad_ad12(POWER_FAIL)<800){
                                power_fail();
                                goto STANDBY;
                                }*/
                         ad_init();
                         }

        WAITING_FOR_OCC:
        strcpy(output_block_buffer,"r0r");
        ser_out();
        output_timer=OUTPUT_INTERVAL;
        while(v!=3){
```

```
        //  Waiting for OCC to take control
                if(portt_rd()&LNEU)goto STANDBY;
                if(serhit1(c1))ser_inp();
                if(block_count){
                        if((v=decode_input())==1)goto STANDBY;
                        block_count=0;
                        }
                if(output_timer==0){
                        ser_out();
                        output_timer=OUTPUT_INTERVAL;
                        }
                }


/*      BEGINNING OF REMOTE MODE        */


        REMOTE_ENTRY_POINT:
        max_loop_time=0;
        input_timer=INPUT_INTERVAL;
        rtd=0;
        arm_height=0;
        auto_arm=0;
        float_flag=0;
        strcpy(output_block_buffer,"r0r");
        output_block_buffer[3]=0x0d;
        output_block_buffer[4]=0;
        t=20;
/*Stay in remote mode until vehicle transmission lever is moved or
  OCC commands a return to standby */
         while(1){
                if(loop_timer>max_loop_time)max_loop_time=loop_timer;
                loop_timer=0;
                // set up to read a/d converter
                pokeb(0xfff0,0x11,0x40);
                pokeb(0xfff0,0x10,0xf7);
                ad_ad12(6);
                steering_angle=ad_ad12(7);  //read ch 6, convert ch 7
                arm_position=ad_ad12(8);  //read ch 7, convert ch 8
                bucket_position=ad_ad12(POWER_FAIL);  //read ch 8, convert ch 10
                /*if((ad_ad12(POWER_FAIL)<800)&&(!ssr2&ENG_START)){
                        power_fail();
                        goto STANDBY;
                        }*/
                // return to original configuration
                ad_init();
                //scale steering data
                steering_angle=steering_angle*steering_slope+steering_offset;
                if(steering_angle<0)steering_angle=0;
                if(steering_angle>255)steering_angle=255;
                //scale bucket data
                bucket_position=bucket_position*bucket_slope-bucket_offset;
                if(bucket_position<0)bucket_position=0;
                if(bucket_position>255)bucket_position=255;
                //scale arm data
                arm_position=arm_position*arm_slope-arm_offset;
                if(arm_position<0)arm_position=0;
                if(arm_position>255)arm_position=255;



/* If valid 'command' block is not received within 500 msec, goto STANDBY */
                if(input_timer==0)goto STANDBY;

/* Send 'system OK' message every 400 msec  */
                if(output_timer==0){
                        if(record==2) strcpy(output_block_buffer,"t0t"); //recording
mode
                        else if(play==2) strcpy(output_block_buffer,"101");
//playback mode
                        else strcpy(output_block_buffer,"r0r"); //standard remote
mode
                        output_timer=OUTPUT_INTERVAL;
                        ser_out();
```

```
                        }
/* when a new message is received from the field unit, call decode_input */
                if(serhit1(c1))ser_inp();
                if(block_count){
                // set bits 0 and 1 of port3 for timing purposes
                        outportb(0xc102,3);
                        /*if it is a valid 'command' message, reset input timer */
                        if((v=decode_input())==3)input_timer=INPUT_INTERVAL;
                        if(v==10){
                                record=0;
                                initial_condition=0;
                                play=0;
                                rec_counter=0;
                                inc=0;
                                stopplay=0;
                                goto STANDBY; //Emergency Stop
                        }
                        block_count=0;
                }
                //time check section
                if(v==3) outportb(0xc102,1); // if valid cmd, turn off bit 1
                else outportb(0xc102,0); // if not valid, turn off both bits
                //

                /* set gear and direction  */
                xmission=desired_gear+desired_direction|REMOTE_ENABLE;
                /* disengage clutch if brake level > 1, otherwise engage it  */
                if(brake_level>1)xmission&=CLUTCH_DISENGAGE;
                else xmission|=CLUTCH_ENGAGE;
                for(i=1;i<8;i++)ad_hv(i,(xmission>>i)&1); /* activate desired
transmission bits */

                /* check to see if steering angle is within 8 unit of correct
value.
                   If so, disable both steering solenoids, otherwise do nothing.
*/
                s=steering_angle-desired_steering;
                if((s<8)&&(s>-8))ssr2&=STEER_0;

                // Disengage starter if engine is running
                if(!(portt_rd()&OIL_PRESSURE))ssr2&=(~ENG_START);

 //  This section checks to see if the loader's arm should be moving
 //  automatically.

                if(auto_arm==2){
                        if(arm_height<desired_height)ssr=(ssr&ARM_STOP)|ARM_UP;
                        else
if(arm_height>desired_height)ssr=(ssr&ARM_STOP)|ARM_DOWN;
                        else ssr&=ARM_STOP.auto_arm=0;
                        }
                arm_sensor=(portt_rd()&(LCC_TH|LCC_DH));
                if(arm_sensor==LCC_TH)arm_height=1;
                else if(arm_sensor==LCC_DH)arm_height=3;
                else{
                        if(arm_height==1){
                                if(ssr&ARM_UP)arm_height=2;
                                else if(ssr&ARM_DOWN)arm_height=0;
                                }
                        else if(arm_height==3){
                                if(ssr&ARM_UP)arm_height=4;
                                else if(ssr&ARM_DOWN)arm_height=2;
                                }
                        }
                if((float_flag)&&!(auto_arm))ssr=(ssr&ARM_STOP)|ARM_FLOAT;
                if(!(float_flag))ssr&=NO_FLOAT;

                if((xmission&DIRECTION)==DIR_REV)ssr2|=VIDEO_SWITCH;
                else ssr2&=NO_VIDEO_SWITCH;

                outportb(SSR,~ssr); /* activate SSR bits */
                outportb(SSR2,~ssr2);
                for(i=1;i<8;i++)ad_hv(i,(xmission>>i)&1);

                /* send desired throttle position to DAC channel 0 */
                ad_da12(0,desired_throttle);
```

```
                outportb(0xc102,0); // turn off timing bit 0
                }
        /* When one of the conditions necessary for remote opperation is not
        met, return to the STANDBY mode.  */

        goto STANDBY;
        }


/*  This is the timebase counter interrupt service routine.
    Each of the timer variables is decremented every millisecond until
    a value of 0 is reached. */

void interrupt far tb_isr(void)
{
        if(input_timer)input_timer--;
        if(output_timer)output_timer--;
        if(aux_timer)aux_timer--;
        if(rattle_timer)rattle_timer--;
        if(v==3)loop_timer++;
        if(rec_counter)rec_counter++;
        fint();
        }
void power_fail(void)
{
        int pwr_fail=0;
        char ob_save[20];
        outportb(SSR2,~EMERG_STOP); //apply full brakes and kill engine
        outportb(SSR,0xff);
        strcpy(ob_save,output_block_buffer);//save last output block
        strcpy(output_block_buffer,"p0p");//signal LCC power failure
        ser_out();//send power fail message to occ
        pokeb(0xfff0,0x11,0x40);//set up to read A/D
        pokeb(0xfff0,0x10,0xf7);
        while(pwr_fail<1600){
                while(output_timer);//wait for output interval to expire
                output_timer=OUTPUT_INTERVAL;
                ser_out();
                ad_ad12(POWER_FAIL);
                pwr_fail=ad_ad12(POWER_FAIL);
                }
        strcpy(output_block_buffer,ob_save); //restore output_block_buffer
        ad_init();
        }


/*   This is the serial input service routine.  When a character is received,
     it is placed in the input buffer.  When the received character is a
     new line character, or the input buffer is full, the characters received up
     to that point are transferred to the block buffer and the block count is
     set to show the number of characters in the buffer.  The input count is
     then set to 0 and the process starts over.  */

int ser_inp(void)
{
        while(serhit1(c1)){
                if((input_buffer[input_count]=getser1(c1))==13){
                        block_count=input_count+1;
                        input_count=0;
                        input_buffer[block_count]=0;
                        strcpy(input_block_buffer,input_buffer);
                        }
                else input_count++;
                }
        return block_count;
        }

void ser_out(void)
{
        for(i=0;i<3;i++)while(!(putser1(output_block_buffer[i],c1)));
        while(!(putser1(0x0d,c1)));
        }


/*  The decode_input() function evaluates the received input block and returns
    a value as follows:  1=valid 'keep alive' message, 2=OCC asking for control,
```

```
    3=valid control message received, -1=incorrect character count, -2=checksum
    error, -3=first character was not a legal command, 10=Emergency Stop.
    If the input block is a valid command message, the function decodes it and
    sets the appropriate bits in the system control variables .     */

int decode_input(void)
{
        unsigned return_val=0,checksum=0,i,ary[20];

        if((block_count!=3)&&(block_count!=21))return -1;
        for(i=0;i<(block_count-2);i++)checksum+=input_block_buffer[i];
        if((checksum&0xf)!=(input_block_buffer[i]&0xf))return -2;
        switch(input_block_buffer[0]){
        case 'e': //Emergency stop
                return_val=10;
                break;
        case 'i': //Keep Alive message
                if(block_count==3)return_val=1;
                else return_val=-3;
                break;
        case 'p': // OCC asking for control
                if(block_count==3)return_val=2;
                else return_val=-3;
                break;
        case 'c': //Command message
                if(block_count!=21)return_val=-3;
                else{
                        return_val=3;
                        for(i=1;i<19;i+=2)ary[(i-
1)/2]=((input_block_buffer[i]&0xf)<<4)|(input_block_buffer[i+1]&0xf);


                          /* Playback section - to repeat recorded operator
information */
                          exit_initial_condition=1;
                        switch(play){
                                case 0:  /* No play action -- wait for button press
*/
                                        if(ary[7]&0xf&PLAYBACK) play=1;
                                        break;
                                case 1:  /* Play button pressed, wait for release */
                                        if(ary[7]&0xf&PLAYBACK) break;
                                        else{
                                                play=2;
                                                initial_condition=1;
                                                //rec_counter=1;
                                        }
                                        break;
                                case 2:  /* Enter playback mode */
                                        /* Terminate playback if playback button
presses */
                                        switch(stopplay){
                                                case 0:
                                                        if(ary[7]&0xf&PLAYBACK)
stopplay=1;
                                                        break;
                                                case 1:
                                                        if(ary[7]&0xf&PLAYBACK) break;
                                                        else{
                                                                play=0;
                                                                rec_counter=0;
                                                                inc=0;
                                                                stopplay=0;


initial_condition=0;
                                                        }
                                                        break;
                                                default:
                                                        stopplay=0;
                                        }


                                        /*if playback is just starting, initialize
the machine
                                        components to their proper positions*/
                                        if(initial_condition){
```

```
                                        check_steering=init_steering-
steering_angle;
                                        check_arm=init_arm-arm_position;
                                        check_bucket=init_bucket-
bucket_position;

                                        //determine required arm movements,
if necessary
                                        if(check_arm>10){ //arm below desired
                                                arm_flag=0;
                                                desired_arm=255; //move arm up
                                        }
                                        else if(check_arm<-10){  //arm above
desired
                                                arm_flag=0;
                                                desired_arm=0; //move arm down
                                        }
                                        else{
                                                arm_flag=1;
                                                desired_arm=127; //don't move
arm
                                        }
                                        //determine required bucket
movements, if necessary
                                        if(check_bucket<-10){
                                                bucket_flag=0;
                                                desired_bucket=255; //roll
back bucket
                                        }
                                        else if(check_bucket>10){
                                                bucket_flag=0;
                                                //move bucket if it does not
conflict with
                                                //the current arm position
if((arm_position<50)&&(arm_flag==0)) desired_bucket=127;
                                                else desired_bucket=0; //dump
bucket
                                        }
                                        else{
                                                bucket_flag=1;
                                                desired_bucket=127; //don't
move bucket
                                        }
                                        //check if steering is in proper
position
        if((check_steering>20)||(check_steering<-20)){
                                                steering_flag=0;
                                                desired_steering=rec[inc][0];
                                        }
                                        else{
                                                steering_flag=1;
                                                desired_steering=rec[inc][0];
                                        }
                                           //set throttle and brake to zero
during initial positioning
                                           desired_throttle=0;
                                           desired_brake=0;
                                        //if initial positions are satisfied,
start playback
                                        //and counter
        if(steering_flag&&bucket_flag&&arm_flag){
                                                initial_condition=0;
                                                exit_initial_condition=0;
                                                rec_counter=1;
                                        }
                                }
                                else{
                                        exit_initial_condition=1;

                                        //perform trimming operation, or
normal playback
                                        //trim steering
                                        if(ary[0]>167){
```

```
                                                        ary[0]=255; //steer right
                                                            //get steering angle at
beginning of trim operation
                                                        initial_angle=steering_angle;
                                                            //steering_trim_flag=1;
                                                    }
                                                else if(ary[0]<87){
                                                        ary[0]=0; //steer left
                                                        initial_angle=steering_angle;
                                                            //steering_trim_flag=1;
                                                    }
                                                else{
                                                        //if a trimming operation was
performed, get the
                                                        //new steering angle and
change downstream state
                                                            //data accordingly
                                                        //if(steering_trim_flag){
                                                        //
        new_angle=steering_angle;
                                                        //
        for(i=inc;i<=incmax;i++){
                                                        //
        rec[i][0]=rec[i][0]-initial_angle+new_angle;
                                                        //
        if(rec[i][0]>510) rec[i][0]=0;
                                                        //
        if((rec[i][0]>255)&&(rec[i][0]<=510)) rec[i][0]=255;
                                                        //      }
                                                        //      steering_trim_flag=0;
                                                            //}
                                                        ary[0]=rec[inc][0]; //use
recorded value
                                                    }

                                                //trim brakes - use trim value only
if greater than
                                                //recorded value

if(ary[1]<(rec[inc][1]+10))ary[1]=rec[inc][1];
                                                //trim throttle - use trim value only
if greater than
                                                //recorded value

if(ary[2]<(rec[inc][2]+10))ary[2]=rec[inc][2];
                                                //trim arm
                                                if(ary[3]>147) ary[3]=255; //arm up
                                                else if(ary[3]<107) ary[3]=0; //arm
down
                                                else ary[3]=rec[inc][3]; //use
recorded arm value
                                                //trim bucket
                                                if(ary[4]>147) ary[4]=255; //roll
back bucket
                                                else if(ary[4]<107) ary[4]=0; //dump
bucket
                                                else ary[4]=rec[inc][4]; //use
recorded bucket value
                                                //set desired direction and
keypresses to recorded values
                                                ary[5]=rec[inc][5];
                                                ary[6]=rec[inc][6];
                                                /*Check to see if it is time to step
forward in the recorded
                                                array of operations */

        if((inc<incmax)&&(state_counter[inc+1]<=rec_counter)) inc++;

                                                /*If counter has reached the maximum
value from the
                                                record action, then stop playback */
                                                if(rec_counter>=rec_counter_max){
                                                        play=0;
                                                        rec_counter=0;
                                                        inc=0;
                                                }
```

```
                        }

                        break;
                default:
                        play=0;
        }

        /* set desired control actions */
        if((!initial_condition)&&(exit_initial_condition)){
                desired_steering=ary[0];
                desired_brake=ary[1];
                desired_throttle=ary[2];
                if(desired_throttle>180)desired_throttle=180;
                desired_arm=ary[3];
                desired_bucket=ary[4];
        }

        /*Record section - OCC movements are monitored and stored in
          an array */
        switch(record){
                case 0:  /*No record action -- wait for button press
*/
                        if(ary[7]&0xf&REC_STFIN) record=1;
                        break;
                case 1:  /*record button pressed, wait for release
*/
                        if(ary[7]&0xf&REC_STFIN) break;
                        else{
                                //get initial position information
                                init_steering=steering_angle;
                                init_arm=arm_position;
                                init_bucket=bucket_position;

                                rec_counter=1;
                                record=2;
                                inc=0;

                                //get initial values
                                for(i=0;i<=6;i++) rec[inc][i]=ary[i];
                                state_counter[inc]=rec_counter;
                        }
                        break;
                case 2:  /*enter record mode */
                        /*see if record start/finish button has been
                          pressed a second time */
                        switch(stoprecord){
                                case 0:
                                        if(ary[7]&0xf&REC_STFIN)
stoprecord=1;
                                        break;
                                case 1:
                                        if(ary[7]&0xf&REC_STFIN)
break;
                                        else{
                                                incmax=inc;
                                                record=0;

        rec_counter_max=rec_counter;
                                                rec_counter=0;
                                                inc=0;
                                                stoprecord=0;
                                        }
                                        break;
                                default:
                                        stoprecord=0;
                        }

                        //initialize flag
                        changeflag=0;

                        /*if the maximum number of allowable
operation changes
                          has been reached, terminate recording */
                         if(inc>=199){
                                record=0;
                                incmax=inc;
```

```
                                        rec_counter_max=rec_counter;
                                        rec_counter=0;
                                        inc=0;
                                }
                                else if(rec_counter!=0){
                                        /*check to see if operator is
changing anything,
                                          and if so, update the rec[][]
array*/
                                        keychange1=abs(rec[inc][5]-ary[5]);
                                        keychange2=abs(rec[inc][6]-ary[6]);
                                        if((keychange1>=4)||(keychange2>=4))
changeflag=1;
                                        for(i=0;i<=6;i++){
                                                change=abs(rec[inc][i]-
ary[i]);
                                                if(change>=20) changeflag=1;
                                        }
                                        if(changeflag==1){
                                                inc++;
                                                for(i=0;i<=6;i++)
rec[inc][i]=ary[i];
        state_counter[inc]=rec_counter;
                                        }
                                        /*check to see if rec_counter has
reached max
                                          allowable value, terminate if so */
                                        if(rec_counter>=90000){
                                                record=0;
                                                incmax=inc;
                                                rec_counter_max=rec_counter;
                                                rec_counter=0;
                                                inc=0;
                                        }
                                }
                                break;
                        default:
                                record=0;
                }

                if((ary[5]&GEAR_MASK)==GEAR_1)desired_gear=FIRST_GEAR;
                else desired_gear=0;

//  Check desired direction
                desired_direction=(ary[5]&DIRECTION_MASK)>>5;

/* if neither FWD or REV selected, set direction NEUTRAL */
                if(desired_direction==0)desired_direction=8;


/*Joystick function selection.
If the command value is between the switch points, the solenoids
are disabled.  If the value exceeds one of the base switch points,
the switch point is moved 2 units toward the center to provide
hysteresis and the appropriate solenoid is activated. */

                if(desired_arm>up_switch){
                        up_switch=UP_SWITCH_0-10;
                        down_switch=DOWN_SWITCH_0;
                        ssr=(ssr&ARM_STOP)|ARM_UP;
                        auto_arm=0;
                        float_flag=0;
                        }
                else if(desired_arm<down_switch){
                        up_switch=UP_SWITCH_0;
                        down_switch=DOWN_SWITCH_0+10;
                        ssr=(ssr&ARM_STOP)|ARM_DOWN;
                        auto_arm=0;
                        float_flag=0;
                        }
                else{
                        up_switch=UP_SWITCH_0;
                        down_switch=DOWN_SWITCH_0;
                        ssr&=ARM_STOP;
                        }
```

```
if(ary[6]&RATTLE){
        if(!(rattle_timer)){
                if(rattle_flag)ssr=ssr^0xc0;
                else{
                        rattle_flag=1;
                        ssr=(ssr&BUCKET_STOP)|ROLL_BACK;
                        }
                rattle_timer=200;
                }
        }
else{
        rattle_flag=0;
        if(desired_bucket>rb_switch){
                rb_switch=RB_SWITCH_0-2;
                dump_switch=DUMP_SWITCH_0;
                ssr=(ssr&BUCKET_STOP)|ROLL_BACK;
                rtd=0;  /*terminate return to dig if enabled */
                }
        else if(desired_bucket<dump_switch){
                rb_switch=RB_SWITCH_0;
                dump_switch=DUMP_SWITCH_0+2;
                ssr=(ssr&BUCKET_STOP)|DUMP;
                rtd=0;  /*terminate return to dig if enabled */
                }
        else{
                rb_switch=RB_SWITCH_0;
                dump_switch=DUMP_SWITCH_0;
                ssr&=BUCKET_STOP;
                }
        }
switch(rtd){
case 0:  /*No return to dig action--wait for button press */
        if(ary[5]&OCC_RTD)rtd=1;
        break;
case 1:  /*rtd button pressed, wait for release  */
        if(ary[5]&OCC_RTD)break;
        /*determine if bucket should roll back or dump */
        if((portt_rd()&LCC_RTD))rtd=2; /*dump */
        else rtd=3;  /*roll back */
case 2:  /*bucket must first dump to clear sensor and then roll
back */
        if((portt_rd()&LCC_RTD))ssr=(ssr&BUCKET_STOP)|DUMP;
        else rtd=3;
        break;
case 3:  /*bucket must roll back until sensor encountered */
        if((portt_rd()&LCC_RTD)){
                ssr&=BUCKET_STOP;
                rtd=0; /* bucket in dig position */
                }
        else ssr=(ssr&BUCKET_STOP)|ROLL_BACK;
        break;
default:
        rtd=0;
        ssr&=BUCKET_STOP;
}

switch(auto_arm){
case 0:  /*No movement--wait for TH or DH to be pressed */
        if((ary[5]&OCC_TH)|(ary[6]&OCC_DH)){
                auto_arm=1;
                if(ary[5]&OCC_TH){
                        if(arm_height)desired_height=0;
                        else desired_height=1;
                        }
                else if(arm_height<3)desired_height=3;
                else desired_height=2;
                }
        break;
case 1:  /*TH or DH pressed, wait for release */
        if(!((ary[5]&OCC_TH)|(ary[6]&OCC_DH)))auto_arm=2;
        break;
case 2: /*in motion */
        break;
default:
        auto_arm=0;
```

```
                }
        if((ary[6]&OCC_FLOAT)&&!(float_flag)){
                float_flag=1;
                if(arm_height){
                        desired_height=0;
                        auto_arm=2;
                        }
                }

                /*Brake Level Selection
                  The variable brake_level determines the current braking level.
The
                  switching points of that level are adjusted so as to widen
                  the current level to provide hysteresis.  If the value of
                  desired_brake is outside of one of these switch points, the
                  value of brake_level is incremented or decremented by 1.*/


                switch(brake_level){
                case 3: /* Highest level, check lower switch point only.  */
                        if(desired_brake<(BRAKE_3_SWITCH-10)){
                                brake_level=2; /*move to brake level 2 */
                                ssr=(ssr&BRAKE_0)|BRAKE_2;
                                }
                        /* otherwise, remain in level 3 */
                        else ssr=(ssr&BRAKE_0)|BRAKE_3;
                        break;
                case 2:
                        if(desired_brake<(BRAKE_2_SWITCH-10)){
                                brake_level=1;/* move down to level 1 */
                                ssr=(ssr&BRAKE_0)|BRAKE_1;
                                }
                        else if(desired_brake>=(BRAKE_3_SWITCH+5)){
                                brake_level=3; /* move up to level 3 */
                                ssr=(ssr&BRAKE_0)|BRAKE_3;
                                }
                        /* otherwise, remain in level 2 */
                        else ssr=(ssr&BRAKE_0)|BRAKE_2;
                        break;
                case 1:
                        if(desired_brake<(BRAKE_1_SWITCH-10)){
                                brake_level=0; /*move down to level 0 */
                                ssr=ssr&BRAKE_0;
                                }
                        else if(desired_brake>=(BRAKE_2_SWITCH+5)){
                                brake_level=2;/* move up to level 2 */
                                ssr=(ssr&BRAKE_0)|BRAKE_2;
                                }
                        /* otherwise, remain in level 1 */
                        else ssr=(ssr&BRAKE_0)|BRAKE_1;
                        break;
                case 0: /* no braking action at all */
                        if(desired_brake>(BRAKE_1_SWITCH+5)){
                                brake_level=1; /*move up to level 1 */
                                ssr=(ssr&BRAKE_0)|BRAKE_1;
                                }
                        /* otherwise, remain in level 0 */
                        else ssr=ssr&BRAKE_0;
                        break;
                default:brake_level=3;
                }

                /*  Compute throttle position value */
                desired_throttle=4095-(desired_throttle*6);
        // send power to throttle actuator and transmission mux
                ssr2|=THROTTLE_POWER|XMISSION_EN;
                /* Check engine start switch (transmission must be in neutral) */
                if
((ary[6]&OCC_ENG_START)&&(desired_direction==8)&&(portt_rd()&OIL_PRESSURE))ssr2|=E
NG_START;
                else ssr2&=(~ENG_START);

                /* if desired steering differs from machine angle by 20 units
                   or more, activate appropriate steering solenoid.  */
                if((desired_steering-steering_angle)>20){
                        ssr2=(ssr2&STEER_0)|STEER_RIGHT;
```

```
                }
        else if((desired_steering-steering_angle)<-20){
                ssr2=(ssr2&STEER_0)|STEER_LEFT;
                }
        }
        break;
default:return_val=-3;}
return return_val;
}
```

Appendix D : OCC Control Program

```
/*          OCC Program for remote control of the Case Loader  */


#include <stdio.h>
#include <dos.h>        /* Enable/disable functions */
#include <string.h>
#include "\STEBC31\ve.h"          /* V25_engine initialization */
#include "\STEBC31\sw.h"          /* LCD function/data prototypes */
#include "\STEBC31\lcd.h"
#include "\STEBC31\ser1.h"

//  KEYBOARD INPUTS

#define         EN_DIS          0x10
#define         ASTR            0x100
#define         RECORD          0x20
#define         PLAYBACK        0x200
#define         GEAR_1          0x1000
#define         GEAR_2          0x2000
#define         FWD             0x4000
#define         REV             0x8000
#define         CAL_MASK        0x110

//  ANALOG INPUT CHANNELS

#define         BRAKE           3
#define         THROTTLE        4
#define         STEERING        5
#define         V_JOY           6
#define         H_JOY           7

#define         OUTPUT_INTERVAL 50
#define         INPUT_INTERVAL  1000

#define         BLANK_LINE      "                    "

float brake_slope,throttle_slope,steering_slope,h_joy_slope,v_joy_slope;
int brake_offset,throttle_offset,steering_offset,h_joy_offset,v_joy_offset;
int b0,t0,hj0,vj0,cal_1,cal_2,cur_brake,cur_throttle,cur_steering;
int cur_v_joy,cur_h_joy,m_n,beep_time,input_timer,output_timer,timer1,timer2;
int loader_status,comm_disp,in_block_counter,lcd_vee,learn_mode,power_fail;
float raw_atod;
int test;
unsigned char in_buf[1024],out_buf[1024],in_block[40],out_block[50],in_char;
char display_1[22],display_2[22],display_3[22],display_4[22];
unsigned char* in_block_ptr,out_block_ptr;

extern COM ser1_com;
COM* c1;


int kb_scan(void);
void calibrate(void);
void e_stop(void);
void interrupt far tb_isr(void);
void remote_control(void);
void get_serial_data(void);
void err_display(int error_number);
void send_data(void);
void display_out(void);
void main(void)
{
        c1=&ser1_com;
        output_timer=OUTPUT_INTERVAL;
        input_timer=INPUT_INTERVAL;
        ve_init();
        lcd_init();
        time_base_init(1);
        time_base_interrupt(1,tb_isr);
        sw_da10(lcd_vee);
        s1_init(0xc9,8,in_buf,1024,out_buf,1024,c1);
```

```
        if((m=kb_scan())&ASTR)calibrate();
        strcpy(display_1,"CASE LOADER CONTROL");
        strcpy(display_3,BLANK_LINE);
        strcpy(display_4,BLANK_LINE);
        comm_disp=10; //cause display to be updated
        loader_status=0;
         power_fail=0;
        learn_mode=0;
        beep_time=50;

        while(1){

// Clear emergency stop switch if activated
                if(!(port_rd(1)&2))e_stop();

// If serial data is present, get it.
                if(serhit1(c1))get_serial_data();

// Display current communication status if neceessary
                if(loader_status!=comm_disp){
                        comm_disp=loader_status;
                        switch(loader_status){
                        case 0:
                                strcpy(display_2,"  No Communication  ");
                                  if(power_fail) strcpy(display_3,"   POWER FAILURE!
");
                                break;
                        case 1:
                                  strcpy(display_2,"  Not in Neutral    ");
                                break;
                        case 2:
                                strcpy(display_2,"  Loader in STANDBY ");
                                strcpy(display_3,"    This IS the     ");
                                strcpy(display_4,"     NEXT STEP      ");
                                break;
                        case 3:
                                strcpy(display_2,"  Loader in REMOTE  ");
                                break;
                                }
                        display_out();  //update display
                        }
                if((m=kb_scan())&0x0fff){
                        strcpy(display_3,BLANK_LINE);
                        strcpy(display_4,BLANK_LINE);
                        display_out();
                        }
                if(m&EN_DIS)remote_control();
                if(output_timer==0){
                        output_timer=OUTPUT_INTERVAL;
                        putser1('i',c1);
                        putser1('i',c1);
                        putser1(0x0d,c1);
                        }
                if(input_timer==0){
                        input_timer=INPUT_INTERVAL;
                        loader_status=0;
                        }



                }
        }
//
//  CALIBRATION ROUTINE
//
//  Sets slope and offset paramaters for the analog inputs to
//  normalize them to 256 units full scale.
//  This section is entered by having the asterisk key pressed
//  when power is applied to the OCC unit.
//  Pressing the EN/DIS key at any time will terminate the
//  calibrate mode, saving the parameters that have been set
//  up to that time.
//
 void calibrate(void)
{
        int cal_flag=0;
```

```
        lcdcmd(0x01);  // Clear display
        lcdcmd(0x80);  //select line 1
        printf("  OCC  CALIBRATION    Press  EN/DIS at  Press * to continue.  any
time to exit. ");
        while((m=kb_scan())&ASTR);  // Wait for * release
        while(!((m=kb_scan())&EN_DIS)){
                if((m & ASTR)){
    // if the * key is pressed, procede according to cal_flag
                        while((m=kb_scan())&ASTR);  // Wait for * release
                        switch(cal_flag){
                        case 0: // Set lcd contrast using FWD/REV lever
                                lcdcmd(0x94);
                                printf("to set LCD contrast  Use FWD/REV lever
then press *   ");
                                while(!((m=kb_scan())&ASTR)){
                                        if(m&FWD){
                                                if(lcd_vee>0)lcd_vee--;
                                                sw_da10(lcd_vee);
                                                }
                                        if(m&REV){
                                                if(lcd_vee<254)lcd_vee++;
                                                sw_da10(lcd_vee);
                                                }
                                        timer1=10;
                                        while(timer1);
                                        }
                                cal_flag=1;
                                beep_time=200;
                                break;

                        case 1: //Ask for controls in neutral
                                lcdcmd(0x94);
                                printf(" throttle, and brake  Release  joystick,
and press *   ");
                                cal_flag=2;
                                beep_time=200;
                                break;
                        case 2: //Read neutral positions and ask for throttle
                                t0=sw_ad12a(adch(THROTTLE),1);
                                  t0=sw_ad12a(adch(THROTTLE),1);
                                b0=sw_ad12a(adch(BRAKE),1);
                                vj0=sw_ad12a(adch(V_JOY),1);
                                hj0=sw_ad12a(adch(H_JOY),1);
                                lcdcmd(0x94);
                                printf("to full throttle posDepress throttle bar");
                                cal_flag=3;
                                beep_time=200;
                                break;
                        case 3: //Read full throttle and ask for full brake
                                cal_1=sw_ad12a(adch(THROTTLE),1);
                                  cal_1=sw_ad12a(adch(THROTTLE),1);
                                if((cal_1-t0)<20){
                                        beep_time=1000;
                                        break;
                                        }
                                throttle_slope=256.0/(cal_1-t0);
                                throttle_offset=(int)(-t0*throttle_slope);
                                lcdcmd(0x94);
                                printf("  to full brake pos Squeeze brake lever ");
                                cal_flag=4;
                                beep_time=200;
                                break;
                        case 4: //Read full brake and ask for steering left
                                cal_1=sw_ad12a(adch(BRAKE),1);
                                if((cal_1-b0)<20){
                                        beep_time=1000;
                                        break;
                                        }
                                brake_slope=256.0/(cal_1-b0);
                                brake_offset=(int)(-b0*brake_slope);
                                lcdcmd(0x94);
                                printf(" control full left     Move steering    ");
                                cal_flag=5;
                                beep_time=200;
                                break;
                        case 5: //Read steering left and ask for steering right
```

```
                cal_1=sw_ad12a(adch(STEERING),1);
                lcdcmd(0x94);
                printf(" control full right    Move steering    ");
                cal_flag=6;
                beep_time=200;
                break;
        case 6: //Read steering right and ask for joystick forward
                cal_2=sw_ad12a(adch(STEERING),1);
                if((cal_2-cal_1)<20){
                        beep_time=1000;
                        break;
                        }
                steering_slope=256.0/(cal_2-cal_1);
                steering_offset=(int)(-cal_1*steering_slope);
                lcdcmd(0x94);
                printf(" forward (arm down) Move joystick fully ");
                cal_flag=7;
                beep_time=200;
                break;
        case 7: //Read joystick fwd and ask for joystick back
                cal_1=sw_ad12a(adch(V_JOY),1);
                lcdcmd(0x94);
                printf("   back (arm up)    ");
                cal_flag=8;
                beep_time=200;
                break;
        case 8: //Read joystick back and ask for joystick right
                cal_2=sw_ad12a(adch(V_JOY),1);
                if((cal_2-cal_1)<20){
                        beep_time=1000;
                        break;
                        }
                v_joy_slope=256.0/(cal_2-cal_1);
                v_joy_offset=(int)(128-(vj0*v_joy_slope));
                lcdcmd(0x94);
                printf("   right (dump)     Move joystick fully ");
                cal_flag=9;
                beep_time=200;
                break;
        case 9: //Read joystick right and ask for joystick left
                cal_1=sw_ad12a(adch(H_JOY),1);
                lcdcmd(0x94);
                printf("  left (roll back)  Move joystick fully ");
                cal_flag=11;
                beep_time=200;
                break;
        case 11: //Read joystick right and quit calibration routine
                cal_2=sw_ad12a(adch(H_JOY),1);
                if((cal_2-cal_1)<20){
                        beep_time=1000;
                        break;
                        }
                h_joy_slope=256.0/(cal_2-cal_1);
                h_joy_offset=(int)(128-(hj0*h_joy_slope));
                lcdcmd(0x01); // Clear display
                lcdcmd(0x94);
                printf("CALIBRATION COMPLETE");
                lcdcmd(0xd4);
                printf("   Press * to exit");
                cal_flag=12;
                beep_time=200;
                break;
        case 12:  //return
                return;
        default:cal_flag=1;
        }
    while(beep_time);
    }
  }
}

void e_stop(void)
{
     int disp_flag=1;
     lcdcmd(0x01);
     lcdcmd(0x94);
```

```
        printf("   EMERGENCY STOP   ");
        beep_time=500;
        timer1=1000;
        putser1('e',c1);
        putser1('e',c1);
        putser1(0x0d,c1);
        output_timer=OUTPUT_INTERVAL;
            while(!(port_rd(1)&2)){
          if(serhit1(c1))get_serial_data();
          if((loader_status==3)&&(timer1==0)&&(disp_flag)){
                  sw_relay(1);  //activate KILL transmitter
                  lcdcmd(0xd4);
                  printf("KILL xmtr activated ");
                  disp_flag=0;
                  }
          if(output_timer==0){
                  putser1('e',c1);
                  putser1('e',c1);
                  putser1(0x0d,c1);
                  output_timer=OUTPUT_INTERVAL;
                  }
          }
        sw_relay(0);  //turn off KILL transmitter
        comm_disp=10;

        return;
        }
int kb_scan(void)
{
        int key,k1,k2,k3,k4;
//  set all enable lines high
        outportb(0xe0,0);
        outportb(0xe1,0);
        outportb(0xe2,0);
        outportb(0xe4,0);

//  set h1 low and read keyboard

        outportb(0xe0,1);
        k1=(0xf0&peekb(0xfff0,0x10));
        outportb(0xe0,0);

//  set h2 low and read keyboard

        outportb(0xe1,1);
        k2=(0xf0&peekb(0xfff0,0x10));
        outportb(0xe1,0);

//  set h3 low and read keyboard

        outportb(0xe2,1);
        k3=(0xf0&peekb(0xfff0,0x10));
        outportb(0xe2,0);

//  set kh low and read keyboard

        outportb(0xe4,1);
        k4=(0xf0&peekb(0xfff0,0x10));
        outportb(0xe4,0);

//  combine key presses and return

        key=k2|(k1>>4)|(k3<<4)|(k4<<8);
        return (~key);
        }

void get_serial_data(void)
{
        int checksum,i;
        while(serhit1(c1)){
                in_block[in_block_counter++]=in_char=getser1(c1);
                if(in_char==0x0d)ser1_com.in_tail=ser1_com.in_head;
                }
        if(in_char!=0x0d)return;
        if(in_block_counter<3){
                in_block_counter=0;
```

```
                return;
                }
        for(i=0,checksum=0;i<in_block_counter-2;i++)checksum+=in_block[i];
        if((checksum&0x0f)!=(in_block[i]&0x0f)){
                in_block_counter=0;
                return;
                }
        in_block_counter=0;
        switch(in_block[0]){
         //not in neutral
        case 'n':
                loader_status=1;
                input_timer=INPUT_INTERVAL;
                break;
         //standby mode
        case 's':
                loader_status=2;
                input_timer=INPUT_INTERVAL;
                break;
         //remote mode
        case 'r':
                loader_status=3;
                learn_mode=0;
                input_timer=INPUT_INTERVAL;
                break;
         //remote mode - recording
         case 't':
                loader_status=3;
                learn_mode=1;
                input_timer=INPUT_INTERVAL;
                break;
         //remote mode - playback
         case 'l':
                loader_status=3;
                learn_mode=2;
                input_timer=INPUT_INTERVAL;
                break;
         //power fail
        case 'p':
                loader_status=0;
                 power_fail=1;
                input_timer=INPUT_INTERVAL;
                 break;
        default:
                break;
                }

        }

void send_data()
{
        int i;
        out_block[0]='c';  //first char signals 'control'
// Read steering position and store as two 4-bit characters
        i=(int)((float)sw_ad12a(adch(STEERING),1)*steering_slope+steering_offset);
        if(i>255)i=255;
        if(i<0)i=0;
        out_block[1]=(char)(((i&0xf0)>>4)|0x30);
        out_block[2]=(char)((i&0x0f)|0x30);

// Read brake position and store as two 4-bit characters
        i=(int)((float)sw_ad12a(adch(BRAKE),1)*brake_slope+brake_offset);
        if(i>255)i=255;
        if(i<0)i=0;
        out_block[3]=(char)(((i&0xf0)>>4)|0x30);
        out_block[4]=(char)((i&0x0f)|0x30);

// Read throttle position and store as two 4-bit characters
        i=(int)((float)sw_ad12a(adch(THROTTLE),1)*throttle_slope+throttle_offset);
        if(i>255)i=255;
        if(i<0)i=0;
        out_block[5]=(char)(((i&0xf0)>>4)|0x30);
        out_block[6]=(char)((i&0x0f)|0x30);

// Read arm position and store as two 4-bit characters
        i=(int)((float)sw_ad12a(adch(V_JOY),1)*v_joy_slope+v_joy_offset);
```

```
        if(i>255)i=255;
        if(i<0)i=0;
        out_block[7]=(char)(((i&0xf0)>>4)|0x30);
        out_block[8]=(char)((i&0x0f)|0x30);

// Read bucket position and store as two 4-bit characters
        i=(int)((float)sw_ad12a(adch(H_JOY),1)*h_joy_slope+h_joy_offset);
        if(i>255)i=255;
        if(i<0)i=0;
        out_block[9]=(char)(((i&0xf0)>>4)|0x30);
        out_block[10]=(char)((i&0x0f)|0x30);

// Scan keyboard and send four 4-bit characters
        i=kb_scan();
        out_block[11]=(char)(((i&0xf000)>>12)|0x30);
        out_block[12]=(char)(((i&0x0f00)>>8)|0x30);
        out_block[13]=(char)(((i&0x0f0)>>4)|0x30);
        out_block[14]=(char)((i&0x0f)|0x30);
        out_block[15]=out_block[16]=out_block[17]=out_block[18]=0x30;
//  Read video joystick and set block 16 accordingly
        if(sw_di(3)==0)out_block[16]|=1;
        if(sw_di(4)==0)out_block[16]|=2;
        if(sw_di(5)==0)out_block[16]|=4;
        if(sw_di(6)==0)out_block[16]|=8;

//  video keyboard patch--use 4 keyboard keys for video switches.
        out_block[16]|=(i&3);
        if(i&0x20)out_block[16]|=4;
        else if(i&0x200)out_block[16]|=8;

        out_block[19]=0;
        for(i=0;i<19;i++)out_block[19]+=out_block[i];
        out_block[19]=(out_block[19]&0xf)|0x30;
        out_block[20]=0x0d;
        for(i=0;i<21;i++)putser1((unsigned char)out_block[i],c1);
        return;
        }


void remote_control(void)
{
// Wait for EN/DIS button release
        while((m=kb_scan())&EN_DIS);

// Check to see if loader is in the Standby Mode--if not then error #1
        if(loader_status!=2){
                err_display(1);
                return;
                }
// Check to see that direction lever is in Neutral-- if not then error#2
        if(m&(FWD|REV)){
                err_display(2);
                return;
                }
// Check to see that gear selector switch is in the '1' position--
// if not, error #3
        if((m&(GEAR_1|GEAR_2))!=GEAR_1){
                err_display(3);
                return;
                }
// Check to see if brake is fully applied--if not, error #4
        raw_atod=sw_ad12a(adch(BRAKE),1);
        if(raw_atod*brake_slope+brake_offset<200){
                err_display(4);
                return;
                }
        WAIT_FOR_REMOTE:
        output_timer=OUTPUT_INTERVAL; // Send 'CONTROL' command
        putser1('p',c1);
        putser1('p',c1);
        putser1(0x0d,c1);
        timer1=5000;  // Allow 5 sec for the loader to respond
        strcpy(display_2," Asking for Control ");
        display_out();
        while(loader_status!=3){
              if(!output_timer){
```

```
                        output_timer=OUTPUT_INTERVAL; // Send data block
                        putser1('p',c1);
                        putser1('p',c1);
                        putser1(0x0d,c1);
                        }
                if(serhit1(c1))get_serial_data();
                if(timer1==0){
                        err_display(5);
                        break;
                        }
                }
        if(loader_status!=3)return;
        lcdcmd(0xc0);
        strcpy(display_2,"   OCC IN CONTROL   ");
        lcdcmd(0x94);


        while(!((m=kb_scan())&EN_DIS)){
                if(!(port_rd(1)&2)){
                        e_stop();
                        break;
                        }
                if(output_timer==0){
                        send_data();
                        output_timer=OUTPUT_INTERVAL;
                        }
                if(serhit1(c1)){
                         if(learn_mode==0){
                              strcpy(display_4,"      NORMAL          ");
                              }
                         if(learn_mode==1){
                              strcpy(display_4,"     RECORDING!       ");
                              }
                         if(learn_mode==2){
                              strcpy(display_4,"     PLAYBACK!        ");
                              }
                         display_out();
                         get_serial_data();
                         }
                if(input_timer==0){
                        beep_time=500;
                        lcdcmd(0x94);
                        printf("      LOSS OF        ");
                        lcdcmd(0xd4);
                        printf("   COMMUNICATION     ");
                        loader_status=0;
                        goto WAIT_FOR_REMOTE;
                        }
                if(loader_status!=3){
                        beep_time=500;
                        if(loader_status==2){
                                lcdcmd(0x94);
                                printf(" Loader In Standby  ");
                                lcdcmd(0xd4);
                                printf("                    ");
                                goto WAIT_FOR_REMOTE;
                                }
                        else break;
                        }
                }
// Wait for EN/DIS release
        while(kb_scan()&EN_DIS);
        comm_disp=10;
        return;
        }

void err_display(int error_number)
{
        beep_time=500;
        lcdcmd(0x94);
        printf("       ERROR!       ");
        lcdcmd(0xD4);
        switch(error_number){
        case 1:
                printf("Loader Not In Remote");
```