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

DEVELOPMENT OF A TETHERED MOBILE ROBOT FOR HIGHWAY MAINTENANCE

STEVEN A. VELINSKY
DAEHIE HONG
KEITH A. MUELLER

AHMCT Research Report
UCD-ARR-94-10-31-01

Final Report of Contract
IA65Q168-MOU 92-9

October 31, 1994

Advanced Highway Maintenance and Construction Technology Center

DISCLAIMER/DISCLOSURE

"The research reported herein was performed as part of the Advanced Highway Maintenance and Construction Technology Program (AHMCT), within the Department of Mechanical and Aeronautical Engineering at the University of California, Davis and the Division of New Technology and 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."

ABSTRACT

This document reports on the development of the Tethered Mobile Robot (TMR) at the Advanced Highway Maintenance and Construction Technology (AHMCT) Center at the University of California, Davis. This application of automation has the primary objective of improving the level of safety of a variety of highway maintenance and construction activities.

The TMR system is a self-propelled wheeled mobile robot that works in close proximity to a support vehicle for purposes of power, etc. The robot's position relative to the support vehicle is measured with high accuracy. As such, the support vehicle contains the associated maintenance supplies (sealant, etc.), power supply (hydraulic power supply, electrical generator, etc.), and, in many cases, the primary maintenance operation sensing devices (e.g., machine vision for crack sealing operations) and/or path planning components.

The TMR project has involved the following: a detailed literature search, development of global machine specifications, development of system design concept, and the design and construction of a downsized prototype TMR for the initial development of both the mechanical system and the required controls. This was followed with the evaluation of the first generation prototype, which led to the design and construction of a full-sized prototype TMR and relative position system. This report addresses each of the development aspects, and includes discussions of operational requirements, system configuration, system modeling, downsized system testing, control system architecture and approach, and test results of the full-size unit. Technical drawings are included as are control software listings. Finally, recommendations for further work are discussed.

EXECUTIVE SUMMARY

Many highway maintenance operations involve the use of materials and tooling within close proximity to a support vehicle. Other operations involve the use of tools which are powered by a supply on the support vehicle, etc. Considering the weight of many road maintenance devices, such as routers, and the forces that occur during their operation, the use of conventional robot/end effectors is not possible. Highway maintenance activities almost always require an end-effector to follow a specific path as opposed to merely moving from one location to another without path following requirements as in most manufacturing automation applications. Another aspect relates to the fact that most highway maintenance operations require the specific placement of the device relative to the pavement (e.g., paint nozzles, routers, etc.), which additionally complicates the use of conventional robots.

Accordingly, unique concepts have been developed to overcome the inherent disadvantages of the use of conventional robots for highway maintenance operations. This document reports on the development of the Tethered Mobile Robot (TMR) at the Advanced Highway Maintenance and Construction Technology (AHMCT) Center at the University of California, Davis. The TMR involves the use of a self-propelled robot working in close proximity to a support vehicle for purposes of power, etc., and allowing for the measurement of the robot's position relative to the support vehicle with high accuracy. As such, the support vehicle contains the associated maintenance supplies (sealant, etc.), power supply (hydraulic power supply, electrical generator, etc.), and, in many cases, the primary maintenance operation sensing devices (e.g., machine vision for crack sealing operations). Furthermore, a support system accurately determines the location of the robot relative to the support vehicle.

This development effort has involved a detailed literature search (Kochekali and Velinsky, 1994), development of global machine specifications, development of system design concept (Winters and Velinsky, 1992; Winters, et al., 1994), design and construction of a downsized prototype TMR for the initial development of both the mechanical system and the required controls (Hong, et al., 1994a), and design and construction of a full-size prototype TMR and relative position system. Additionally, significant effort has addressed both control issues (Zhang and Velinsky, 1994a & 1994b; Hong, 1994a; Hong, et al., 1994b) and accurate dynamic modeling of such a wheeeled mobile robot (Boyden and Velinsky, 1993, 1994a, 1994b). In the TMR development, we have used the crack sealing task as the primary application. Accordingly, a mobile robot sized to carry the sealant dispenser was developed, and the control algorithms are highly related to crack following.

This document concisely reviews some of the important aspects of the TMR system development. The operational requirements are first presented. Next, the TMR configuration is discussed in detail, including the unique controller hardware configuration. A passive linkage is one of the approaches employed for measuring the robot's position with respect to the support vehicle, and design details are presented including error analysis.

A downsized TMR was built to examine the configuration and to perform initial control system development. Test results of this system are presented. Since the TMR differs from previously researched wheeled mobile robots due to the anticipated high loads, detailed dynamic modeling has been part of this study. Herein, the important features of the devloped models are discussed as are model limitations.

Due to the unique nature of the TMR and its potential applications, much effort has gone towards the development of control algorithms. There are three basic modes of TMR control. The first is *Manual Control with Joystick* performed by a human operator. This control mode can be used for cases when the operator needs to manually place the TMR at a specific position or for manual path tracing of the TMR. The second is *Automatic Trajectory Tracking Control.* Using this mode, the TMR can automatically track a specific path without any manual operations. The reference path for this mode can be a pre-defined curve in a computer file. Also, the path can be generated with a real-time sensor, laser range finding sensor for the crack sealing operation for example, which forms a sensor-based real-time navigation problem. The third mode is *Robust TMR Velocity Control.* Many highway maintenance operations require a

control system that is very robust to external force disturbances, which are hard to estimate. The important features of the control algorithms are presented, and detailed testing of the TMR for the first two modes is additionally reported. Finally, recommendations for future development efforts are presented.

TABLE OF CONTENTS

LIST OF FIGURES viii

CHAPTER 1 - INTRODUCTION 1

CHAPTER 2 - OPERATIONAL REQUIREMENTS
- 2.1 Introduction 4
- 2.2 Tractive Force Ability 4
- 2.3 Robustness 5
- 2.4 Controllability 5
- 2.5 Versatility 5

CHAPTER 3 - TMR CONFIGURATION
- 3.1 Introduction 6
- 3.2 Wheel/Robot Configuration 6
- 3.3 Tracking System Configuration 7
- 3.4 Control System Configuration 12

CHAPTER 4 - LINKAGE/WORKSPACE CONSIDERATIONS 15

CHAPTER 5 - PHYSICAL MODEL CONCEPT TESTING 17

CHAPTER 6 - DYNAMIC MODELS 19

CHAPTER 7 - CONTROL ALGORITHMS 22

CHAPTER 8 - EXPERIMENTAL RESULTS
- 8.1 Introduction 31
- 8.2 Manual Control with Joystick 31
- 8.3 Automatic Trajectory Tracking Control 31
- 8.4 Sliding Mode Control 33

CHAPTER 9 - CONCLUSIONS AND RECOMMENDATIONS 39

CHAPTER 10 - REFERENCES 42

APPENDICIES
- A Technical Drawings 46
- B Critical Commercially Purchased Components 65
- C Servo Motor Controller Diagrams 67
- D TMR Control Software Code 76

LIST OF FIGURES

Figure 3-1. Conceptual TMR System Configuration 9
Figure 3-2. Conceptual TMR Layout 9
Figure 3-3. TMR Schematic 10
Figure 3-4. TMR Photograph 11
Figure 3-5. Close-up View of TMR 12
Figure 3-6. General Control System Configuration 13
Figure 5-1. Closed-Loop Test Using Table Data as Reference 18
Figure 7-1a. Flow Chart of Control Program, Main Program 26
Figure 7-1b Flow Chart of Control Program, Tracking Control 27
Figure 7-1c. Flow Chart of Control Program, Get Current Posture of TMR 28
Figure 7-1d. Flow Chart of Control Program, Joystick Control 29
Figure 7-1e. Flow Chart of Control Program, Tracking Control with Laser 30
Figure 8-1. Joystick Control Example 32
Figure 8-2. TMR Control System Software Structure 32
Figure 8-3. Trajectory Tracking Control Example 34
Figure 8-4. Crack Tracking Test Example 35
Figure 8-5. Controller Robustness to Coefficient Uncertainty 36
Figure 8-6. Controller Robustness to External Force Uncertainty 37

CHAPTER 1

INTRODUCTION

Many highway maintenance operations involve the use of materials and tooling within close proximity to a support vehicle. For example, crack sealing operations involve maintenance personnel dispensing sealant from a wand that is attached to a vehicle housing the sealant melter. Other operations involve the use of tools, which are powered by a supply on the support vehicle, etc. While the use of conventional robots seems at first consistent with many of the positional requirements of maintenance tasks, their use is hindered due to several reasons. First and foremost, commercial robots have relatively low load carrying capacity relative to their weight. Considering the weight of many road maintenance devices, such as routers, and the forces that occur during their operation, the use of conventional robot/end effectors is not possible. Highway maintenance activities almost always require an end-effector to follow a specific path as opposed to merely moving from one location to another without path following requirements as in most manufacturing automation applications. Another aspect relates to the fact that most highway maintenance operations require the specific placement of the device relative to the pavement (e.g., paint nozzles, routers, etc.), which additionally complicates the use of conventional robots.

Accordingly, unique concepts have been developed to overcome the inherent disadvantages of the use of conventional robots for highway maintenance operations. A prime example is the positioning system concept used on the Automated Crack Sealing Machine developed through the Strategic Highway Research Program's SHRP H-107A project (Velinsky, 1993). In this concept, a conventional SCARA manipulator was inverted and mounted on a linear slide to provide a redundant degree of freedom allowing the manipulator to avoid singular positions in its motion and move through any prescribed path in its dexterous workspace. In a routing configuration, the SCARA manipulator is used to guide process carts over the pavement along specific paths (following cracks). Such an approach provides accurate and consistent relative positioning between the maintenance device and the pavement, and additionally relieves the manipulator of the burden of carrying the weight of that maintenance device. The robot/slideway's use is restricted to the movement of the carts within a plane, and the determination of the carts' location through the robot joint positioning. One problem is that the mechanical advantage of the robot is dependent on its joint positions.

A natural evolution of the SHRP H-107A concept involves the use of self-propelled robot working in close proximity to a support vehicle for purposes of power, etc., and allowing for the measurement of the robot's position relative to the support vehicle with high accuracy. As such, the support vehicle would contain the associated maintenance supplies (sealant, etc.), power supply (hydraulic power supply, electrical generator, etc.), and in many cases the primary maintenance operation sensing devices (e.g., machine vision for crack sealing operations). The Tethered Mobile Robot (TMR) concept was thus developed which involves the supply of necessary maintenance materials and power through a tether to the support vehicle. Furthermore a support system accurately determines the location of the robot relative to the support vehicle, and this relative position system could be based on any of a variety of technologies; i.e., it could be through a mechanical connection (e.g., linkage), an optical connection, etc.

The TMR project has developed such a system. This development effort has involved a detailed literature search (Kochekali and Velinsky, 1994), development of global machine specifications, development of system design concept (Winters and Velinsky, 1992; Winters, et al., 1994), design and construction of a downsized prototype TMR for the initial development of both the mechanical system and the required controls (Hong, et al., 1994a), and design and construction of a full-size prototype TMR and relative position system. Additionally, significant effort has addressed both control issues (Zhang and Velinsky, 1994a & 1994b; Hong, 1994; Hong, et al., 1994b) and accurate dynamic modeling of such a robot (Boyden and Velinsky, 1993, 1994a, 1994b). In the TMR development, we have used the crack sealing task as the primary application. Accordingly, a mobile robot sized to carry the sealant dispenser was developed, and the control algorithms are highly related to crack following.

This document concisely reviews some of the important aspects during the TMR system development. The interested reader is referred to Boyden and Velinsky (1993), Hong (1994), Kochekali and Velinsky (1994), Winters and Velinsky (1992), and Zhang and Velinsky (1994b), which are detailed interim reports of this project and provide significant detail on all of the areas covered.

CHAPTER 2

OPERATIONAL REQUIREMENTS

2.1 Introduction

The objectives of the Tethered Mobile Robot (TMR) are as follows: self-propelling, controllable, robust, and compact, with the ability to accurately follow a designated path. Since the TMR will be used primarily on asphalt and/or concrete roadways, which are considered fairly smooth and hard surfaces, only wheels for self-propelling will be considered. This is because wheels are much more energy efficient than legged or treaded concepts under these conditions (Muir and Neuman, 1987). The cart must be steerable, being able to follow a defined path in the roadway. The ability to control the position, velocity, and acceleration of the cart is a must. Robustness, the ability to attenuate disturbances, is also of prime importance. Disturbances, such as irregular surfaces and sealant affixations (for highway maintenance), must not affect the TMR's performance.

2.2 Tractive Force Ability

In order for the TMR to be self-propelled, it must be able to produce enough tractive force (between the drive wheels and contact surface) to counteract the tool's resultant force (for highway maintenance) and other applied loads, while still accelerating the TMR to proper speeds. For example, during the routing operation, the resultant force from the router's blade is approximately 200 lbf (890 N) in the direction opposing the forward motion. Additionally, the total normal force necessary to ensure proper pavement cutting is approximately 500 lbf (2224 N). From this information and assuming an adequate acceleration and minimal tire slip, the minimum tire tractive force needed to propel the cart is about 300 lbf (1334 N); i.e., 200 lb (890 N) to overcome the router force plus 100 lbf (445 N) for .2 g acceleration of the 500 lbf (2224 N) vehicle.

2.3 Robustness

Robustness, as defined here, will mean the ability of the wheel configuration (and wheels) to handle disturbances and irregularities without effecting the TMR's overall performance. In addition to the wheel configuration, the other components, such as control hardware, tracking sensors, etc., must also exhibit this characteristic. In highway maintenance work, wheel disturbances will consist of, but are not limited to problems with paint and sealant making contact with the wheel and the ability to travel on irregular surfaces, such as roadways.

2.4 Controllability

Although the TMR is expected to move at relatively slow speeds, the forces acting upon it may be high. As such, the method of ignoring vehicle dynamics, which is used widely in control strategies for automatic guided vehicles (AGV's) due to their low speeds and accelerations (Smith and Starkey, 1991), is not valid; this subject has been studied in detail as part of this project (Boyden & Velinsky, 1993, 1994a, 1994b). Furthermore, actuating the different wheel configurations may be difficult depending upon the selected approach; e.g., ball wheels are difficult to actuate.

2.5 Versatility

In order to be able to perform a wide variety of highway maintenance tasks, the TMR should operate in several control modes, including: operator joystick control, local sensor based feedback control, and planned path control.

CHAPTER 3

TMR CONFIGURATION

3.1 Introduction

The TMR's configuration includes the configurations of the wheels, tracking system, sensing, and controls, all of which are discussed below.

3.2 Wheel/Robot Configuration

Many wheeled mobile robot architectures have been discussed in detail (e.g., Muir and Neuman, 1987; Alexander and Maddocks, 1989; Feng and Krough, 1991). Of the many wheel configurations, three main wheel types have been used: conventional, omnidirectional, and ball wheels.

Gentile and Mangialardi (1992) have classified conventional wheeled mobile robots with three or more wheels into nine configurations, and they have discussed the characteristics of each. The conventional wheeled mobile robot is the simplest to construct, and this type of configuration generally allows for two degrees of freedom (DOF) in travel. Both the omnidirectional wheel and the ball wheel allow for three DOF. However, these are generally much more difficult to construct and/or actuate than conventional wheels.

The TMR configuration must allow for common highway maintenance operations. The most challenging task and that closest to application is pavement routing. This provides two main constraints to the system configuration selection. First, the TMR must always allow for the router to move tangent to the path for proper cutting, and secondly, the TMR's wheel configuration must produce enough tractive force to overcome the routing blade's resultant force.

The tractive force ability of each wheel configuration is a determining factor concerning the efficiency of each configuration. If the wheel configuration cannot produce enough tractive force for our application, it is not practical. Of the various configurations considered, it was concluded that all systems which use some type of pneumatic or semi-pneumatic tire could produce adequate force, and all other non-compliant (plastics, metal, etc.) tires would most likely not. It was further determined that the conventional wheel configuration is more robust than the omnidirectional configuration when considering surface irregularities and sealant problems.

The wheel configuration selected is most similar to that of a previous design noted as Newt (see Muir and Neuman, 1987). The TMR has front caster(s) and two parallel driven rear wheels. The TMR is differentially steered, that is, the drive wheels are driven at different speeds to cause the vehicle to steer. Such a system has two DOF and it can follow a path in a plane, but has singularities in its workspace. The configuration does not allow for motion perpendicular to the rear wheels' centerline. Accordingly, added path planning, for example, will be required to place the robot at the start of a crack.

3.3 Tracking System Configuration

In order to control the position and path trajectory of the TMR, the spatial position must be known very accurately. Based on the nature of crack sealing and other maintenance operations and the intended use of the TMR, the primary sensing system will be housed on the support vehicle, and it will only be necessary to locate the relative position of the TMR. Tracking devices for autonomous and semi-autonomous vehicles have been discussed extensively in the literature. The majority of autonomous and semi-autonomous vehicles use dead reckoning as their tracking method, while other systems include vision systems, infrared systems, sonar systems, and knowledge based systems (McGillem and Rappaport, 1988; Sugimoto, et al., 1988; Smith and Starkey, 1991; Zelinsky, 1991; Dainis and Juberts, 1985). Of these methods, the equipment cost for dead reckoning is lowest, but it suffers from error accumulation, and such error accumulation would be greatly amplified based on the high tractive forces that will be necessary to accomplish the maintenance tasks of interest.

An alternative approach to the electric/electronic methods noted could be the use of a mechanical linkage between the support vehicle and the Tethered Mobile Robot. In this configuration, ideally, no forces are applied to the linkage which freely follows the motion of the TMR. The linkage system, is a passive device which allows the TMR to be tracked very accurately. It should be noted that the concept of tethering the mobile robot to a support vehicle through the use of a linkage for position measurement has not been discussed in the literature, although water and air based systems have used umbilical cords as a method of tethering vehicles. The main advantage of the linkage arrangement is its absolute positioning determination, without error accumulation. Another advantage is that it provides a means of running power and other needed cables to the TMR from the support vehicle. The disadvantages of this system would be the possible errors involved, such as linkage deflection and resolution which could not be taken into account easily.

After examining the different possibilities in tracking the TMR, it was determined that the popular methods, such as dead reckoning, infrared optics, and sonar would not meet the needed accuracy and reliability of the tracking system requirements. Therefore, the initial design concept chosen has the TMR connected to the support vehicle via a mechanical linkage. Since the TMR has self-driving capabilities, passive elements without driving capabilities are adequate for the linkage system which can be adapted to handle irregular road surfaces and a wide range of workspaces. The general configuration of the planar linkage has an encoder mounted on each joint in order to calculate the relative position of the TMR to the support vehicle. The TMR system concept configuration is depicted in Figure 3-1, and the robot configuration is shown in Figure 3-2.

We have also been testing the use of triangulation via linear encoders. In this approach, we measure relative position of the robot from two different locations on the support vehicle. It is likely that a combination of the techniques may be employed to add redundancy to the system. Additionally, a laser range finder based sensor is employed to profile the surface in the vicinity of cracks, and this sensor then provides feedback allowing the TMR to follow the crack's path. In this project, the laser range sensor was purchased, which has a significantly larger field of view than that used in the Automated Crack Sealing Machine developed in SHRP H-107A.

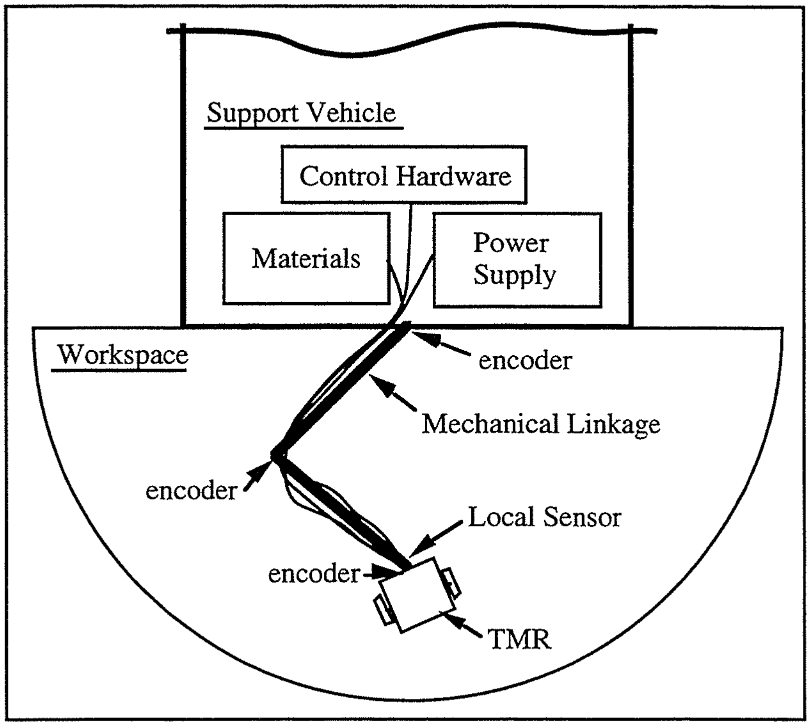


Figure 3-1. Conceptual TMR System Configuration

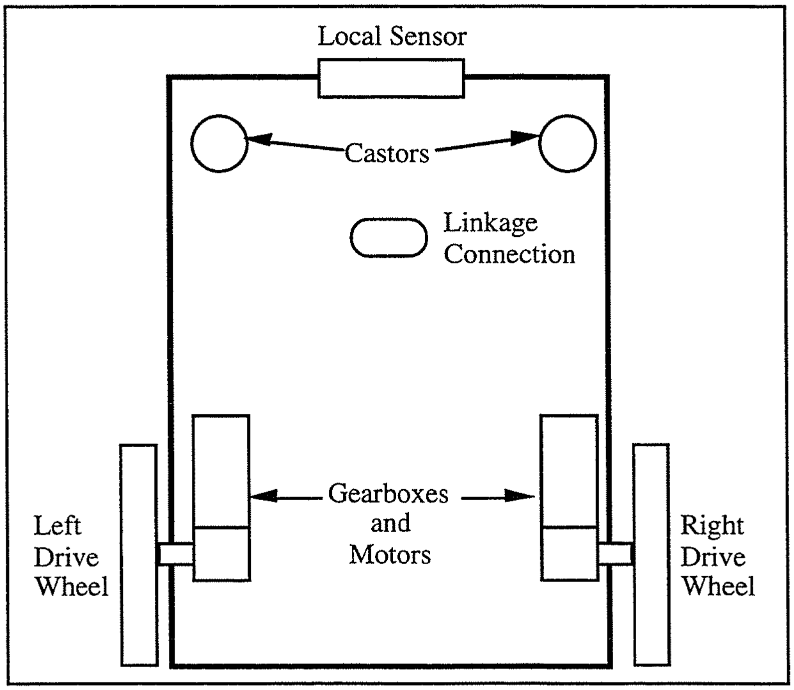


Figure 3-2. Conceptual TMR Layout

Figure 3-3 is a diagram of the TMR showing the configuration of the linkage and linear transducers. The laser range finder based local sensor is additionally depicted on the robot. The prototype TMR system is shown in Figure 3-4. Figure 3-5 is a close-up view of the local sensor on the TMR. Appendix A includes detailed drawings of linear transducer components, linkage system components, and system mounts. Numerous components have been purchased commercially, and Appendix B is a list of the most critical commercially purchased components.

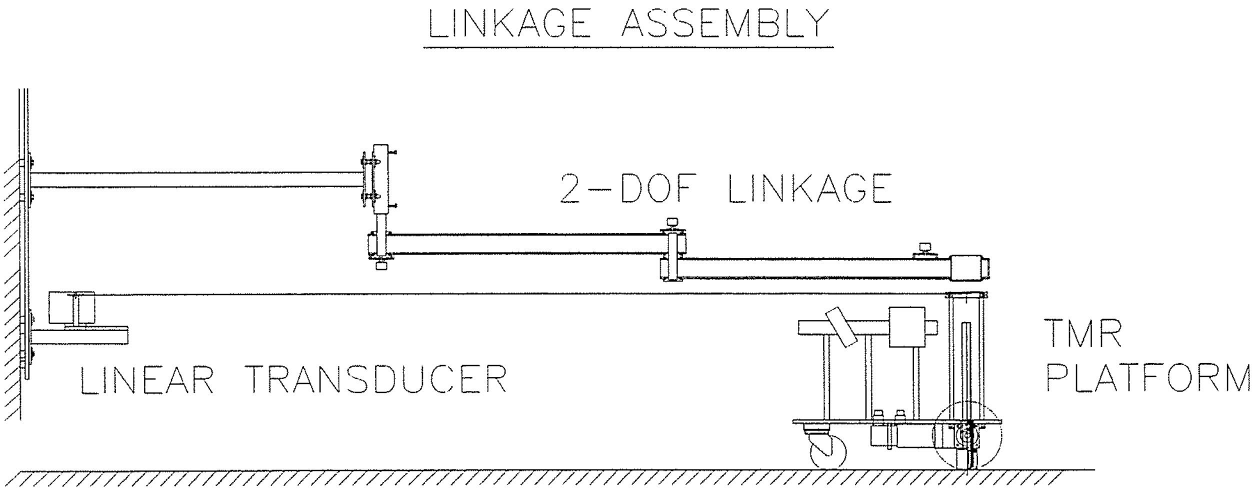


Figure 3-3. TMR Schematic

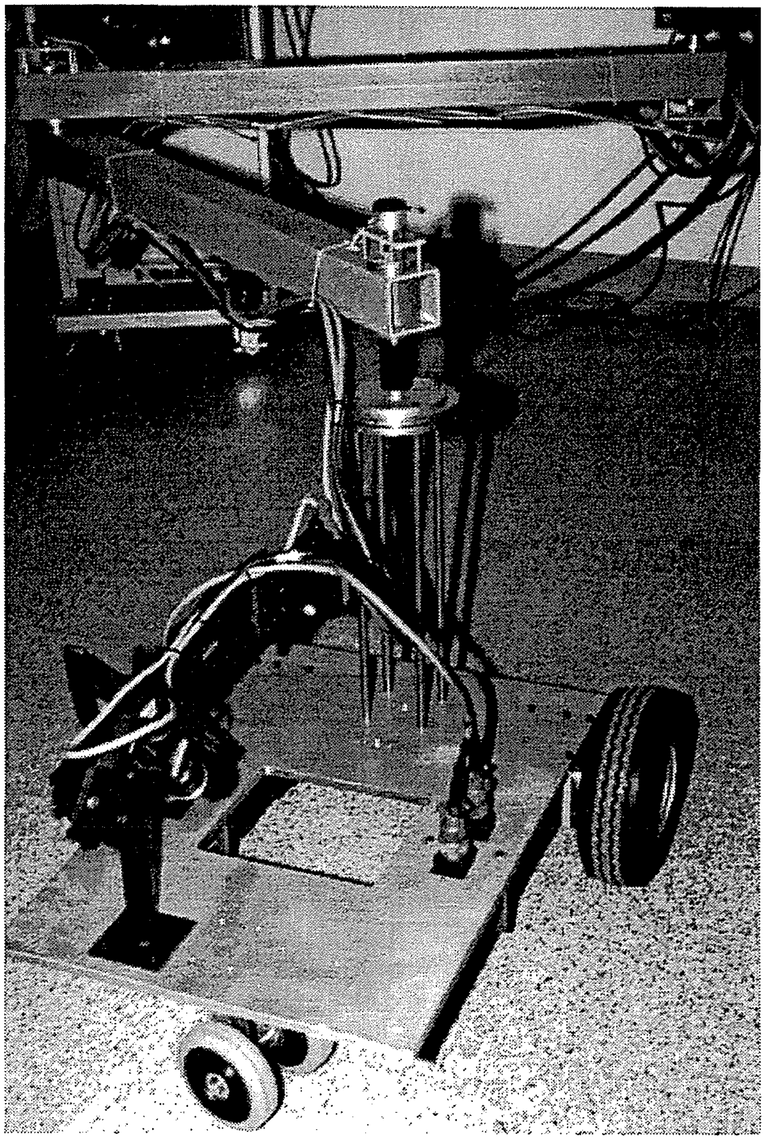


Figure 3-4. TMR Photograph

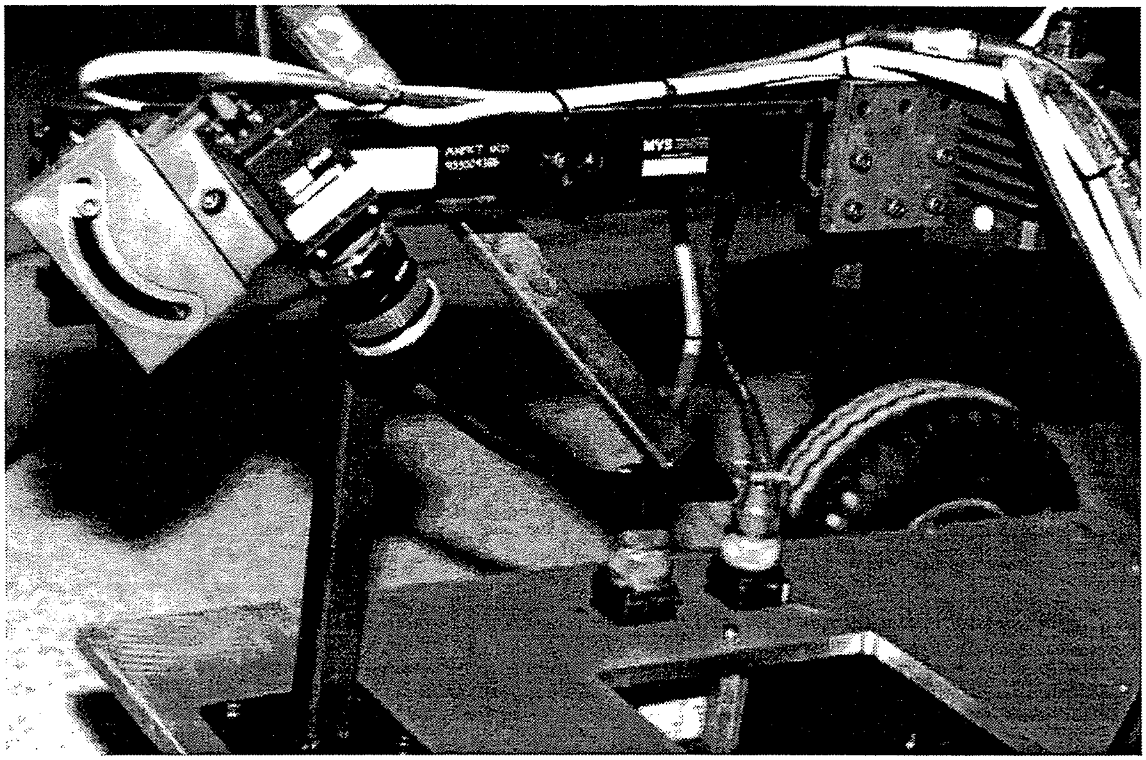


Figure 3-5. Close-up View of the Local Sensor on the TMR

3.4 Control System Configuration

The newest method for controller hardware is that of Application Specific Integrated Circuit (ASIC) technology. This high density semiconductor design method provides a highly integrated microelectronics component for industrial use, which results in higher reliability, lower cost and compactness of the control equipment. Besides the compaction of hardware, these newer controllers allow flexible implementation of software for control purposes (Yamazaki and Numazawa, 1990). The control system configuration, shown in Figure 3-6, is based on this new technology.

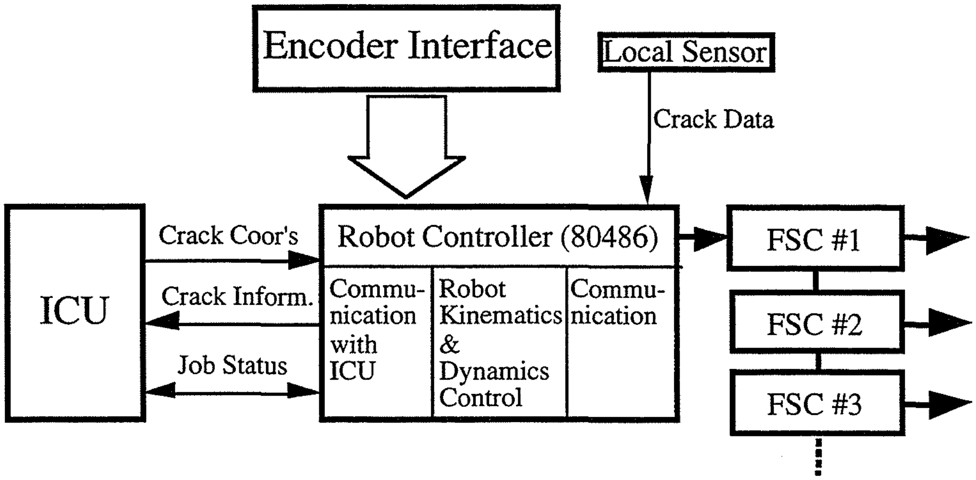


Figure 3-6. General Control System Configuration

In one method of operation, the initial location of the TMR will be based on the assumed crack starting position as determined by vision system data and the Integration and Control Unit (ICU) of the Automated Crack Sealing Machine. Actual placement of the TMR will occur through the control of each wheels. The local sensor, located at the front of the TMR, determines exact crack position and provides this data to the robot controller, which then places the TMR on this crack position. The control system consists of the robot controller, the actuators that drive the TMR wheels, the actuator controllers, and the corresponding control algorithms and software.

The 80486-based robot controller produces the control commands of the actuators by performing the motion kinematics, path planning, display functions, and information transfer to each actuator controller through ISA bus connection. Linkage joint angle data is provided by the optical encoders, and with this data, the posture vector (that is defined as a row vector containing the x-y position and the angular lotation of the TMR) are calculated. The posture vector is fed back to the tracking control to produce appropriate control input commands. Based on their superior performance, brushless DC (BLDC) motors are used to drive the TMR wheels, and gear reduction is employed to achieve high wheel torque.

The current state-of-the-art in motor control technology, Flexible Servo Control (FSC), directly controls each driving motor (Yamazaki, et al., 1987; Yamazaki, et al., 1988). The FSC has been developed by using the ASIC (Application Specific Integrated Circuit) technology and can do the current loop servo control calculation, including the feed forward control within 60 microseconds. This technology has been modified for use in the TMR as discussed in Hong, et al. (1994a, 1994b).

The motor controller employs multi-loop feedback with the use of two sensors: a Hall sensor to detect the current and an incremental optical encoder to detect the position and velocity. Once a position command is given for generating a motion, the position loop controller calculates a velocity command so as to eliminate the position error. This velocity command is supplied to the velocity loop, and, having a similar control treatment in the velocity loop and in the current loop, the final voltage output is supplied via the PWM (Pulse Width Modulation) circuit to the motor. The velocity controller controls the velocity of the rotor inertia and compensates for external forces, such as frictional forces, load forces, etc. The current controller controls the current in the R-L circuit and compensates for the back electromotive force generated inside the motor proportional to the motor velocity.

CHAPTER 4

LINKAGE/WORKSPACE CONSIDERATIONS

In designing the configuration for the linkage system, initially, some general constraints for the workspace had to be addressed. It was decided that two main workspace areas would be needed in order to have a versatile and usable highway maintenance system; one is 12 ft (3.7 m) wide, allowing the system to address a full lane width, and the second is 8 ft (2.4 m) wide, confining work to fall behind the transport vehicle and within its width, but of greater depth than the first area. The linkage system needs to be stored easily within the confines of the truck and the linkage must not have any singularity points within the workspace. Based on an extensive literature review and on the planar nature of the linkage, a general serial two degree of freedom manipulator, which would allow the largest usable work area and the simplest configuration, was selected. The manipulator link lengths are equal and each joint allows approximately 360 degrees of rotation without restriction. The design method of Gosselin and Guillot (1991) was then employed to arrive at an optimal 7 ft (2.1 m) total linkage length (3.5 ft [1.07 m] for each link). This corresponds to a workspace with a total area of 153 ft^2 (14.2 m^2), and the defined workspaces of 12 ft (3.7 m) and 8 ft (2.4 m) having depths of 3.6 ft (1.1 m) and 5.7 ft (1.7 m), respectively.

In order to properly size and design the linkage, one must understand the general bias or systematic errors. These errors include items such as calibration errors, deformation errors, and limitations of system resolution to name a few. Generally, the tooling of the TMR will be required to operate in specific locations in Cartesian space. As such, the limitations imposed upon the Cartesian coordinates by system resolution follows.

For a simple planar two degree of freedom manipulator, the uncertainty, $Y_{uncert.}$, in the end effector ordinate, y, is expressed as

$$Y_{uncert.} = \frac{Y_{error}}{y}$$

$$= \frac{l_1 \cos(\theta_1)\delta\theta_1 + \delta l_1 \sin(\theta_1) + l_2 \cos(\theta_1 + \theta_2)(\delta\theta_1 + \delta\theta_2) + \delta l_2 \sin(\theta_1 + \theta_2)}{l_1 \sin(\theta_1) + l_2 \sin(\theta_1 + \theta_2)} \tag{4.1}$$

where: θ_1, θ_2 = link angles, l_1, l_2 = link lengths, δl_1 = discrete uncertainty of link l_1, δl_2 = discrete uncertainty of link l_2, $\delta\theta_1$ = discrete uncertainty of angle θ_1, and $\delta\theta_2$ = discrete uncertainty of angle θ_2.

Evaluating Eqn. (4.1) numerically will give a general description of the uncertainty for a given set of parameters. A corresponding x-component error can similarly be written as

$$\begin{aligned} X_{uncert.} &= \frac{X_{error}}{x} \\ &= \frac{-l_1 \sin(\theta_1)\delta\theta_1 + \delta l_1 \cos(\theta_1) - l_2 \sin(\theta_1 + \theta_2)(\delta\theta_1 + \delta\theta_2) + \delta l_2 \cos(\theta_1 + \theta_2)}{l_1 \cos(\theta_1) + l_2 \cos(\theta_1 + \theta_2)}. \end{aligned} \tag{4.2}$$

Equations (4.1) and (4.2) can now be used to show the positional uncertainty as a function of angular orientation. Values for the encoders and the link length uncertainties must be included to determine the actual values.

CHAPTER 5

PHYSICAL MODEL CONCEPT TESTING

A scaled-down physical model of the TMR was built in order to examine the performance of the controllers, kinematics, and path trajectories. The down-sized TMR wheel configuration chosen is similar to a tricycle having two driven wheels at one end and one passive castor at the other. The main frame of the TMR prototype is made of aluminum and is approximately 7.1 in (18 cm) wide by 9.8 in (25 cm) long. Attached to the frame are two D.C. motors that are connected to the two driven wheels. The wheels are aluminum and employ very tractive type tires with radii equal to 1.16 in (2.95 cm). The track width of the driven wheels is approximately 8.3 in (21 cm) and the castor is located 5.3 in (13.5 cm) from the driven wheels. Stated otherwise, this configuration has two diametrically opposed drive wheels and a single free-rolling castor. This general configuration will allow two dimensional motion; therefore, any path in a plane may be traced.

The down-sized TMR was exercised in both open loop and closed loop tests. The main purpose of the open-loop testing was to determine how well the TMR's trajectory, under constant velocity, followed the theoretical path derived using the same velocities. The closed loop test showed the TMR's ability to follow a defined path given simulated error data similar to that provided by the local sensor in the actual system. In the closed-loop test, the tracking system (linkage) was ignored, and supplemented with dead reckoning as the means of positional information; this did not cause any significant errors due to the short duration of each test.

Figure 5-1 shows the results from a closed loop test which used a sinusoidal reference for simulated local sensor data. Overall the TMR tracked the reference very well (errors are sufficiently small that they are not visible in the figure), showing only minor errors while turning sharp corners, and proving the basic validity of the approach. These errors could possibly be reduced by adjusting the control gains or using a better algorithm.

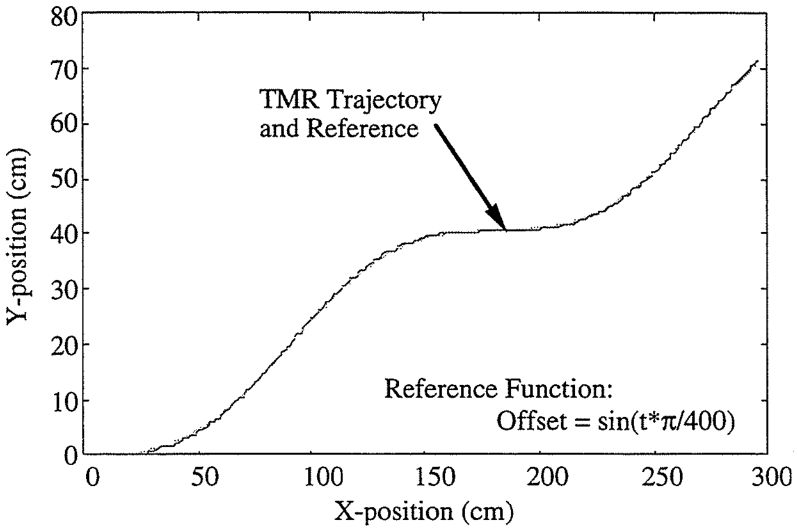


Figure 5-1 Closed-Loop Test Using Table Data as Reference

CHAPTER 6

DYNAMIC MODELS

A significant amount of research is being done and has been published on the subject of Wheeled Mobile Robots (WMRs). Of this work, a great deal is dedicated toward the development of control strategies for tracking WMRs and for the generation of path planning techniques. A fundamental aspect of research in this area involves the manner in which the WMR's position and orientation is tracked. Most researchers have used kinematic models to accomplish this task (Segovia et al., 1991; Alexander and Maddocks, 1989), arguing that because of the low speeds, low accelerations, and lightly loaded conditions under which WMRs operate, these kinematic models are valid. However, as WMRs are designed to perform heavy duty work and travel at higher speeds, dynamic modeling of these vehicles becomes increasingly important. A few researchers have derived dynamic models for wheeled mobile robots (Hemami et al., 1990; Hamdy and Badreddin, 1992), but for large, high load vehicles these models may not be valid due to imposed restrictions and potentially inaccurate tire models. As such, during the course of the TMR project, the importance of dynamic modeling of WMRs and the determination under which conditions it is necessary to use a complex tire model for high load applications has been investigated (Boyden & Velinsky, 1993, 1994a, 1994b).

The work of Boyden and Velinsky shows the importance and significance of dynamic modeling of wheeled mobile robots for high load applications. The kinematic model, which is the method most researchers have used to track WMRs, was compared to a dynamic model through computer simulation. For both differentially and conventionally steered WMRs, it was found that a kinematic model cannot accurately predict the position and orientation of a 'working' WMR under almost any conditions. Use of the kinematic model must be limited to lightweight vehicles that operate under very low speeds, very low accelerations, and under lightly loaded conditions. From these results, it is concluded that dynamic modeling of any 'working' WMR is absolutely necessary and a kinematic representation should never even be considered.

The kinematic model is also used to track the position of real vehicles by measuring wheel speeds directly, a process called dead reckoning. It has been shown that for both differentially and conventionally steered WMRs, the potential error associated with this method is large, even when the tires are producing forces at only a fraction of their friction potential, where dead reckoning is generally assumed to be relatively accurate. This demonstrates that vehicle tracking through the use of dead reckoning for a WMR when wheel speeds are measured off of the driving wheels is much less accurate than commonly assumed. As such, dead reckoning is only valid on small, lightweight vehicles that operate under very low speeds, very low accelerations, and are lightly loaded.

Another aspect of this work was to determine the importance of using a complex tire representation when dynamically modeling a WMR. A dynamic model without a tire model was created to investigate this. For differentially steered vehicles, this dynamic (without tire) model was surprisingly accurate, even approaching the friction limits of the tires. A complete and accurate tire model (such as the Dugoff tire model) is irreplaceable for any complete dynamic model that may encounter situations where the tire limits are approached or reached. However, for a differentially steered WMR which is known to stay well within the traction limits of the tires, very simple tire models can be used with excellent accuracy. A reasonable rule of thumb for *differentially steered* WMRs is as follows: if the WMR is known to stay within 50% of the tire traction limits, simple tire models (or no tire model) provide excellent accuracy and can be used with confidence; if the WMR is expected to exceed 50% of the traction limits, a more accurate tire model (one that incorporates the friction circle concept) is in order to ensure accurate simulation results.

For conventionally steered WMRs, the dynamic (without tire) model was only accurate to a fraction of the tire friction limits. In fact, the dynamic (without tire) model had an accuracy range similar to that of the dead reckoning method. Because the limits of accuracy were found to be at such low tire friction levels, it is suggested that dynamic modeling incorporating the use of an accurate tire model should always be used when simulating *conventionally steered* wheeled mobile robots. This result is expected since a tire develops force through slip which is significant in high load conditions.

CHAPTER 7

CONTROL ALGORITHMS

Since the TMR is a differentially steered wheeled mobile robot, control algorithms specific to this type of configuration have been investigated. Two detailed reports have been generated (Zhang and Velinsky, 1994b; Hong, 1994), and to follow is a synopsis of the subject.

The development of a control algorithm is always based on a mathematical model, and kinematic and dynamic models of WMRs have been studied by many authors. Muir and Neuman (1987), and Alexander and Maddocks (1990) studied the general WMR kinematics problem. Hamdy and Badreddin (1992), and Boyden and Velinsky (1994a & 1994b) developed dynamic models for WMRs. Related to control algorithm research, various approaches have been taken for the development of tracking control algorithms from PID types to fuzzy and neural network approaches. The knowledge-based algorithms are mainly suitable for real environment guidance type problems. However, for the problem of accurate path tracking control, kinematic or dynamic model based control algorithms are essential. For actual applications, the dynamic model of a WMR may contain considerable uncertainty arising from the driven force and payload, and furthermore, the computational complexity of a dynamics model is considerable.

Accordingly, kinematic models have played an important role in WMR control algorithm development. Kanayama, Nilipour, and Lelm (1988) proposed a PID control algorithm for WMR tracking control. The kinematic model was used to transform the posture error in world-coordinates to robot-fixed coordinates. A similar control algorithm was developed by Lee and Williams (1993). Kanayama, et al. (1990) proposed a nonlinear control algorithm based on a kinematic model, and they also gave a proof of the stability of the algorithm. Winters and Velinsky (1992) used a PID based control algorithm on a WMR tracking control study, and this included both simulation and experiment on their prototype "Tethered Mobile Robot."

Taking the kinematic features as a basis, the influence of WMR kinematics on tracking control performance was studied in Zhang and Velinsky, 1994a. Based on its inherent kinematics, the control structure of a differentially steered wheeled mobile robot (WMR) was studied. A global exponentially convergent control algorithm and a globally convergent tracking control algorithm were developed according to the position of the tracking point on the robot. These algorithms can track any differentiable reference path with zero steady-state error. An orientation equation under the condition of exact position tracking was proposed, and the fundamental relation between position tracking and orientation tracking was given. When the control system exactly tracks a reference path, the orientation of the WMR is completely determined by the location of the tracking point on the robot, and its desired speed and acceleration along the reference path. Examples have illustrated the position tracking control ability and orientation behavior of the developed control algorithms. Finally, it can be concluded that the exact tracking of position and orientation concurrently is not possible except for the special case when the tracking point is on the center of the baseline of the WMR.

Although dynamic model based tracking control algorithm is vital for accurate tracking control of wheeled mobile robots, it is impossible to use a full dynamic model for the control system design because of the computational complexity. Also described in Zhang and Velinsky (1994b), a systematic method for the dynamic model based tracking control algorithm for differentially driven wheeled mobile robots was developed. This method ensures the application of the nonlinear robust control system design method to the dynamic model based tracking control problem of wheeled mobile robots. On the basis of a full dynamic model, a reduced order dynamic model was developed and the influence of uncertainty was analyzed. This led to a tracking control algorithm which is exponentially convergent and robust to both unmodeled dynamics and external disturbances while maintaining the calculation simplicity. Numerical simulations showed the tracking control ability under uncertainties, such as unmodeled dynamics and payload disturbances.

Concurrently, Hong (1994) reports on the control issues which are briefly discussed below. In order to cope with all possible situations that the TMR would meet for highway maintenance operations, three different modules were proposed for the TMR control system. The first is *Manual Control with Joystick* performed by a human operator. This control mode can be used for cases when the operator needs to manually place the TMR at a specific position or for manual path tracing of the TMR. The second is *Automatic Trajectory Tracking Control.* Using this mode, the TMR can automatically track a specific path without any manual operations. The reference path for this mode can be a pre-defined curve in a computer file. The roadway sign stenciling, for example, needs a sequence of pre-defined reference path commands to draw a pre-defined roadway sign. Also, the path can be generated with a real-time sensor, such as a laser range finding sensor for the crack sealing operation, which forms a sensor based real-time navigation problem. The third mode is *Robust TMR Velocity Control.* Many highway maintenance operations require a control system that is very robust to external force disturbances that are hard to estimate. The routing process for the crack sealing operation is a good example. The routing force is difficult to predict and also severely fluctuates during the process.

A new path tracking control algorithm for a 2 DOF differentially steered mobile robot is presented for the Automatic Trajectory Tracking Control and its exponential stability is proven in this research. There has been no research that has applied the feedback linearization method for non-holonomic mobile robot control due to the non-square nature of the governing equations of motion. A new idea is proposed to overcome the inherent problem and an exponentially stable non-linear control law is successfully derived using the feedback linearization. This exponentially stable control algorithm would be more appropriate to the applications to highway maintenance operations than the control algorithms developed for general purpose mobile robots.

In the TMR control, the biggest concerns are the non-linear terms due to centrifugal forces and the routing force that is very hard to estimate. However, the upper and lower bounds of the fluctuating routing force can be estimated using an appropriate experimental method. Therefore, the robust control using the sliding mode technique is very appropriate for the TMR control problem. Much research has been conducted about wheeled mobile robot control, but there have not been any papers concerning the application of sliding mode control to the wheeled mobile robot. In this research, sliding mode control for the wheeled mobile robot is formulated.

To implement the developed algorithms, a new mobile robot controller is optimally designed using an 80486 CPU and the motor controller equipped with the Flexible Servo Controller ASIC (Application Specific Integrated Circuit) chip developed by de Schepper, et al., 1990. The motor controller is the most important part in the TMR controller hardware. The motor controller board is designed with the Flexible Servo Controller (FSC) chip and fabricated using the printed circuit board CAD software (OrCAD). The TMR controller has the unique feature of utilizing the advantages of the servo motor controller board equipped with the FSC.

Appendix C provides the circuit diagrams for the servo motor controller in addition to the printed circuit board traces. Appendix D provides software listings of Hong's control programs. The flow charts of the main program and some important subroutines are shown in the Figure 7-1. The other control software listings appear in Zhang and Velinsky, 1994b.

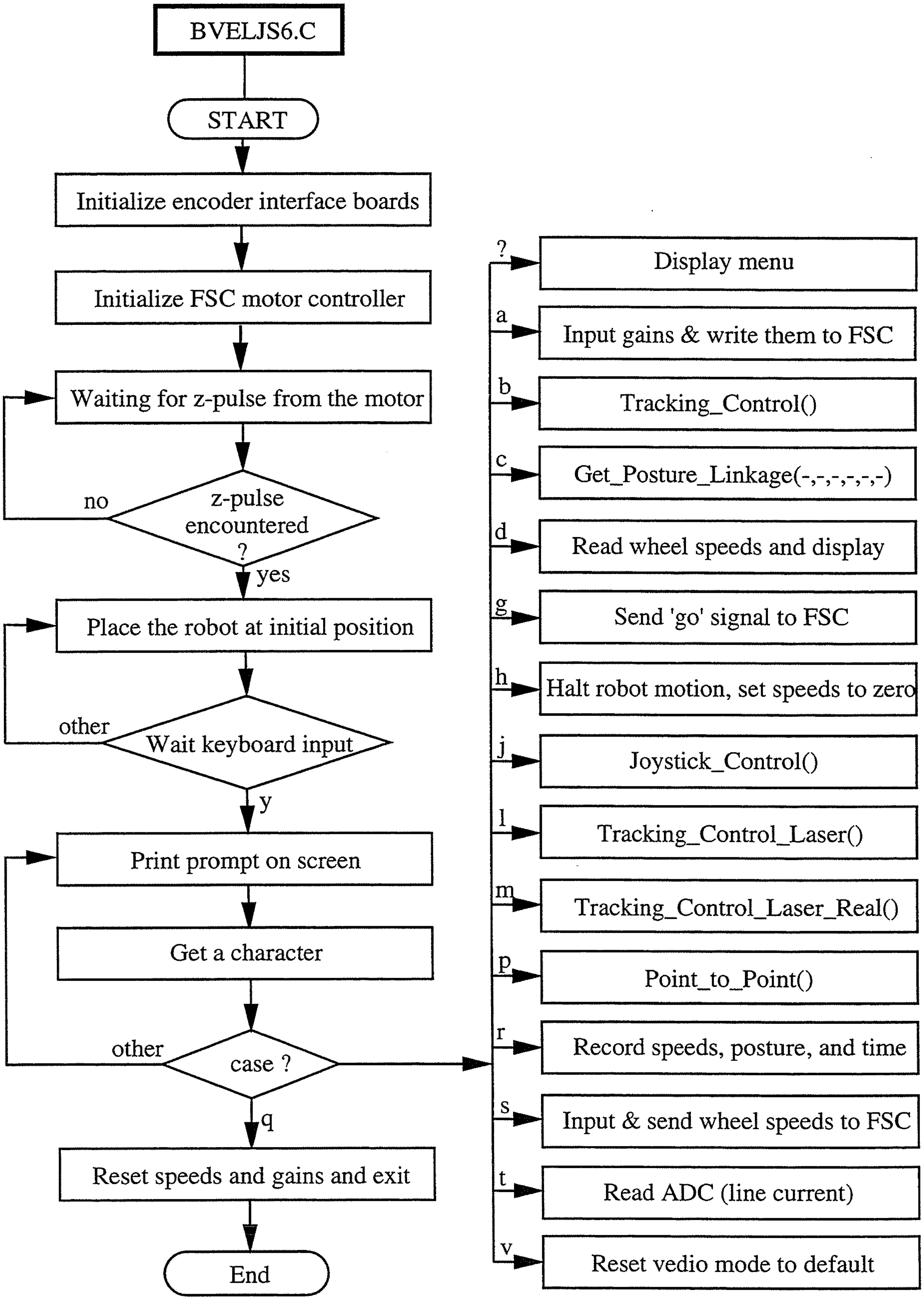


Figure 7-1a. Flow Chart of Control Program, Main Program

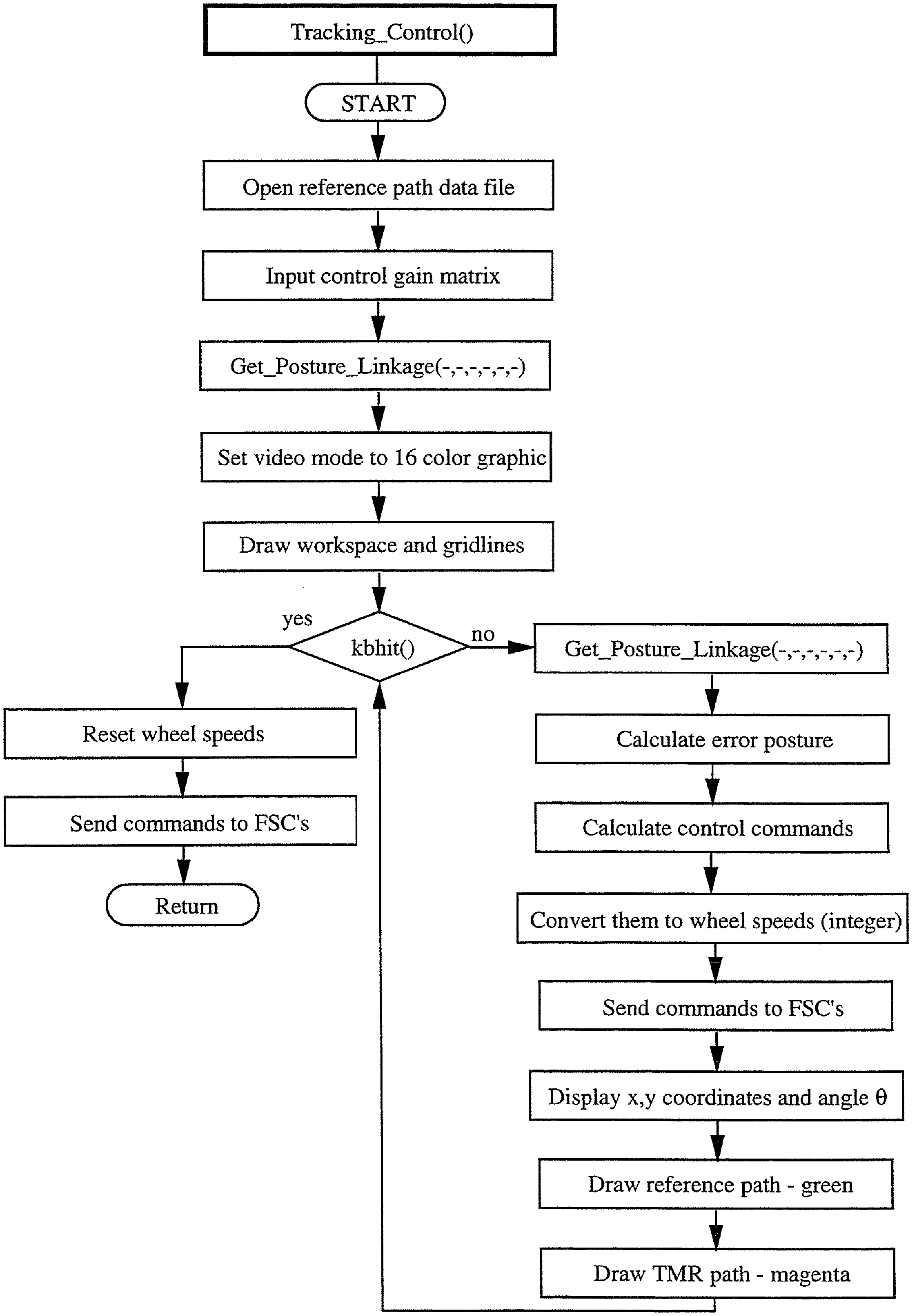


Figure 7-1b. Flow Chart of Control Program, Tracking Control

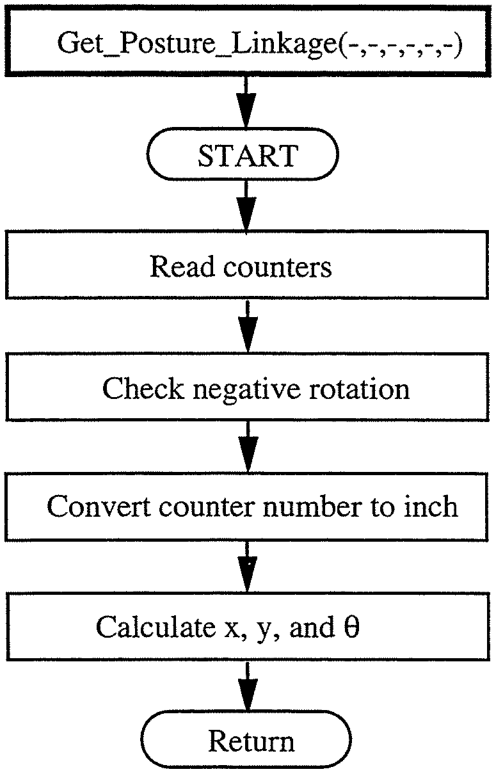


Figure 7-1c. Flow Chart of Control Program, Get Current Posture of TMR

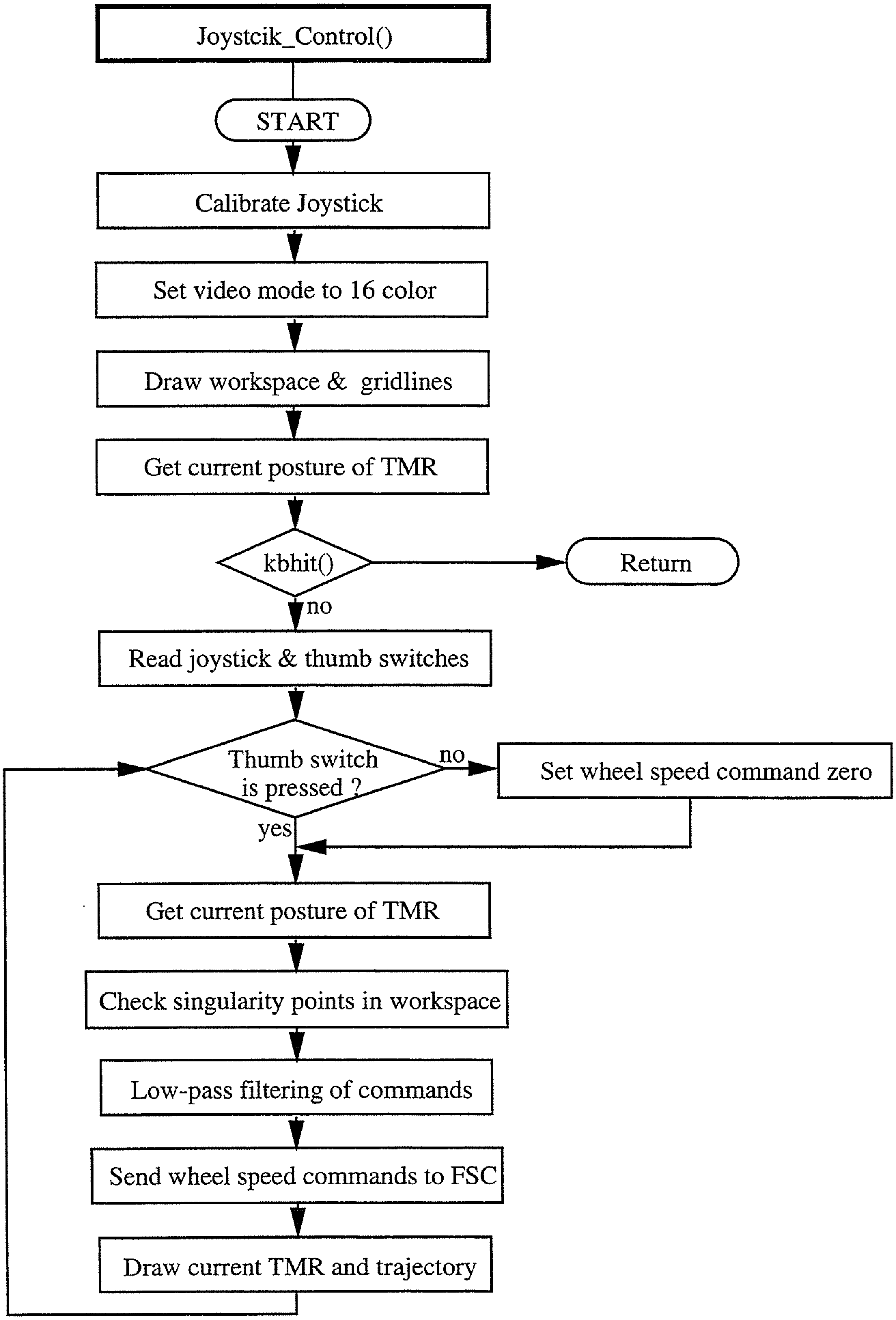


Figure 7-1d. Flow Chart of Control Program, Joystick Control

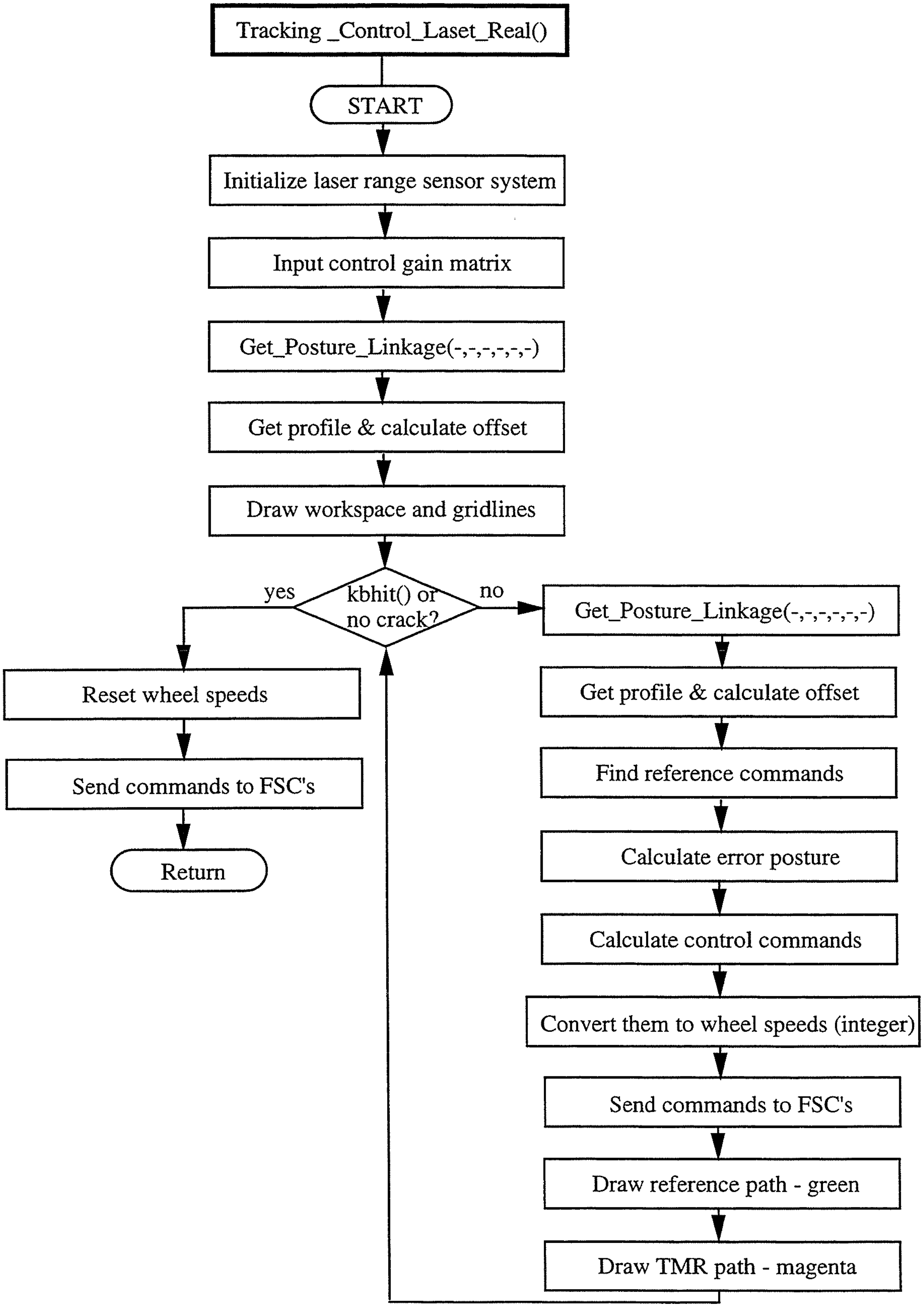


Figure 7-1e. Flow Chart of Control Program, Tracking Control with Laser

CHAPTER 8

EXPERIMENTAL RESULTS

8.1 Introduction

The developed tracking control law is implemented with the TMR platform and controller discussed earlier. Additional detail of the TMR controller hardware used is explained in Hong, 1994.

8.2 Manual Control with Joystick

The Manual Control with Joystick is successfully implemented with the TMR controller. The joystick is connected to the game port of the 80486 computer. The game port can be read using DOS BIOS interrupt routines. The joystick values are converted to the linear velocity and the angular velocity. These velocity commands are then filtered with a second order Butterworth filter. These filtered velocity commands are sent to the motor controller boards through the ISA bus interface. The position and orientation of the TMR are displayed on the screen. The control sampling time was about 0.029 second including the display. Figure 8-1 shows an experimental result of Manual Control with Joystick. The plot in the figure contains the unfiltered velocity command that is transformed with the joystick output, its filtered velocity command, and the real command following for the left wheel. The control algorithm for the wheel motor control is the Proportional and Integral (PI) control law. The performance of the joystick control was good when the proportional control gain was set to 20000 and the integral control gain to 1500.

8.3 Automatic Trajectory Tracking Control

The Automatic Trajectory Tracking Control mode is also implemented with the TMR platform and controller. The TMR control software consists of two levels, robot control and actuator (BLDC motor) control as shown in Figure 8-2. The flexible servo motor control board performs motor current and speed regulations. The TMR tracking control law resides in the 80486 machine. The TMR tracking control law produces the speed commands for each driving

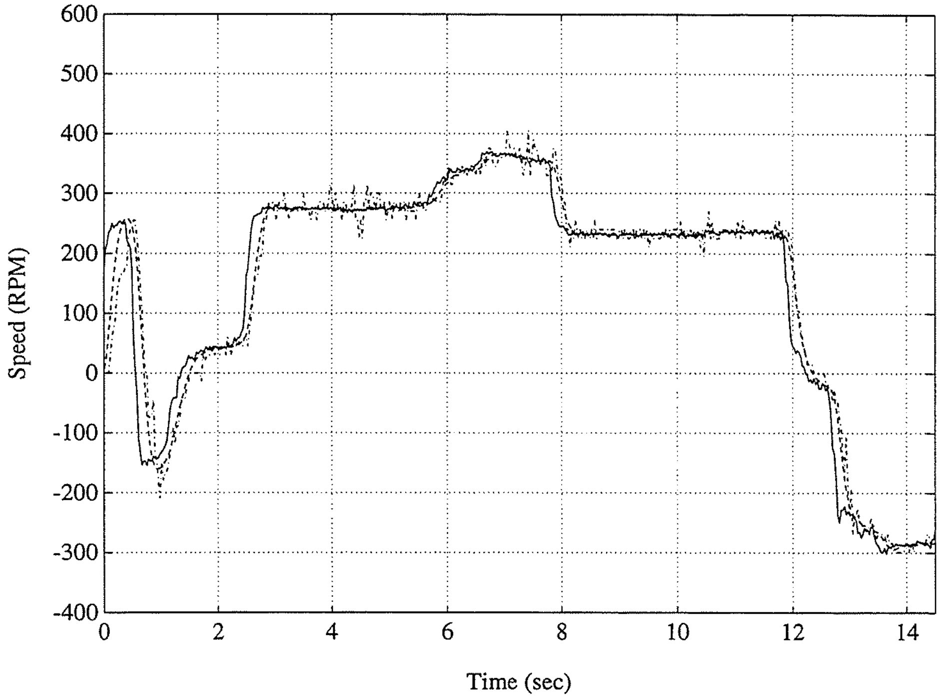


Figure 8-1. Joystick Control Example

Joystick command following of the left wheel when K_P=20000, K_I=1500, solid: unfiltered reference command from joystick, dashed: filtered reference command, dashdot: actual response of the left wheel motor.

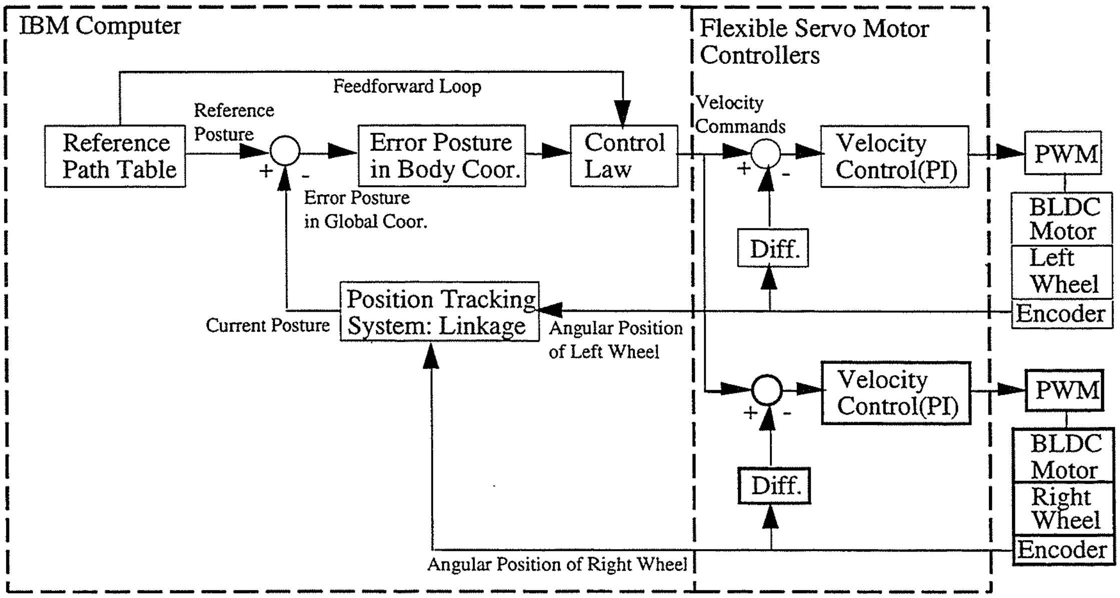


Figure 8-2. TMR Control System Software Structure

wheels and sends them to the servo motor controller. The speed command is controlled using the PI control algorithm with fast sampling time, 500 μs, in the servo motor controller.

The reference table method for a reference posture is applied for the tracking control performance test. The reference posture is generated in an off-line program and stored in a disk file. Then, this reference posture is read every control sampling time within the tracking control program. There are two purposes for which this table reference method is tested. First of all, this method can be utilized for the applications that require tracking a pre-programmed path, such as a highway stenciling operation. We can draw a roadway sign by making the TMR follow a pre-defined path. Another purpose is that it is a pre-test before implementing the tracking control with a real-time sensor, the laser range sensor for the crack sealing operation. The reference path shown in Figure 8-3a is used for the test. The reference path consists of two sinusoidal curves. The first part of the reference curve is one cycle of a sine curve and the other is a negative sine back to the initial position. Each graph shows trajectory tracking on the *X-Y* plane, angular position tracking, tracking errors, and control inputs while the TMR is following the reference posture.

Figure 8-4a shows artificial cracks routed on plywood in order to test the tracking control with the laser range sensor in a laboratory environment. Three different shapes of cracks, A, B, and C, are used for the test. Crack A is 6.35 mm (.25 in) wide. Cracks B and C are 12.7 mm (.5 in) wide. Figure 8-4b shows tracking control results along cracks A, B, and C. The linear speed was 127 mm/sec (5 inch/sec). The lateral deviation of the actual TMR trajectory from the reference crack is very small along the crack except along the high curvature region. The lateral error around a steep curve region is large relative to that of a smooth region. This is unavoidable to some extent for a non-holonomic cart moving forward with a constant speed. This error is mainly caused by the inertia of the robot platform.

8.4 Sliding Mode Control

The sliding mode control law is tested through computer simulations. The following reference trajectories are used to assess the tracking performance,

$$q_{1d} = 0.2 + 0.1 sin(2\pi t / 5)$$
$$q_{2d} = 0.5 sin(2\pi t / 5)$$

whose units are meters per second and radians per second, respectively. The following linear relationships are assumed for the external forces,

$$P_x = -100u$$
$$P_y = -10r$$
$$M_P = -10r.$$

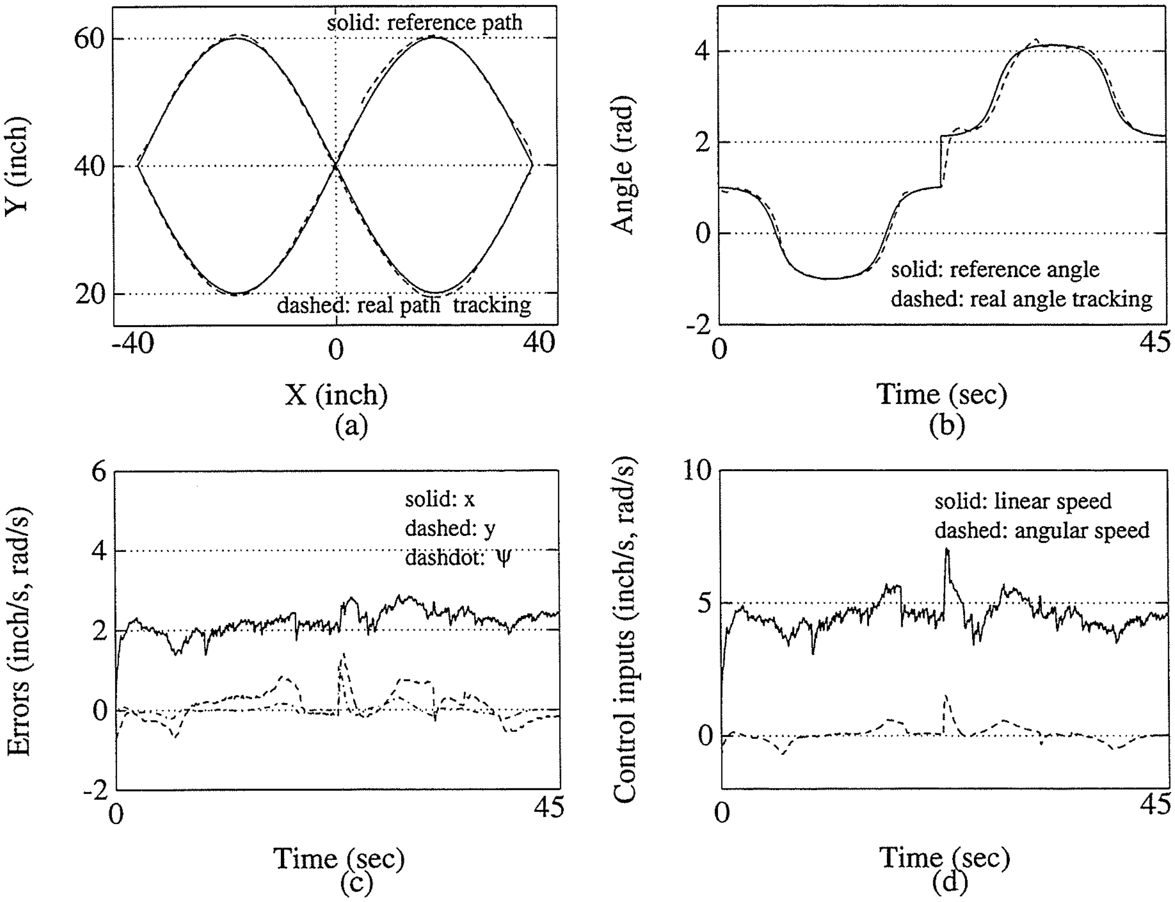
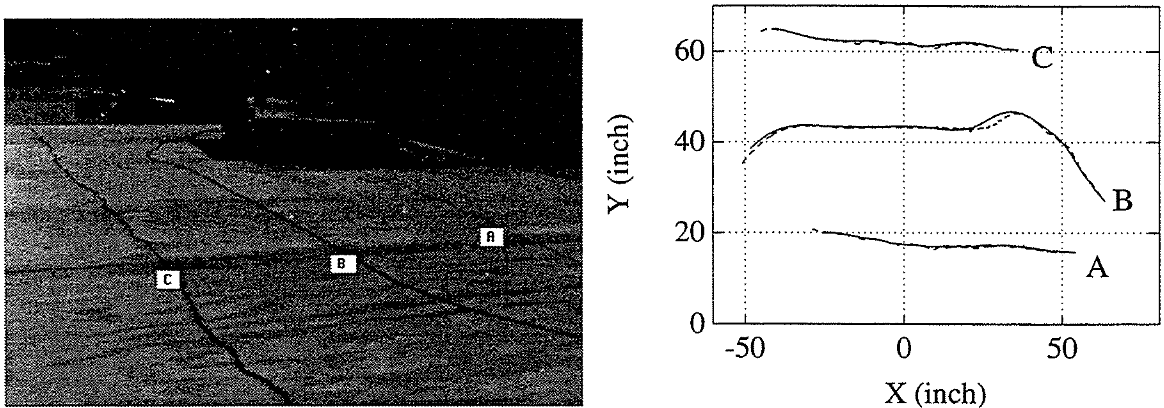


Figure 8-3. Trajectory Tracking Control Example
Reference posture table, K=[20 0; 0 20]. (a) reference path tracking, (b) reference angle tracking, (c) tracking errors, and (d) control inputs.



(a) Cracks on plywood. (b) Trajectory tracking.

Figure 8-4. Crack Tracking Test Example
Solid: actual TMR trajectory; Dashed: crack path detected with laser sensor

The previous relationships state that the routing forces and moment are proportional to the linear velocity and the angular velocity. The first linear equation about P_x has been proved through a simple test.

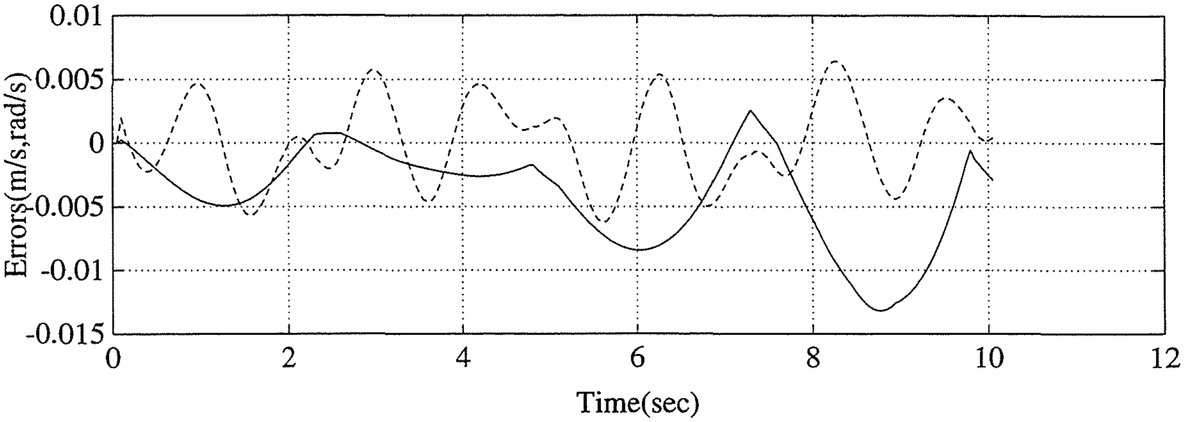
In order to see the robustness of the control law to the uncertainties, some parameters are varied and then control simulations are performed. The equations of motion are rewritten for easy programming as

$$\dot{u} = a_1 r^2 + a_2 P_x + a_3(\tau_l + \tau_r)$$
$$\dot{r} = b_1 ur + b_2 P_y + b_3 M_P + b_4(\tau_l - \tau_r)$$

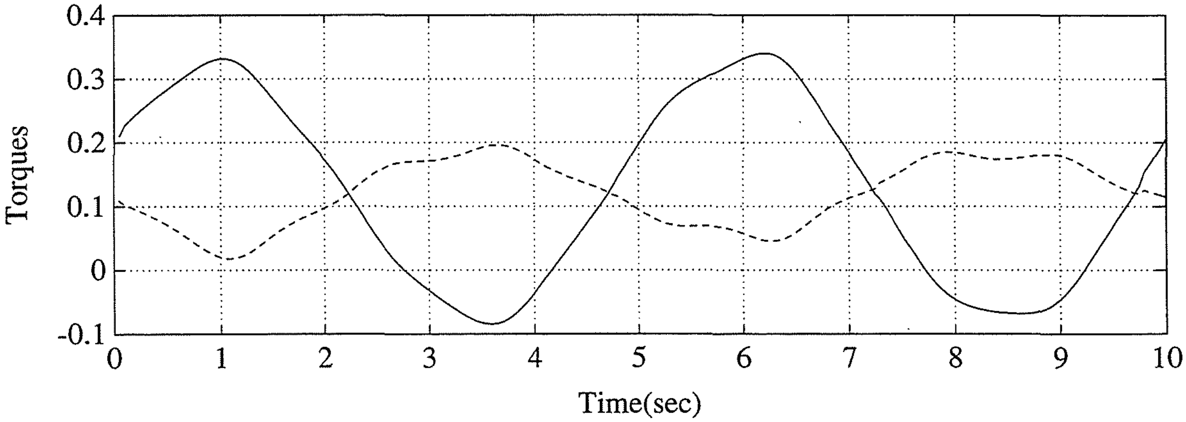
where the coefficients a_1, a_2, a_3, b_1, b_2, b_3, and b_4 are easily obtained from the original equations of motion. The coefficients a_1, a_3, b_1, and b_4 are varied as

$$a_1 = \hat{a}_1(1 + 0.1 sin(\pi t)),$$
$$a_3 = \hat{a}_3(1 + 0.1 sin(0.5\pi t)),$$
$$b_1 = \hat{b}_1(1 + 0.1 sin(2\pi t)), \text{ and}$$
$$b_4 = \hat{b}_4(1 + 0.1 sin(1.5\pi t))$$

and the control simulation is performed in order to see the robustness to the parametric uncertainties. Figure 8.5a shows the tracking errors and 8.5b represents the control inputs. We can see that the tracking control performance and the robustness to the parametric uncertainties are very good.



(a) Tracking errors; solid: linear speed; dotted: angular speed.



(b) Smooth control inputs; solid: left wheel torque; dotted: right wheel torque, unit: N-m.

Figure 8-5. Controller Robustness to Coefficient Uncertainty

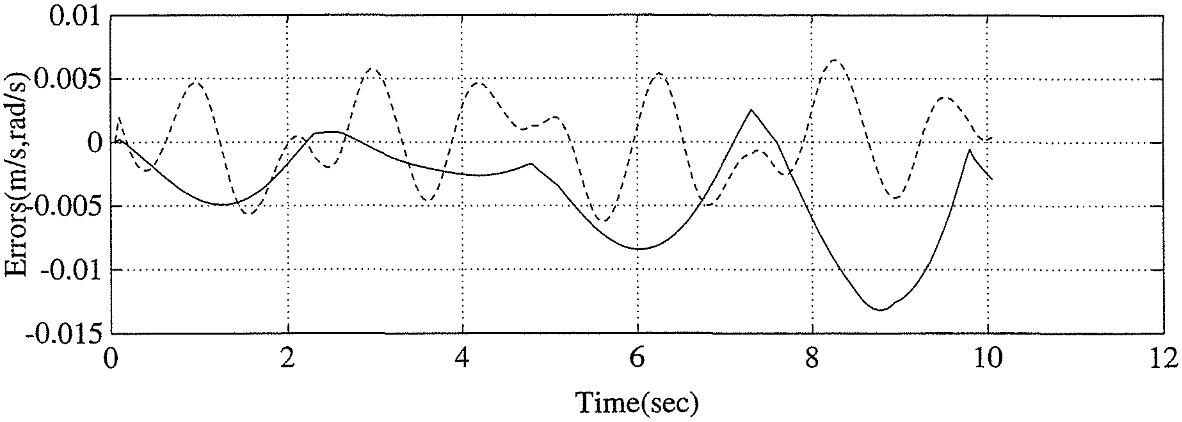
Smooth control inputs and resulting control performances when including coefficient uncertainties, such that: λ=[20 0; 0 20], η=[0.1; 0.1], Φ=0.8

Figure 8-6 shows the tracking control performance when the uncertainties on the external forces and moment are included as

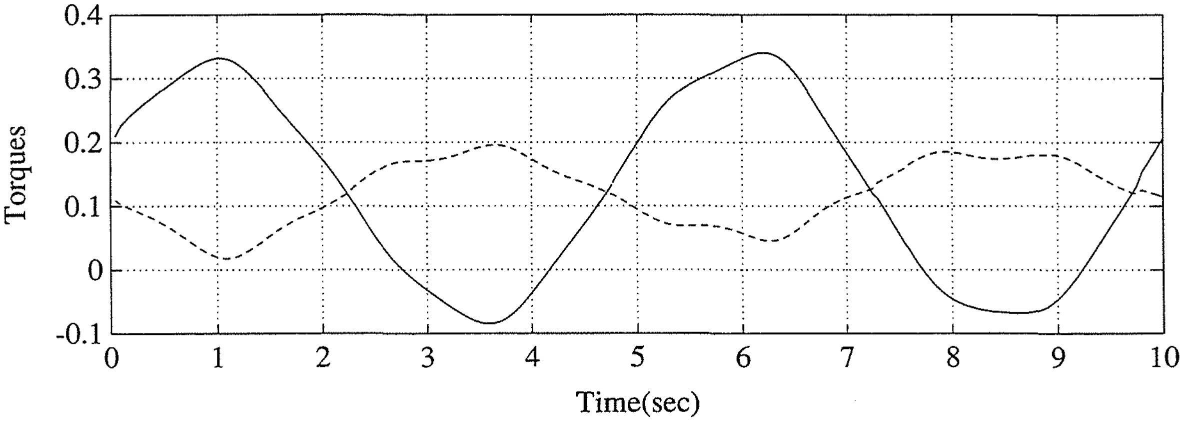
$$P_x = \hat{P}_x(1 + 0.1sin(\pi t)),$$

$$P_y = \hat{P}_y(1 + 0.1sin(2\pi t)), \text{ and}$$

$$M_P = \hat{M}_P(1 + 0.1sin(1.5\pi t)).$$



(a) Tracking errors; solid: linear speed; dotted: angular speed.



(b) Smooth control inputs; solid: left wheel torque; dotted: right wheel torque, unit: N-m.

Figure 8-6. Controller Robustness to External Force Uncertainty

Smooth control inputs and resulting control performances when including the uncertainties in the external forces, such that: λ=[20 0; 0 20], η=[0.1; 0.1], Φ=0.8.

The plots of tracking errors show that the control performance is very good even though the external forces are not exactly estimated. Also, control chattering disappears as shown in Figure 8.6b.

From several simulations assessing tracking control performance and robustness, we can conclude that the developed sliding mode controller is very appropriate for our purposes.

CHAPTER 9

CONCLUSIONS AND RECOMMENDATIONS

9.1 Conclusions

The Tethered Mobile Robot (TMR) has been developed at the Advanced Highway Maintenance and Construction Technology (AHMCT) Center at the University of California, Davis to address the many unique requirements of highway maintenance and construction activities. The TMR involves the use of a self-propelled robot working in close proximity to a support vehicle for purposes of power, etc., and allowing for the measurement of the robot's position relative to the support vehicle with high accuracy. As such, the support vehicle contains the associated maintenance supplies (sealant, etc.), power supply (hydraulic power supply, electrical generator, etc.), and, in many cases, the primary maintenance operation sensing devices (e.g., machine vision for crack sealing operations). Furthermore, a support system accurately determines the location of the robot relative to the support vehicle.

This development effort has involved a detailed literature search (Kochekali and Velinsky, 1994), development of global machine specifications, development of system design concept (Winters and Velinsky, 1992; Winters, et al., 1994), design and construction of a downsized prototype TMR for the initial development of both the mechanical system and the required controls (Hong, et al., 1994a), and design and construction of a full-size prototype TMR and relative position system. Additionally, significant effort has addressed both control issues (Zhang and Velinsky, 1994a & 1994b; Hong, 1994; Hong, et al., 1994b) and accurate dynamic modeling of such a wheeled mobile robot (Boyden and Velinsky, 1993, 1994a, 1994b). In the TMR development, we have used the crack sealing task as the primarily application. Accordingly, a mobile robot sized to carry the sealant dispenser was developed and the control algorithms were highly related to crack following. This document has concisely reviewed some of the important aspects of the TMR system development. The interested reader is referred to the noted reports for additional detail.

9.2 Recommendations

While the TMR concept has considerable applicability to the highway maintenance area, a system comprised of the coordinated actions of multiple wheeled mobile robots will provide much additional flexibility. Such a system will allow for completion of the majority of maintenance tasks that involve several subtasks. For example, it will allow for the entire crack sealing process to be accomplished including routing and sealing. Accordingly, it is recommended that a second wheeled mobile robot with adequate size and power to accomplish such tasks as routing be built, and the two robots be integrated for coordinated tasks. Furthermore, software developments and system architectures should be developed in such a manner to later accommodate the coordinated actions of several wheeled mobile robots.

The coordinated activities of multiple robots has been the subject of numerous recent studies; e.g., Xia, et al. (1994), Ishida, et al. (1994), Borenstein (1994). Much like the previous wheeled mobile robot work outside of the AHMCT Center, the research on coordinated multiple wheeled robots has been mostly concerned with laboratory research type vehicles. The published papers do not address high load and/or high speed practical applications. It is likely that some of the approaches do, however, have the potential to be extended to allow the control of multiple "working" wheeled mobile robots. As such, the detailed literature review should be continued with emphasis on multiple robot systems. Due to the rapidly changing technology, this is an essential component.

Concurrently, other critical tasks should be undertaken. First, the implementation of more complex and robust control algorithms should occur on the existing TMR. This should be coupled with extensive testing of the relative position measurement systems. Design specifications for the enlarged WMR needs to be developed to best meet the needs of a variety of possible operations, including routing. Also, potential applications outside of crack sealing should be identified with the long term plan of identifying a specific task for TMR implementation.

The initial TMR control hardware is based on the Flexible Servo Control (FSC) architecture as discussed earlier in this report. Use of this approach has allowed rapid development of the TMR. However, the motor control hardware is tied to the specific FSC computer chip which is not commercially available. While this approach has expedited the TMR demonstration, the long term interests are better served with the use of commercially available and more general hardware. Thus a transition to such hardware is necessary.

To couple a second mobile robot to the system, a relative measurement system between the two mobile robots will be necessary. It would be desirable to utilize many of the same components on the robots and on the measurement systems. Also, control algorithms for the coordinated activities of multiple wheeled mobile robots should be developed. Based on the intended application area, the controllers need to be particularly robust, and methods for analytically incorporating uncertainties in the models needs to be continued.

It is recommended that any developments be first tested in a laboratory environment to ensure proper operation under controlled conditions. Following the laboratory test, the integrated system should also be tested outside of the laboratory with the use of a support vehicle.

The ultimate goal of future development should be the demonstration of the coordinated multiple wheeled mobile robot system in an actual highway maintenance task. Potential application tasks include, but are not limited to: crack routing and sealing, automated highway system's magnetic sensor installation, and mud-jacking of pavement slabs.

CHAPTER 10

REFERENCES

Alexander, J.C.; Maddocks, J.H. (1989) "On the Kinematics of Wheeled Mobile Robots," *Intnl. J. of Robotics Research*, 8(5).

Borenstein, J. (1994) "The CLAPPER: A dual-drive mobile robot witrh internal correction of dead-reckoning errors," *Proc. of the IEEE Conference on Robotics and Automation*, pp. 3085-3090.

Boyden, D. and Velinsky, S.A. (1993) "Dynamic Modeling of Wheeled Mobile Robots," *AHMCT Research Report UCD-ARR-93-10-5-01*, University of California at Davis.

Boyden, D. and Velinsky, S.A. (1994a) "Dynamic Modeling of Wheeled Mobile Robots for High Load Applications," *Proc. of the IEEE International Conference on Robotics and Automation*, pp. 3071-3078.

Boyden, D. and Velinsky, S.A. (1994b) "Limitations of Wheeled Mobile Robot Kinematic Models," *Proc. of the 4th International Workshop on Advances in Robot Kinematics*, pp. 3071-3078.

Dainis, A.; Juberts, M. (1985) "Accurate Remote Measurement of Robot Trajectory Motion," *Proc. IEEE Conf. on Robotics and Automation*, pp. 92-99.

Feng, D.; Krogh, B.H. (1991) "Dynamic Steering Control of Conventionally Steered Mobile Robots," *J.of Robotics Systems*, 8(5), pp. 699-721.

Gentile, A.; Mangialardi, L. (1992) "Comparisons Between Different Mobile Robot Configurations," *Proc. Third Intnl. Workshop on Advances in Robot Kinematics*, Ferrara Italy.

Gosselin, C.M.; Guillot, M. (1991) "The Synthesis of Manipulators with Prescribed Workspace," *J. of Mechanical Design*, 113(4), pp. 451-455.

Hamdy, A. and Badreddin, E., 1992, "Dynamic Modelling of a Wheeled Mobile Robot for Identification, Navigation, and Control", Robotics and Flexible Manufacturing Systems, Elsevier Science Publisher B. V. (North-Holland), pp. 119-128.

Hemami, A., Mehrabi, M.G. and Cheng, R.M.H., 1990, "A New Control Strategy for Tracking in Mobile Robots and AGV's", IEEE, pp. 1122-1127.

Hong, D. (1994a) "Control System Development for Tethered Mobile Robot for Automating Highway Maintenance Operation," *AHMCT Research Report UCD-ARR-94-09-12-01*, University of California at Davis.

Hong, D., Yamazaki, K., and Velinsky, S.A. (1994b) "S-Cube: Super Servo System for Intelligent Motion Control for Mechatronics Systems," Intelligent Automation and Soft Computing (M. Jamshidi, C.C. Nguyen, R. Lumia, and J. Yuh, eds.), Vol. 1, TSI Press, pp. 505-510.

Ishida, Y., Asama, H., Tomita, S., Ozaki, K., Matsumoto, A., and Endo, I. (1994) "Functional complement by cooperation of multiple autonomous robots," *Proc. of the IEEE Conference on Robotics and Automation*, Vol. 3, pp. 2476-2481.

Kanayama, Y., Nilipour, A., and Lelm, C. (1988) "A locomotion control method for autonomous vehicles," *Proc. of the IEEE Conference on Robotics and Automation*, Vol. 2, pp. 1315-1317.

Kanayama, Y., Kimura, Y., Miyazaki, F., and Noguchi, T. (1990) "A stable tracking control method for an autonomous mobile robot," *Proc. of the IEEE International Conference on Robotics and Automation*, Vol. 1, pp. 384-389.

Kochekali, H. and Velinsky S.A. (1994) "Adaptation of wheeled mobile robots to highway maintenance operations," *AHMCT Research Report UCD-ARR-94-1-5-01*, University of California at Davis.

Lee, S.S. and Williams, J.H. (1993) "A fast tracking error control method for an autonomous mobile robot," *Robotica*, Vol. 11, pp. 205-215.

McGillem, C.D.; Rappaport, T.S. (1988) "Infra-red Location System for Navigation of Autonomous Vehicles," *Proc. IEEE Conf. on Robotics and Automation*, pp. 1236-1238.

Muir, P.F.; Neuman, C.P. (1987) "Kinematic Modeling of Wheeled Mobile Robots," *J.of Robotic Systems*, 4(2), pp. 281-340.

Segovia, A., Rombaut, M., Preciado, A. and Meizel, D. (1991) "Comparative Study of the Different Methods of Path Generation for a Mobile Robot in a Free Environment", *1991 IEEE Conference*, pp. 1667-1670.

Smith, D.E.; Starkey, J.M. (1991) "Overview of Vehicles Models, Dynamics, and Control Applied to Automated Vehicles," Advanced Automotive Technologies, ASME Pub.# DE-Vol. 40, pp. 69-88.

Sugimoto, G.; et al. (1988) "Practical Course Follow Performance of an AGV without Fixed Guideways." *Proc. of USA-Japan Symposium on Flexible Automation - Crossing Bridges: Advances in Flexible Automation and Robotics*, pp. 651- 655.

Velinsky, S.A. (1993) "Fabrication and Testing of Maintenance Equipment Used for Pavement Surface Repairs," Final Report of SHRP H-107A, National Research Council Strategic Highway Research Program, Washington DC.

Winters, S.E. and Velinsky, S.A. (1992) "Development of a tethered mobile robot (TMR) for highway maintenance," *AHMCT Research Report UCD-ARR-92-11-25-01*, University of California at Davis.

Winters, S.E., Hong, D., Velinsky, S.A., and Yamazaki, K. (1994) "A new robotic system concept for automating highway maintenance operations," *Proc. of SPACE'94 - ASCE Conference on Robotics for Challenging Environments*, pp. 374-382.

Xia, F., Velastin, S.A., and Davies, A.C. (1994) "A parallel simulation of multiple mobile robots using the DORIS design method," *Proc. of the IEEE Conference on Robotics and Automation*, Vol. 3, pp. 2482-2487.

Yamazaki, K.; de Schepper, F; Kamiyama, M. (1987) "Development of Flexible Actuator Controller for Advanced Machine Tool and Robot Control," *Annals of the CIRP*, 36(1), pp. 285-288.

Yamazaki, K.; et al. (1988) "Application of ASIC-Technology to Mechatronics Control: Development of the Flexible Servo Peripheral Chip," *Annals of the CIRP*, 37(1), pp. 329-332.

Yamazaki, K.; Numazawa, S. (1990) "High Speed and High Accuracy Control of a Direct Drive Robot," Automation of Manufacturing Processes, ASME Pub.#DSC-Vol. 22, pp.83-88.

Zelinsky, A. (1991) "Mobile Robot Map Making Using Sonar," *J. of Robotic Systems*, 8(5), pp. 557-578.

Zhang, Y.L. and Velinsky S.A. (1994a) "On the Tracking Control of Differentially Steered Wheeled Mobile Robots," submitted for publication.

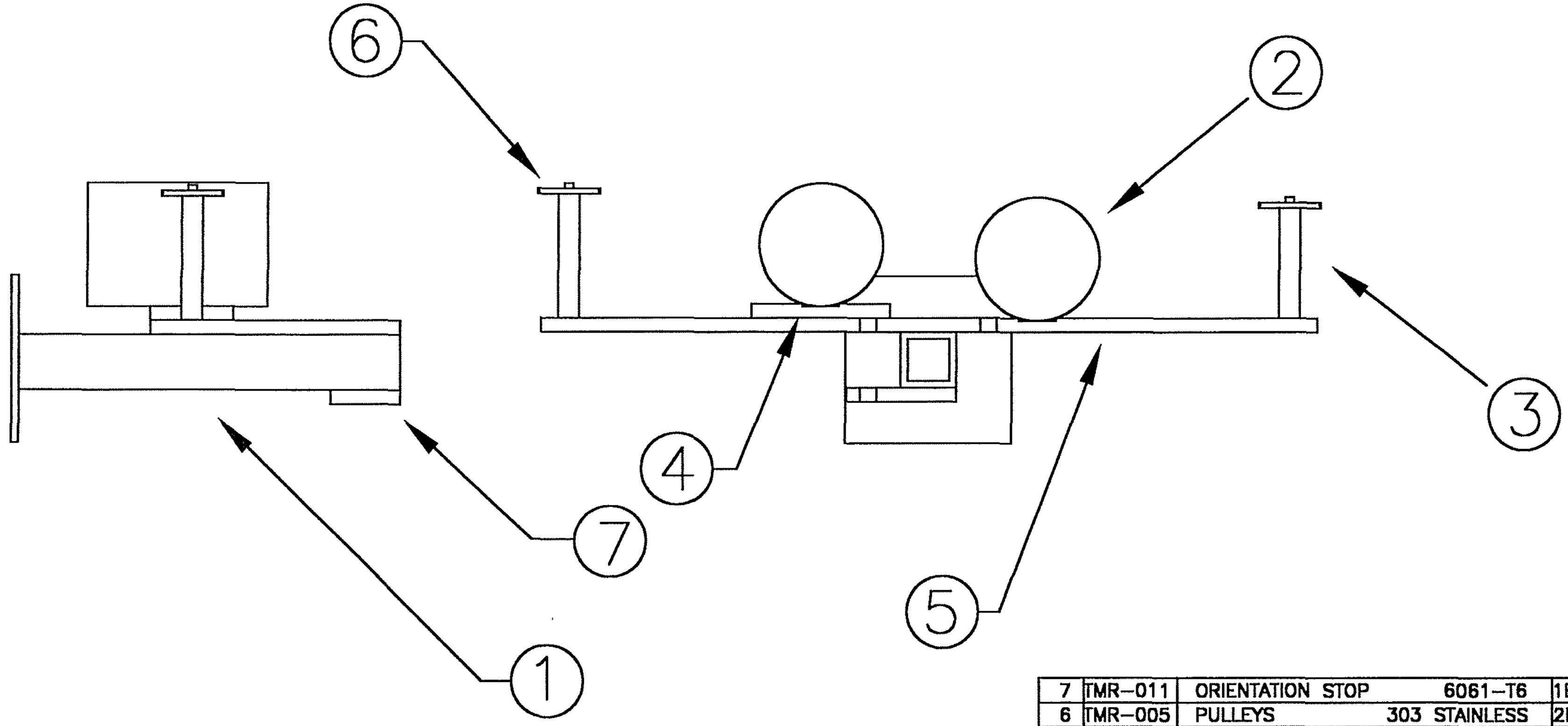
Zhang, Y.L. and Velinsky, S.A. (1994b) "Tracking Control Algorithms for the Tethered Mobile Robot," *AHMCT Research Report UCD-ARR-94-06-30-01*, University of California at Davis.

APPENDIX A

TECHNICAL DRAWINGS

Drawing #	Title	Description
TMR-001	Overall View	Linear Transducer Assembly Picture
TMR-002	Applicator Plate	Linear Transducer Component
TMR-003	Linear Transducer Base	Linear Transducer Component
TMR-004	Set Off Block	Linear Transducer Component
TMR-005	Pulley	Linear Transducer Component
TMR-006	Pulley Standoffs	Linear Transducer Component
TMR-007	Rotation Assembly	Linkage/Lin. Trans. Mount to TMR
TMR-008	Line Connection	TMR Rotation Mount Component
TMR-009	Rotary Components	TMR Rotation Mount Component
TMR-010	Encoder Platform	TMR Rotation Mount Component
TMR-011	Extra Components	TMR Rotation Mount Component
TMR-100	Arm Assembly	Linkage Assembly Picture
TMR-101	Arm Base Mount	Linkage Component
TMR-102	Vari Plate	Linkage Component
TMR-103	Height Adj. Rod/..	Linkage Component
TMR-104	Bearing Capture	Linkage Component
TMR-105	Arm Tubes	Linkage Component
TMR-106	Outside Pipe	Linkage Component

		REVISION			
LTR	ZONE	DESCRIPTION	DATE	DRWN	APP
A		INITIAL RELEASE	3/9/94	KJM	DAB



NOTES:

1) DRAWING FOR REFERENCE ONLY

ITEM NO.	PART / ID NO.	NOMENCLATURE / DESCRIPTION	MATERIAL / SPECIFICATION	QTY REQD
7	TMR-011	ORIENTATION STOP	6061-T6	1EA
6	TMR-005	PULLEYS	303 STAINLESS	2EA
5	TMR-002	TRANSDUCER PLATE	TOOLING PLATE	1EA
4	TMR-004	SET OFF BLOCK	6061 ALUMINUM	1EA
3	TMR-006	PULLEY STANDOFFS	303 STAINLESS	2EA
2		LINEAR CABLE TRANSDUCERS		
1	TMR-003	BASE MOUNT	1018 STEEL	1EA

PART LIST

UNLESS OTHERWISE NOTED:
TOLERANCES
.XX ± 0.01 ANGLES ± 0.5°
.XXX ± 0.005
64

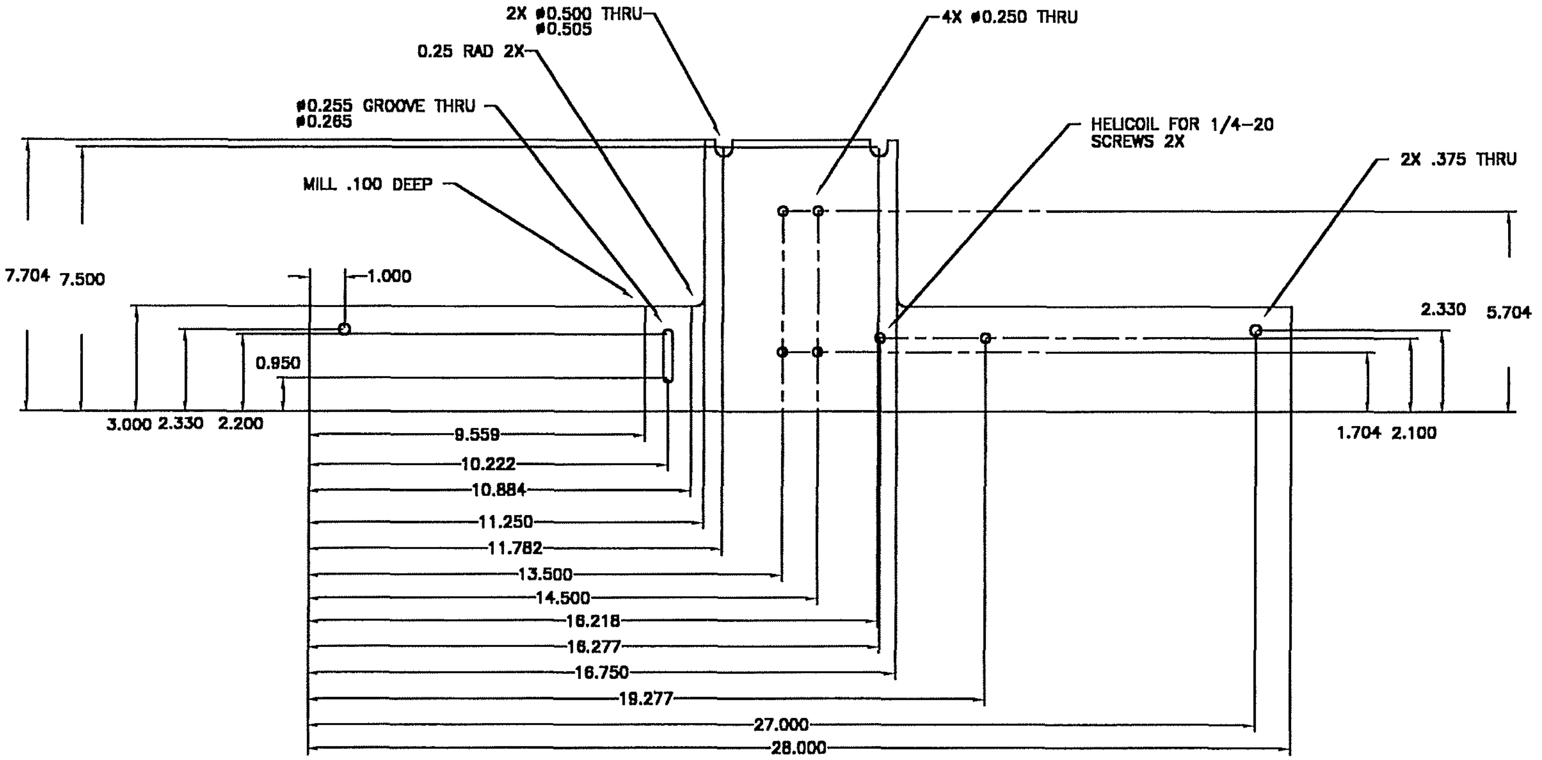
DESIGN KJM DATE 3/9/94
DRAWN KJM
APPRV
RELEASE

UNIVERSITY OF CALIFORNIA, DAVIS
MECHANICAL ENGINEERING DEPARTMENT

TITLE OVERALL VIEW
TMR ASSEMBLY

QTY 1EA
SCALE NA
DRAWING NO. TMR-001
SIZE C
SHT
PROJECT TMR PROJECT

REVISION					
LTR	ZONE	DESCRIPTION	DATE	DRWN	APP
A		INITIAL RELEASE	3/8/94	KJM	DAB



1

NOTES: UNLESS OTHERWISE SPECIFIED

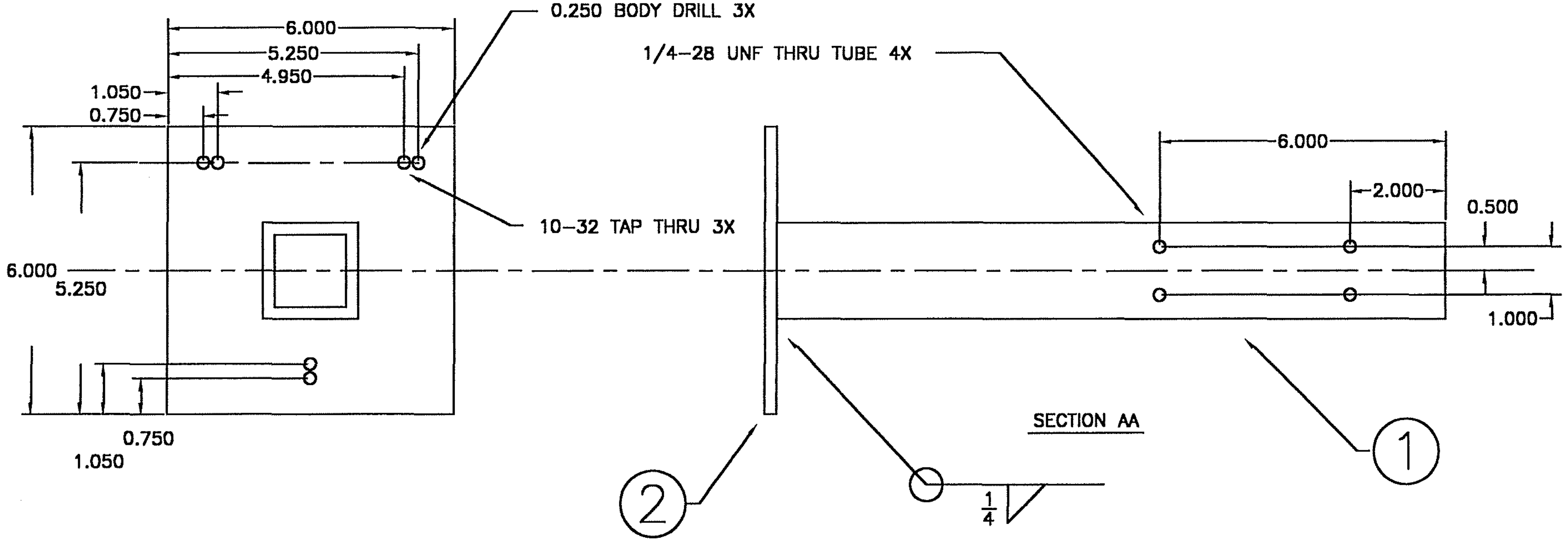
1) BREAK ALL SHARP EDGES

1		1/2" THICK TOOLING PLATE	6061-T6	1
ITEM NO.	PART / ID NO.	NOMENCLATURE / DESCRIPTION	MATERIAL / SPECIFICATION	QTY REQD

PART LIST

UNLESS OTHERWISE NOTED: TOLERANCES .XX ± .01 ANGLES ± 0.5° .XXX ± .005		UNIVERSITY OF CALIFORNIA, DAVIS MECHANICAL ENGINEERING DEPARTMENT	
		TITLE APPLICATOR PLATE TMR ASSEMBLY	
DESIGN KJM	DATE 3/8/94		
DRAWN KJM		QTY 1	DRAWING NO. TMR-002
APPRV DAB		SCALE 1:1	
RELEASE		SIZE B · SHT	PROJECT TMR PROJECT

		REVISION			
LTR	ZONE	DESCRIPTION	DATE	DRWN	APP
A		INITIAL RELEASE	3/6/94	KJM	DAB



△1

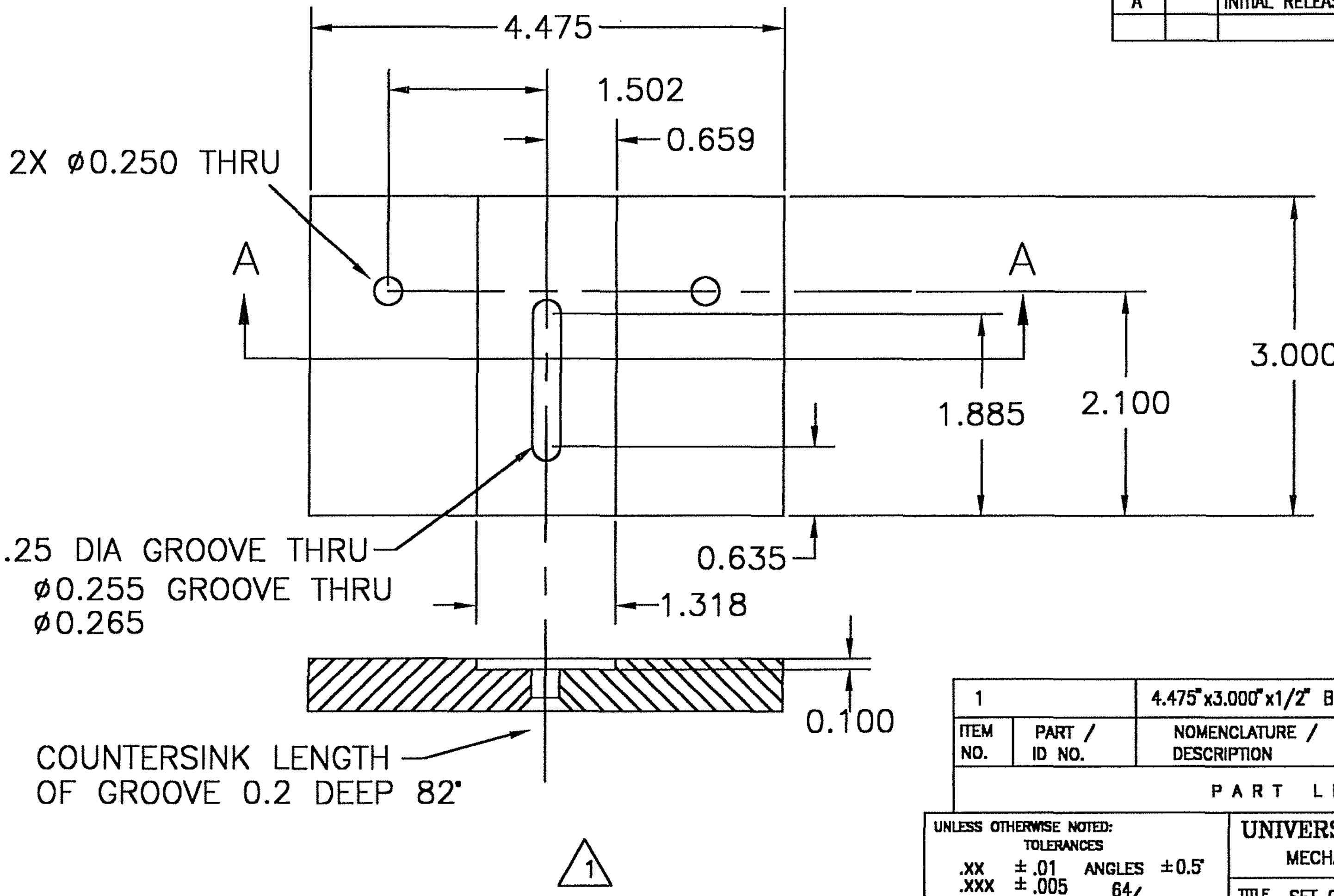
NOTES: UNLESS OTHERWISE SPECIFIED
1) BREAK ALL SHARP EDGES
2) 1 WELDED TO 2 90° +-0.5°

ITEM NO.	PART / ID NO.	NOMENCLATURE / DESCRIPTION	MATERIAL / SPECIFICATION	QTY REQD
2		6"x6"X1/4" THICK PLATE	1018 STEEL	1
1		2"x2"x1/4" TUBING 14" LONG	1018 STEEL	1

PART LIST

UNLESS OTHERWISE NOTED: TOLERANCES .XX ± .01 ANGLES ±0.5° .XXX ± .005 64√		UNIVERSITY OF CALIFORNIA, DAVIS MECHANICAL ENGINEERING DEPARTMENT	
		TITLE LINEAR TRANSDUCER BASE MOUNT TMR ASSEMBLY	
DESIGN KJM	DATE 3/6/94		
DRAWN KJM		QTY 1	DRAWING NO. TMR-003
APPRV DAB		SCALE 1:1	
RELEASE		SIZE B / SHT	PROJECT TMR PROJECT

REVISION					
LTR	ZONE	DESCRIPTION	DATE	DRWN	APP
A		INITIAL RELEASE	3/6/94	KJM	DAB



ITEM NO.	PART / ID NO.	NOMENCLATURE / DESCRIPTION	MATERIAL / SPECIFICATION	QTY REQD
1		4.475"x3.000"x1/2" BLOCK	6061-T6 ALUMINUM	1

PART LIST

UNLESS OTHERWISE NOTED:
TOLERANCES
.XX ± .01 ANGLES ±0.5°
.XXX ± .005
64√

DESIGN KJM	DATE 3/8/94	
DRAWN KJM		
APPRV DAB		
RELEASE		

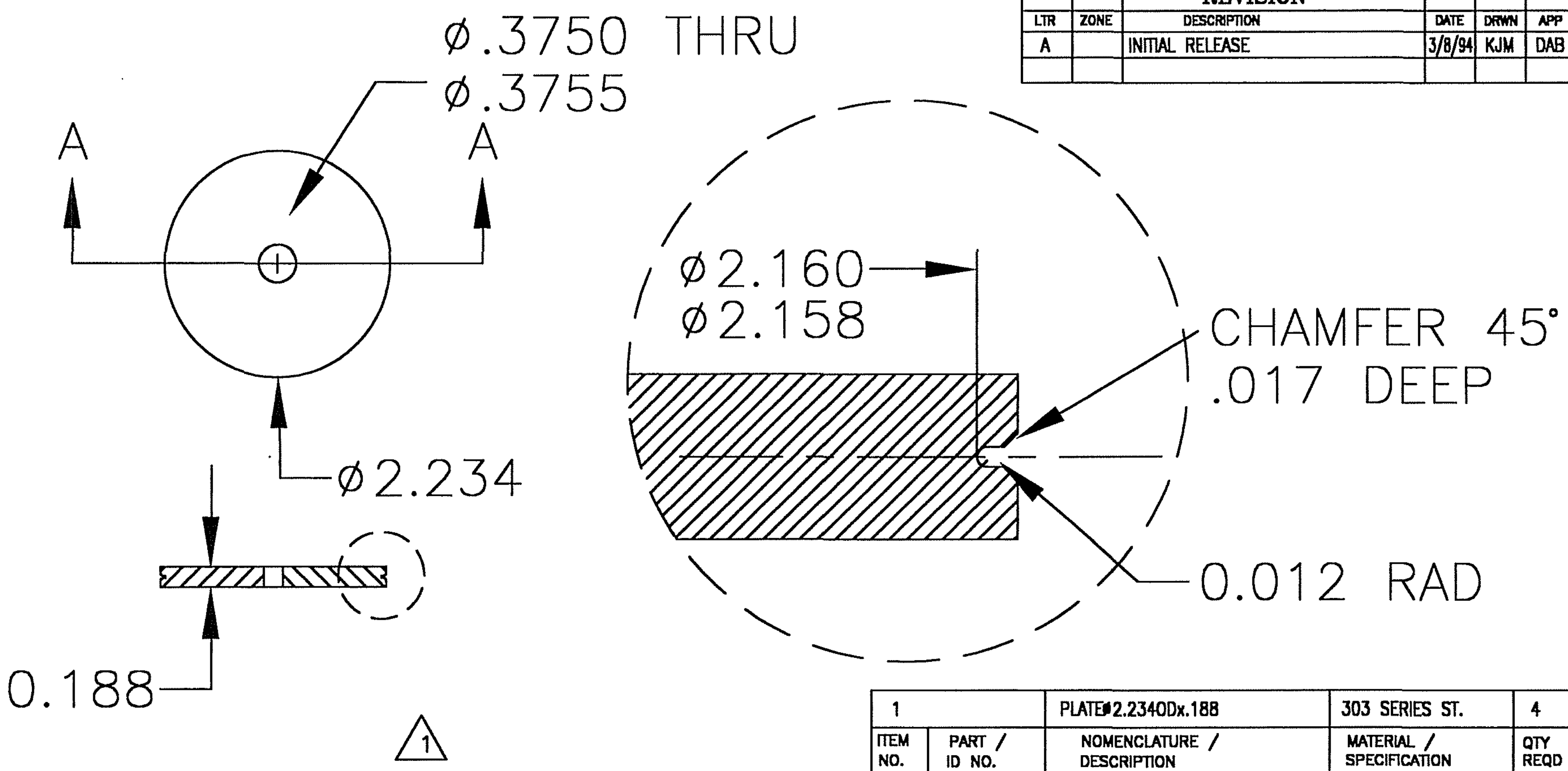
UNIVERSITY OF CALIFORNIA, DAVIS
MECHANICAL ENGINEERING DEPARTMENT

TITLE SET OFF BLOCK
TMR ASSEMBLY

QTY 1	DRAWING NO. TMR-004
SCALE 1:1	
SIZE B	SHT
PROJECT TMR PROJECT	

NOTES: UNLESS OTHERWISE SPECIFIED
1) BREAK ALL SHARP EDGES

REVISION					
LTR	ZONE	DESCRIPTION	DATE	DRWN	APP
A		INITIAL RELEASE	3/8/94	KJM	DAB



NOTES: UNLESS OTHERWISE SPECIFIED
1) BREAK ALL SHARP EDGES

1		PLATEø2.234ODx.188	303 SERIES ST.	4
ITEM NO.	PART / ID NO.	NOMENCLATURE / DESCRIPTION	MATERIAL / SPECIFICATION	QTY REQD

PART LIST

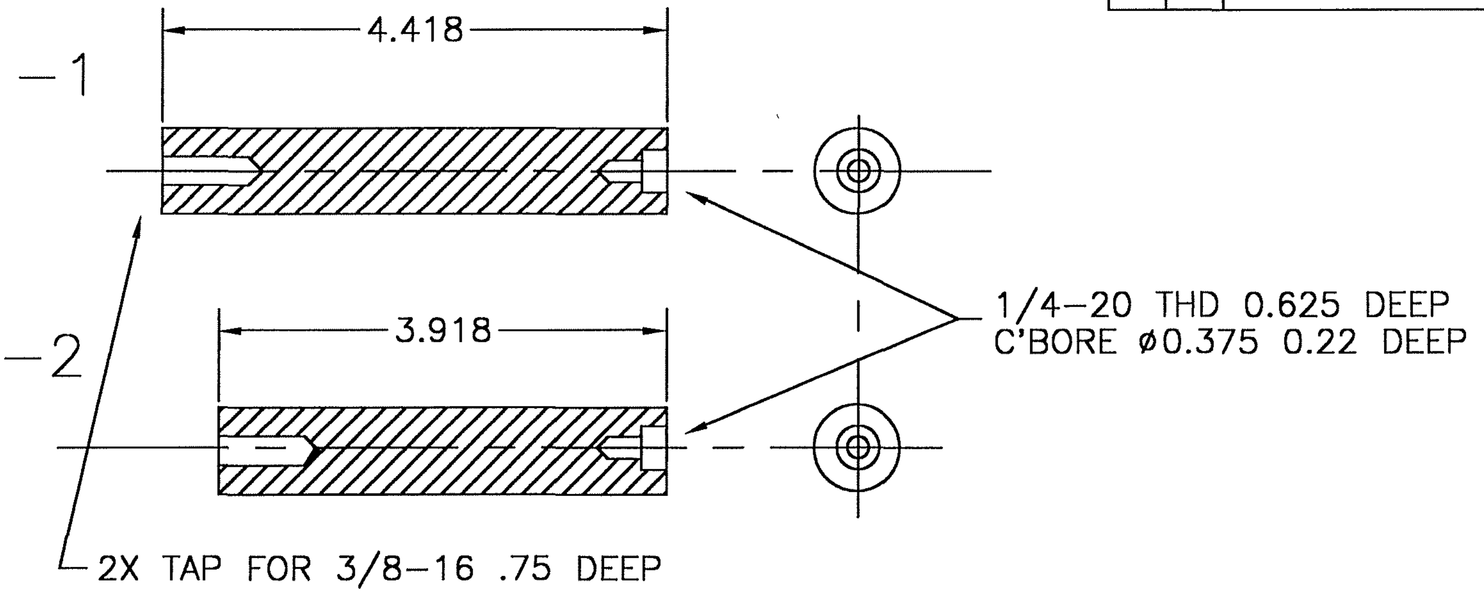
UNLESS OTHERWISE NOTED:
TOLERANCES
.XX ± .01 ANGLES ±0.5°
.XXX ± .001
64

UNIVERSITY OF CALIFORNIA, DAVIS
MECHANICAL ENGINEERING DEPARTMENT

TITLE PULLEY
TMR ASSEMBLY

DESIGN KJM	DATE 3/8/94			
DRAWN KJM		QTY 1	DRAWING NO. TMR-005	
APPRV DAB		SCALE 1:1		
RELEASE		SIZE B	SHT	PROJECT TMR PROJECT

REVISION					
LTR	ZONE	DESCRIPTION	DATE	DRWN	APP
A		INITIAL RELEASE	2/5/94	KJM	DAB



△1 △2

NOTES: UNLESS OTHERWISE SPECIFIED
1) BREAK ALL SHARP EDGES

ITEM NO.	PART / ID NO.	NOMENCLATURE / DESCRIPTION	MATERIAL / SPECIFICATION	QTY REQD
−2		Ø0.750x3.912 ROD	303 SERIES ST.	1
−1		Ø0.750x4.412 ROD	303 SERIES ST.	1

PART LIST

UNLESS OTHERWISE NOTED:
TOLERANCES
.XX ±.01 ANGLES ±0.5°
.XXX ±.003
64√

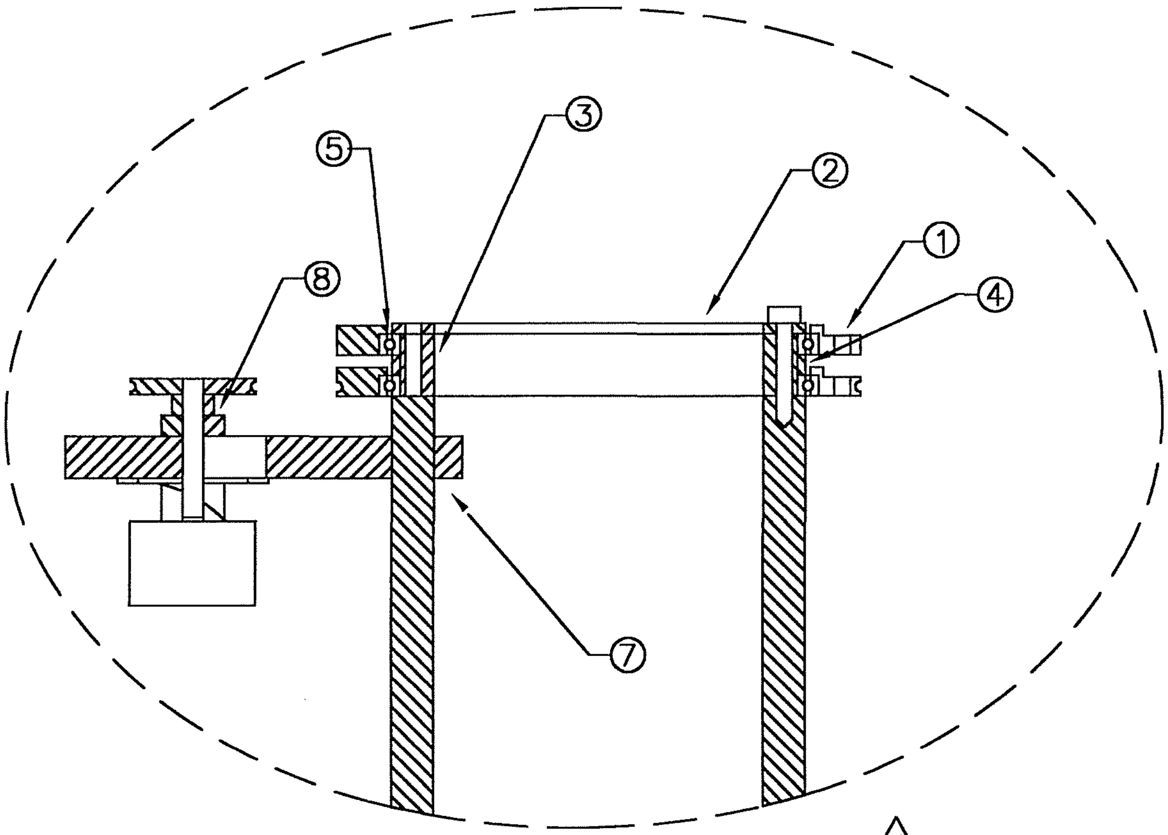
DESIGN KJM
DATE 3/11/94
DRAWN KJM
APPRV DAB
RELEASE

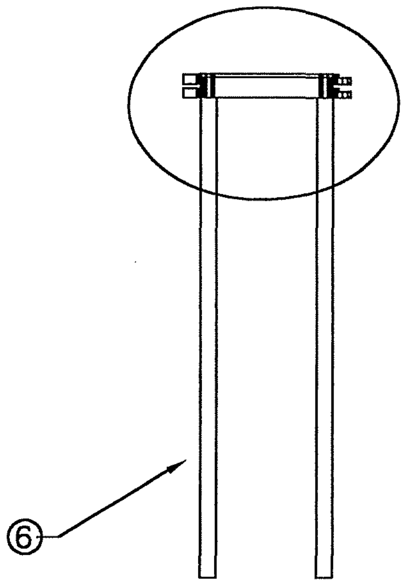
UNIVERSITY OF CALIFORNIA, DAVIS
MECHANICAL ENGINEERING DEPARTMENT

TITLE PULLEY STANDOFFS
TMR ASSEMBLY

QTY 1
DRAWING NO. TMR−006
SCALE 1:1
SIZE B
SHT
PROJECT TMR PROJECT

REVISION					
LTR	ZONE	DESCRIPTION	DATE	DRWN	APP
A		INITIAL RELEASE	8/2/92	KJM	DAB





△1

NOTES: UNLESS OTHERWISE SPECIFIED
1) DRAWING FOR REFERENCE ONLY

ITEM NO.	PART / ID NO.	NOMENCLATURE / DESCRIPTION	MATERIAL / SPECIFICATION	QTY REQD
8	TMR-010	ENCODER SLIDE	OIL IMPREG BRONZE	2
7	TMR-010	ENCODER PLATFORM	6061-T6	1
6	TMR-011	1/2" ROD 17" LONG	1018 STEEL	4
5		KAYDON BEARINGS		2
4	TMR-008	⌀4.936X0.250	6061-T6	1
3	TMR-008	⌀4.750X0.750	6061-T6	1
2	TMR-008	⌀4.936X0.125	6061-T6	1
1	TMR-008	⌀0.25X.35	6061-T6	2

PART LIST

UNLESS OTHERWISE NOTED:
TOLERANCES
.XX ±.01 ANGLES ±0.5°
.XXX ±.005
64

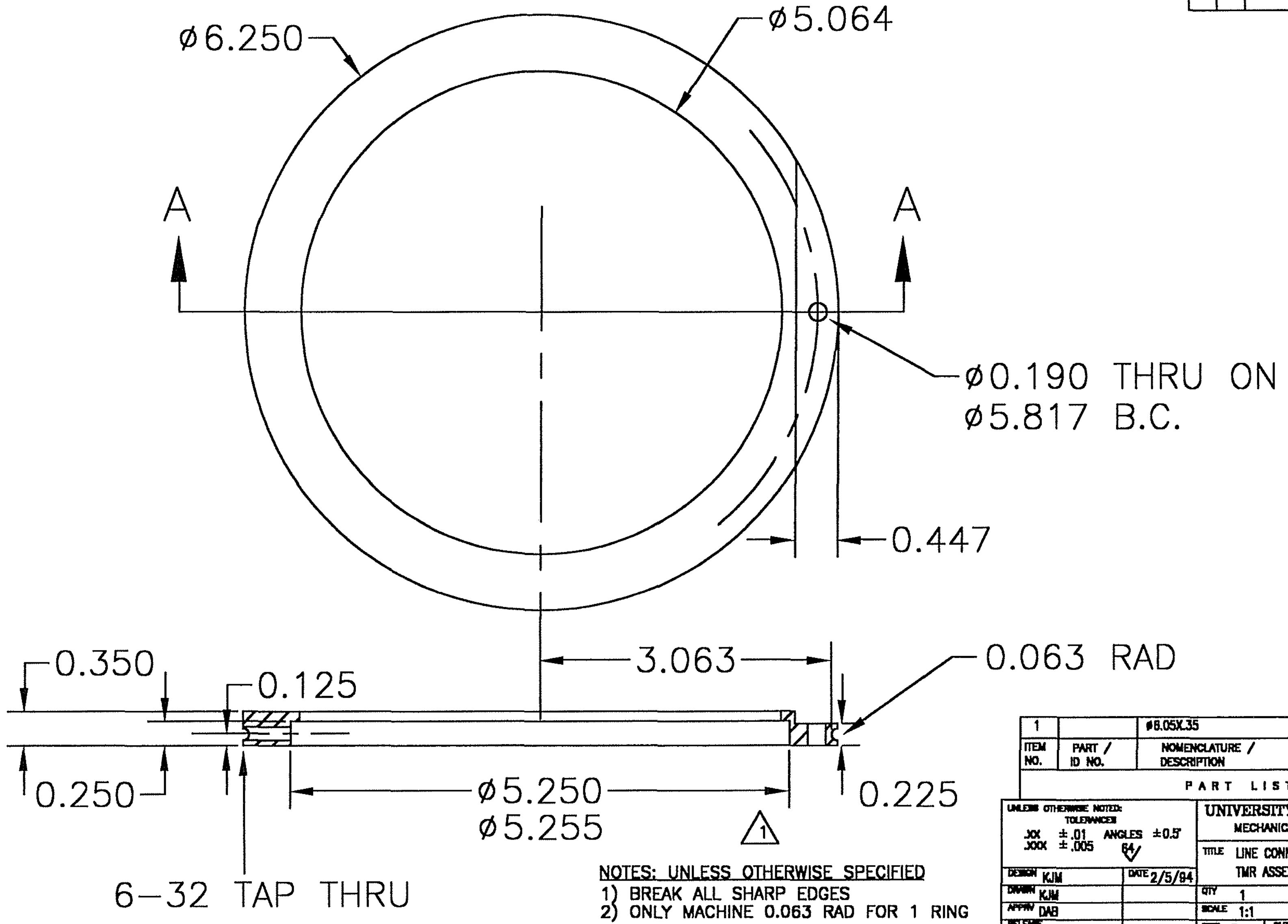
DESIGN KJM DATE 3/8/94
DRAWN KJM
APPRV DAB
RELEASE

UNIVERSITY OF CALIFORNIA, DAVIS
MECHANICAL ENGINEERING DEPARTMENT

TITLE ROTATION ASSEMBLY
TMR ASSEMBLY

QTY 1
SCALE 1:1
DRAWING NO. TMR-007
SIZE B
SHT
PROJECT TMR PROJECT

REVISION					
LTR	ZONE	DESCRIPTION	DATE	DRWN	APP
A		INITIAL RELEASE	2/5/94	KJM	DAB



NOTES: UNLESS OTHERWISE SPECIFIED
1) BREAK ALL SHARP EDGES
2) ONLY MACHINE 0.063 RAD FOR 1 RING

1		ø6.05X.35	6061-T6	2
ITEM NO.	PART / ID NO.	NOMENCLATURE / DESCRIPTION	MATERIAL / SPECIFICATION	QTY REQD

PART LIST

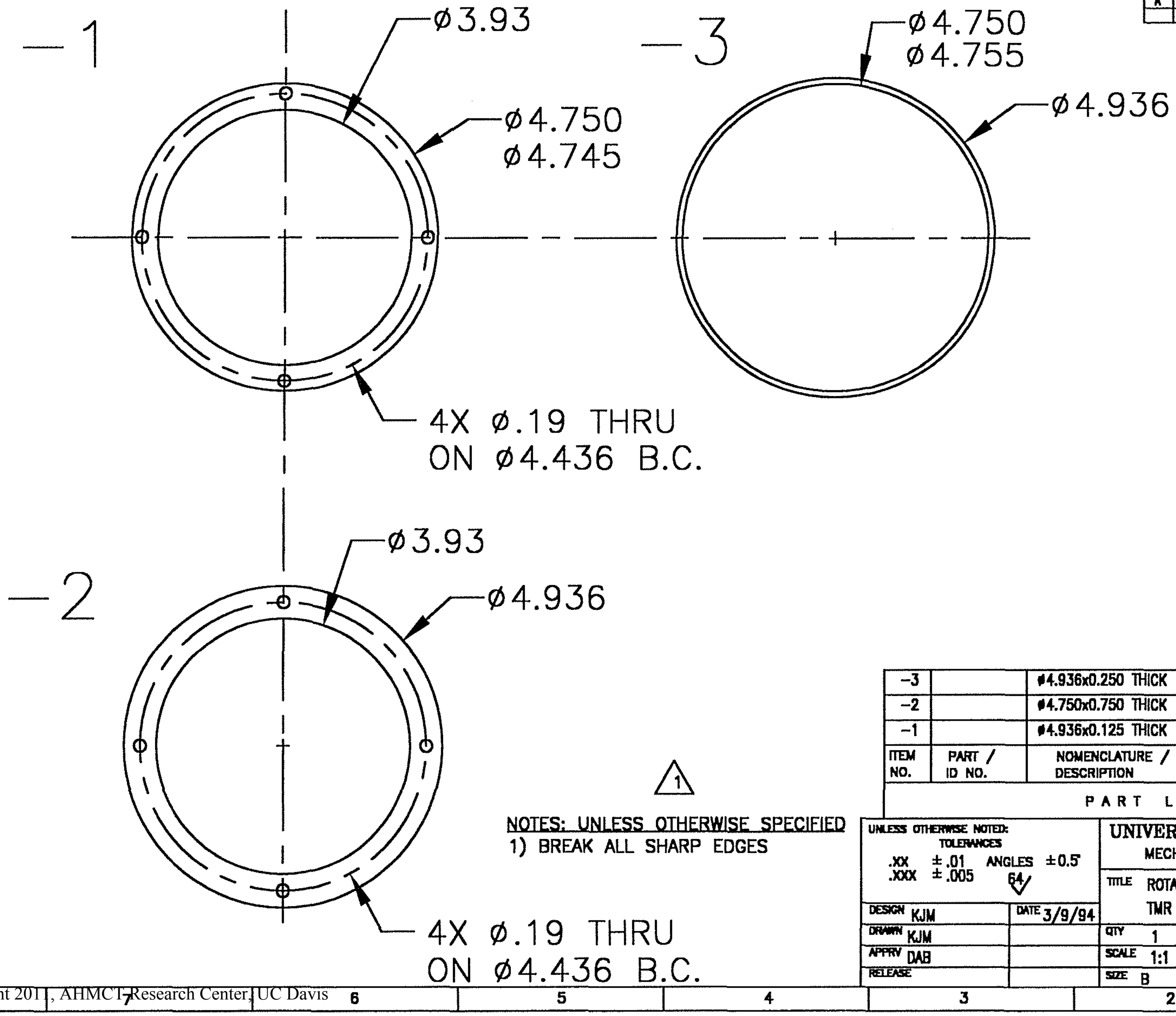
UNLESS OTHERWISE NOTED:
TOLERANCES
.XX ±.01 ANGLES ±0.5°
.XXX ±.005
64√

UNIVERSITY OF CALIFORNIA, DAVIS
MECHANICAL ENGINEERING DEPARTMENT

TITLE LINE CONNECTION
TMR ASSEMBLY

DESIGN KJM
DATE 2/5/94
DRAWN KJM
APPRV DAB
RELEASE
QTY 1
SCALE 1:1
DRAWING NO. TMR-008
SIZE B
SHT
PROJECT TMR PROJECT

REVISION					
LTR	ZONE	DESCRIPTION	DATE	DRWN	APP
A		INITIAL RELEASE	3/9/94	KJM	DAB



NOTES: UNLESS OTHERWISE SPECIFIED
1) BREAK ALL SHARP EDGES

−3		ø4.936x0.250 THICK	6061−T6	1
−2		ø4.750x0.750 THICK	6061−T6	1
−1		ø4.936x0.125 THICK	6061−T6	1
ITEM NO.	PART / ID NO.	NOMENCLATURE / DESCRIPTION	MATERIAL / SPECIFICATION	QTY REQD

PART LIST

UNLESS OTHERWISE NOTED:
TOLERANCES
.XX ± .01 ANGLES ±0.5°
.XXX ± .005 64√

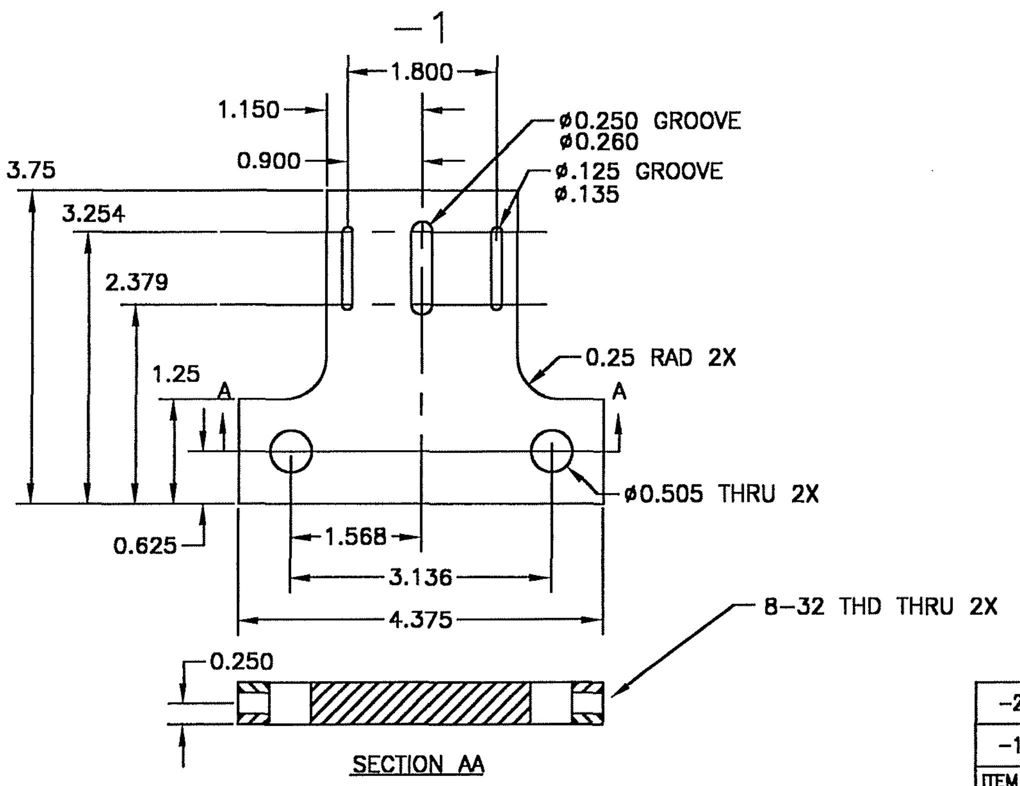
DESIGN KJM DATE 3/9/94
DRAWN KJM
APPRV DAB
RELEASE

UNIVERSITY OF CALIFORNIA, DAVIS
MECHANICAL ENGINEERING DEPARTMENT

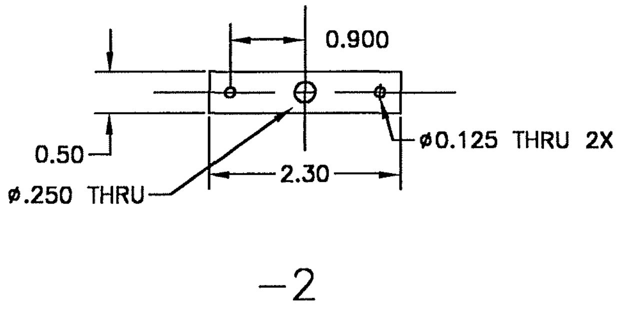
TITLE ROTARY COMPONENTS
TMR ASSEMBLY

QTY 1
SCALE 1:1
DRAWING NO. TMR−009
SIZE B
SHT
PROJECT TMR ASSEMBLY

		REVISION			
LTR	ZONE	DESCRIPTION	DATE	DRWN	APP
A		INITIAL RELEASE	3/14/94	KJM	DAB



SECTION AA



△1 △2

NOTES: UNLESS OTHERWISE SPECIFIED

1) BREAK ALL SHARP EDGES

−2		0.50x2.30x.125 THICK	OIL IMPREG. BRONZE	2
−1		3.75x4.375x0.500	6061−T6 ALUMINUM	1
ITEM NO.	PART / ID NO.	NOMENCLATURE / DESCRIPTION	MATERIAL / SPECIFICATION	QTY REQD

PART LIST

UNLESS OTHERWISE NOTED:
TOLERANCES
.XX ± .01 ANGLES ±0.5°
.XXX ± .005
64√

DESIGN KJM	DATE 3/14/94		
DRAWN KJM			
APPRV DAB			
RELEASE			

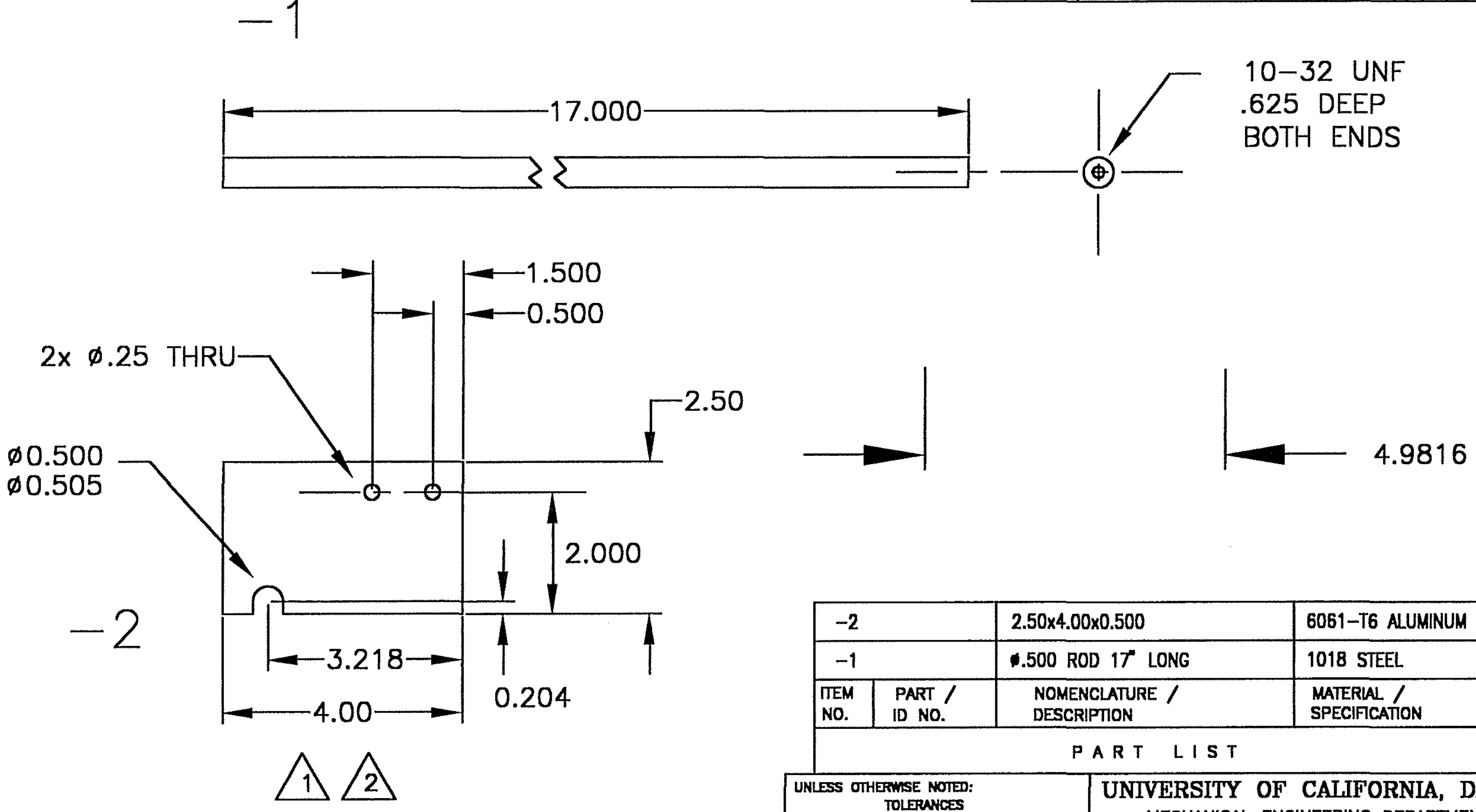
UNIVERSITY OF CALIFORNIA, DAVIS
MECHANICAL ENGINEERING DEPARTMENT

TITLE ENCODER PLATFORM
TMR ASSEMBLY

QTY 1	DRAWING NO. TMR−010	
SCALE 1:1		
SIZE B	SHT	PROJECT TMR PROJECT

REVISION

LTR	ZONE	DESCRIPTION	DATE	DRWN	APP
A		INITIAL RELEASE	3/11/94	KJM	DAB
B		DIMENSIOS FOR HOLES ADDED	4/7/94	KJM	DAB



NOTES: UNLESS OTHERWISE SPECIFIED

1) BREAK ALL SHARP EDGES

ITEM NO.	PART / ID NO.	NOMENCLATURE / DESCRIPTION	MATERIAL / SPECIFICATION	QTY REQD
−2		2.50x4.00x0.500	6061−T6 ALUMINUM	1
−1		ø.500 ROD 17" LONG	1018 STEEL	4

PART LIST

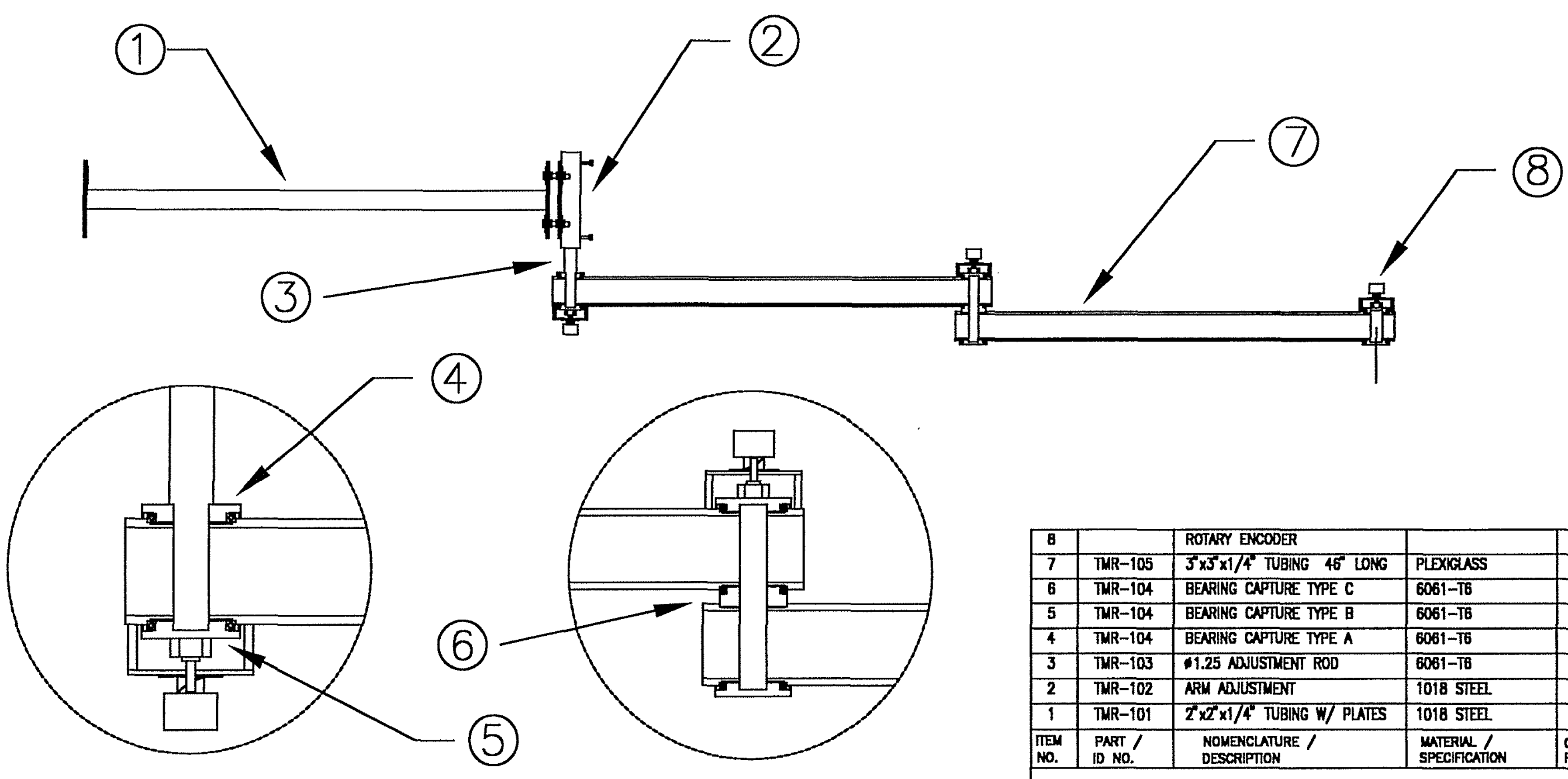
UNLESS OTHERWISE NOTED:
TOLERANCES
.XX ±.01 ANGLES ±0.5°
.XXX ±.005
64√

UNIVERSITY OF CALIFORNIA, DAVIS
MECHANICAL ENGINEERING DEPARTMENT

TITLE EXTRA COMPONENTS
TMR ASSEMBLY

DESIGN KJM	DATE 3/11/94	QTY 1		DRAWING NO. TMR−011	
DRAWN KJM		SCALE 1:1			
APPRV DAB					
RELEASE		SIZE B	SHT	PROJECT TMR PROJECT	

REVISION					
LTR	ZONE	DESCRIPTION	DATE	DWN	APP
A		INITIAL RELEASE	3/11/94	KJM	DAB



ITEM NO.	PART / ID NO.	NOMENCLATURE / DESCRIPTION	MATERIAL / SPECIFICATION	QTY REQD
8		ROTARY ENCODER		3
7	TMR-105	3"x3"x1/4" TUBING 46" LONG	PLEXIGLASS	2
6	TMR-104	BEARING CAPTURE TYPE C	6061-T6	1
5	TMR-104	BEARING CAPTURE TYPE B	6061-T6	5
4	TMR-104	BEARING CAPTURE TYPE A	6061-T6	1
3	TMR-103	ø1.25 ADJUSTMENT ROD	6061-T6	1
2	TMR-102	ARM ADJUSTMENT	1018 STEEL	1
1	TMR-101	2"x2"x1/4" TUBING W/ PLATES	1018 STEEL	1

PART LIST

NOTES:
1) DRAWING FOR REFERENCE ONLY
2) HALF VIEW OF ENTIRE ARM

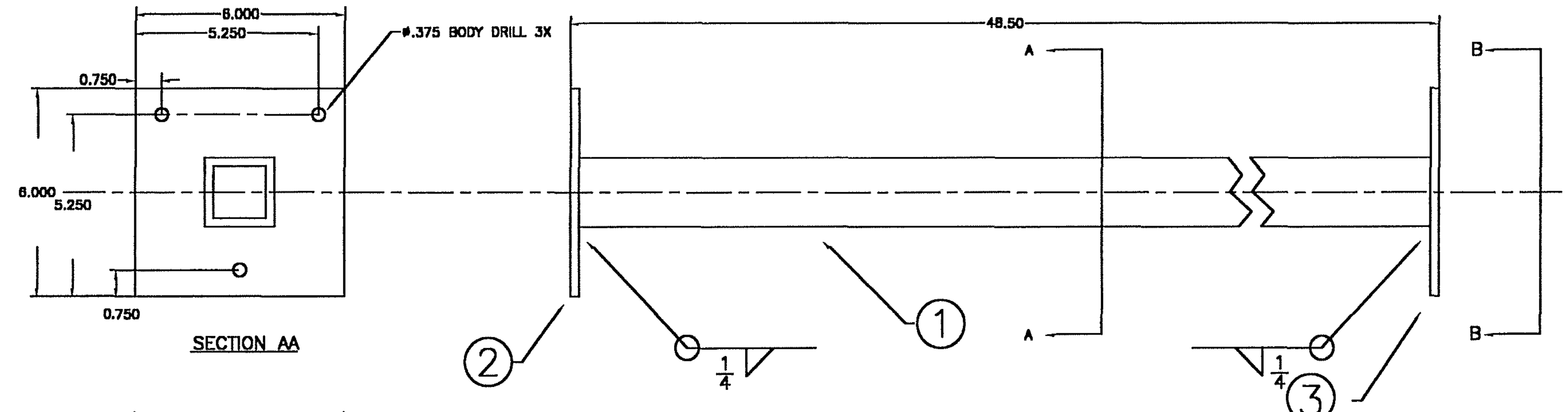
UNLESS OTHERWISE NOTED:
TOLERANCES
.XX ± .01 ANGLES ±0.5°
.XXX ± .005
64√

UNIVERSITY OF CALIFORNIA, DAVIS
MECHANICAL ENGINEERING DEPARTMENT

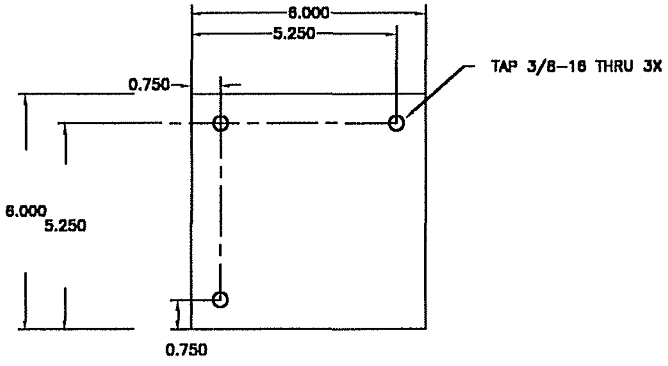
TITLE ARM ASSEMBLY
TMR ARM ASSEMBLY

DESIGN KJM
DATE 3/15/94
DRAWN KJM
APPRV DAB
RELEASE
QTY 1
SCALE 1:1
DRAWING NO. TMR-100
SIZE B
SHT
PROJECT TMR PROJECT

REVISION					
LTR	ZONE	DESCRIPTION	DATE	DRWN	APP
A		INITIAL RELEASE	3/1/94	KJM	DAB



SECTION AA



SECTION BB

△1

NOTES: UNLESS OTHERWISE SPECIFIED
1) BREAK ALL SHARP EDGES
2) 1 WELDED TO 2 90° +-0.5°
3) 1 WELDED TO 3 90° +-0.5°

ITEM NO.	PART / ID NO.	NOMENCLATURE / DESCRIPTION	MATERIAL / SPECIFICATION	QTY REQD
3		6"x6"X1/4" THICK PLATE	1018 STEEL	1
2		6"x6"X1/4" THICK PLATE	1018 STEEL	1
1		2"x2"x1/4" TUBING 14" LONG	1018 STEEL	1

PART LIST

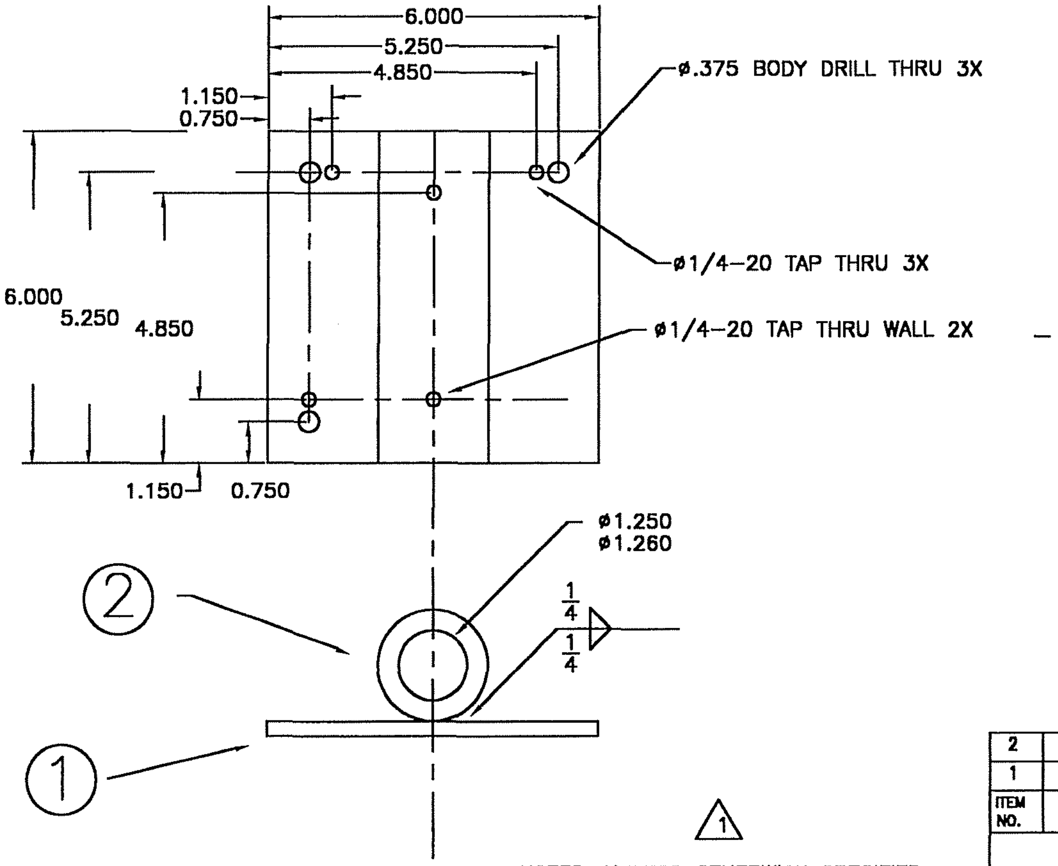
UNLESS OTHERWISE NOTED:
TOLERANCES
.XX ± .01 ANGLES ±0.5°
.XXX ± .005
64√

UNIVERSITY OF CALIFORNIA, DAVIS
MECHANICAL ENGINEERING DEPARTMENT

TITLE ARM BASE MOUNT
TMR ASSEMBLY

DESIGN KJM
DATE 3/14/94
DRAWN KJM
APPRV DAB
RELEASE
QTY 1
SCALE 1:1
DRAWING NO. TMR-101
SIZE B
SHT
PROJECT TMR PROJECT

REVISION					
LTR	ZONE	DESCRIPTION	DATE	DRWN	APP
A		INITIAL RELEASE	3/8/94	KJM	DAB



NOTES: UNLESS OTHERWISE SPECIFIED
1) BREAK ALL SHARP EDGES
2) 1 WELDED TO 2 90° −+0.5°

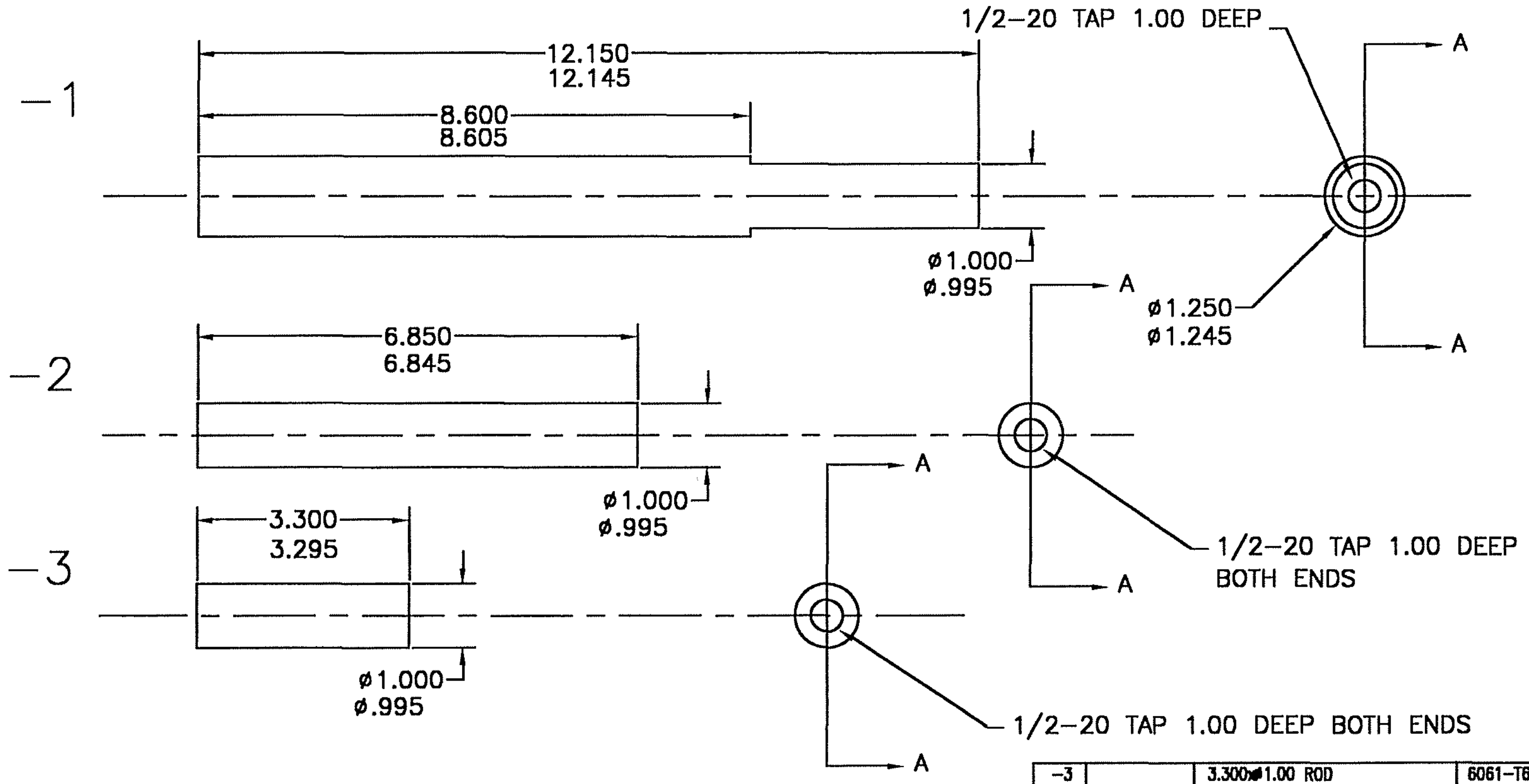
2		6"x ø2" ø1.25 HOLE ROD	1018 STEEL	1
1		6"x6"x1/4" THICK PLATE	1018 STEEL	1
ITEM NO.	PART / ID NO.	NOMENCLATURE / DESCRIPTION	MATERIAL / SPECIFICATION	QTY REQD

PART LIST

UNLESS OTHERWISE NOTED:
TOLERANCES
.XX ±.01 ANGLES ±0.5°
.XXX ±.005
64√

DESIGN KJM	DATE 3/10/94	UNIVERSITY OF CALIFORNIA, DAVIS MECHANICAL ENGINEERING DEPARTMENT	
		TITLE VARI PLATE TMR ARM ASSEMBLY	
DRAWN KJM		QTY 1	DRAWING NO. TMR–102
APPRV DAB		SCALE 1:1	
RELEASE		SIZE B	SHT	PROJECT TMR PROJECT

		REVISION			
LTR	ZONE	DESCRIPTION	DATE	DRWN	APP
A		INITIAL RELEASE	[illegible]	KJM	DAB



1

NOTES: UNLESS OTHERWISE SPECIFIED

1) BREAK ALL SHARP EDGES

–3		3.300xø1.00 ROD	6061–T6	1
–2		7.100xø1.00 ROD	6061–T6	1
–1		12.150xø1.25 ROD	6061–T6	1
ITEM NO.	PART / ID NO.	NOMENCLATURE / DESCRIPTION	MATERIAL / SPECIFICATION	QTY REQD

PART LIST

UNLESS OTHERWISE NOTED:
TOLERANCES
.XX ±.01 ANGLES ±0.5°
.XXX ±.005
64✓

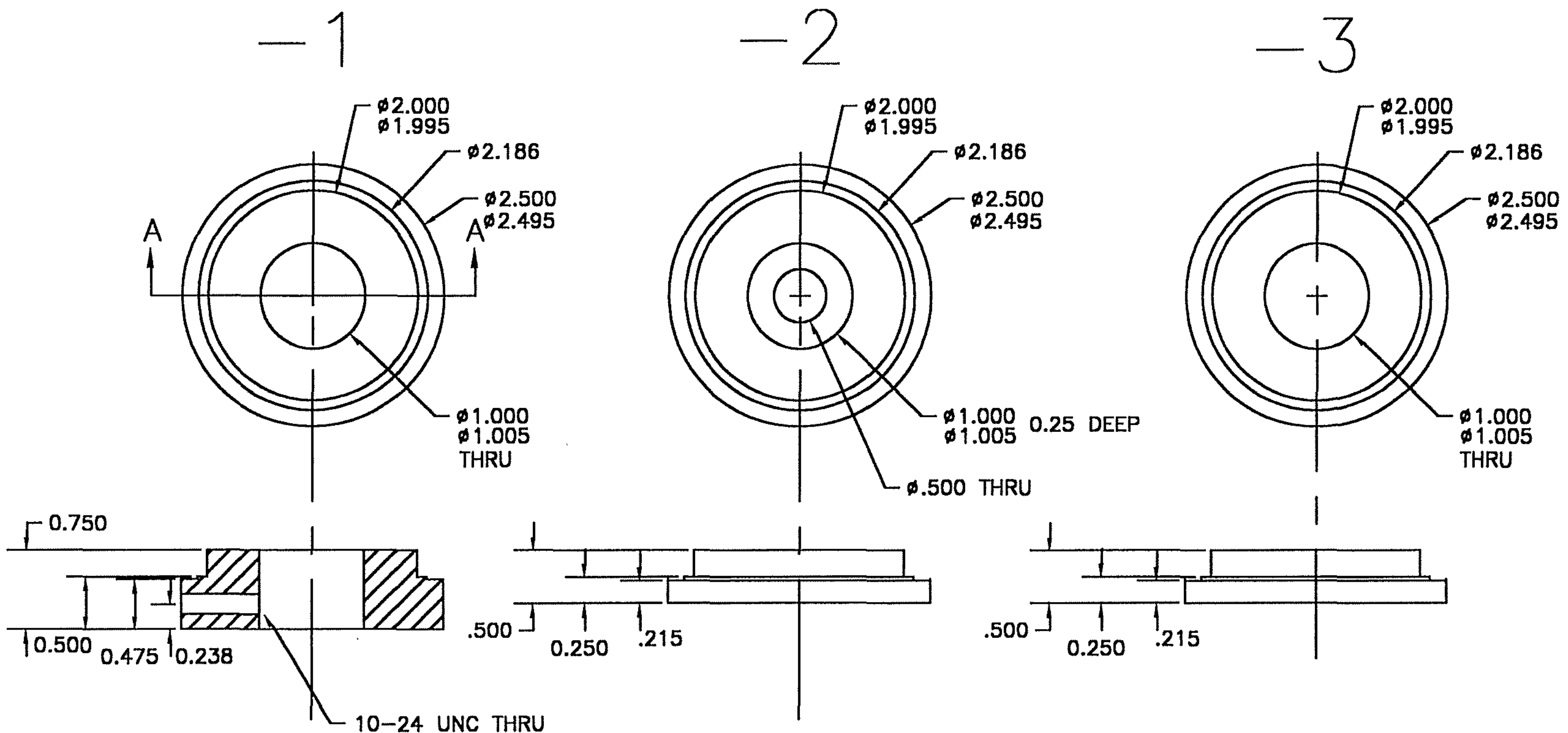
DESIGN KJM
DATE 3/15/94
DRAWN KJM
APPRV DAB
RELEASE

UNIVERSITY OF CALIFORNIA, DAVIS
MECHANICAL ENGINEERING DEPARTMENT

TITLE HEIGHT ADJUSTING ROD/JOINT RODS
TMR ARM ASSEMBLY

QTY 1
SCALE 1:1
DRAWING NO. TMR–103
SIZE B
SHT
PROJECT TMR PROJECT

REVISION					
LTR	ZONE	DESCRIPTION	DATE	DRWN	APP
A		INITIAL RELEASE	[illegible]	KJM	DAB



−3		⌀2.500x0.500	6061−T6	1
−2		⌀2.500x0.500	6061−T6	5
−1		⌀2.500x⌀1.000	6061−T6	1
ITEM NO.	PART / ID NO.	NOMENCLATURE / DESCRIPTION	MATERIAL / SPECIFICATION	QTY REQD

PART LIST

△1

NOTES: UNLESS OTHERWISE SPECIFIED
1) BREAK ALL SHARP EDGES

UNLESS OTHERWISE NOTED:
TOLERANCES
.XX ±.01 ANGLES ±0.5°
.XXX ±.005
64√

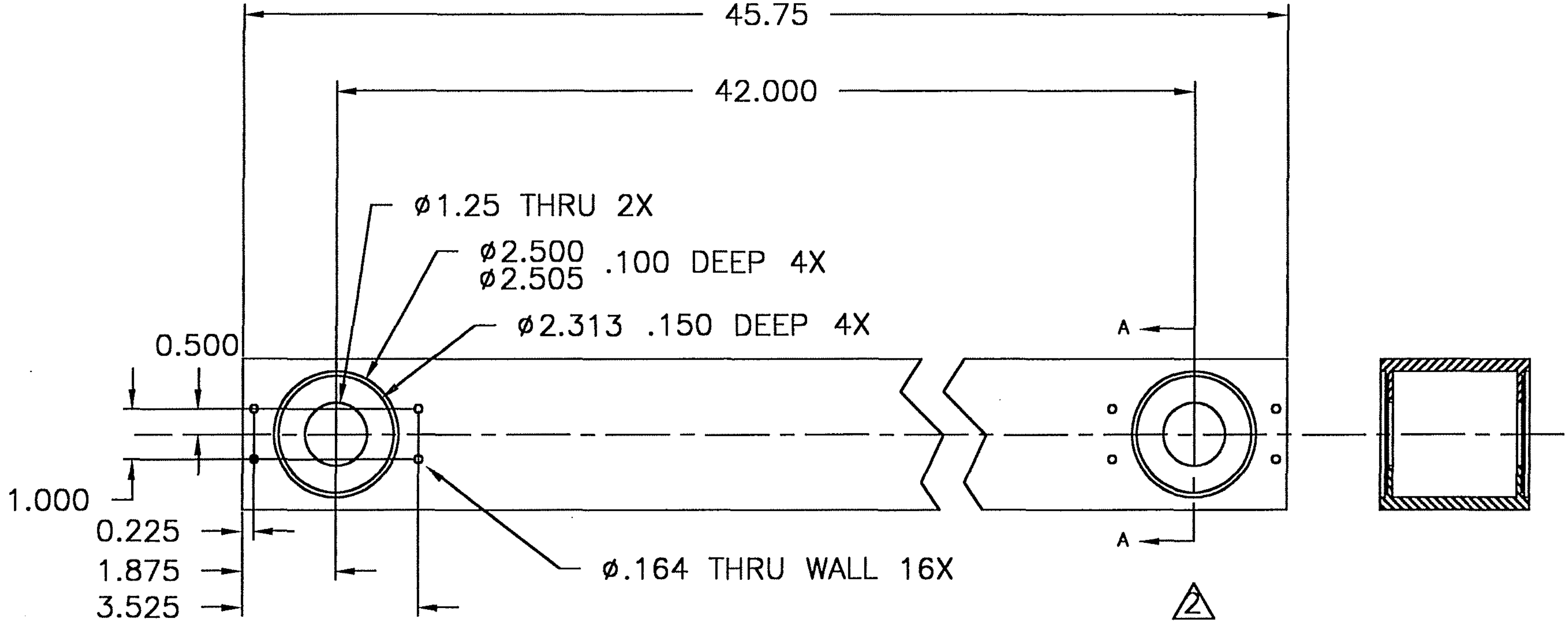
DESIGN KJM
DATE 3/15/94
DRAWN KJM
APPRV DAB
RELEASE

UNIVERSITY OF CALIFORNIA, DAVIS
MECHANICAL ENGINEERING DEPARTMENT

TITLE BEARING CAPTURE VERSIONS
TMR ARM ASSEMBLY

QTY 1
SCALE 1:1
DRAWING NO. TMR-104
SIZE B
SHT
PROJECT TMR PROJECT

REVISION					
LTR	ZONE	DESCRIPTION	DATE	DRWN	APP
A		INITIAL RELEASE	3/16/94	KJM	DAB



1

NOTES: UNLESS OTHERWISE SPECIFIED

1) BREAK ALL SHARP EDGES

2) NOTE THAT HOLES AND C'BORES ARE REPEATED ON OPPOSITE SIDES. C'BORES TO BE PARALLEL, HOLES TO BE COINCENTRIC.

1		3"x3"x1/4" SQ. TUBING 45.75"	PLEXIGLASS	2
ITEM NO.	PART / ID NO.	NOMENCLATURE / DESCRIPTION	MATERIAL / SPECIFICATION	QTY REQD

PART LIST

UNLESS OTHERWISE NOTED:
TOLERANCES
.XX ± .01 ANGLES ±0.5°
.XXX ± .005 64

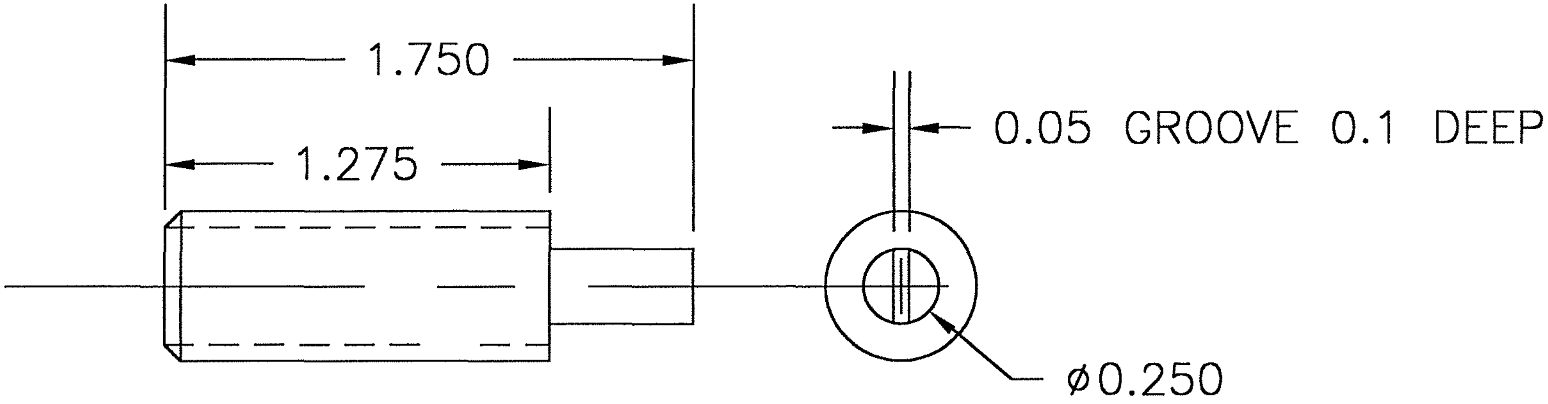
DESIGN KJM DATE 3/16/94
DRAWN KJM
APPRV DAB
RELEASE

UNIVERSITY OF CALIFORNIA, DAVIS
MECHANICAL ENGINEERING DEPARTMENT

TITLE ARM TUBES
TMR ASSEMBLY

QTY 1
SCALE 1:1
DRAWING NO. TMR-105
SIZE B
SHT
PROJECT TMR PROJECT

REVISION					
LTR	ZONE	DESCRIPTION	DATE	DRWN	APP
A		INITIAL RELEASE	3/20/94	KJM	DAB



ITEM NO.	PART / ID NO.	NOMENCLATURE / DESCRIPTION	MATERIAL / SPECIFICATION	QTY REQD
1		1/2-20 THREAD STOCK 1.75"	STEEL	8

PART LIST

NOTES: UNLESS OTHERWISE SPECIFIED
1) BREAK ALL SHARP EDGES

UNLESS OTHERWISE NOTED: TOLERANCES .XX ± .01 ANGLES ±0.5° .XXX ± .005 64		UNIVERSITY OF CALIFORNIA, DAVIS MECHANICAL ENGINEERING DEPARTMENT	
		TITLE OUTSIDE PIPE LINK SCREW	
DESIGN KJM	DATE 3/20/94		
DRAWN KJM		QTY 1	DRAWING NO. TMR-106
APPRV		SCALE 1:1	
RELEASE DAB		SIZE B SHT	PROJECT TMR PROJECT

APPENDIX B

CRITICAL COMMERCIALLY PURCHASED COMPONENTS

Manufacturer	**Component**	**Model**
Reliance Motion Control 6950 Washington Avenue South Eden Prairie, Minnesota 55344 Tel. (612) 942-3600 Fax (612) 942-3636	Brushless DC Motor Stall torque: 20 lb-in Max continuous speed: 5000 rpm Torque const.: 2.5 lb-in/A Voltage const.: 34 V/kRPM	Electro-Craft S-3016-N-H00AA
BAYSIDE CONTROL INC. 27 Seaview Blvd. Port Washington, NY 11050 Tel 516-484-5353 Fax 516-484-5496	Gear Box Rated torque: 800 lb-in Peak torque: 1328 lb-in Rated input speed: 4000 rpm Std backlash: 10 minutes Minimum efficiency: 85% Radial load: 600 lbs Axial load: 600 lbs	RA90-010
Encoder Products Co. Sandpoint, IDAHO Tel 208-263-8541	Encoders for Linkage 2500 pulse per revolution Differential line drive A, B quadrature output and Z index pulses	ACCU-CODER 755A
Celesco 7800 Deering Ave. P.O Box 7964 Canoga Park, CA 91309-7964 Tel 800-423-5483 Fax 818-340-1175	Linear Transducer Encoder based unit; two channel square wave quadrature output Cable tension: 25 oz. typical Max vibration: 5 G's RMS	PT9150-0150-111-1110
APPRO International Inc. 3687 Enochs St. Santa Clara, CA 95051 Tel 408-732-6091 Fax 408-732-6096	80486 Computer 486DX-33MHz motherboard w/64K cache 4Meg. RAM 213MB IDE HDD FDD and HDD controllers 2 serial & 1 parallel ports 1 MB VGA card 14" VGA color monitor Rackmount keyboard and case w/250 watt	486DX-33MHz Rackmount System
KEITHLEY METRABYTE 440 Myles Standish Blvd. Taunton, MA 02780 Tel 508-880-3000 Fax 508-880-0179	Encoder Interface Board Accepts inputs from incremental or quadrature encoders Digitally filtered inputs 333 kHz max quadrature input pulse rate 24-bit pre-settable counters 3-axis version	M5312
Modular Vision System Inc. 3195 De Miniac Montreal, Quebec, H4S 1S9 Tel 514-333-0140 Fax 333-8636	Laser Range Finding Sensor Resolution along scan: 0.0625in Vertical resolution: 0.0625 in Accuracy of crack position: 0.125 in Field of view: 12 in System response freq.: 18 Hz	Laser Vision System

APPENDIX C

SERVO MOTOR CONTROLLER DIAGRAMS

Servo Motor Controller Circuit Diagram 1/5 69

Servo Motor Controller Circuit Diagram 2/5 70

Servo Motor Controller Circuit Diagram 3/5 71

Servo Motor Controller Circuit Diagram 4/5 72

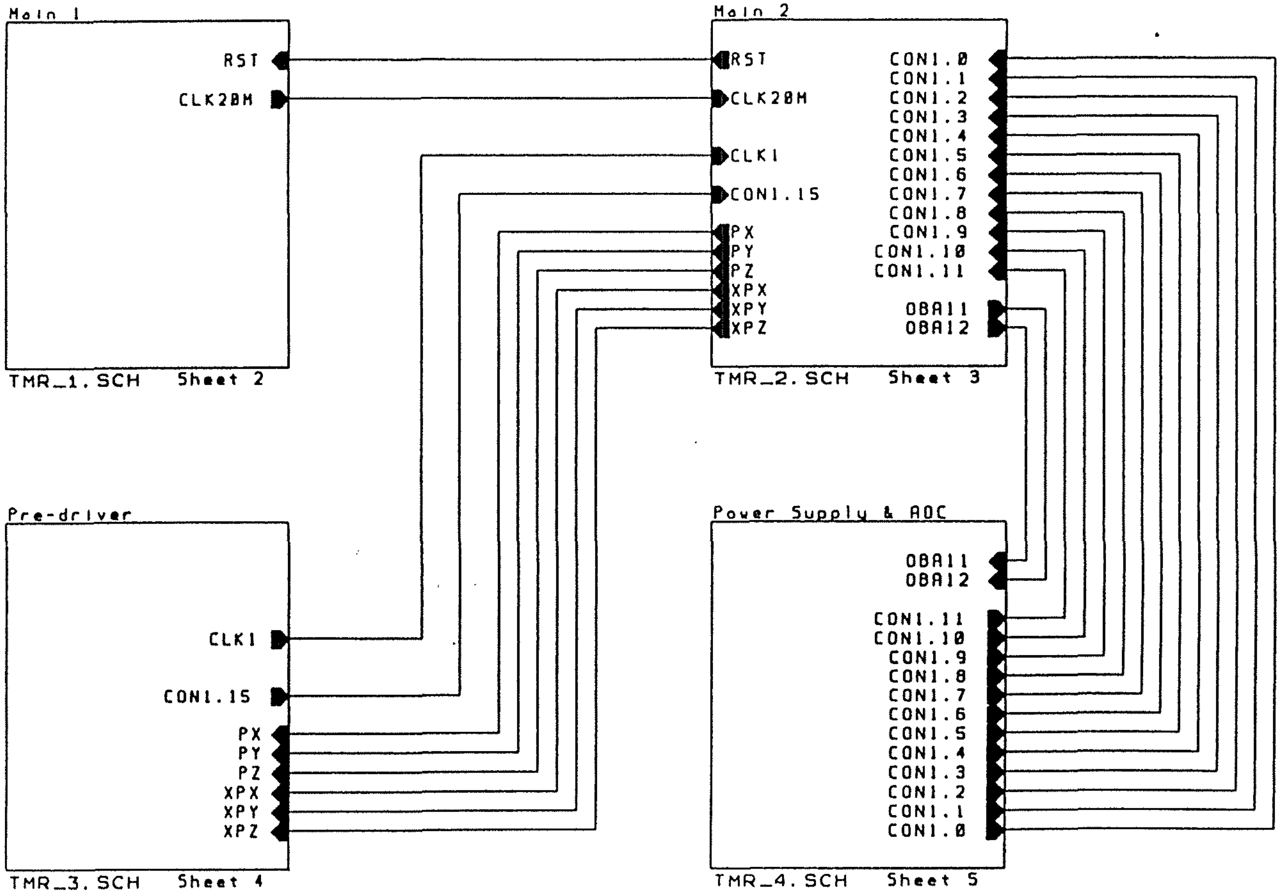
Servo Motor Controller Circuit Diagram 5/5 73

Servo Motor Controller Printed Circuit Board Traces, Component Copper Layer 74

Servo Motor Controller Printed Circuit Board Traces, Solder Copper Layer 75

Servo Motor Controller Printed Circuit Board Traces, Assembly Drawing 76

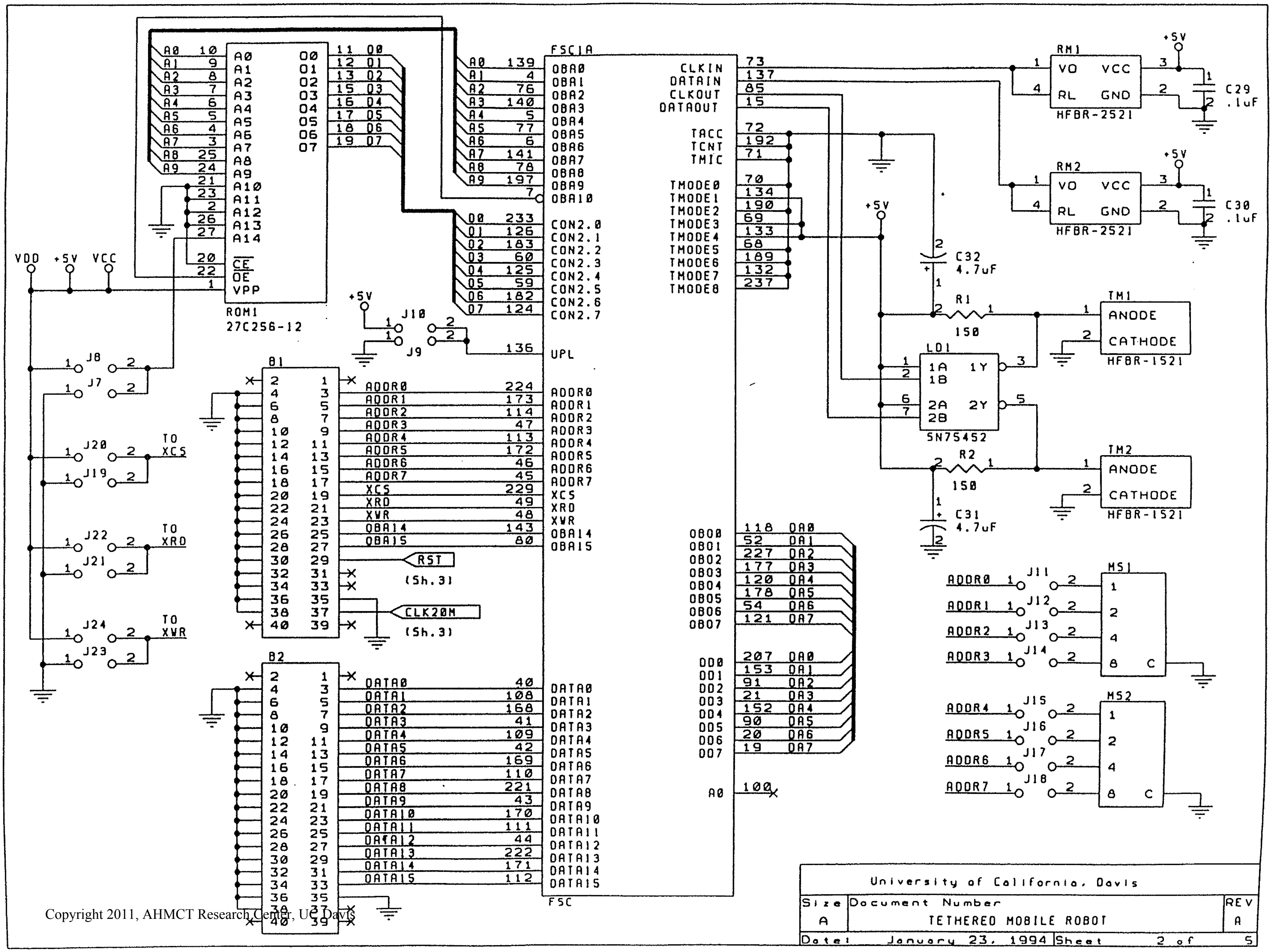
SUBCIRCUIT CONNECTIONS



University of California, Davis

Size	Document Number	REV
A	TETHERED MOBILE ROBOT	A

Date: January 23, 1994 Sheet 1 of 5



University of California, Davis

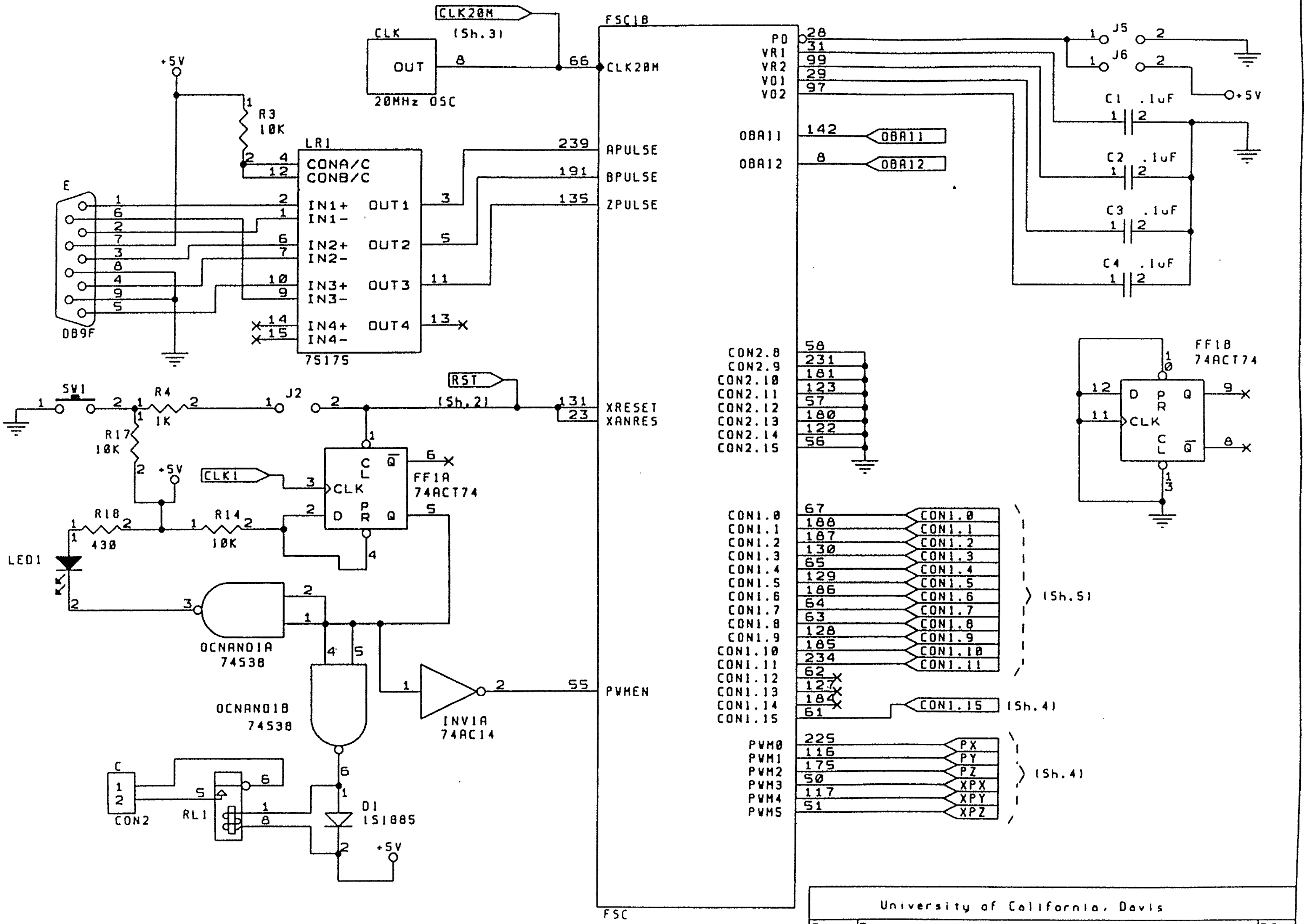
Size: A

Document Number: TETHERED MOBILE ROBOT

REV: A

Date: January 23, 1994

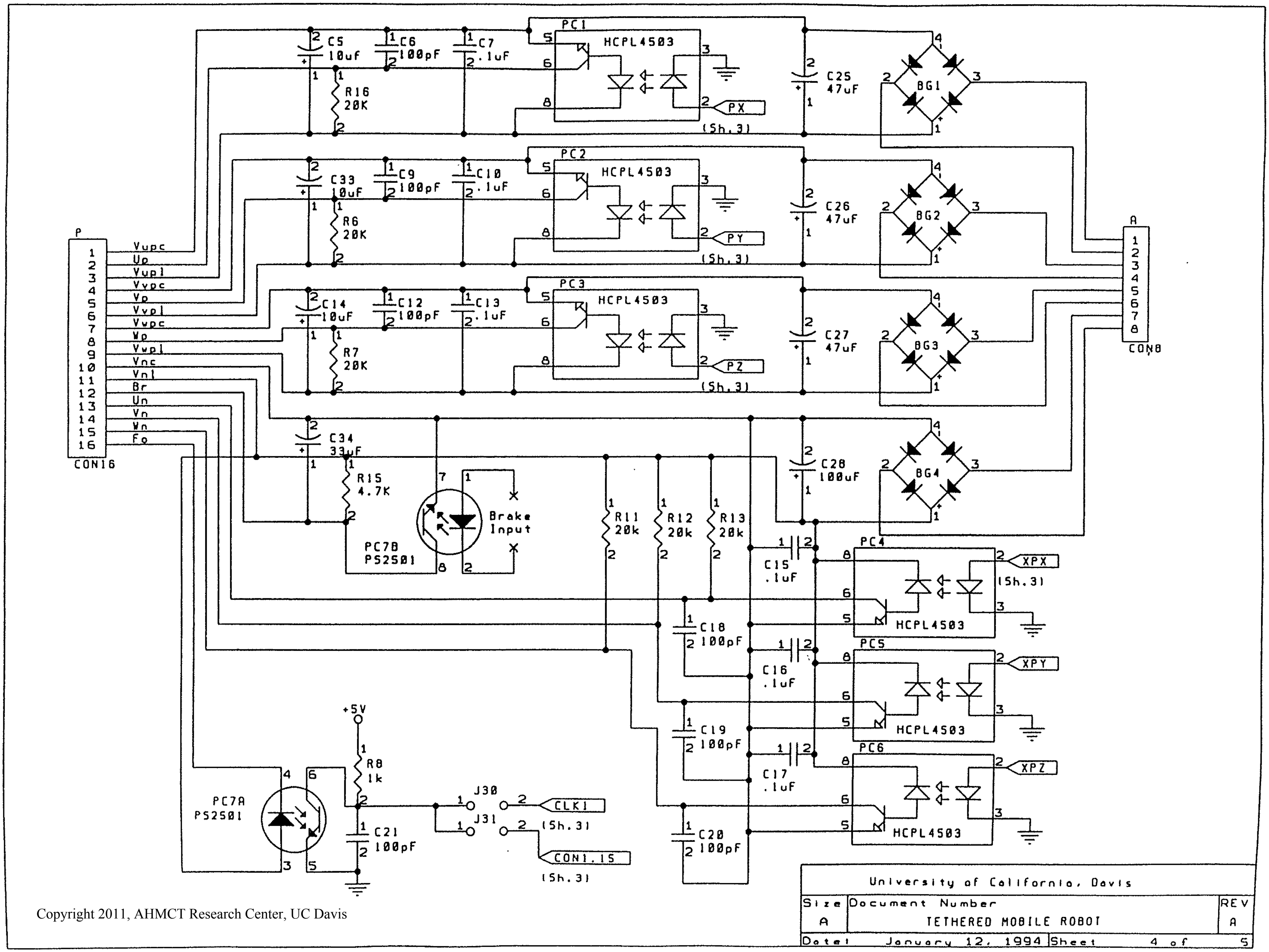
Sheet 2 of 5

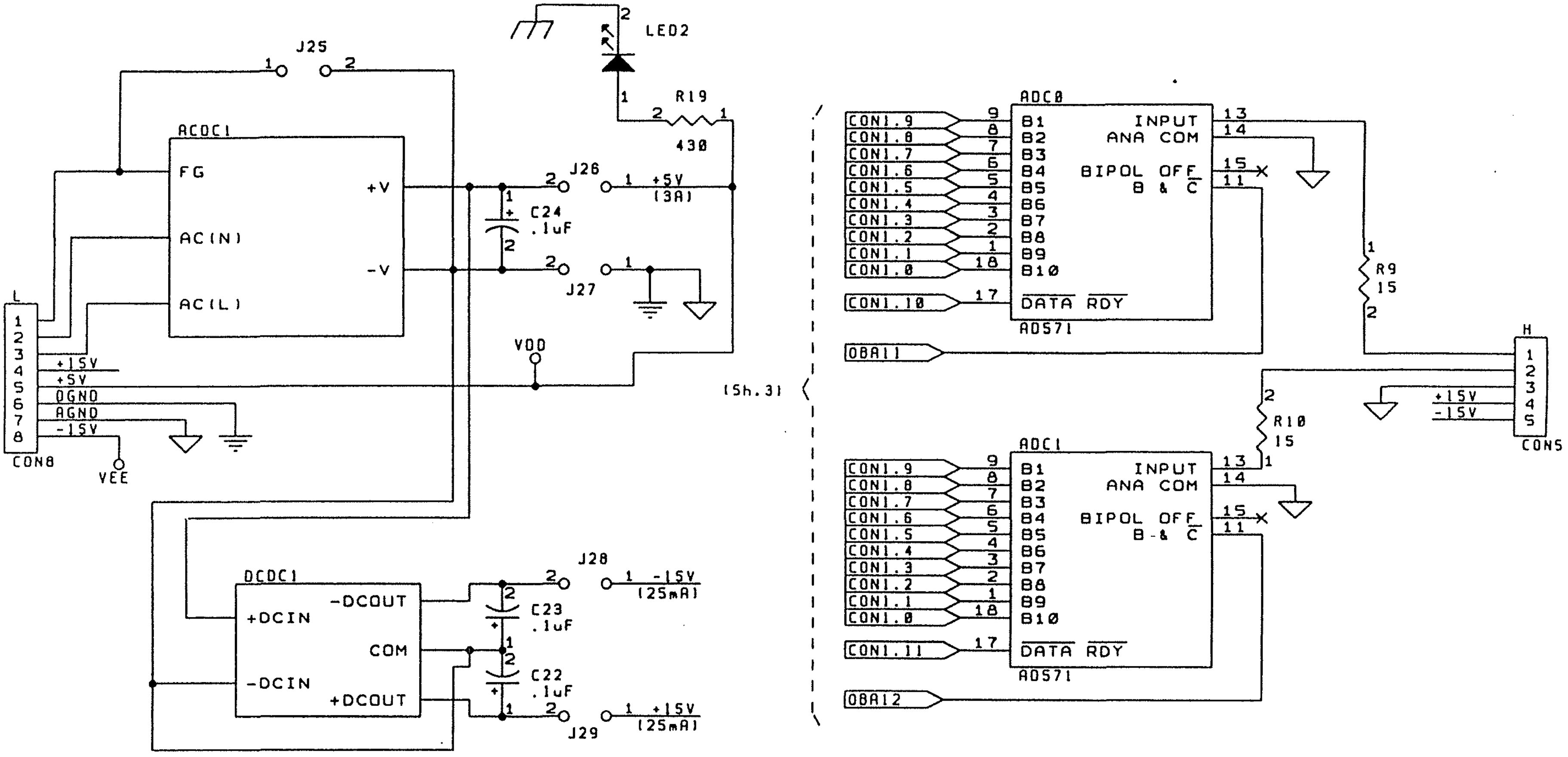


University of California, Davis

Size	Document Number	REV
A	TETHERED MOBILE ROBOT	A

Date: January 15, 1994 Sheet 3 of 5

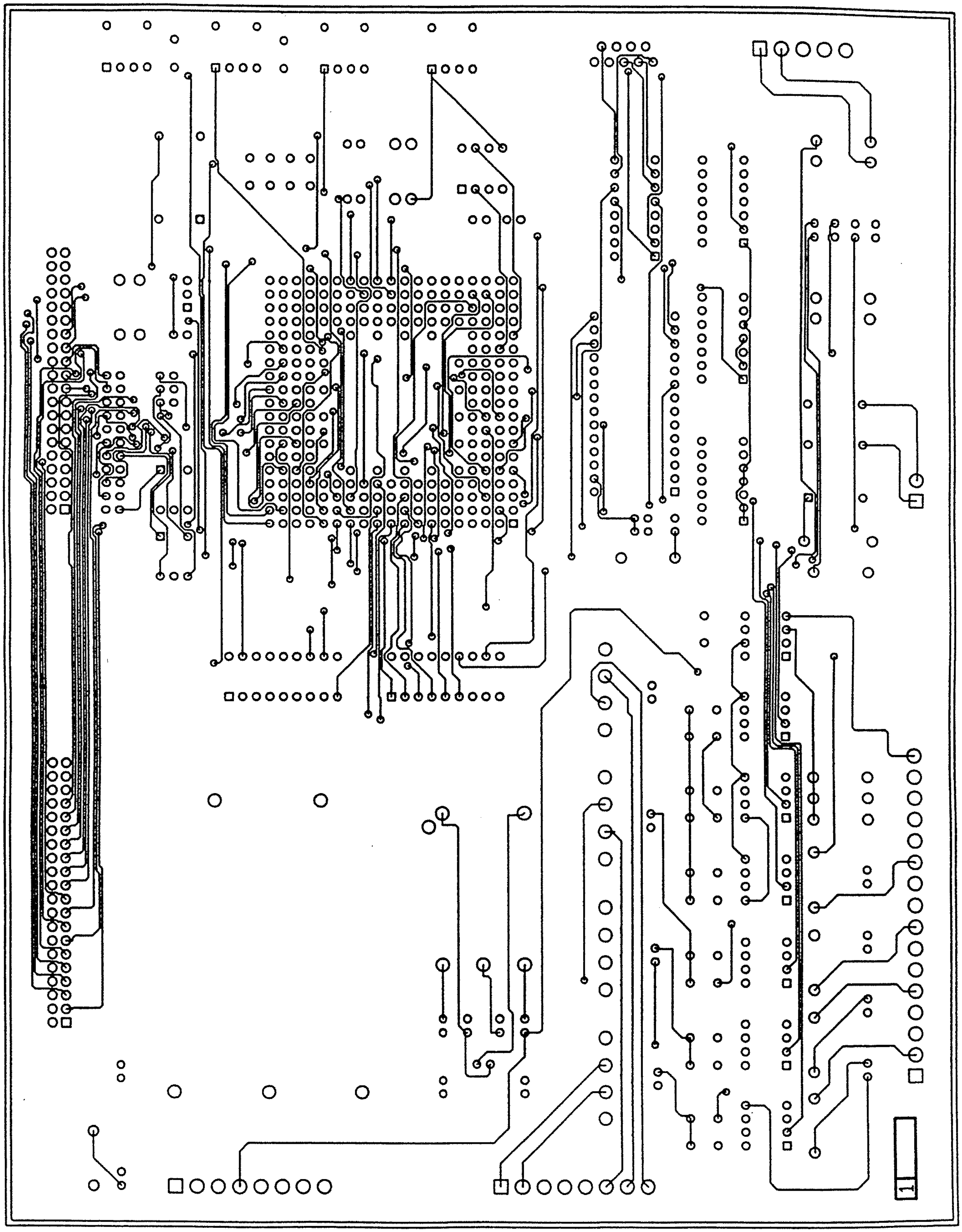

P
CON16
1 Vupc
2 Up
3 Vupl
4 Vvpc
5 Vp
6 Vvpl
7 Vwpc
8 Wp
9 Vwpl
10 Vnc
11 Vnl
12 Br
13 Un
14 Vn
15 Wn
16 Fo
C5 10uF
C6 100pF
C7 .1uF
R16 20K
PC1 HCPL4503
PX (Sh.3)
C25 47uF
BG1
C33 10uF
C9 100pF
C10 .1uF
R6 20K
PC2 HCPL4503
PY (Sh.3)
C26 47uF
BG2
C14 10uF
C12 100pF
C13 .1uF
R7 20K
PC3 HCPL4503
PZ (Sh.3)
C27 47uF
BG3
C34 33uF
R15 4.7K
PC7B PS2501
Brake Input
C28 100uF
BG4
A
CON8
R11 20k
R12 20k
R13 20k
C15 .1uF
C16 .1uF
C17 .1uF
C18 100pF
C19 100pF
C20 100pF
PC4 HCPL4503
XPX (Sh.3)
PC5 HCPL4503
XPY
PC6 HCPL4503
XPZ
+5V
R8 1k
C21 100pF
PC7A PS2501
J30
J31
CLK1 (Sh.3)
CON1.15 (Sh.3)
University of California, Davis
Size A
Document Number TETHERED MOBILE ROBOT
REV A
Date: January 12, 1994
Sheet 4 of 5

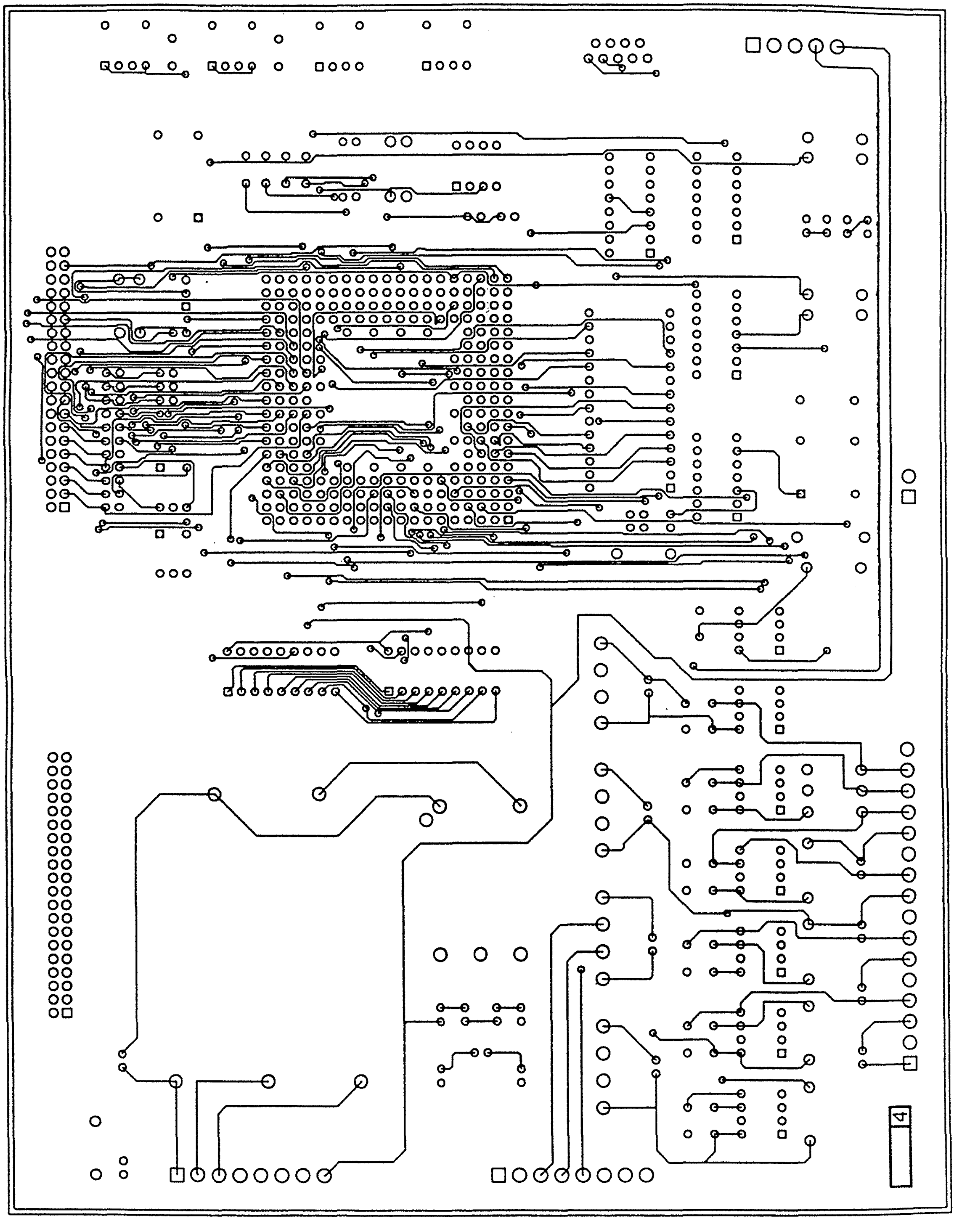


University of California, Davis

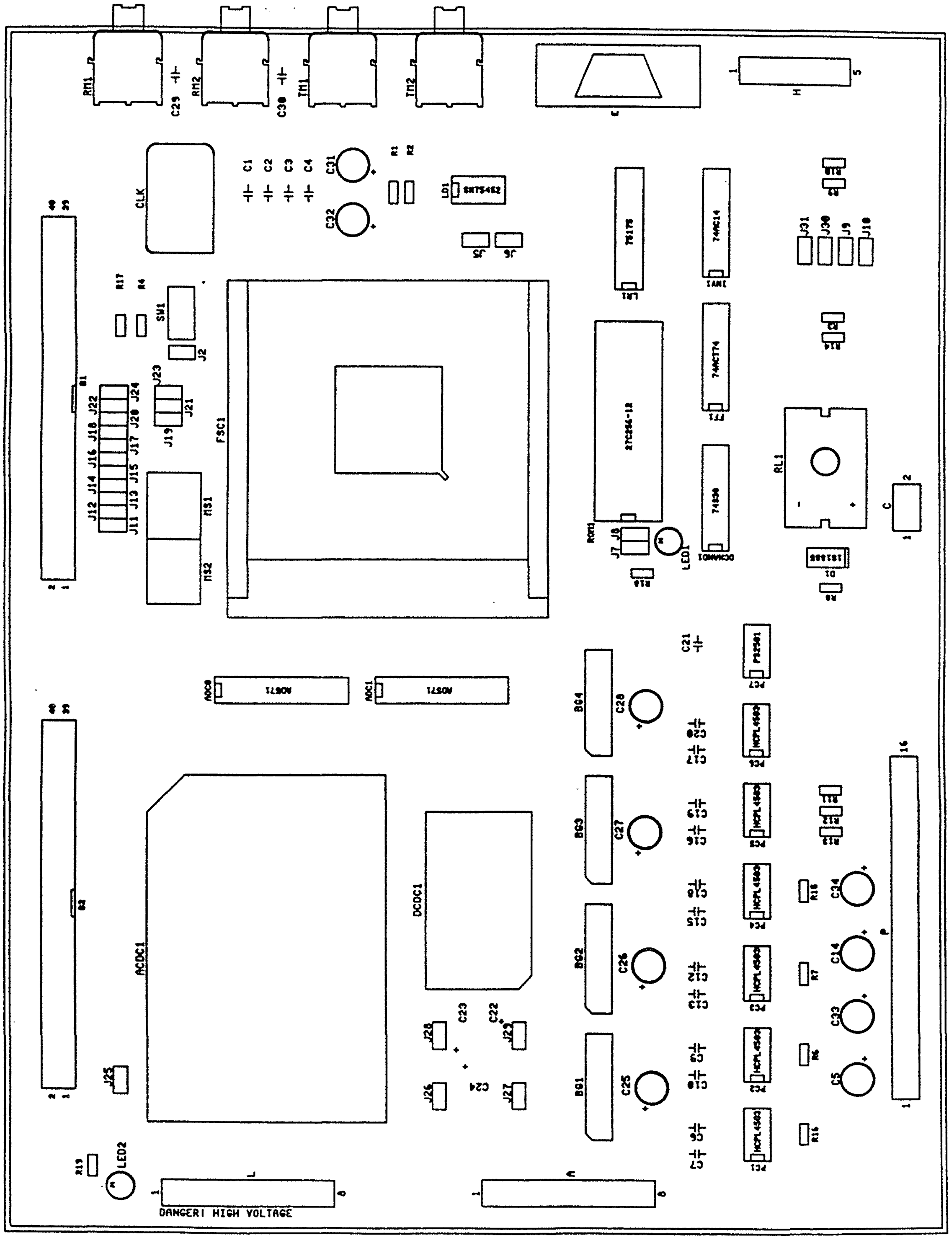
Size	Document Number	REV
A	TETHERED MOBILE ROBOT	A

Date: January 15, 1994 Sheet 5 of 5


1



4


RM1
C29
RM2
C30
TM1
TM2
E
1
5
H
CLK
C1
C2
C3
C4
C31
C32
R1
R2
LD1
SN75452
J5
J6
75176
LR1
74HC14
INV1
J31
J30
J9
J10
40
39
R17
R4
SW1
J2
J23
J21
J19
J24
J22
J20
J18
J17
J16
J15
J14
J13
J12
J11
B1
FSC1
MS1
MS2
27C256-12
ROM1
J8
J7
LED1
74HCT74
FF1
74C30
DCNAND1
RL1
C
2
1
D1
2
1
C21
PS2501
PC7
ADC0
AD571
ADC1
AD571
BG4
C28
C20
C17
HCPL4503
PC6
16
BG3
C27
C19
C16
PC5
B2
ACDC1
DCDC1
BG2
C26
C18
C15
PC4
R18
C34
C12
C13
PC3
R7
C14
P
C33
J28
C23
C22
J29
C24
J26
J27
BG1
C25
C9
C10
PC2
R6
C5
J25
2
1
C6
C7
PC1
R16
R13
LED2
1
J
8
DANGER! HIGH VOLTAGE
1
A
8

APPENDIX D

TMR CONTROL SOFTWARE CODE

```
/****************************************************************        */
/* File Name           BVELJS6.C                                         */
/* Function            TMR control program                               */
/*          ? : menu\n");                                                */
/*          a : set motor velocity control gains                         */
/*          b : tracking control - table reference                       */
/*          d : display speeds of each wheels                            */
/*          g : go signal (reset all gains and power                     */
/*          h : stop the motors                                          */
/*          j : Joystick operation                                       */
/*          l : Tracking control with offset table                       */
/*          m : Tracking control with laser sensor                       */
/*          p : point-to-point control                                   */
/*          q : quit and exit                                            */
/*          r : record speeds of both motors                             */
/*          s : change speeds of both motors                             */
/*          t : ADC reading                                              */
/*          v : return to default video mode (text screen)               */
/* Programed by        Dahie Hong                                        */
/* Date                Aug., 30, 1994                                    */
/* Version             6.0                                               */
/****************************************************************        */

#include <stdio.h>
#include <math.h>
#include <dos.h>
#include <time.h>
#include <graph.h>
#include <conio.h>
#include <malloc.h>
#include <stdlib.h>
#include <c:\hong\tmr\fscreg.h>            /* attached            */
#include "c:\encoder\source\m5312.c"       /* refer M5312 manual  */
#include "laser.c"                         /* attached            */

// define sound generation related parameters
#define TIMER_FREQ          1193180L
#define TIMER_COUNT         0x42
#define TIMER_MODE          0x43
#define TIMER_OSC                   0xb6
#define OUT_8255            0x61
#define SPKRON              3

// define display related parameters
#define dashed              0xA0A0
#define solid               0xFFFF
#define red                 4
#define cyan                3
#define green                       2
#define yellow              14
#define white                       7
#define lt_blue             9
#define brt_green           0
```

```
#define lt_magenta              13
#define black                          0
#define blue                    1
#define no_vert_lines           18
#define xconstant               35
#define no_horiz_lines          11
#define yconstant               35
#define pi                      3.1415927
#define cga             2.0833333      /* distance from front of TMR to applicator in inches */
#define cgb             0.416667       /* distance from rear of TMR to applicator in inches */

// define joystick and gameport related parameters
#define THUMB                   0x10
#define TRIGGER                 0x20
#define JS                      0x15
#define NAVE                    10
#define ticks                   3

// define 2-nd order filter coefficients
#define a2                      -1.561
#define a3                      0.6414
#define b1                      0.0201
#define b2                      0.0402
#define b3                      0.0201

#define S_limit                 500     /* speed limit of motor, rpm   */
#define CRACK_END               10000
#define PI                      3.141592654

extern unsigned short  enc_base;       /* globoal address for int_handler for encoder */
int     x_pos, y_pos, x0, x1, x2, x3, x4, yy0, yy1, y2, y3, y4;
int     tx1, tx2, tx3, tx4, tx5, tx6, tx7, tx8;
int     ty1, ty2, ty3, ty4, ty5, ty6, ty7, ty8;
int     s_laser_init;

double  Xs[2000], Ys[2000], Ts[2000], rr[2000];

int main()
{
void    init_encoder(int, int, int, int, int, short);
void    load_cntr(int, long, short);
long    read_cntr(int, short);
int     read_ip(int, short);

void    sound_on(unsigned);
void    sound_off(void);
double aatan(double, double, double, double);
void    Display_Menu(void);
void    init_motors(void);
void    Joystick_Control(void);
void    Point_to_Point(void);
int     Tracking_Control(void);
void    Send_Velocity_Commands(int, int);
void    draw_tmr(double, double, double, int, int, int, int, int, int);
```

```
void    create_replace_grid(void);
void    mark_origin(int, int);
void    set_workspace(void);
void    draw_tmr_trajectory(void);
int     Tracking_Control_Laser(void);
int     Tracking_Control_Laser_real(void);

int     j,k, xorigin, yorigin;
int     left_corner_x, left_corner_y, right_corner_x, right_corner_y;
int     x_old, x_new, y_old, y_new;
int     hor_lines, ver_lines, line_start_x, line_start_y;
char    xvalue, yvalue, thetavalue;
double  angular_pos, x_pos, y_pos, x, y;
FILE    *infile;

double  Xc, Yc, Tc, th1, th2, th3;
int     i, n_dp,wl,wr, ii;
double  Xr[10], Yr[10], Tr[10], X0, Y0, T0, Tt, ddt, Td;
double  wwc, vvc, aw, ww,ddtt;

char    sf[20];

float   wc=0.8; /* cart's angular velocity (rad/s) */
float   vf=0.8; /* cart's forward velocity (ft/s) */
float   gr=10.; /* gearbox ratio */
float   r=.4167;        /* radius of wheel (ft) */
float   d=2.;   /* axle length (ft) */

int     cmd, kc1, kc2, i_q_r, torque_angle, kv1, kv2;
int     vel_r,vel_r2, ik, ij, vel_tmp;
int     cmd_n,pwm_onoff,vel_fb_1,vel_fb_2,tmp_read;
char    cch, ch_tmp, cc1;

clock_t         ticksnow;
double          tused, dt1;

FILE    *f2, *f1;

        f2=fopen("c:\\hong\\tmr\\joy.m","w");

/* Initialization           */
        s_laser_init = 0;

/**     get initial parameters  **/
        printf("\nEnter the base address the 5312 is strapped at in hexadecimal - ");
        scanf("%x", &enc_base);

/** initialize each encoder - see manual for detail parameter setting **/
        init_encoder(AXIS_A, MCR, ICR, OCCR, QR, enc_base); /* init a          */
        init_encoder(AXIS_B, MCR, ICR, OCCR, QR, enc_base); /* init b          */
        init_encoder(AXIS_C, MCR, ICR, OCCR, QR, enc_base); /* init c          */

        init_motors();
```

```
printf("\n");
do
{
        printf("......waiting for z pulse\r");
}while(inpw(FSC_z_encountered) != 0);
printf("\n z pulse is encountered\n");
do {
        printf("\rPlace the robot at the initial position and type y when ready");
        cc1 = getchar();
}while( cc1 != 'y');

load_cntr(WR_ALL, 0l, enc_base);   /* zero counters          */

printf("\n\n\n");

kc1 = 4900;
kc2 = 600;

printf("\n\nType '?' for help.\n\n");
do
{
        printf("\nTMR>> ");
        cch = getchar();
        switch (cch) {
                case '?' :
                        Display_Menu();
                        break;

                case 'a' :
                        printf("Type kv1, kv2, vel_r1, vel_r2. ");
                        scanf("%d %d %d %d",&kv1,&kv2,&vel_r,&vel_r2);
                        do {
                                outpw(FSC_kc1_r,kc1);
                                tmp_read = inpw(FSC_kc1_r);
                        }while( tmp_read != kc1);
                        do {
                                outpw(FSC_kc2_r,kc2);
                                tmp_read = inpw(FSC_kc2_r);
                        }while( tmp_read != kc2);
                        do {
                                outpw(FSC_kv1_r,kv1);
                                tmp_read = inpw(FSC_kv1_r);
                        }while( tmp_read != kv1);
                        do {
                                outpw(FSC_kv2_r,kv2);
                                tmp_read = inpw(FSC_kv2_r);
                        }while( tmp_read != kv2);
                        do {
                                outpw(FSC_vel_r_r,vel_r);
                                tmp_read = inpw(FSC_vel_r_r);
                        }while( tmp_read != vel_r);
                        do {
                                outpw(FSC_vel_r_r_1,vel_r2);
                                tmp_read = inpw(FSC_vel_r_r_1);
```

```
            }while( tmp_read != vel_r2);
            break;

    case 'b' :
            ik = Tracking_Control();
            if ( ik != 0)
                    printf("\n......Tracking sinusoidal curve.
                    Type any key to stop.");
            else
                    printf("\n......Error on tracking control mode.");
            break;

    case 'c' :
            Get_Posture_Linkage(&th1,&th2,&th3,&Xc,&Yc,&Tc);
            printf("\nth1: %lf, th2: %lf, th3: %lf, Xc: %lf,Yc: %lf, Tc: %lf",
            th1,th2,th3,Xc,Yc,Tc);
            break;

    case 'd' :
            printf("\n\n\nVel 1      Vel 2 \n");
            do {
                    vel_fb_1 = inpw(FSC_vel);
                    vel_fb_2 = inpw(0xcf16);

                    for ( ik = 0; ik <100; ik++) {
                            printf("%5d %5d\r",vel_fb_1,vel_fb_2);
                    }
            }while(!kbhit());
            break;

    case 'g' :
            cmd = 3;
            do {
                    outpw(FSC_wait_or_go,cmd);
                    tmp_read = inpw(FSC_wait_or_go);
            }while(tmp_read != cmd);
            break;

    case 'h' :
            Send_Velocity_Commands(0,0);
            break;

    case 'j' :
            Joystick_Control();
            break;

    case 'l' :
            ik = Tracking_Control_Laser();
            if ( ik != 0)
                    printf("\n......Tracking crack path. Type any key to stop.");
            else
            printf("\n......Error on tracking control with laser.");
            break;
```

```
case 'm' :
        ik = Tracking_Control_Laser_real();
        if ( ik != 0)
                printf("\n......Tracking crack path. Type any key to stop.");
        else
                printf("\n......Error on tracking control with laser.");
        break;

case 'p' :
        Point_to_Point();
        break;

case 'q' :
        kc1=0; kc2=0; kv1=0; kv2=0; vel_r=0; vel_r2=0;
        do {
                outpw(FSC_kc1_r,kc1);
                tmp_read = inpw(FSC_kc1_r);
        }while( tmp_read != kc1);
        do {
                outpw(FSC_kc2_r,kc2);
                tmp_read = inpw(FSC_kc2_r);
        }while( tmp_read != kc2);
        do {
                outpw(FSC_kv1_r,kv1);
                tmp_read = inpw(FSC_kv1_r);
        }while( tmp_read != kv1);
        do {
                outpw(FSC_kv2_r,kv2);
                tmp_read = inpw(FSC_kv2_r);
        }while( tmp_read != kv2);
        do {
                outpw(FSC_vel_r_r,vel_r);
                tmp_read = inpw(FSC_vel_r_r);
        }while( tmp_read != vel_r);
        do {
                outpw(FSC_vel_r_r_1,vel_r2);
                tmp_read = inpw(FSC_vel_r_r_1);
        }while( tmp_read != vel_r2);
        exit(1);

case 'r' :
        printf("Input the file name to save data : ");
        scanf("%s",sf);
        f1=fopen(sf,"w");
        printf("\n\nInput vel_r1, vel_r2: ");
        scanf("%d %d",&vel_r, &vel_r2);
        do {
                outpw(FSC_vel_r_r,vel_r);
                tmp_read = inpw(FSC_vel_r_r);
        }while( tmp_read != vel_r);
        do {
                outpw(FSC_vel_r_r_1,vel_r2);
                tmp_read = inpw(FSC_vel_r_r_1);
        }while( tmp_read != vel_r2);
```

```
                                        fprintf(f1,"wl = %d,  wr = %d",vel_r,vel_r2);
                                        fprintf(f1,"Xc  Yc  Tc  FSC timer\n");

                                        for ( ik = 0; ik < 500; ++ik)
                                        {
                                                Get_Posture_Linkage(&th1,&th2,&th3,&Xc,&Yc,&Tc);
                                                fprintf(f1,"%lf %lf %lf %u\n",
                                                Xc,Yc,Tc,inpw(FSC_current_time));
                                                printf("....................................%d\r",ik);
                                        }
                                        break;

                                case 's' :
                                        printf("\n\nInput vel_r1, vel_r2: ");
                                        scanf("%d %d",&vel_r, &vel_r2);
                                        do {
                                                outpw(FSC_vel_r_r,vel_r);
                                                tmp_read = inpw(FSC_vel_r_r);
                                        }while( tmp_read != vel_r);
                                        do {
                                                outpw(FSC_vel_r_r_1,vel_r2);
                                                tmp_read = inpw(FSC_vel_r_r_1);
                                        }while( tmp_read != vel_r2);
                                        break;

                                case 't' :
                                        do {
                                                printf("ADC reading: %d %d\n",
                                                inpw(FSC_adc0),inpw(FSC_adc1));
                                        }while(1);
                                        break;

                                case 'v' :
                                        _setvideomode( _DEFAULTMODE );
                                        break;

                        };

                }while(1);

        }

        int GetJoystick(x,y)    /* read gameport        */
        int *x, *y;
        {
                union REGS regs;

                regs.h.ah = 0x84;       /* Joystick interrupt */
                regs.x.dx = 0x01;       /* 0 = switches, 1 = position */
                int86(JS, &regs, &regs); /* Issue DOS interrupt */
                *x = regs.x.ax;
                *y = regs.x.bx;
                return regs.x.cflag;
```

```
}

int GetSwitches(s)      /* read status of joystick switches      */
char *s;
{
        union REGS regs;

        regs.h.ah = 0x84;       /* Joystick interrupt */
        regs.x.dx = 0x00;       /* 0 = switches, 1 = position */
        int86(JS, &regs, &regs); /* Issue DOS interrupt */
        *s = regs.h.al;
        return regs.x.cflag;
}

// read encoders on the linkage and calculate the posture of the robot
Get_Posture_Linkage(double *th1, double *th2, double *th3,
                        double *Xc, double *Yc, double *Tc)
{
        long            a_cnt, b_cnt, c_cnt;    /* counter values */
        double          th1_o = 3.56483, th2_o = 2.295119, th3_o = -4.289152, L = 42.0;
        double          CF = 0.000628319;

        a_cnt = read_cntr(AXIS_A, enc_base); /* read counters       */
        b_cnt = read_cntr(AXIS_B, enc_base);
        c_cnt = read_cntr(AXIS_C, enc_base);
        if (a_cnt > 20000) a_cnt = a_cnt - 16777215;
        if (b_cnt > 20000) b_cnt = b_cnt - 16777215;
        if (c_cnt > 20000) c_cnt = c_cnt - 16777215;
        *th1 = (double) a_cnt;
        *th2 = (double) b_cnt;
        *th3 = (double) c_cnt;
        *th1 = CF * *th1 + th1_o;
        *th2 = CF * *th2 + th2_o;
        *th3 = CF * *th3 + th3_o;
        *Xc = L * cos(*th1) + L * cos(*th1 + *th2);
        *Yc = L * sin(*th1) + L * sin(*th1 + *th2);
        *Tc = *th1 + *th2 + *th3;
}

void Display_Menu()                     /* display menu after the prompt TMR >>     */
{
        printf("\n\n  ? : menu\n");
        printf("  a : set motor velocity control gains\n");
        printf("  b : tracking control - table reference\n");
        printf("  d : display speeds of each wheels\n");
        printf("  g : go signal (reset all gains and power\n");
        printf("  h : stop the motors\n");
        printf("  j : Joystick operation\n");
        printf("  l : Tracking control with offset table\n");
        printf("  m : Tracking control with laser sensor\n");
        printf("  p : point-to-point control\n");
        printf("  q : quit and exit\n");
        printf("  r : record speeds of both motors\n");
        printf("  s : change speeds of both motors\n");
```

```
        printf("  t : ADC reading\n");
        printf("  v : return to default video mode (text screen)\n");

}

void    init_motors()  /* initialize motor control parameters              */
{
        int torque_angle=2000, tmp_read;
        int cmd=2, kc1=0, kc2=0, kv1=0, kv2=0, vel_r=0, vel_r2=0;

        do {
                outpw(FSC_torque_angle,torque_angle);
                tmp_read = inpw(FSC_torque_angle);
        }while( tmp_read != torque_angle);
        cmd=2;
        do {
                outpw(FSC_wait_or_go,cmd);
                tmp_read = inpw(FSC_wait_or_go);
        }while( tmp_read != cmd);

        do {
                outpw(FSC_kc1_r,kc1);
                tmp_read = inpw(FSC_kc1_r);
        }while( tmp_read != kc1);
        do {
                outpw(FSC_kc2_r,kc2);
                tmp_read = inpw(FSC_kc2_r);
        }while( tmp_read != kc2);
        do {
                outpw(FSC_kv1_r,kv1);
                tmp_read = inpw(FSC_kv1_r);
        }while( tmp_read != kv1);
        do {
                outpw(FSC_kv2_r,kv2);
                tmp_read = inpw(FSC_kv2_r);
        }while( tmp_read != kv2);
        do {
                outpw(FSC_vel_r_r,vel_r);
                tmp_read = inpw(FSC_vel_r_r);
        }while( tmp_read != vel_r);
        do {
                outpw(FSC_vel_r_r_1,vel_r2);
                tmp_read = inpw(FSC_vel_r_r_1);
        }while( tmp_read != vel_r2);
}

void    Joystick_Control()
{
        char    sw;
        int     thumb,trigger,wlt,wrt,errs,errj,xcent,ycent,wl,wr;
        int     x,y,xsum,ysum,err,i,tmp_read,safty;
        float   wrtf,wltf,xx, yy;

        float   wlxn=0.0, wlxn_1=0.0, wlxn_2=0.0;
```

```
        float   wlyn=0.0, wlyn_1=0.0, wlyn_2=0.0;
        float   wrxn=0.0, wrxn_1=0.0, wrxn_2=0.0;
        float   wryn=0.0, wryn_1=0.0, wryn_2=0.0;

        float   wc=0.8; /* cart's angular velocity (rad/s) */
        float   vf=0.8; /* cart's forward velocity (ft/s) */
        float   gr=10.; /* gearbox ratio */
        float   r=.4167;        /* radius of wheel (ft) */
        float   d=2.;   /* axle length (ft) */

        int     j,k, xorigin, yorigin;
        int     left_corner_x, left_corner_y, right_corner_x, right_corner_y;
        int     x_old, x_new, y_old, y_new;
        int     hor_lines, ver_lines, line_start_x, line_start_y;
        char    xvalue, yvalue, thetavalue, buffer2[100];
        double          angular_pos, x_pos, y_pos, xc, yc,th1,th2,th3, dist,dist_old, ang_old;
        FILE    *infile, *joyout;
        char    sf[20];
        double tused;
        clock_t ticksnow;

//      printf("Input the file name to save data : ");
//      scanf("%s",sf);
//      joyout=fopen(sf,"w");

        printf("Calibrating Joystick. Please Wait\n");
        xsum = ysum = 0;
        for(i=0;i<NAVE;i++){
        err = GetJoystick(&x,&y);
        if(err){
                printf("Error reading joystick. Exiting\n");
                exit(1);
        }
        xsum += x;
        ysum += y;
        }
        xcent = xsum / NAVE;
        ycent = ysum / NAVE;
        printf("Done calibrating joystick. Press Enter to proceed with manual operation.\n");
        printf("You must hold the trigger in order to control the mobile robot.\n");

        _setvideomode( _VRES16COLOR );
        set_workspace();
        k = 0;

/*  MAKES GRIDLINES FOR WORKSPACE  */
        xorigin = 9+no_vert_lines/2*xconstant;
        yorigin = 65+(no_horiz_lines-2)*yconstant;
        mark_origin(xorigin, yorigin);

/*  SCANS FIRST POSTURE OF TMR  */
        Get_Posture_Linkage(&th1,&th2,&th3,&xc,&yc,&angular_pos);
        x_old = xorigin+(int)(xc*xconstant/12);
        y_old = yorigin-(int)(yc*yconstant/12);
```

```
dist_old = sqrt(xc*xc+yc*yc);
ang_old = angular_pos;
do {
        errj = GetJoystick(&x,&y);
        errs = GetSwitches(&sw);
        trigger = !(THUMB&sw);
        thumb = !(TRIGGER&sw);
        x=x-xcent;
        y=y-ycent;
        y=-y;
        if (trigger){
                xx= (float) x;
                yy= (float) y;
        }
        else {
                xx=0.;
                yy=0.;
        }

        Get_Posture_Linkage(&th1,&th2,&th3,&xc,&yc,&angular_pos);

        dist = sqrt(xc*xc + yc*yc);
        safty = 0;
        if ( (dist >= 72) && (dist > dist_old) ) {
                yy = yy*0.1;
                safty = 1;
        }
        if ( (dist <= 18) && (dist < dist_old) ) {
                yy = yy*0.02;
                safty = 2;
        }

        if ( (angular_pos < -1.571) && (angular_pos < ang_old) ) {
                xx = xx * 0.1;
                safty = 3;
        }
        if ( (angular_pos > ang_old) && (angular_pos > 4.712) ) {
                xx = xx * 0.1;
                safty = 3;
        }

        if (safty == 0) sound_off();
        if (safty == 1) sound_on(500);
        if (safty == 2) sound_on(1000);
        if (safty == 3) sound_on(2000);

        dist_old = dist;
        ang_old = angular_pos;


        wltf=gr/r*(d/2.*wc/63.*xx+vf/75.*yy)*9.55;
        wrtf=gr/r*(-d/2.*wc/63.*xx+vf/75.*yy)*9.55;
```

```
// 2-nd order filteration
                    wlxn_2 = wlxn_1; wlxn_1 = wlxn; wlxn = wltf;
                    wlyn = b1*wlxn + b2*wlxn_1 + b3*wlxn_2 - a2*wlyn_1 - a3*wlyn_2;
                    wlyn_2 = wlyn_1; wlyn_1 = wlyn;

                    wrxn_2 = wrxn_1; wrxn_1 = wrxn; wrxn = wrtf;
                    wryn = b1*wrxn + b2*wrxn_1 + b3*wrxn_2 - a2*wryn_1 - a3*wryn_2;
                    wryn_2 = wryn_1; wryn_1 = wryn;

                    wrt= (int) wryn;
                    wlt= (int) wlyn;
                    wr = (int) wrtf;          wr = -wr;
                    wl = (int) wltf;
                    wrt = -wrt;

                    Send_Velocity_Commands(wlt,wrt);
                    ticksnow = clock();
                    tused = (double) ticksnow / CLK_TCK;
//                  fprintf(joyout,"%lf\n",tused);

//                  fprintf(joyout,"%d %d %d %d %d %u\n",
                    wl, wr, wlt,wrt,inpw(FSC_vel),inpw(0xcf16),inpw(FSC_current_time));

                    k++;
                    if (k > 9) {
                              x_new = xorigin+(int)(xc*xconstant/12);
                              y_new = yorigin-(int)(yc*yconstant/12);
                              draw_tmr(xc, yc, angular_pos, xorigin, yorigin,
                              x_old, x_new, y_old, y_new);
                              x_old = x_new;
                              y_old = y_new;
                              k = 0;
                    }


          }while(!kbhit());
          _setvideomode( _DEFAULTMODE );

}


void      Point_to_Point()        /* point to point control mode          */
{
          double          Xc, Yc, Tc, th1, th2, th3;
          int             i, n_dp,wl,wr, ii, tmp_read;
          double          Xr[10], Yr[10], Tr[10], X0, Y0, T0, Tt, ddt, Td;
          double          wwc, vvc, aw, ww,ddtt;

          Get_Posture_Linkage(&th1,&th2,&th3,&Xc,&Yc,&Tc);
          Xr[0] = Xc; Yr[0] = Yc, Tr[0] = Tc;
          printf("\nPresent posture : (%8.3lf, %8.3lf, %5.3lf)",Xc,Yc,Tc);
          printf("\nInput number of destination points: ");
          scanf("%d",&n_dp);
          for (i=1;i<=n_dp;i++) {
```

```
		printf("\nInput posture for point %d : ",i);
		scanf("%lf %lf %lf",&Xr[i],&Yr[i],&Tr[i]);
	}

	for (i=0;i<n_dp;i++) {
		if (((Xr[i+1]-Xc) >= 0) && ((Yr[i+1]-Yc) >= 0))
			Td = atan((Yr[i+1]-Yc)/(Xr[i+1]-Xc));
		if (((Xr[i+1]-Xc) < 0) && ((Yr[i+1]-Yc) >= 0))
			Td = 3.141592 - atan((Yr[i+1]-Yc)/(Xc-Xr[i+1]));
		if (((Xr[i+1]-Xc) < 0) && ((Yr[i+1]-Yc) < 0))
			Td = 3.141592 + atan((Yr[i+1]-Yc)/(Xr[i+1]-Xc));
		if (((Xr[i+1]-Xc) >= 0) && ((Yr[i+1]-Yc) < 0))
			Td = - atan((Yc-Yr[i+1])/(Xr[i+1]-Xc));

		Tt = Td - Tc;
/*		ticksnow = clock();
		tused = (double) ticksnow/CLK_TCK;		*/
		wwc = 0; aw = 0.1;
		if (Tt > 0) {
			wl = -25; wr = wl;
			do {
				Get_Posture_Linkage(&th1,&th2,&th3,&Xc,&Yc,&Tc);
				do {
					outpw(FSC_vel_r_r,wl);
					tmp_read = inpw(FSC_vel_r_r);
				}while(tmp_read != wl);
				do {
					outpw(FSC_vel_r_r_1,wr);
					tmp_read = inpw(FSC_vel_r_r_1);
				}while(tmp_read != wr);
			}while( Tc < Td);
		}
		if (Tt < 0) {
			wl = 25; wr = wl;
			do {
				Get_Posture_Linkage(&th1,&th2,&th3,&Xc,&Yc,&Tc);
				do {
					outpw(FSC_vel_r_r,wl);
					tmp_read = inpw(FSC_vel_r_r);
				}while(tmp_read != wl);
				do {
					outpw(FSC_vel_r_r_1,wr);
					tmp_read = inpw(FSC_vel_r_r_1);
				}while(tmp_read != wr);
			}while( Tc  >  Td);
		}

		for (ii=0;ii<100;ii++) {
			wl = 0; wr = 0;
			do {
				outpw(FSC_vel_r_r,wl);
				tmp_read = inpw(FSC_vel_r_r);
			}while(tmp_read != wl);
			do {
```

```
                                outpw(FSC_vel_r_r_1,wr);
                                tmp_read = inpw(FSC_vel_r_r_1);
                        }while(tmp_read != wr);
                }

                ddt = sqrt((Xr[i+1]-Xc)*(Xr[i+1]-Xc)+(Yr[i+1]-Yc)*(Yr[i+1]-Yc));
                X0=Xc; Y0=Yc; T0=Tc;
                ddtt = 0;
                do {
                        Get_Posture_Linkage(&th1,&th2,&th3,&Xc,&Yc,&Tc);
                        ddtt = sqrt((Xc-X0)*(Xc-X0)+(Yc-Y0)*(Yc-Y0));
                        if (ddtt <= 0.2*ddt) wwc = 25;
                        if ((ddtt > 0.2*ddt)&&(ddtt<0.8*ddt)) wwc = 50;
                        if (ddtt>0.8*ddt) wwc = 25;
                        wl = (int) wwc; wr = -wl;
                        do {
                                outpw(FSC_vel_r_r,wl);
                                tmp_read = inpw(FSC_vel_r_r);
                        }while(tmp_read != wl);
                        do {
                                outpw(FSC_vel_r_r_1,wr);
                                tmp_read = inpw(FSC_vel_r_r_1);
                        }while(tmp_read != wr);
                }while( ddtt < ddt);

                for (ii=0;ii<100;ii++) {
                        wl = 0; wr = 0;
                        do {
                                outpw(FSC_vel_r_r,wl);
                                tmp_read = inpw(FSC_vel_r_r);
                        }while(tmp_read != wl);
                        do {
                                outpw(FSC_vel_r_r_1,wr);
                                tmp_read = inpw(FSC_vel_r_r_1);
                        }while(tmp_read != wr);
                }

        }

}


int     Tracking_Control(void)          /* tracking control mode with reference table */
{
        int             wli, wri;
        double          th1,th2,th3, Xr,Yr,Tr, ur,rr, Xc,Yc,Tc, x,y,p, X0,Y0,T0;
        double          e = 0, c = 1.2, T2 = 12.5, R = 5.0;
        double          invE[2][2], f[2], K[2][2], z, u[2], u1[2], wl, wr;
        char            sf[20];
        char            buffer1[100], buffer2[100];

        int             j,k, xorigin, yorigin;
        int             left_corner_x, left_corner_y, right_corner_x, right_corner_y;
        int             x_old, x_new, y_old, y_new, x_old_r, y_old_r, x_new_r, y_new_r;
```

```
        int             hor_lines, ver_lines, line_start_x, line_start_y;
        char            xvalue, yvalue, thetavalue;
        double          angular_pos, x_pos, y_pos;

        double          tused;
        clock_t         ticksnow;

        FILE            *f1, *ft, *ft2;

//      printf("Input the file name to save data : ");
//      scanf("%s",sf);
//      ft=fopen(sf,"w");

        if ( (f1 = fopen("c:\\hong\\tmr\\control\\ref_path.dat","r")) == NULL ) {
                printf("\nError! Reference path data file can not be opened");
                return(0);
        }
        ft = fopen(sf,"w");
        ft2 = fopen("c:\\hong\\tmr\\control\\rt.out","w");

//      fprintf(ft,"Xr,Yr,Tr,Xc,Yc,Tc,x,y,p,u[1],u[2]\n");

        /* Set control gain matrix  */
        printf("\nInput control gains( K(1,1),K(1,2),K(2,1),K(2,2)): ");
        scanf("%lf %lf %lf %lf",&K[0][0],&K[0][1],&K[1][0],&K[1][1]);

        Get_Posture_Linkage(&th1,&th2,&th3,&X0,&Y0,&T0);
        _setvideomode( _VRES16COLOR );
        set_workspace();

/*  MAKES GRIDLINES FOR WORKSPACE  */
        xorigin = 9+no_vert_lines/2*xconstant;
        yorigin = 65+(no_horiz_lines-2)*yconstant;
        mark_origin(xorigin, yorigin);
/*  SCANS FIRST POSTURE OF TMR  */
        x_old = xorigin+(int)(X0*xconstant/12);
        y_old = yorigin-(int)(Y0*yconstant/12);

        fscanf(f1,"%lf %lf %lf %lf %lf\n",&Xr,&Yr,&Tr,&ur,&rr);
        x_old_r = xorigin+(int)(Xr*xconstant/12);
        y_old_r = yorigin-(int)(Yr*yconstant/12);

        while( !kbhit() && ((fscanf(f1,"%lf %lf %lf %lf %lf\n",&Xr,&Yr,&Tr,&ur,&rr))
        != EOF) ) {
                Get_Posture_Linkage(&th1,&th2,&th3,&Xc,&Yc,&Tc);
/* Calculate error posture in body coordinate */
                x = (Xr - Xc)*cos(Tc) + (Yr - Yc)*sin(Tc);
                y = -(Xr - Xc)*sin(Tc) + (Yr - Yc)*cos(Tc);
                p = Tr - Tc;
                invE[0][0] = -1;
                invE[0][1] = -y/(e+x+c);
                invE[1][0] = 0;
                invE[1][1] = -1/(e+x+c);
                f[0] = ur*cos(Tc);
```

```
                    f[1] = ur*sin(Tc) + c*rr;

                    z = y + c*p;
                    u1[0] = -f[0] - K[0][0] * x -K[0][1] * z;
                    u1[1] = -f[1] - K[1][0] * x -K[1][1] * z;
                    u[0] = invE[0][0] * u1[0] + invE[0][1] * u1[1];
                    u[1] = invE[1][0] * u1[0] + invE[1][1] * u1[1];
                    wl = ( u[0] - T2*u[1] ) / R * 9.55;
                    wr = ( u[0] + T2*u[1] ) / R * 9.55;

                    wli = (int)wl; wri = (int)wr; wri = -wri;

                    Send_Velocity_Commands(wli,wri);

                    x_new = xorigin+(int)(Xc*xconstant/12);
                    y_new = yorigin-(int)(Yc*yconstant/12);
                    x_new_r = xorigin+(int)(Xr*xconstant/12);
                    y_new_r = yorigin-(int)(Yr*yconstant/12);

/*  DISPAYS X AND Y COORDINATES AND THETA  */
                    _settextcolor(white);
                    _settextposition(3,10);
                    sprintf(buffer2,"x: %lf in       y: %lf in        theta: %lf degrees",
                    Xc, Yc, Tc*180/pi);
                    _outtext(buffer2);

/*  DRAWS REFERENCE PATH     */
                    _setcolor(green);
                    _moveto(x_old_r, y_old_r);
                    _lineto(x_new_r, y_new_r);

/*  DRAWS TMR PATH  */
                    _setcolor(lt_magenta);
                    _moveto(x_old, y_old);
                    _lineto(x_new, y_new);

                    x_old = x_new;
                    y_old = y_new;

                    x_old_r = x_new_r;
                    y_old_r = y_new_r;

                    ticksnow = clock();
                    tused = (double) ticksnow / CLK_TCK;
//                  fprintf(ft,"%lf %lf %lf %lf %lf %lf %lf %lf %lf %lf %lf\n",
                    Xr,Yr,Tr,Xc,Yc,Tc,x,y,p,u[0],u[1]);
//                  fprintf(ft2,"%lf\n",tused);

          }

          wli = 0; wri = 0;
          Send_Velocity_Commands(wli,wri);
          fclose(ft);
          fclose(f1);
```

```
        return(1);

}

int     Tracking_Control_Laser(void)        /* tracking control mode with offset table */
{
        int             wli, wri;
        double          th1,th2,th3, Xr,Yr,Tr, ur, Xc,Yc,Tc, x,y,p, X0,Y0,T0;
        double          e, c, T2, R;
        double          invE[2][2], f[2], K[2][2], z, u[2], u1[2], wl, wr;
        char            sf[20];
        char            buffer1[100], buffer2[100];

        int             j,k, xorigin, yorigin;
        int             left_corner_x, left_corner_y, right_corner_x, right_corner_y;
        int             x_old, x_new, y_old, y_new, x_old_r, y_old_r, x_new_r, y_new_r;
        int             hor_lines, ver_lines, line_start_x, line_start_y;
        char            xvalue, yvalue, thetavalue;
        double          angular_pos, x_pos, y_pos;

        double          tused;
        clock_t         ticksnow;

        FILE            *f1, *ft, *ft2, *ff1;

        int             nd, i, ii, kn, kn1, rind, found_ref_value, status = 0;
        int             Ntt;
        double          D, offset, ds, sinT, cosT;
        double          rrd, dt, xnd, ynd, Ts0, Xs0, Ys0, ndf, di;

        double          Xcx, Xcxn, Xcxn_1, Xcxn_2, Xcyn_1, Xcyn_2;
        double          Ycx, Ycxn, Ycxn_1, Ycxn_2, Ycyn_1, Ycyn_2;
        double          Tcx, Tcxn, Tcxn_1, Tcxn_2, Tcyn_1, Tcyn_2;
        double          Xsx, Xsxn, Xsxn_1, Xsxn_2, Xsyn_1, Xsyn_2;
        double          Ysx, Ysxn, Ysxn_1, Ysxn_2, Ysyn_1, Ysyn_2;
        double          Tsx, Tsxn, Tsxn_1, Tsxn_2, Tsyn_1, Tsyn_2;
        double          an2, an3;
        double          bn1, bn2, bn3;

        an2 = -1.926; an3 = 0.9286;
        bn1 = 0.0007; bn2 = 0.0013; bn3 = 0.0007;

        e = 0; c = 1.2; T2 = 12.5; R = 5.0;
        Ntt = 2000; D = 9.75;

        for (i = 0;i < Ntt;i++) {
                Xs[i] = 0.0;
                Ys[i] = 0.0;
                Ts[i] = 0.0;
                rr[i] = 0.0;
        }

        if ( (f1 = fopen("c:\\hong\\tmr\\control\\ref_off.dat","r")) == NULL ) {
                printf("\nError! Reference offset data file can not be opened");
```

```
            return(0);
        }
        printf("Input the file name to save data : ");
        scanf("%s",sf);
        ft=fopen(sf,"w");

        ft2 = fopen("c:\\hong\\tmr\\control\\rt.out","w");
        ff1 = fopen("c:\\hong\\tmr\\control\\off.out","w");

        /*Set linear speed*/
        printf("\nInput linear speed( inch/sec ): ");
        scanf("%lf",&ur);
        /*Set sampling time*/
        printf("\nInput sampling time( sec ): ");
        scanf("%lf",&dt);

        /* Set control gain matrix  */
        printf("\nInput control gains( K(1,1),K(1,2),K(2,1),K(2,2)): ");
        scanf("%lf %lf %lf %lf",&K[0][0],&K[0][1],&K[1][0],&K[1][1]);

        ds = ur*dt;

        Get_Posture_Linkage(&th1,&th2,&th3,&X0,&Y0,&T0);

        Xcxn = X0; Xcxn_1 = X0; Xcxn_2 = X0; Xcyn_1 = X0; Xcyn_2 = X0;
        Ycxn = Y0; Ycxn_1 = Y0; Ycxn_2 = Y0; Ycyn_1 = Y0; Ycyn_2 = Y0;
        Tcxn = T0; Tcxn_1 = T0; Tcxn_2 = T0; Tcyn_1 = T0; Tcyn_2 = T0;

        _setvideomode( _VRES16COLOR );
        set_workspace();

/*  MAKES GRIDLINES FOR WORKSPACE  */
        xorigin = 9+no_vert_lines/2*xconstant;
        yorigin = 65+(no_horiz_lines-2)*yconstant;
        mark_origin(xorigin, yorigin);

//      Xsxn = X0; Xsxn_1 = X0; Xsxn_2 = X0; Xsyn_1 = X0; Xsyn_2 = X0;
//      Ysxn = Y0; Ysxn_1 = Y0; Ysxn_2 = Y0; Ysyn_1 = Y0; Ysyn_2 = Y0;
//      Tsxn = T0; Tsxn_1 = T0; Tsxn_2 = T0; Tsyn_1 = T0; Tsyn_2 = T0;

        fscanf(f1,"%lf\n",&offset);
        Xs0 = X0 + D*cos(T0) - offset*sin(T0);
        Ys0 = Y0 + D*sin(T0) + offset*cos(T0);
        Ts0 = T0 + atan(offset/D);

        nd = D / ds;
        ndf = (double) nd;
        xnd = (Xs0 - X0) / ndf;
        ynd = (Ys0 - Y0) / ndf;
        ii = 0;
        for(i = 0;i < nd;i++) {
                di = (double) i;
                Xs[i] = X0 + xnd*di;
                Ys[i] = Y0 + ynd*di;
```

```
                    Ts[i] = Ts0;

/*                  Xsxn_2 = Xsxn_1; Xsxn_1 = Xsxn; Xsxn = Xsx;
                    Xs[ii] = bn1*Xsxn + bn2*Xsxn_1 + bn3*Xsxn_2 - an2*Xsyn_1 - an3*Xsyn_2;
                    Xsyn_2 = Xsyn_1; Xsyn_1 = Xs[ii];

                    Ysxn_2 = Ysxn_1; Ysxn_1 = Ysxn; Ysxn = Ysx;
                    Ys[ii] = bn1*Ysxn + bn2*Ysxn_1 + bn3*Ysxn_2 - an2*Ysyn_1 - an3*Ysyn_2;
                    Ysyn_2 = Ysyn_1; Ysyn_1 = Ys[ii];

                    Tsxn_2 = Tsxn_1; Tsxn_1 = Tsxn; Tsxn = Tsx;
                    Ts[ii] = bn1*Tsxn + bn2*Tsxn_1 + bn3*Tsxn_2 - an2*Tsyn_1 - an3*Tsyn_2;
                    Tsyn_2 = Tsyn_1; Tsyn_1 = Ts[ii];
 */
                    ii++;
            }

/*  SCANS FIRST POSTURE OF TMR  */
            x_old = xorigin+(int)(X0*xconstant/12);
            y_old = yorigin-(int)(Y0*yconstant/12);

            kn = 1;
            Xr = Xs[kn];
            Yr = Ys[kn];
            Tr = Ts[kn];


            x_old_r = xorigin+(int)(Xr*xconstant/12);
            y_old_r = yorigin-(int)(Yr*yconstant/12);

            while( !kbhit() ) {

                    Get_Posture_Linkage(&th1,&th2,&th3,&Xc,&Yc,&Tc);
 /*                 Xcxn_2 = Xcxn_1; Xcxn_1 = Xcxn; Xcxn = Xcx;
                    Xc = bn1*Xcxn + bn2*Xcxn_1 + bn3*Xcxn_2 - an2*Xcyn_1 - an3*Xcyn_2;
                    Xcyn_2 = Xcyn_1; Xcyn_1 = Xc;

                    Ycxn_2 = Ycxn_1; Ycxn_1 = Ycxn; Ycxn = Ycx;
                    Yc = bn1*Ycxn + bn2*Ycxn_1 + bn3*Ycxn_2 - an2*Ycyn_1 - an3*Ycyn_2;
                    Ycyn_2 = Ycyn_1; Ycyn_1 = Yc;

                    Tcxn_2 = Tcxn_1; Tcxn_1 = Tcxn; Tcxn = Tcx;
                    Tc = bn1*Tcxn + bn2*Tcxn_1 + bn3*Tcxn_2 - an2*Tcyn_1 - an3*Tcyn_2;
                    Tcyn_2 = Tcyn_1; Tcyn_1 = Tc;
*/
                    sinT = sin(Tc);
                    cosT = cos(Tc);

                    if( (fscanf(f1,"%lf\n",&offset)) != EOF ) {

                            Xs[ii] = Xc + D*cosT - offset*sinT;
//                          Xsxn_2 = Xsxn_1; Xsxn_1 = Xsxn; Xsxn = Xsx;
//                          Xs[ii] = bn1*Xsxn + bn2*Xsxn_1 + bn3*Xsxn_2 - an2*Xsyn_1
                            - an3*Xsyn_2;
```

```
//                          Xsyn_2 = Xsyn_1; Xsyn_1 = Xs[ii];

                            Ys[ii] = Yc + D*sinT + offset*cosT;
//                          Ysxn_2 = Ysxn_1; Ysxn_1 = Ysxn; Ysxn = Ysx;
//                          Ys[ii] = bn1*Ysxn + bn2*Ysxn_1 + bn3*Ysxn_2 - an2*Ysyn_1 -
                            an3*Ysyn_2;
//                          Ysyn_2 = Ysyn_1; Ysyn_1 = Ys[ii];

                            Ts[ii] = aatan( Xs[ii-1], Ys[ii-1], Xs[ii], Ys[ii] );
                            if ( ii == 0 ) Ts[ii] = aatan( Xs[Ntt-1], Ys[Ntt-1], Xs[ii], Ys[ii] );
//                          Tsxn_2 = Tsxn_1; Tsxn_1 = Tsxn; Tsxn = Tsx;
//                          Ts[ii] = bn1*Tsxn + bn2*Tsxn_1 + bn3*Tsxn_2 - an2*Tsyn_1 -
                            an3*Tsyn_2;
//                          Tsyn_2 = Tsyn_1; Tsyn_1 = Ts[ii];

//                          fprintf(ff1,"%lf %lf %lf %lf %lf %lf %lf\n",
                            offset,Xc,Yc,Tc, Xs[ii], Ys[ii], Ts[ii]);
                    }
                    else Xs[ii] = CRACK_END;

                    ii++;
                    if (ii >= Ntt ) ii = 0;

/*********** Searching reference values ****************************/

                    if (Xs[kn] == CRACK_END) {
                            wli = 0; wri = 0;
                            Send_Velocity_Commands(wli,wri);
                            fclose(ft);
                            fclose(f1);
                            fclose(ft2);
                            return(1);
                    }

                    Xr = Xs[kn];
                    Yr = Ys[kn];
                    Tr = Ts[kn];
                    rrd = rr[kn];

                    kn++;
                    if ( kn >= Ntt ) kn = 0;

/************************************************************************/

            /* Calculate error posture in body coordinate */
                    x = (Xr - Xc)*cosT + (Yr - Yc)*sinT;
                    y = -(Xr - Xc)*sinT + (Yr - Yc)*cosT;
                    p = Tr - Tc;

                    invE[0][0] = -1;
                    invE[0][1] = -y/(e+x+c);
                    invE[1][0] = 0;
                    invE[1][1] = -1/(e+x+c);
                    f[0] = ur*cos(Tc);
```

```
                f[1] = ur*sin(Tc) + c*rrd;

                z = y + c*p;
                u1[0] = -f[0] - K[0][0] * x -K[0][1] * z;
                u1[1] = -f[1] - K[1][0] * x -K[1][1] * z;
                u[0] = invE[0][0] * u1[0] + invE[0][1] * u1[1];
                u[1] = invE[1][0] * u1[0] + invE[1][1] * u1[1];
                wl = ( u[0] - T2*u[1] ) / R * 9.55;
                wr = ( u[0] + T2*u[1] ) / R * 9.55;

                wli = (int)wl; wri = (int)wr; wri = -wri;

                Send_Velocity_Commands(wli,wri);

                x_new = xorigin+(int)(Xc*xconstant/12);
                y_new = yorigin-(int)(Yc*yconstant/12);
                x_new_r = xorigin+(int)(Xr*xconstant/12);
                y_new_r = yorigin-(int)(Yr*yconstant/12);

/*  DISPAYS X AND Y COORDINATES AND THETA  */
                _settextcolor(white);
                _settextposition(3,10);
                sprintf(buffer2,"x: %lf in       y: %lf in         theta: %lf degrees",
                Xc, Yc, Tc*180/pi);
                _outtext(buffer2);

/*  DRAWS REFERENCE PATH     */
                _setcolor(green);
                _moveto(x_old_r, y_old_r);
                _lineto(x_new_r, y_new_r);

/*  DRAWS TMR PATH  */
                _setcolor(lt_magenta);
                _moveto(x_old, y_old);
                _lineto(x_new, y_new);

                x_old = x_new;
                y_old = y_new;

                x_old_r = x_new_r;
                y_old_r = y_new_r;

                ticksnow = clock();
                tused = (double) ticksnow / CLK_TCK;
                fprintf(ft,"%lf %lf %lf %lf %lf %lf %lf %lf %lf\n",Xc,Yc,Tc,Xr,Yr,Tr,x,y,p);
                fprintf(ft2,"%lf\n",tused);

        }

        wli = 0; wri = 0;
        Send_Velocity_Commands(wli,wri);
        fclose(ft);
        fclose(f1);
        return(1);
```

```
}

int       Tracking_Control_Laser_real(void) /* tracking control mode with laser sensor */
{
          int            wli, wri;
          double         th1,th2,th3, Xr,Yr,Tr, ur, Xc,Yc,Tc, x,y,p, X0,Y0,T0;
          double         e, c, T2, R;
          double         invE[2][2], f[2], K[2][2], z, u[2], u1[2], wl, wr;
          char           sf[20];
          char           buffer1[100], buffer2[100];

          int            j,k, xorigin, yorigin;
          int            left_corner_x, left_corner_y, right_corner_x, right_corner_y;
          int            x_old, x_new, y_old, y_new, x_old_r, y_old_r, x_new_r, y_new_r;
          int            hor_lines, ver_lines, line_start_x, line_start_y;
          char           xvalue, yvalue, thetavalue;
          double         angular_pos, x_pos, y_pos;

          double         tused;
          clock_t        ticksnow;

          FILE           *f1, *ft, *ft2, *ff1;

          int            nd, i, ii, kn, kn1, rind, found_ref_value, status = 0;
          int            Ntt;
          double         D, offset, ds, sinT, cosT;
          double         rrd, dt, xnd, ynd, Ts0, Xs0, Ys0, ndf, di;

          double         Xcx, Xcxn, Xcxn_1, Xcxn_2, Xcyn_1, Xcyn_2;
          double         Ycx, Ycxn, Ycxn_1, Ycxn_2, Ycyn_1, Ycyn_2;
          double         Tcx, Tcxn, Tcxn_1, Tcxn_2, Tcyn_1, Tcyn_2;
          double         Xsx, Xsxn, Xsxn_1, Xsxn_2, Xsyn_1, Xsyn_2;
          double         Ysx, Ysxn, Ysxn_1, Ysxn_2, Ysyn_1, Ysyn_2;
          double         Tsx, Tsxn, Tsxn_1, Tsxn_2, Tsyn_1, Tsyn_2;
          double         an2, an3;
          double         bn1, bn2, bn3;

          float          tolerance;
          float          avg;
          float          offs;
          int            run_status = END_PROG;
          char           reply[10];

#ifdef FILE_OUTPUT
          char           out_file[32];

          if ( s_laser_init != 1) {
                    printf("Enter the name of the output file>>\n\t");
                    scanf("%s", out_file);

                    data_ptr = fopen(out_file, "w");        /* sets pointer to output file    */
                    off_set_ptr = fopen("off_set.out", "w");
```

```
#ifdef DEBUG
                    lisa = fopen("lisa.out", "w");
                    w = fopen("widths.out", "w");
#endif

// if fopen returns a NULL then there was an error opening the        */
// file and the program is exited          */
                    if (data_ptr == (FILE *)NULL)
                    {
                            printf("ERROR OPENING %s", "output file");
                            exit(-1);
                    }

#endif /* FILE_OUTPUT */

                    do
                    {
                            check_start(&run_status);
                    }while (run_status = END_PROG);

                    init(&tolerance, &avg);
//                  printf("\n\nTolerance Value: %f,      Average Value: %f\n\n",tolerance,avg);
//                  printf("Enter New Tolerance Value:\t");
//                  scanf("%s", reply);
//                  tolerance = atof(reply);
                    s_laser_init = 1;

          }

          an2 = -0.4803; an3 = 0.2127;
          bn1 = 0.1831; bn2 = 0.3662; bn3 = 0.1831;

          e = 0; c = 1.2; T2 = 12.5; R = 5.0;
          Ntt = 2000; D = 9.75;

          for (i = 0;i < Ntt;i++) {
                    Xs[i] = 0.0;
                    Ys[i] = 0.0;
                    Ts[i] = 0.0;
                    rr[i] = 0.0;
          }

//        if ( (f1 = fopen("c:\\hong\\tmr\\control\\ref_off.dat","r")) == NULL ) {
//                  printf("\nError! Reference offset data file can not be opened");
//                  return(0);
//        }

          printf("Enter the name of the output file>>\n\t");
          scanf("%s", out_file);

          ff1 = fopen(out_file, "w");      /* sets pointer to output file    */

          ft = fopen("c:\\hong\\tmr\\control\\traj.out","w");
          ft2 = fopen("c:\\hong\\tmr\\control\\rt.out","w");
```

```
		/*Set linear speed*/
		printf("\nInput linear speed( inch/sec ): ");
		scanf("%lf",&ur);
		/*Set sampling time*/
		printf("\nInput sampling time( sec ): ");
		scanf("%lf",&dt);

		/* Set control gain matrix  */
		printf("\nInput control gains( K(1,1),K(1,2),K(2,1),K(2,2)): ");
		scanf("%lf %lf %lf %lf",&K[0][0],&K[0][1],&K[1][0],&K[1][1]);

		ds = ur*dt;

		Get_Posture_Linkage(&th1,&th2,&th3,&X0,&Y0,&T0);

		Xcxn = X0; Xcxn_1 = X0; Xcxn_2 = X0; Xcyn_1 = X0; Xcyn_2 = X0;
		Ycxn = Y0; Ycxn_1 = Y0; Ycxn_2 = Y0; Ycyn_1 = Y0; Ycyn_2 = Y0;
		Tcxn = T0; Tcxn_1 = T0; Tcxn_2 = T0; Tcyn_1 = T0; Tcyn_2 = T0;

		_setvideomode( _VRES16COLOR );
		set_workspace();

	/*  MAKES GRIDLINES FOR WORKSPACE  */
		xorigin = 9+no_vert_lines/2*xconstant;
		yorigin = 65+(no_horiz_lines-2)*yconstant;
		mark_origin(xorigin, yorigin);

	//	fscanf(f1,"%lf\n",&offset);

		wait_for_profile();
		find_clean_crack(tolerance,avg,&offs);
		offset = (double) offs;

		Xs0 = X0 + D*cos(T0) - offset*sin(T0);
		Ys0 = Y0 + D*sin(T0) + offset*cos(T0);
		Ts0 = T0 + atan(offset/D);

		nd = D / ds;
		ndf = (double) nd;
		xnd = (Xs0 - X0) / ndf;
		ynd = (Ys0 - Y0) / ndf;
		ii = 0;
		for(i = 0;i < nd;i++) {
			di = (double) i;
			Xs[i] = X0 + xnd*di;
			Ys[i] = Y0 + ynd*di;
			Ts[i] = Ts0;
			ii++;
		}

	/*  SCANS FIRST POSTURE OF TMR  */
		x_old = xorigin+(int)(X0*xconstant/12);
		y_old = yorigin-(int)(Y0*yconstant/12);
```

```
		kn = 1;
		Xr = Xs[kn];
		Yr = Ys[kn];
		Tr = Ts[kn];

/*		Xsxn = Xs[ii-1]; Xsxn_1 = Xs[ii-2]; Xsxn_2 = Xs[ii-3];
		Xsyn_1 = Xs[ii-1]; Xsyn_2 = Xs[ii-2];
		Ysxn = Ys[ii-1]; Ysxn_1 = Ys[ii-2]; Ysxn_2 = Ys[ii-3];
		Ysyn_1 = Ys[ii-1]; Ysyn_2 = Ys[ii-2];
		Tsxn = Ts[ii-1]; Tsxn_1 = Ts[ii-2]; Tsxn_2 = Ts[ii-3];
		Tsyn_1 = Ts[ii-1]; Tsyn_2 = Ts[ii-2];
*/
		x_old_r = xorigin+(int)(Xr*xconstant/12);
		y_old_r = yorigin-(int)(Yr*yconstant/12);

		while( !kbhit() ) {

			Get_Posture_Linkage(&th1,&th2,&th3,&Xc,&Yc,&Tc);
/*			Xcxn_2 = Xcxn_1; Xcxn_1 = Xcxn; Xcxn = Xcx;
			Xc = bn1*Xcxn + bn2*Xcxn_1 + bn3*Xcxn_2 - an2*Xcyn_1 - an3*Xcyn_2;
			Xcyn_2 = Xcyn_1; Xcyn_1 = Xc;

			Ycxn_2 = Ycxn_1; Ycxn_1 = Ycxn; Ycxn = Ycx;
			Yc = bn1*Ycxn + bn2*Ycxn_1 + bn3*Ycxn_2 - an2*Ycyn_1 - an3*Ycyn_2;
			Ycyn_2 = Ycyn_1; Ycyn_1 = Yc;

			Tcxn_2 = Tcxn_1; Tcxn_1 = Tcxn; Tcxn = Tcx;
			Tc = bn1*Tcxn + bn2*Tcxn_1 + bn3*Tcxn_2 - an2*Tcyn_1 - an3*Tcyn_2;
			Tcyn_2 = Tcyn_1; Tcyn_1 = Tc;
*/
			sinT = sin(Tc);
			cosT = cos(Tc);

			wait_for_profile();
			find_clean_crack(tolerance,avg,&offs);
			offset = (double) offs;


			if ( offset != NO_CRACK  ) {

				Xs[ii] = Xc + D*cosT - offset*sinT;
//				Xsxn_2 = Xsxn_1; Xsxn_1 = Xsxn; Xsxn = Xsx;
//				Xs[ii] = bn1*Xsxn + bn2*Xsxn_1 + bn3*Xsxn_2 - an2*Xsyn_1 -
				an3*Xsyn_2;
//				Xsyn_2 = Xsyn_1; Xsyn_1 = Xs[ii];

				Ys[ii] = Yc + D*sinT + offset*cosT;
//				Ysxn_2 = Ysxn_1; Ysxn_1 = Ysxn; Ysxn = Ysx;
//				Ys[ii] = bn1*Ysxn + bn2*Ysxn_1 + bn3*Ysxn_2 - an2*Ysyn_1 -
				an3*Ysyn_2;
//				Ysyn_2 = Ysyn_1; Ysyn_1 = Ys[ii];

				Ts[ii] = aatan( Xs[ii-1], Ys[ii-1], Xs[ii], Ys[ii] );
```

```
                    if ( ii == 0 ) Ts[ii] = aatan( Xs[Ntt-1], Ys[Ntt-1], Xs[ii], Ys[ii] );
//                  Tsxn_2 = Tsxn_1; Tsxn_1 = Tsxn; Tsxn = Tsx;
//                  Ts[ii] = bn1*Tsxn + bn2*Tsxn_1 + bn3*Tsxn_2 - an2*Tsyn_1 -
                    an3*Tsyn_2;
//                  Tsyn_2 = Tsyn_1; Tsyn_1 = Ts[ii];

                    fprintf(ff1,"%lf %lf %lf %lf %lf %lf %lf\n", offset,Xc,Yc,Tc, Xr, Yr, Tr);
            }
            else Xs[ii] = CRACK_END;

            ii++;
            if (ii >= Ntt ) ii = 0;

/*********** Searching reference values ****************************/

            if (Xs[kn] == CRACK_END) {
                    wli = 0; wri = 0;
                    Send_Velocity_Commands(wli,wri);
                    fclose(ft);
//                  fclose(f1);
                    fclose(ft2);
                    printf("Stop at crack end!!!!!!!!!");
                    return(1);
            }

            Xr = Xs[kn];
            Yr = Ys[kn];
            Tr = Ts[kn];
            rrd = rr[kn];

            kn++;
            if ( kn >= Ntt ) kn = 0;

/**************************************************************************/

        /* Calculate error posture in body coordinate */
            x = (Xr - Xc)*cosT + (Yr - Yc)*sinT;
            y = -(Xr - Xc)*sinT + (Yr - Yc)*cosT;
            p = Tr - Tc;

            invE[0][0] = -1;
            invE[0][1] = -y/(e+x+c);
            invE[1][0] = 0;
            invE[1][1] = -1/(e+x+c);
            f[0] = ur*cos(Tc);
            f[1] = ur*sin(Tc) + c*rrd;

            z = y + c*p;
            u1[0] = -f[0] - K[0][0] * x -K[0][1] * z;
            u1[1] = -f[1] - K[1][0] * x -K[1][1] * z;
            u[0] = invE[0][0] * u1[0] + invE[0][1] * u1[1];
            u[1] = invE[1][0] * u1[0] + invE[1][1] * u1[1];
            wl = ( u[0] - T2*u[1] ) / R * 9.55;
            wr = ( u[0] + T2*u[1] ) / R * 9.55;
```

```
                wli = (int)wl; wri = (int)wr; wri = -wri;

                Send_Velocity_Commands(wli,wri);

                x_new = xorigin+(int)(Xc*xconstant/12);
                y_new = yorigin-(int)(Yc*yconstant/12);
                x_new_r = xorigin+(int)(Xr*xconstant/12);
                y_new_r = yorigin-(int)(Yr*yconstant/12);

/*  DISPAYS X AND Y COORDINATES AND THETA  */
                _settextcolor(white);
                _settextposition(3,10);
                sprintf(buffer2,"x: %lf in       y: %lf in         theta: %lf degrees",
                Xc, Yc, Tc*180/pi);
                _outtext(buffer2);

/*  DRAWS REFERENCE PATH     */
                _setcolor(green);
                _moveto(x_old_r, y_old_r);
                _lineto(x_new_r, y_new_r);

/*  DRAWS TMR PATH  */
                _setcolor(lt_magenta);
                _moveto(x_old, y_old);
                _lineto(x_new, y_new);

                x_old = x_new;
                y_old = y_new;

                x_old_r = x_new_r;
                y_old_r = y_new_r;

                ticksnow = clock();
                tused = (double) ticksnow / CLK_TCK;
                fprintf(ft,"%lf %lf %lf %lf %lf\n",x,y,p,u[0],u[1]);
                fprintf(ft2,"%lf\n",tused);

        }

        wli = 0; wri = 0;
        Send_Velocity_Commands(wli,wri);
        fclose(ft);
        fclose(f1);
        return(1);

}

void Send_Velocity_Commands(int wl, int wr)  /* send velocity commands to the motor drive */
{
        int      tmp_read;

        if (wl > S_limit)        wl = S_limit;
        if (wl < -S_limit)       wl = -S_limit;
```

```
        if (wr > S_limit)       wr = S_limit;
        if (wr < -S_limit)      wr = -S_limit;

        do {
                outpw(FSC_vel_r_r,wl);
                tmp_read = inpw(FSC_vel_r_r);
        }while(tmp_read != wl);
        do {
                outpw(FSC_vel_r_r_1,wr);
                tmp_read = inpw(FSC_vel_r_r_1);
        }while(tmp_read != wr);
}


void set_workspace(void)
{
        _setbkcolor (_BLACK);
        _clearscreen(_GCLEARSCREEN);
        _settextcolor(white);
        _settextposition(1,30);
        _outtext ("WORKSPACE OF TMR");
        create_replace_grid();
}

/*  MARKS ORIGIN  */
void mark_origin(int xorigin, int yorigin)
{
        xorigin = 9+no_vert_lines/2*xconstant;
        yorigin = 65+(no_horiz_lines-2)*yconstant;
        _setcolor(white);
        _setlinestyle(solid);
        _moveto(xorigin-xconstant, yorigin);
        _lineto(xorigin+xconstant, yorigin);
        _moveto( xorigin,yorigin-yconstant);
        _lineto( xorigin,yorigin+yconstant);
}

/*  REPLACES PARITALLY LOST GRID AND ORIGIN MARKER  */
void create_replace_grid(void)
{
        int i;
        _setcolor(cyan);
        _rectangle(_GBORDER, 9, 65, 639, 450);
        _settextcolor(white);
        _settextposition(13,0);
        _outtext ("60\n\n\n\n\n\n\n\n\n\n\n0");
        _settextposition(30,18);
        _outtext ("-60  0          60");

/*  HORIZONTAL LINES  */
        for(i = 1; i< no_horiz_lines; i++)
        {
                _setcolor(red);
                _setlinestyle(dashed);
```

```
                _moveto(9, 65+i*yconstant);
                _lineto (639, 65+i*yconstant);
        }

/*  VERITICAL LINES  */
        for(i = 1; i< no_vert_lines; i++)
        {
                _setcolor(red);
                _setlinestyle(dashed);
                _moveto(9+i*xconstant, 65);
                _lineto (9+i*xconstant, 450);
        }

/*  DRAWS HALF-CIRCLE  */
        _setcolor(yellow);
        _arc(79,135,569,625,569,380,79,380);
}

void draw_tmr(double x, double y, double angular_pos, int xorigin, int yorigin, int x_old, int
x_new, int y_old, int y_new)
{
        int i, j;
        extern int x_pos, y_pos, x0, x1, x2, x3, x4, yy0, yy1, y2, y3, y4;
        extern int tx1, tx2, tx3, tx4, tx5, tx6, tx7, tx8;
        extern int ty1, ty2, ty3, ty4, ty5, ty6, ty7, ty8;
        float tmrlength, tmrwidth, halflength, halfwidth, tirelength, tirewidth;
        char buffer1[100], buffer2[100];

        tmrlength = 30.0/12;
        tmrwidth = 20.0/12;
        halfwidth = tmrwidth/2;
        halflength = tmrlength/2;
        tirelength = 10.5/12;
        tirewidth = 3.5/12;

/*      REDRAWS THE TMR IN BLACK TO HIDE IT     */
        _setcolor(black);
        _moveto(x1, yy1);
        _lineto(x2, y2);
        _lineto(x3, y3);
        _lineto(x4, y4);
        _lineto(x1, yy1);
        _moveto(tx1,ty1);
        _lineto(tx2, ty2);
        _lineto(tx3, ty3);
        _lineto(tx4, ty4);
        _lineto(tx1, ty1);
        _moveto(tx5,ty5);
        _lineto(tx6, ty6);
        _lineto(tx7, ty7);
        _lineto(tx8, ty8);
        _lineto(tx5, ty5);
        _ellipse(_GBORDER, (int)(x_pos-4), (int)(y_pos+4), (int)(x_pos+4), (int)(y_pos-4));
```

```
        x_pos = xorigin+(int)(x/12*xconstant);
        y_pos = yorigin-(int)(y/12*yconstant);

/*  DISPAYS X AND Y COORDINATES AND THETA  */
        _settextcolor(white);
        _settextposition(3,10);
        sprintf(buffer2,"x: %lf in        y: %lf in          theta: %lf degrees",
        x, y, angular_pos*180/pi);
        _outtext(buffer2);

/*  CALCULATES THE CORNER COORDINATES OF THE TMR  */
        x0 = (int)(x_pos + xconstant*cga*cos(angular_pos));
        yy0 = (int)(y_pos - yconstant*cga*sin(angular_pos));
        x1 = (int)(x0 + xconstant*halfwidth*sin(angular_pos));
        yy1 = (int)(yy0 + yconstant*halfwidth*cos(angular_pos));
        x2 = (int)(x1 - xconstant*tmrlength*cos(angular_pos));
        y2 = (int)(yy1 + yconstant*tmrlength*sin(angular_pos));
        x3 = (int)(x2 - xconstant*tmrwidth*sin(angular_pos));
        y3 = (int)(y2 - yconstant*tmrwidth*cos(angular_pos));
        x4 = (int)(x1 - xconstant*tmrwidth*sin(angular_pos));
        y4 = (int)(yy1 - yconstant*tmrwidth*cos(angular_pos));

/*  CALCULATES THE TMR'S TIRE COORDINATES */
        tx1 = (int)(x3 - 5*sin(angular_pos));
        ty1 = (int)(y3 - 5*cos(angular_pos));
        tx2 = (int)(tx1 - xconstant*tirewidth*sin(angular_pos));
        ty2 = (int)(ty1 - yconstant*tirewidth*cos(angular_pos));
        tx3 = (int)(tx2 + xconstant*tirelength*cos(angular_pos));
        ty3 = (int)(ty2 - yconstant*tirelength*sin(angular_pos));
        tx4 = (int)(tx1 + xconstant*tirelength*cos(angular_pos));
        ty4 = (int)(ty1 - yconstant*tirelength*sin(angular_pos));
        tx5 = (int)(x2 + 5*sin(angular_pos));
        ty5 = (int)(y2 + 5*cos(angular_pos));
        tx6 = (int)(tx5 + xconstant*tirewidth*sin(angular_pos));
        ty6 = (int)(ty5 + yconstant*tirewidth*cos(angular_pos));
        tx7 = (int)(tx6 + xconstant*tirelength*cos(angular_pos));
        ty7 = (int)(ty6 - yconstant*tirelength*sin(angular_pos));
        tx8 = (int)(tx5 + xconstant*tirelength*cos(angular_pos));
        ty8 = (int)(ty5 - yconstant*tirelength*sin(angular_pos));

/*  DRAWS TMR  */
        _setcolor(lt_blue);
        _moveto(x1, yy1);
        _lineto(x2, y2);
        _lineto(x3, y3);
        _lineto(x4, y4);
        _lineto(x1, yy1);
//      _setcolor(blue);
        _moveto(tx1,ty1);
        _lineto(tx2, ty2);
        _lineto(tx3, ty3);
        _lineto(tx4, ty4);
        _lineto(tx1, ty1);
        _moveto(tx5,ty5);
```

```
        _lineto(tx6, ty6);
        _lineto(tx7, ty7);
        _lineto(tx8, ty8);
        _lineto(tx5, ty5);
        _setcolor(green);
        _ellipse(_GBORDER, (int)(x_pos-4), (int)(y_pos+4), (int)(x_pos+4), (int)(y_pos-4));
        gcvt(x, 7, buffer1);     /* gcvt converts a double to a string */
        gcvt(y, 7, buffer1);
        gcvt(angular_pos, 7, buffer1);

/* DRAWS TMR PATH */
        _setcolor(lt_magenta);
        _moveto(x_old, y_old);
        _lineto(x_new, y_new);

/* ADDING TO DISPLAY TIME */
 /*     for(i = 0; i<20000; i++)
        {
                for(j = 0; j<25; j++);
        } */


/* REPLACES THE ENTIRE GRID AND ORIGIN */
        create_replace_grid();
        mark_origin(xorigin, yorigin);
}

void sound_on(unsigned freq)
{
        unsigned status, ratio, part_ratio;

        status = inp(OUT_8255);
        outp(TIMER_MODE,TIMER_OSC);
        ratio = (unsigned)(TIMER_FREQ/freq);
        part_ratio = ratio & 0xff;
        outp(TIMER_COUNT,part_ratio);
        part_ratio = (ratio >> 8) & 0xff;
        outp(TIMER_COUNT,part_ratio);
        outp(OUT_8255,(status | SPKRON));
}

void sound_off(void)
{
        unsigned status;
        status = inp(OUT_8255);
        outp(OUT_8255,(status & ~SPKRON));
}

double aatan(double x1, double y1, double x2, double y2)
{
        double dx, dy, ang, dydx;

        dx = x2 - x1;
        dy = y2 - y1;
```

```
		if ( dx == 0 ) {
			if ( dy > 0 ) ang = PI/2;
			else if ( dy < 0 ) ang = 3*PI/2;
			else ang = 1000;
		}

		else if ( dx > 0 ) {
			if ( dy == 0 ) ang = 0;
			else if ( dy > 0 ) {
				dydx = dy/dx;
				if ( dydx > 100000 ) ang = PI/2;
				else ang = atan(dydx);
			}
			else {
				dydx = dy/dx;
				if ( dydx < -100000 ) ang = 3*PI/2;
				else ang = 2*PI - atan(-dydx);
			}
		}

		else {
			if ( dy == 0 ) ang = PI;
			else if ( dy > 0 ) {
				dydx = dy/dx;
				if ( dydx < -100000 ) ang = PI/2;
				else ang = PI - atan(-dydx);
			}
			else {
				dydx = dy/dx;
				if ( dydx > 100000 ) ang = 3*PI/2;
				else ang = PI + atan(dydx);
			}
		}
		return(ang);
	}
```

```
/*******************************************************************/
/* File Name         FSCreg.h                                       */
/* Function          Interface Memory Map for FSC Registers         */
/* Programed by      Dahie Hong                                     */
/* Date              Aug., 30, 1993                                 */
/* Version           1.0                                            */
/* Revision History  None                                           */
/*******************************************************************/

/*****************************************************************/
/*      Registers in CPU RW, FSC R         */
/*      Addr   0x00 - 0x7f    */
/*      FSC    read    */
/*      External        read & write    */
/*****************************************************************/
#define FSC_index              0x320  // index register
#define FSC_data               0x322  //
#define FSC_kode               0x322  //
#define FSC_address            0x324  //
#define FSC_wait_or_go         0x326  // control flag from host
#define FSC_init_rotvel        0x720  // initial rotational speed from host
#define FSC_pwmmax             0x722  // max value of pwm
#define FSC_pwmmin             0x724  // min value of pwm
#define FSC_q_pwm_range        0x726   //
#define FSC_pwm_period_2       0xb20  // half pwm period
#define FSC_torque_angle       0xb24  // torque angle
#define FSC_kpl                0xb26  // proportional gain of position control
#define FSC_vrlimit            0xf20   // velocity limit
#define FSC_b_p_h_r1           0xf22  // position reference low word
#define FSC_b_p_l_r1           0xf24  // position reference high word
#define FSC_vel_max            0xf26  //
#define FSC_xvel_max           0x1320         //
#define FSC_b_p_h_r2           0x1322         //
#define FSC_b_p_l_r2           0x1324         //
#define FSC_b_p_h_r3           0x1326         //
#define FSC_b_p_l_r3           0x1720         //
#define FSC_pwmst              0x1722         //

#define FSC_kc1_r              0x3320
#define FSC_kc2_r              0x3322
#define FSC_i_q_r_r            0x3324
#define FSC_kv1_r              0x3326
#define FSC_kv2_r              0x3720
#define FSC_vel_r_r            0x3722
#define FSC_vel_r_r_1          0x3724
#define FSC_pwm_onoff          0x3726

#define FSC_act_node           0x7b24         //
#define FSC_w_data1            0x7b26         //  write data from host
#define FSC_rw_addr1           0x7f20         //  address of read data
#define FSC_w_data2            0x7f22         //  write data from host
#define FSC_rw_addr2           0x7f24         //  address of read or write data
#define FSC_cmd                0x7f26         //  cpu intervention command from host
```

```
/****************************************************************/
/*      Registers in CPU R, FSC RW          */
/*      Address         0x80 - 0xff    */
/*      FSC     read & write    */
/*      External        read    */
/****************************************************************/
#define FSC_h0              0x8320          //
#define FSC_l0              0x8322          //
#define FSC_h1              0x8324          //
#define FSC_l1              0x8326          //
#define FSC_h2              0x8720          //
#define FSC_l2              0x8722          //

#define FSC_error           0x8320          //
#define FSC_zeros           0x8322          //
#define FSC_fives           0x8322          //
#define FSC_aes             0x8324          //
#define FSC_mon_type        0x8324          // monitor type
#define FSC_firstaddr       0x8326          // first address used in memory test routine
#define FSC_lastaddr                0x8720          // last address used in memory test routine

#define FSC_extromaddr      0x8724          //
#define FSC_codelow         0x8726          //
#define FSC_codehigh        0x8b20          //
#define FSC_cst200h                 0x8b22          //
#define FSC_bit10           0x8b24          // content is 0000 0100 0000 0000B
#define FSC_answer          0x8b26          //
#define FSC_polefactor      0x8f20          //
#define FSC_theta           0x8f22          // angle
#define FSC_sin             0x8f24          // sin(theta)
#define FSC_cos             0x8f26          // cos(theta)
#define FSC_sin120          0x9320          // sin(theta+120)
#define FSC_cos120          0x9322          // cos(theta+120)
#define FSC_kslip           0x9324          //
#define FSC_slipinc         0x9326          //
#define FSC_sliplow         0x9720          //
#define FSC_sliphigh        0x9722          //
#define FSC_i_u             0x9724          // u-axis current
#define FSC_i_v             0x9726          // v-axis current
#define FSC_i_q             0x9b20          // q-axis current
#define FSC_i_d             0x9b22          // d-axis current
#define FSC_v_q             0x9b24          // q-axis velocity
#define FSC_v_d             0x9b26          // d-axis velocity
#define FSC_pwmu            0x9f20          // u phase pwm
#define FSC_pwmv            0x9f22          // v phase pwm
#define FSC_pwmw            0x9f24          // w phase pwm
#define FSC_pwmoffset       0x9f26          // pwm offset
#define FSC_pfrequency      0xa320          // pwm frequency
#define FSC_pdelay          0xa322          // pwm delay
#define FSC_i_q_r           0xa324          // reference of q-axis current
#define FSC_i_q_e           0xa326          // error of q-axis current
#define FSC_i_q_int         0xa720          // buffer for q-axis current PI control
#define FSC_vqlimit         0xa722          // limit of q-axis velocity
#define FSC_kc1             0xa724          // proportional gain of current control
```

```
#define FSC_kc2            0xa726        // integral gain of current control
#define FSC_i_d_r          0xab20        // reference of d-axis current
#define FSC_i_d_e          0xab22        // error of d-axis current
#define FSC_i_d_int        0xab24        // buffer for d-axis current PI control
#define FSC_vdlimit        0xab26        // limit of d-axis velocity
#define FSC_vel_r          0xaf20        // reference velocity
#define FSC_vel_e          0xaf22        // velocity error
#define FSC_vel_int        0xaf24        // buffer for velocity PI control
#define FSC_iqlimit        0xaf26        // limit of q-axis current
#define FSC_kv1            0xb320        // proportional gain of velocity control
#define FSC_kv2            0xb322        // integral gain of velocity control
#define FSC_pos            0xb324        // new value of position counter(encoder)
#define FSC_pos1           0xb326        // old value of position counter(encoder)
#define FSC_increment      0xb720        //
#define FSC_countmax       0xb722        // max position counter value of encoder
#define FSC_countmax2      0xb724        // half of counter max
#define FSC_xcountmax2     0xb726        // - countermax2
#define FSC_vel            0xbb20        // feedback velocity
#define FSC_velfactor      0xbb22        // velocity factor

#define FSC_adc0           0xbb24        // ADC #define0 result
#define FSC_adc1           0xbb26        // ADC #define1 result
#define FSC_i_u1           0xbf20        // first  filtered value of u-axis current
#define FSC_i_u2           0xbf22        // second filtered value of u-axis current
#define FSC_i_u3           0xbf24        // third  filtered value of u-axis current
#define FSC_i_u4           0xbf26         // fourth filtered value of u-axis current
#define FSC_i_v1           0xc320        // first  filtered value of v-axis current
#define FSC_i_v2           0xc322        // second filtered value of v-axis current
#define FSC_i_v3           0xc324        // third  filtered value of v-axis current
#define FSC_i_v4           0xc326        // fourth filtered value of v-axis current

#define FSC_rot_field_angle          0xc720          // rotational field angle
#define FSC_pos_p          0xc722        //
#define FSC_pos1_p         0xc724        //
#define FSC_increment_p    0xc726        //
#define FSC_z_encountered  0xcb20        // 0 indicates that z pulse is encountered
#define FSC_current_time   0xcb22        // record timer reading evevry current loop

#define FSC_p_e            0xfb24        // position error
#define FSC_p_l_e          0xfb26        // position error low word
#define FSC_p_h_e          0xff20        // position error high word
#define FSC_status         0xff22         // status of cpu intervention
#define FSC_r_data1        0xff24        // receive data1
#define FSC_r_data2        0xff26        // receive data2
```

```
/**************************************************************** */
/* File Name          laser.c                                           */
/* Function           contains functions related to the laser range finding sensor */
/* Programed by       D.A. Krulewich, D. Hong, and L. Matsumoto        */
/* Date               Aug., 30, 1994                                    */
/**************************************************************** */
#include <signal.h>
#include <bios.h>

#include "calib8.h"
#include "profile.h"

/*#define PRINT_ERR*/
#define FILE_OUTPUT
/*#define DEBUG*/
#define TIMING_SAMP_INT

#define TOO_SMALL         1.0      /* width of crack in mm which is too small to seal */
#define TOO_BIG           50.0     /* width of crack in mm which is too large to seal */
#define N                 25
#define NOT_FOUND         0
#define FOUND             1
#define CLEAN             0
#define VEGETATION        1
#define YES               1
#define NO                0
#define INTENSITY         9        /* intensity of laser (1-9) */
#define START_PROG        1
#define END_PROG          0
#define FS_MM             101.6
#define NUM_PTS           nb_line_field  /* number of points in each scan line */
#define FS_BITS           126.0  /* full scale number of bits */
#define SEND_DATA         26
#define NO_CRACK                   1000.0
#define MAX_ERR           (FS_MM/2.4) /* Maximum error for saturation */
#define MAX_NC_SAMPS      330      /* Maximum consecutive samples of
                                   no crack found before really sending
                                   the no crack signal to the robot */

/* define serial communication constants */
#define WORD_LENGTH       _COM_CHR8         /* 8 bits per character */
                                            /* _COM_CHR7 for 7 bits per character */
#define STOP_BITS         _COM_STOP1        /* 1 stop bit */
                                            /* _COM_STOP2 for 2 stop bits */
#define PARITY            _COM_ODDPARITY          /* odd parity */
                                            /* _COM_EVENPARITY for even parity */
                                            /* _NO_PARITY for no parity */
#define BAUD_RATE         _COM_4800  /*4800 baud */
                                   /*_COM_110 0      110 baud */
                                   /*_COM_150 32     150 baud */
                                   /*_COM_300 64     300 baud */
                                   /*_COM_600 96     600 baud */
                                   /*_COM_1200       128     1200 baud */
                                   /*_COM_240 160    2400 baud */
```

```
                                        /*_COM_4800         192     4800 baud */
                                        /*_COM_9600         224     9600 baud */
                                        /* _COM_XXX for XXX baud */
#define COM1                    0       /* com1 port assignment */
#define COM2                    1       /* com2 port assignment */

/* set crack profile filter constants (position filter) */
#define A1                      1.0000
#define A2                      -1.5610
#define A3                      .6414
#define B1                      .0201
#define B2                      .0402
#define B3                      .0201

#define TIME_FILTER
#define SATURATION_CHECK

/* Set time filter constants */
/* fc = 4 Hz, fs = 33, n = 2, r = 0.5 */

#define AT1                     1.0000
#define AT2                     -0.83473496972638
#define AT3                     0.37065462937979
#define BT1                     0.13397991491335
#define BT2                     0.26795982982670
#define BT3                     0.13397991491335

/* fc = 1 Hz, fs = 33, n = 2, r = 0.5 */
/*
#define AT1                     1.0000
#define AT2                     -1.715154
#define AT3                     0.763242
#define BT1                     0.012022
#define BT2                     0.024044
#define BT3                     0.012022
*/

extern void     p_init_all(void);
extern void     wait_for_profile(void);
extern void     s_cam_to_user(F_COOR *ptC, CALIBRATION8 near *ptCal);
void    send_offset(float offset);
void    init_serial_port(void);
void    init(float *tolerance, float *avg);
void    menu2(void);
void    set_video(void);
void    restore_video(void);
int     crack_type(void);
void    find_clean_crack(float tolerance, float avg, float *offset);
void    find_filled_crack(float tolerance, float avg, float *offset);
float   filter(float x[], float y[]);
void    emergency_out(int sig);
void    last_call(void);
void    check_start(int *run_status);
void    check_stop(int *run_status);
```

```
void    send_not_found(void);

#ifdef FILE_OUTPUT
FILE *data_ptr, *off_set_ptr;
#endif

#ifdef DEBUG
FILE *lisa, *w;
#endif


void check_start(int *run_status)
{
        *run_status = START_PROG;
}

void check_stop(int *run_status)
{
        if (kbhit())
                *run_status = END_PROG;
        else
                *run_status = START_PROG;
}

void init_serial_port(void)
{
        unsigned data;
        data = (WORD_LENGTH | STOP_BITS | PARITY | BAUD_RATE);
        _bios_serialcom(_COM_INIT, COM1, data);
}

void send_offset(float offset)
{
        unsigned status;
/*      unsigned scaled_offset;*/
/*      char scaled_offset;*/
        int scaled_offset;
        char scaled_output;
        static int i = 0;
        int data;
        int send = NO;
        char *error;
        char out_error[11];
        int dec, sign;
        int count = 6;
        static float t2=0, t4=0;
        float t1, t3;
        float filt_offset;
        float raw_offset;
        static int no_crack_samps = 0;
        static int prev_offset_sign = 1;
        int filter_it;
        int nc = 0;
```

```
        filter_it = 1;    /* Do filter the signal this time through */
        nc = 0;

#ifdef SATURATION_CHECK
        if((offset > MAX_ERR) || (offset < -MAX_ERR)) {
                                        /* Should indicate "No Crack Found" */
                if(no_crack_samps < MAX_NC_SAMPS){  /* For now, send out saturated
                                                    value */
                        offset = MAX_ERR * prev_offset_sign;
                        no_crack_samps++;
                        nc = 0;
                }
                else {
                        offset = 1000.0; /* Send out No Crack Found. Don't filter this */
                        filter_it = 0;
                        nc = 1;
                }
                }
        else {  /* The sensor did find a crack */
                no_crack_samps = 0;
                nc = 0;
        }
#endif /* SATURATION_CHECK */

        raw_offset = offset;

#ifdef TIME_FILTER
        /* Filter the offset signal */
        /* Currently using a
        ripple factor of 0.5, cutoff frequency of 4 Hz, assumed
        sampling frequency of 33 Hz. Designed using Matlab's
        cheby1() function.  The filter structure being used is
        a canonical form (see Discret Time Signal Processing,
        Oppenheim and Schaffer) */

        if(filter_it){
                if(offset>=0)
                        prev_offset_sign = 1;
                else
                        prev_offset_sign = -1;

//      raw_offset = offset;

                /* Delay taps */
                t3 = t4;
                t1 = t2;

                filt_offset = BT1*offset + t1;
                t4 = BT3*offset - AT3*filt_offset;
                t2 = t3 + BT2*offset - AT2*filt_offset;

                /* Now, reset the input offset value to the filtered value */
                offset = filt_offset;
```

```
		}

	/******** End of Filtering ********************/
	#endif /* TIME_FILTER */

	#ifdef FANCY_OUT
		_settextposition(15,35);

		strcpy(error, fcvt((double)offset, count, &dec, &sign));

		out_error[1] = '\0';
		if (dec <= 0)
		{
			strncat(out_error, " ", 1);
			dec = 0;
		}
		else
		{
			strncat(out_error, error, 1);
		}
		strncat(out_error, ".", 1);
		strncat(out_error, (error + dec), 3);

		if (sign == 0)
			out_error[0] = ' ';
		else
			out_error[0] = '-';

		strncat(out_error, " mm", 3);

		_outtext(out_error);

	#endif /*FANCY_OUT*/

	#ifdef PRINT_ERR
		printf("\nError: %8.3f (mm), %8.3f (raw),",offset,raw_offset);
	#endif

	/*	check to see if "DATA READY" flag is set  */
		status = 0x100 & _bios_serialcom(_COM_STATUS, COM1, 0);

		if (status == 0x100)
		{
			while (status == 0x100)
			{
	/*	get data from serial port	*/
					data = 0xff & _bios_serialcom(_COM_RECEIVE, COM1, 0);
					if (data == SEND_DATA)
						send = YES;
					status = 0x100 & _bios_serialcom(_COM_STATUS, COM1, 0);
			}
			if (send)
			{
	/*	send error signal to RPS	*/
```

```
                              scaled_offset = (int) ((offset*2.0 ) * FS_BITS / FS_MM);

                              /* Convert the integer scaled_offset to the char scaled_output.
                                        If no crack is indicated, send out FS_BITS+1 */
                              if(nc){
                                        scaled_output = (char) (FS_BITS+1);          /* Indicate no crack
               found to RPS */
                                        }
                              else {
                                        if(scaled_offset > FS_BITS)
                                                  scaled_output = (char) FS_BITS;
                                        else if(scaled_offset < -FS_BITS)
                                                  scaled_output = (char) -FS_BITS;
                                        else
                                                  scaled_output = (char) scaled_offset;
                                        }

                              /*if (offset > FS_MM)
                                        scaled_offset = (FS_BITS + 1); */     // no crack found

                              /* wait until transmit holding register empty flag is set */
                              do
                              {
                                        status = 0x2000 & _bios_serialcom(_COM_STATUS, COM1, 0);
                              } while (status != 0x2000);

                              status = _bios_serialcom(_COM_SEND, COM1,(unsigned)
               scaled_output);

               #ifdef PRINT_ERR
                              printf("error: %4d",(int) scaled_offset);
               #endif
                              if ((status & 0x8000) == 0x8000)
                              {
                                        printf("\nError sending offset over serial port!!\n");
                              }
                         }
                    }


               #ifdef FILE_OUTPUT
                    fprintf(data_ptr, "%f %f\n", raw_offset, offset);
               #endif
               }

               void init(float *tolerance, float *avg)
               {
                    int i;

                    if (signal(SIGINT,emergency_out) == SIG_ERR)    /*trap ^C*/
                    {
                         perror("signal failed");
                         exit(0);
                    }
```

```
    atexit(last_call); /* the system will call last_call when the program terminates */

    p_trap_kb();    /*save actual key board fct so on return we restore it*/
    u_set_dir();    /*See II.3 Definition of proper DOS environment*/
    set_LPB_board(0); /*read if present descriptions of the LPB board*/
    p_int_reset();  /* resets interrupts 1-6 enable bit in CTRL1 rgister */
    p_int_init();   /*must be done once only*/
    global_init();  /*read camera parameters*/
    p_init_all();   /* initialize board */

    set_laser(INTENSITY);        /* set laser intensity */

    set_video();

    _setbkcolor(_BLUE);
    _clearscreen(_GCLEARSCREEN);
    _settextcolor((short)_WHITE);

    menu2();
}

void menu2(void)
{
    char dummy;

    _clearscreen(_GCLEARSCREEN);
    _outtext("*************************************************\n");
    _outtext("*     *\n");
    _outtext("*     LASER VISION SYSTEM    *\n");
    _outtext("*     *\n");
    _outtext("*************************************************\n");


    _outtext("\nInitialization complete.  Ready to begin sampling.\n");
    _outtext("Place sensor over section to scanned.\n");
    _outtext("Hit any key and enter to continue.>>");
    scanf("%s", &dummy);
}

int crack_type(void)
{
    return CLEAN;
}

float filter(float x[], float y[])
/* discrete realization for a recursive high pass filter */
{
    float vo;

    vo = B1 * x[2] + B2 * x[1] + B3 * x[0] - A2 * y[1] - A3 * y[0];

    return vo;
}
```

```
void find_clean_crack(float tolerance, float avg, float *offset)
{
        F_COOR f_coor;
        int i, j, max=0, min=0, m, mincell, maxcell;
        int crack_start, crack_end, rise;
        float pos[240], depth[240];
        float init_crack_depth, derivative[239], maxmin;
        float width;
        float vo;
        float x[3], y[2];

        for (i = 0; i < nb_line_field; i++)
        {
                f_coor.line = (float)i;
                f_coor.pixel = (float)address[i];
                s_cam_to_user(&f_coor, calib8);
                pos[i]=f_coor.u;
                depth[i]=f_coor.v;

#ifdef DEBUG
                x[0] = x[1];
                x[1] = x[2];
                x[2] = f_coor.v;

                vo = filter(x, y);

                fprintf(lisa,"%f %f %f %f %f\n", f_coor.line, f_coor.pixel, f_coor.u,f_coor.v,vo);
                printf("%f  %f  %f  %f  %f\n", f_coor.line, f_coor.pixel, f_coor.u, f_coor.v, vo);
#endif

        }

        for(m=0; m<nb_line_field-1; m++)
        {
                derivative[m] = address[m+1]-address[m];
                maxmin=derivative[m];
                if(maxmin > max)
                {       max=maxmin;
                        maxcell=m;
                }
                else if(maxmin<min)
                {       min=maxmin;
                        mincell=m;
                }
        }

/*      for(m=0;m<nb_line_field-1;m++)
                printf("%d  %f  %f  %f\n", m, derivative[m], pos[m], depth[m]);
        printf("max is %d and min is %d\n", max, min);
*/
        if((max>2)&&(min<-2))
        {
                width=fabs(pos[maxcell+2]-pos[mincell-2]);
                *offset = (pos[maxcell+2]+pos[mincell-2])/2;
```

```
                    *offset = (*offset+20.15)*0.0531;
//          send_offset(*offset);
//          printf("crack detected!\ncrack start=line %d  and  end=line %d\nwidth=%f\n", mincell,
maxcell, width);
//          printf("\roffset: %6.2f",*offset);
            }
            else if (min<-2)          /* crack end not found */
            {
                    width = fabs(pos[mincell-2] - pos[nb_line_field-1]);
                    if (width < TOO_SMALL)
                    {
                            printf("Crack too small\n");
                            send_not_found();
                    }
                    else if (width > TOO_BIG)
                    {
                            printf ("Crack too big\n");
                            send_not_found();
                    }
                    else
                    {
                            *offset = (pos[mincell-2] + pos[nb_line_field-1])/2;
                            *offset = (*offset+20.15)*0.0531;
                            send_offset(*offset);
                    }
            }

            else
            {
                    send_not_found();
            }
    }

    void find_filled_crack(float tolerance, float avg, float *offset)
    {
            printf("\nProgram incomplete for filled cracks");
    }

    void send_not_found(void)
    {
            printf("\nNo crack found!!\n");
//          send_offset(NO_CRACK);
    }

    void last_call()
    {
            p_restore_kb();           //must be done, key board ISR has been changed
            p_int_reset();
            restore_vectors();
            u_reset_dir();  //return to previous directory
            set_mode(3);
            restore_video();
    }
```

```
void emergency_out(int sig)
{
        exit(0);
}

void set_video(void)
{
        struct videoconfig video_info;
        short videomode = _HRESBW;

        _getvideoconfig(&video_info);        /* Call to find adapter */
        switch (video_info.adapter)
        {
                case _MDPA:
                        printf("This program needs a graphics adapter.\n");
                        exit(0);
                case _CGA:
                        videomode = _HRESBW;      /* 2 color 640x200 CGA mode */
                        break;
                case _EGA:
                        videomode = _ERESCOLOR; /* 16 color 640x350 EGA mode */
                        break;
                case _VGA:
                        videomode = _VRES16COLOR; /* 16 color 640x480 VGA mode */
                        break;
        }
        /* Set adapter to selected mode */
        _setvideomode(videomode);
        /* Call _getvideoconfig again to find resolution and colors */
        _getvideoconfig(&video_info);
}
void restore_video(void)
{
        _setvideomode(_DEFAULTMODE);
}
```