A Vision-Based End-Point Control for a Two-Link Flexible Manipulator

A THESIS
Presented to
The Academic Faculty

by

Klaus Obergfell

In Partial Fulfillment
of the Requirements for the Degree
Master of Science

Georgia Institute of Technology
School of Mechanical Engineering
August 1991

A Vision-Based End-Point Control for a Two-Link Flexible Manipulator

APPROVED:

Wayne J Book

Wayne J Book, Chairman

Stephen L. Dickerson

Kok-Meng Lee

Date Approved by Chairperson 23 Aug 91

ACKNOWLEDGEMENTS

I would like to express my deepest gratitude to Dr. Book, my thesis advisor, for his invaluable help and encouragement throughout my thesis work. His caring spirit and friendship with his students will always be a great example to me.

I would also like to express my appreciation to Dr. Dickerson for his help and for providing the vision system and to Dr. Lee for his assistance and insightful suggestions.

Special thanks also to all my fellow students at the "tin-building" for their help and beneficial discussions.

Finally I owe special debt to my family for their love and continuing support and to Vicki, Martin and a small group of friends for making life in Atlanta enjoyable.

This work was supported by the German Academic Exchange Service (DAAD), the School of Mechanical Engineering at Georgia Tech, NASA Grant NAG 1-623 and the NEC Corporation.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

This thesis investigates the measurement and control of the end-effector position of a large two-link flexible manipulator. The system implementation is described and an initial algorithm for static end-point positioning is discussed.

Most existing robots are controlled through independent joint controllers, while the end-effector position is estimated from the joint positions using a kinematic relation. End-point position feedback can be used to compensate for uncertainty and structural deflections. Such feedback is especially important for flexible robots.

Computer vision is utilized to obtain end-point position measurements. A look-and-move control structure alleviates the disadvantages of the slow and variable computer vision sampling frequency. This control structure consists of an inner joint-based loop and an outer vision-based loop.

A static positioning algorithm was implemented and experimentally verified. This algorithm utilizes the manipulator Jacobian to transform a tip position error to a joint error. The joint error is then used to give a new reference input to the joint controller. The convergence of the algorithm is demonstrated experimentally under payload variation.

A Landmark Tracking System [Dickerson, et al 1990] is used for vision-based end-point measurements. This system was modified and tested. A real-time control system was implemented on a PC and interfaced with the vision system and the robot.

Future extensions of this work are discussed.

CHAPTER I

INTRODUCTION

1.1. Motivation

This thesis investigates the measurement and control of the end-effector position of a large two-link flexible manipulator and describes the initial system implementation.

Most existing robots are controlled through independent joint controllers, while the end-effector position is estimated from the joint positions using a kinematic relation. If those robots have a stiff structure, structural deflections can be neglected. Disadvantages of this concept are heavy weight, need for powerful actuators, slow operation speed, limited work space and resulting low accuracy.

End-point position feedback can be used to compensate for uncertainty and structural deflections and to increase the positioning accuracy. Such feedback is especially important for flexible robots.

Various methods have been published that claim good results for end-point control of flexible robots. But only some of these

researchers prove their methods with a direct measurement of end-point position, while others use an estimation derived from joint and flexible states. Similarly not all end-point control methods use an end-point position feedback. Lack of readily available sensors for a direct measurement encourage this approach. From those deficits comes the motivation to develop a system for end-point position measurement and control.

Existing robot applications use computer vision for inspection and guidance in partially unstructured environments. The acceptance and availability of vision in robotics suggests to use vision also for the position feedback. An inertial measurement (accelerometer) could alternatively be used. Drawbacks of this method are the error resulting from the integration of the original signal and the lack of an absolute measurement. This argument motivates the use of computer vision for position measurements and feedback control.

1.2. Flexible Manipulators

Robots derived from a concept that allows and considers structural deflections are called flexible robots. The development of flexible robots has several motivations: 1. applications in space which dictate dramatically reduced weight, 2. mobile robots for construction work, maintenance of electricity and communication cables etc. which demand reduced weight, reduced power requirements for actuation and increased work-space size, 3.

handling of heavy parts which requires increased payload-to-weight-ratios, 4. inspection and manufacturing of large structures like airplane bodies etc. which demand increased work-space, 5. robots for high speed operation which require reduced inertias. Existing examples of flexible robots are the space shuttle arm and a variety of robots in research laboratories all over the world, ranging from very flexible study objects to applied prototypes.

The structural flexibility introduces characteristics that have to be considered when designing the control for a flexible robot. First, the links of the robot are a continuous system exhibiting an infinite number of flexible modes. Neglecting higher frequencies and modelling the system with a finite number of modes causes some uncertainty. Second, the dynamic equations are nonlinear, with the nonlinear terms being centrifugal and Coriolis forces. Linearizing those equations for an operating point causes more uncertainty, especially for high speed motion and for operation far away from the operating point. Third, the end-point control is a non-colocated problem. That is, the robot is controlled based on measurements at one end of the kinematic chain by actuators at the other end. According to Cannon and Schmitz (1984), direct feedback of the end-point position tends to make the system unstable or conditionally stable. Therefore the control has to be designed carefully. Fourth, the transfer function from actuators to tip position is non-minimum phase. Mathematically this is described by right half plane zeros, meaning physically that the tip initially moves in the opposite direction of the motor input, i.e. in the "wrong" direction.

1.3. Review of Previous Work

End-point control of flexible manipulators

This section discusses publications presenting the motion control of flexible robots. Some of the described methods use end-point feedback with vision. However, the vision is treated as a black-box providing the position of the end-point, the special characteristics of vision are not considered.

Optimal control design with pure feedback of tip-position, joint rate and strain was introduced by Cannon and Schmitz (1984). Those researchers investigated a one-link flexible robot, a one meter long arm that moved in the horizontal plane. This arm was modelled with the assumed modes method, linearized for an operating point and parameters experimentally identified. The controller was digitally implemented with a sampling frequency of 50 Hz. This is 100 times faster than the natural cantilever frequency of 0.5 Hz. Feedback of the end-point position was provided from an analog sensor. The authors reported that the closed-loop system remained conditionally stable and the feedback of joint rate and strain was helpful in achieving good closed-loop performance. The presented step responses show two characteristics: First, the speed of response is faster than the first natural period, but ultimately limited by the wave-propagation delay of the flexible

beam. Second, the response is that of a non-minimum phase system, that is the tip moves initially in the wrong direction.

The optimal control concept was extended to end-point tracking by Oakley and Cannon (1989) and Oakley and Barratt (1990). In the first paper an LQ regulator and a nonlinear estimator were used to control a two-link robot with a rigid upper arm and a flexible lower arm. In the second paper a two-link robot with two flexible arms was controlled by an augmented LQG controller with FIR filter. In both cases the controller worked synchronously with a vision system for end-point position measurements. A trajectory was specified in Cartesian space for end-point tracking. Desired rigid states for reference commands were computed using rigid inverse kinematics, while the desired flexible states were set to zero. Measured end-point position data shows approximate tracking with some over-shoot and the second controller following the desired trajectory more closely.

Two feedback loops, an inner loop to control motor position and an outer loop to control tip position, were suggested by Rattan, Feliu and Brown (1990). The design procedure was described for a simplified one-link flexible robot, a structure consisting of a "mass-less" link (music wire) with lumped mass. A camera tracking a LED mounted at the tip of the arm was used for end-point measurements. The inner loop is used to remove the effect of friction in the joint. The outer loop design considers only the arm dynamics, since the dynamics of the closed inner loop are fast compared with the arm dynamics. The outer loop is composed of a

model based portion and a servo portion. Feedforward terms are used in the servo portion to follow the reference commands.

A Cartesian control schema was investigated by Lee, Kawamura, Miazaki and Arimoto (1990). The transposed manipulator Jacobian relates a force at the robot tip (Cartesian space) to joint torques (joint space). This was used by the authors to transform a proportional Cartesian control action to an equivalent joint control action. Since a proportional control alone is not sufficient for closed-loop stability, a joint rate feedback is added. To improve closed-loop performance also an additional strain feedback was added. This concept was verified using a two-link robot, but only the response to step changes in the reference position was discussed - end-point tracking was not attempted.

An inverse dynamic method for end-point tracking is presented by Kwon and Book (1990). The inverse dynamic method is a feedforward control and does not use end-point feedback. Feedforward control does not affect the closed-loop poles and therefore has the advantage of not making the closed-loop system unstable. But feedforward control requires a good dynamic model, as any open-loop control does. The feedforward control computes the required torque for tracking the end-point from the inverse dynamics. Therefore it is necessary to specify desired states along the desired trajectory. This presents the problem that desired flexible states have to be known for a flexible robot. Kwon and Book present a method to compute the desired flexible coordinates, accepting a non-causal solution. This is a result of the non-minimum

phase characteristics of the flexible system, i.e. the right half plane zeros, which become poles of the inverse dynamics, would make a causal inverse dynamics solution unstable, if a causal solution is accepted.

Vision in robot control systems:

This paragraph describes some characteristics of vision in feedback control.

Visual servo control structures were classified by Sanderson and Weiss (1983). Fig. 1.1. shows this classification in a diagram.

Two feedback representations, position-based and image-based, were suggested. The feedback signal in the first case is a position, e.g. the position of a target relative to the end-effector. The feedback signal in the second case is an image feature, e.g. an area, a length or an angle. The second representation has the advantage of reduced on-line computation, the desired trajectory is specified in desired image features and can be taught off-line. The position-based feedback has the advantage of appearing more natural to feedback control and desired positions can be computed easily.

In a closed-loop joint control structure (Look-and-Move) a inner feedback loop is closed around the joints and an outer feedback loop is closed from the measured robot position to the desired robot position. The outer control loop sends a joint reference command to the inner control loop. An open-loop joint control structure utilizes only feedback of the measured robot position. An advantage of the first structure is that different sampling rates can be used in the

two control loops and the joint loop can operate with a much faster sampling rate than the "slow" vision. It might also be necessary to close the inner loop for stability reasons, [Cannon and Schmitz, 1984].

Timing between joint and vision measurements was used for a further classification of the look-and-move structure. Vision and joint feedback are completely independent for a static look-and-move structure, i.e. the vision based control updates the joint reference input only after the robot came to a complete stop. In a dynamic look-and-move structure the vision based control updates the joint reference input during motion.

Most existing systems use variations of the look-and-move structure. Coulon and Nougaret (1983) performed a dynamic analysis of a TV camera for feedback control and demonstrated visual control of a X-Y-plotter. Feddema and Mitchell (1989) used an image-based look-and-move structure with a feature-based trajectory generator to track a moving target. Their joint controller received velocity commands from the trajectory generator that matched velocity boundary conditions at vision updates.

These applications used TV cameras and specialized computer vision equipment. Existing sensors for vision were discussed by Dunbar (1986). Dunbar pointed out some handicaps of the existing TV standard for computer vision applications. An interesting new approach to computer vision was chosen by Dickerson, et al (1990). Those researchers designed a landmark tracking system (LTS), consisting of illumination, optics, CCD detector, a processor and

software. Based on the experimental works using both the off-the-shelf vision system (breadboard configuration) and the LTS, K.-M. Lee et al (1991) identified several problems associated with the breadboard configuration and the LTS for machine vision applications. The LTS is currently used in two applications related to this thesis: Nam and Dickerson (1991) use vision and accelerometer to obtain position estimates with increased performance and bandwidth. Smith (1991) applies an integrated vision system, based on the electronic framework of the LTS, to end-point control of a three link in-parallel manipulator.

1.4. Chosen Approach

A look-and-move control structure, consisting of an inner joint-based loop and an outer vision-based loop was chosen for several reasons: First this structure alleviates the disadvantages of the intrinsically slow and varying computer vision sampling frequency. The two control loops operate on different sampling rates with the inner joint-based loop being much "faster" than the outer vision-based loop. Second, joint feedback is suggested for stability of the closed loop system [Cannon and Schmitz, 1984]. Third this structure is easily modified for future extensions.

The Landmark Tracking System (LTS) was used for vision. This independent system processes all time consuming vision computations on the integrated processor. The output of the system

is a landmark position. For this reason positions were chosen as visual feedback representation.

A static positioning algorithm was chosen for the first implementation. This algorithm has the advantage of decreased complexity, especially the stability of the closed-loop system is less crucial, while addressing most implementation issues. The algorithm utilizes the manipulator Jacobian to transform a tip position error to a joint error. The joint error is then used to give a new reference input to the joint controller.

A large two-link flexible manipulator at the Flexible Automation Laboratory at Georgia Tech was used to implement the positioning algorithm. The Landmark Tracking System was mounted stationary and observed the manipulator work space.

Visual Servo Control Structures	Closed-Loop Joint Control (Look-and-Move)		Open-Loop Joint Control (Visual Tracking)
Position-Based Visual Feedback	Static PBLM	Dynamic PBLM	Position-Based Visual Servo (PBVS)
Image-Based Visual Feedback	Image-Based Look-and-Move (IBLM)		Image-Based Visual Servo (IBVS)

Figure 1.1. Classification of visual servo control structures, according to Sanderson and Weiss (1983)

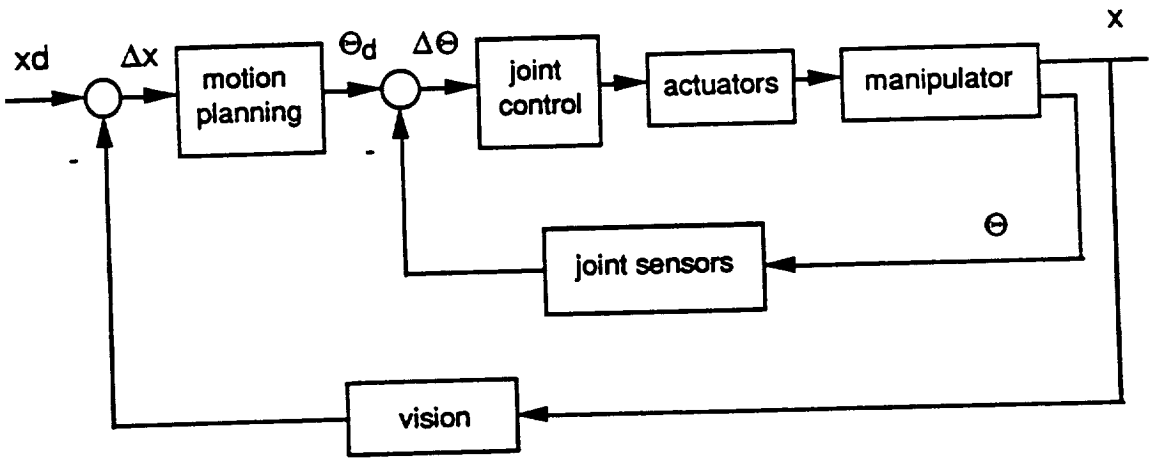


Figure 1.2. Look-and-move control structure for static end-point positioning

CHAPTER II

CONCEPTUAL DESCRIPTION

2.1. Existing Robot Structure and Control System

The investigated robot manipulator is shown in Fig. 2.1. This robot is also known as RALF (Robot Arm Large and Flexible). The manipulator structure consists of two 10 foot long links and a parallel link mechanism. Each link is connected to another by a pin. RALF is actuated with two hydraulic cylinders, one of them is directly connected to the lower link, the other one actuates the upper link through the parallel link mechanism. The motion is restricted to the vertical plane. More structural details can be found in Wilson (1985).

The robot is equipped with sensors for data acquisition and control. Position sensors are mounted parallel to both hydraulic cylinders to measure the extension of the rod. Strain gages are mounted at several locations on the links. A more detailed sensor description can be found in Huggins (1988).

A DEC Micro-Vax was originally used as a centralized data acquisition and control unit. The computer was interfaced with the

robot through a Data Translation analog-to-digital converter board. A decentralized joint controller was implemented with software written in Fortran.

This implementation had two limitations: Low sampling frequency and sequential operation. The maximum sampling frequency of the controller was 120 Hz. This frequency was limited by a slow sampling rate of the converter board and system overhead. An attempt to increase the sampling rate by converting the control software to C was not successful. The different programming language did not produce "faster" code.

The sequential control algorithm restricts the operation to one predefined motion. After completion of this motion the robot moves back to the rest position. With parallel processing it would be possible to perform further motion planning, measurements etc while the robot is still computer controlled.

For those reasons the new control system was implemented on a faster computer with fast data acquisition hardware and designed for quasi-parallel processing, using multi level interrupts.

2.2. End-Point Position Measurements and Vision System Characteristics

The vision system used in this investigation is a prototype of the Landmark Tracking System (LTS) by Dickerson, et al (1990). A landmark is a piece of retroreflective material that is easy to

detect by the vision system. When the vision system detects a landmark in an image, it computes the pixel positions of the landmark center with respect to the image.

For end-point position measurements the vision system was mounted facing the manipulator plane of motion and a landmark was attached to the manipulator tip. Therefore the vision system computes the tip position of the robot with respect to the area observed by the camera, assuming the tip-position is identical to the position of the landmark. A sketch of the position measurement setup is shown in Fig. 2.2.

The LTS is an independent system that communicates with other devices through a serial line. The operation of the vision system is controlled with a command language that calls the integrated software. A set of thresholds and control flags determines operation parameters and output format.

A position measurement sequence consists of sending a command word from the host computer to the LTS and receiving an ASCII character string at the serial port of the host computer after a delay of approximately 50 ms. The character string contains the landmark position information.

The long delay of the position measurement is a second reason to use parallel processing for the new control system. It would not be acceptable to wait for the position measurement data to return while other tasks are pending.

2.3. Integrated Control System

The new control system was implemented on a PC equipped with a data acquisition board and designed to process several tasks in parallel.

Quasi-parallel processing can be implemented on a single processor computer using interrupt functions. An interrupt function is a function that is called by the processor in response to a special event. Such a function can temporarily interrupt the normal operation. Interrupt functions can therefore be used to perform tasks that seem to operate in parallel with other activities.

Two operation modes will be distinguished per Auslander and Tham (1989) for the parallel processing system: Background and foreground operations. A background operation is performed in response to a user input and has a low processing priority. A foreground operation is performed in response to a special event and has a high processing priority.

Two special, time critical functions exist in the control system: Manipulator control and serial communication interface. The manipulator control consists of an inner joint control loop and an outer end-point positioning loop. The time critical inner joint control loop was implemented in the foreground as an timer interrupt function. The end-point positioning loop was implemented in the background as a motion scheduling routine. The serial communication interface consists of a receive and transmit portion.

The receive portion is time critical since a received character could be lost if it is not stored from the serial port buffer to a data buffer before the next character arrives at the serial port. The receive portion of the serial interface is therefore implemented in the foreground as a communication interrupt function. The transmit portion is implemented in the background.

All other tasks were implemented as background functions. Those functions are a supervisor, a setup option, a measurement option and two motion scheduling options. The supervisor of the current implementation displays a menu from which to chose the options. The setup option is used to change control and vision parameters. The measurement option performs single or repeated end-point position measurements. Two motion scheduling options are available: The first option (move robot) moves the robot to a position specified in joint angles, utilizing only joint control. The second option (visual servo) moves the robot to an user specified end-point position, utilizing joint control and end-point position feedback. These background functions communicate with the foreground functions through shared data. Fig. 2.3. shows the data flow of the control system in block diagram form.

The joint controller utilizes desired joint angle reference inputs and joint angle feedback to compute the control action. Each control cycle of the joint controller is started by the system timer. Foreground controller and background motion scheduling routines communicate through a trajectory data structure. This structure consists of a trajectory data array and control flags.

Desired joint angles are computed by a background motion planning function and stored to the trajectory data array. A trajectory control flag is cleared when the complete trajectory is stored. Clearing the flag signals the joint controller that new trajectory data has arrived and the controller utilizes this data to move the robot through the new desired trajectory. The trajectory control flag is set again when the controller has utilized all trajectory data. Setting of the flag signals the background motion scheduling options that a new motion can be scheduled. The joint controller holds the manipulator at the last desired joint position until a new trajectory arrives in the data array.

2.4. Algorithm for Vision Supported Positioning

The manipulator Jacobian relates a small change in joint position (joint space) to a change in end-point position (Cartesian space)

$$\Delta x = J(\Theta)\, \Delta\Theta, \qquad (2.1.)$$

where Δx denotes the vector of small end-point position changes, $J(\Theta)$ denotes the manipulator Jacobian and $\Delta\Theta$ denotes the vector of small joint position changes.

The inverted manipulator Jacobian can therefore be used to transform a tip position error to a joint position error

$$\Delta\Theta = J(\Theta)^{-1}\, \Delta x. \qquad (2.2.)$$

This process can be applied to an positioning algorithm, consisting of the following steps:

1. measure the end-point position
2. compute the tip position error
3. transform the tip position error to joint error, using eq. 2.2.
4. compute new desired joint position
5. command joint controller to move robot to the new desired position

Due to system uncertainty the algorithm is iterated until a desired end-point positioning precision is reached. Between each correction the robot comes to a complete stop to measure the end-point position and plan the next motion.

The conversion of the iterating process is investigated in the following. Starting position, desired position, initial tip position error and tip position error after the first iteration are sketched in Fig. 2.4. The initial tip position error, Δx_0, is transformed to the initial joint error

$$\Delta\Theta_0 = (J^*)^{-1}\ \Delta x_0, \qquad (2.3.)$$

where J^* is an estimation of the exact Jacobian J. This joint error is used to command the robot to the new position, $\Theta_0 + \Delta\Theta_0$. The resulting tip position error after the first iteration is

$$\Delta x_1 = \Delta x_0 - J\ \Delta\Theta_0. \qquad (2.4.)$$

Substituting eq 2.3. in eq. 2.4. yields

$$\Delta x_1 = (I - J(J^*)^{-1})\ \Delta x_0. \qquad (2.5.)$$

Assuming the change of the exact Jacobian J is small and the estimate J^* is constant, the resulting tip position error after n iterations becomes

$$\Delta x_n = (I - J(J^*)^{-1})^n\ \Delta x_0. \qquad (2.6.)$$

By modal transformation of Eq. 2.6. we yield

$$\Delta x_n' = \Lambda^n\ \Delta x_0', \qquad (2.7.)$$

where Λ is the diagonal matrix of eigenvalues of $(I - J(J^*)^{-1})$. The algorithm converges for $n \to \infty$ and the tip position error goes to zero when the largest eigenvalue of $(I - J(J^*)^{-1})$ satisfies the condition

$$|\lambda_{max}| < 1. \qquad (2.8.)$$

If J^* is a good estimate of the true Jacobian J, the product $J(J^*)^{-1}$ is approximately the identity matrix and $(I - J(J^*)^{-1})$ is approximately the null matrix. In this case the condition eq. 2.8. is satisfied. The better the estimate J^*, the faster the algorithm will converge.

When the value of the exact Jacobian undergoes only a small variation in the work space under consideration it is sufficient to use a constant estimation of the Jacobian. This estimate is determined using a calibration described in chapter 4.5.

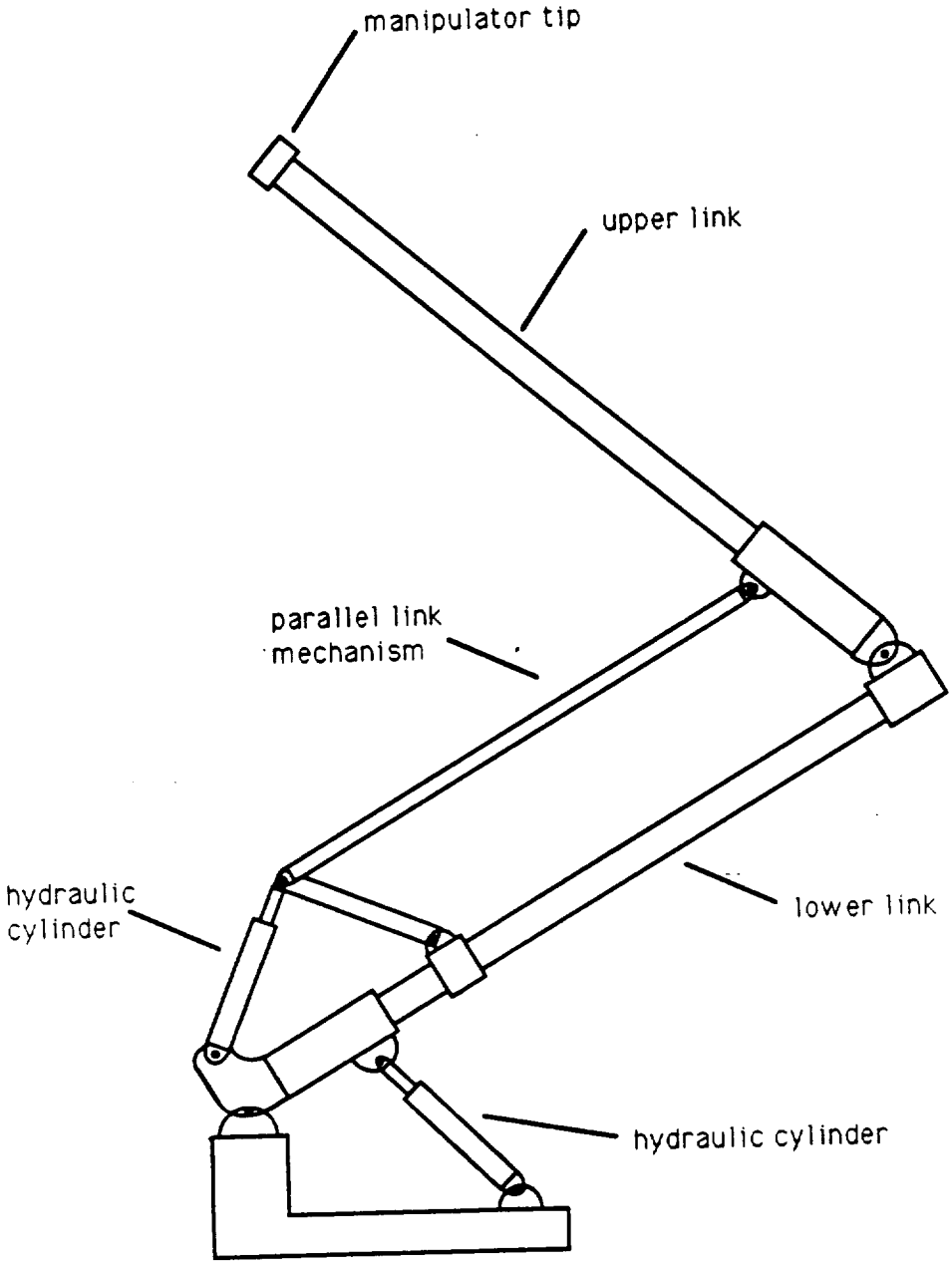


Figure 2.1. RALF (Robot Arm Large and Flexible)

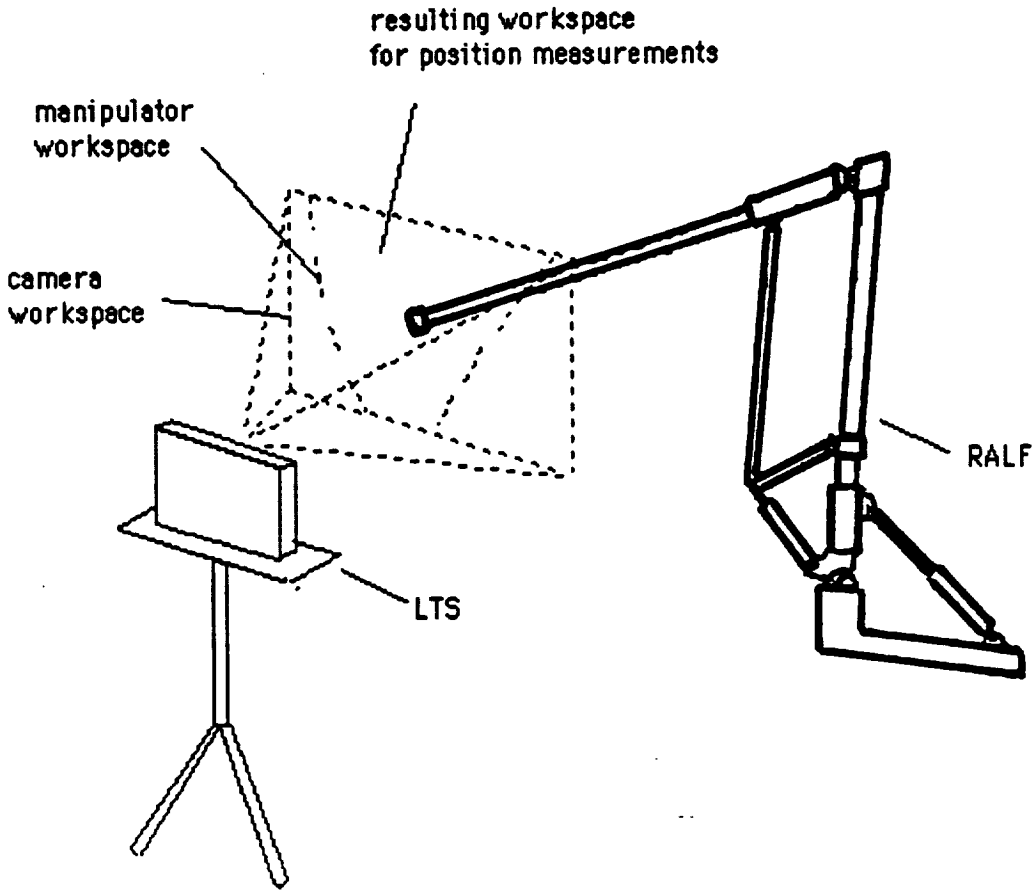


Figure 2.2. Setup for end-point position measurement

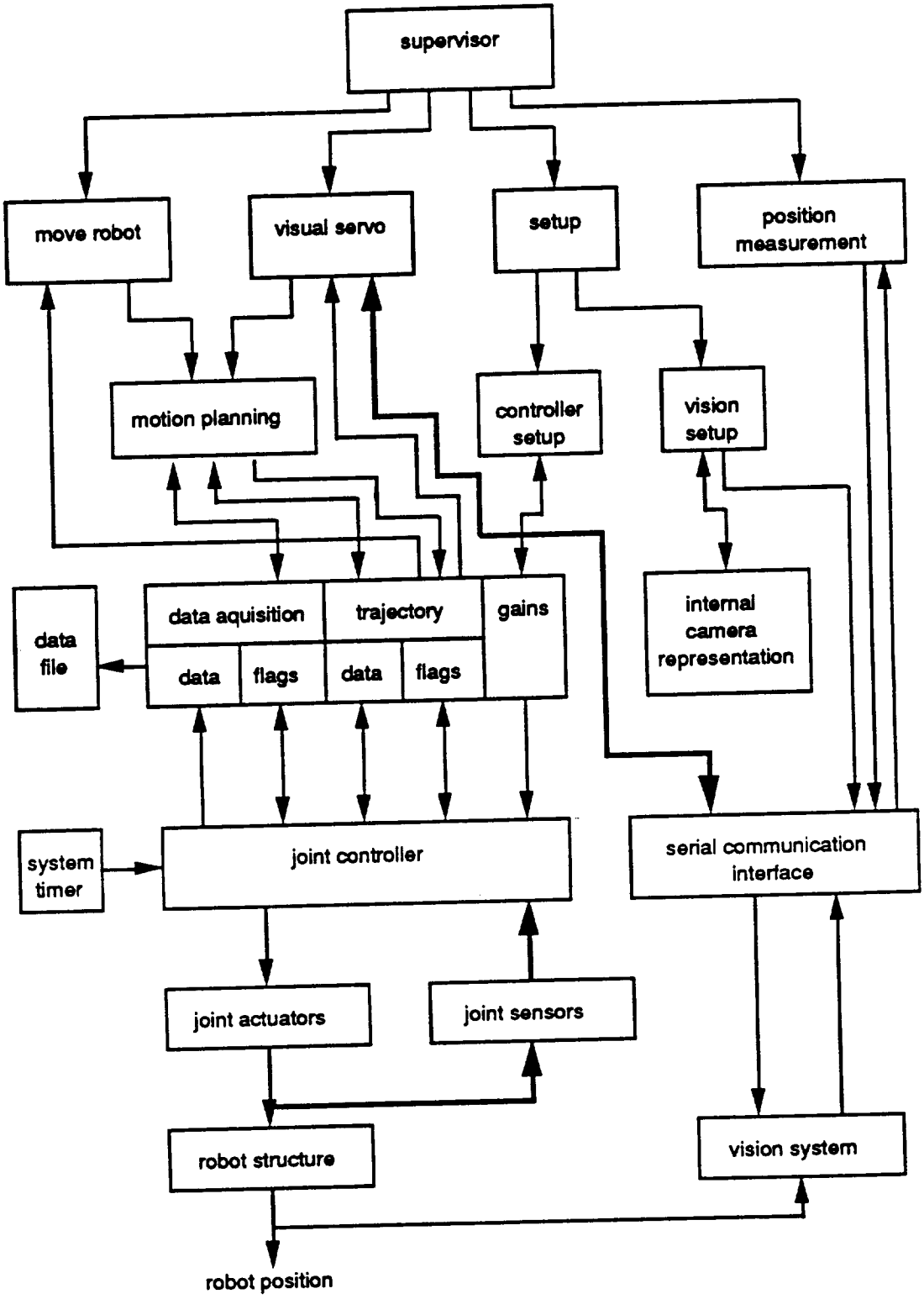


Figure 2.3. Control system block diagram

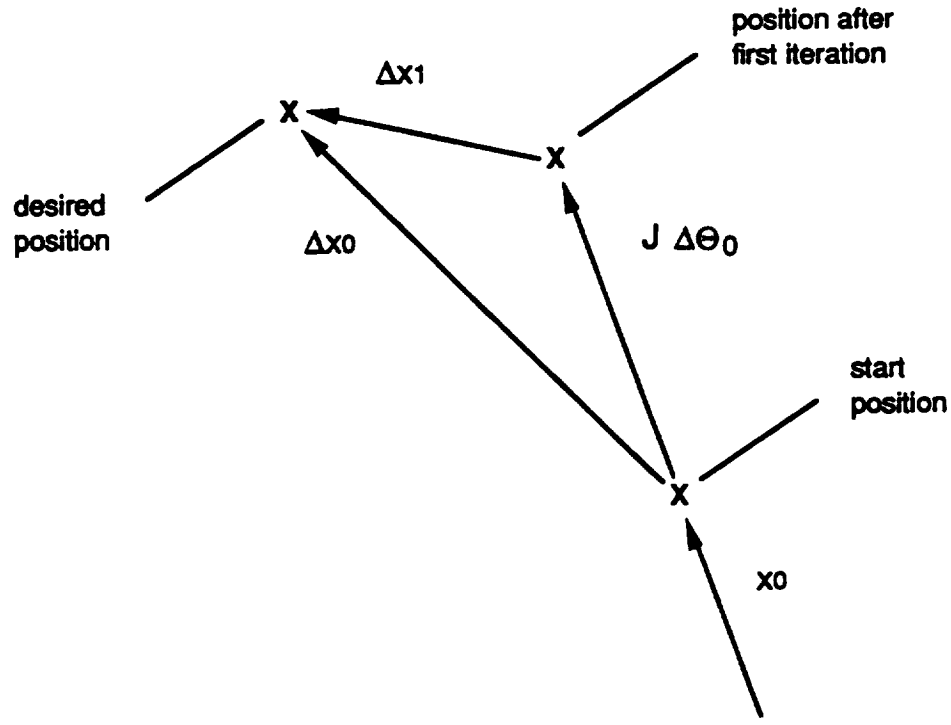


Figure 2.4. End-point positioning, position error after first iteration

CHAPTER III

VISION SYSTEM

3.1. Principle of Landmark-Based Position Measurements

The high cost and complexity of existing vision systems motivated researchers at Georgia Tech to develop a low cost vision system for use in locating objects in a manufacturing environment. This vision system uses landmarks that are attached to these objects to yield precise position measurements quickly.

A landmark is easy to distinguish from the environment and background noise because of it's brightness and known shape. A landmark can be a light source (LED, light bulb) or a piece of reflecting material. The light source has the disadvantage of less flexibility, especially when several landmarks are used. The reflecting material requires a light source for illuminating the landmark, but this can also be an advantage since a short illumination period can be used to "freeze" an image.

Three modes of reflection can be distinguished: Specular reflection (plane mirror), diffuse reflection and retroreflection. Retroreflection is especially useful for landmark tracking because it

does not occur naturally and the reflected light returns along the line of the incident light. The intensity of light returned from a retroreflective landmark is very high, resulting in a good contrast. The LTS is typically used with retroreflective landmarks.

The properties of retroreflection have to be considered in illumination design. The intensity of the reflected light decreases rapidly with increasing observation angle. This angle is explained in Fig. 3.1. Therefore the illumination source has to be very close to the optical axis. The intensity of the reflected light is much less sensitive to a variation of the entrance angle, permitting a large field of view.

3.2. Components of the Landmark Tracking System

3.2.1. Optics and Illumination

Some disadvantages of lenses can be avoided by using a pinhole optics for landmark tracking, e.g. lense optics require focusing and yield an image with barrel or pincushion-like distortions. Pinhole optics give a large depth of field and do not have the geometric distortion of lenses. The typical disadvantages of a pinhole optics, an unsharp image and a high f/stop, are less important or even advantageous for landmark tracking. A sharp image is not necessary to determine the landmark position and a high f/stop is useful to suppress other objects.

Pinhole optics requires a powerful illumination, especially for long range measurements. A very short and intense illumination is desirable for dynamic position measurements to "freeze" the motion and yield a high signal-to-noise ratio. Xenon strobes with a period of light of approximately one millisecond show these properties and are therefore typically used with the LTS, like in the current approach. The strobe is mounted very close to the optical axis to keep the observation angle small.

An electronic flash unit, consisting of a charge circuit, a trigger circuit and timing logic is used to operate the strobe. The flash unit is based on designs by E.H. Lee and Dickerson (1990) and K.-M. Lee, et al (1991). A pulse signal from the LTS processor times the charge time of the capacitors and the triggering of the flash. The energy stored in the capacitors is discharged in the flash tube after a flash is triggered. The intensity of a flash is proportional to the discharged energy.

Two illumination designs were tested, both designs are shown schematically in Fig. 3.2. Design A, the original design for the LTS, was tested with a small flash tube with a maximum input power rating of 1.5 Watt. The maximum input power limits the flash rate since an energy of approximately 0.5 Joule is necessary for sufficient illumination of the landmark at the robot tip. Therefore design B was tested, using an U-shaped flash tube with a maximum input power rating of ten Watts.

Unfortunately design B did not produce flashes with the same intensity as design A. The manufacturer of the flash tubes explained

this resulted from operation of the small flash tube at its maximum trigger voltage, claiming this flash tube would operate beyond it's limits. An attempt to get a brighter flash with design B by increasing the trigger voltage did not succeed. This issue was not investigated further, instead design A was used, limiting the rate of position measurements severely.

Note: A tube similar to the one used in design A is also available with three Watt maximum input power and two tubes could be mounted in parallel on both sides of the sensor. Design B should be reconsidered for a further increase of the flash rate since there is also a different trigger transformer available.

3.2.2. LTS Hardware

The LTS consists of three functional units: Video head, strobe unit and computer. The video head combines pinhole optics, a CCD array sensor and an 8-bit analog-to-digital converter. The strobe unit consists of a flash tube and an electronically controlled power unit as described in the previous paragraph. The computer is a 68000 microprocessor with a 68901 multi-function peripheral device and memory. All operations of the LTS are controlled by the 68000, while the 68901 is utilized as a timing and communication interface. A block diagram of the LTS is shown in Fig. 3.3.

An image acquisition process is started by sending a pulse to the strobe unit to charge the capacitors and trigger the flash. During

the period of strobe discharge the internal software prevents any charge shifting on the CCD sensor. After the illumination time the charges are shifted out of the sensor, amplified, and converted to pixel intensities. The pixel intensities are stored to the video RAM which is disconnected from the 68000 during the shifting process. After the image acquisition process is completed, the 68000 accesses the data in the video RAM for further processing, depending on the software. The LTS contains 64 kB EPROM primarily intended for software and 64 kB additional RAM for data processing.

The 68901 provides an asynchronous receiver-transmitter that is utilized to communicate with an external host computer through a RS-232 serial line. The original LTS was limited to a Baud rate of 2400. Therefore an additional counter was added to the circuit board for the selection of Baud rates up to 57.6 kBaud. The system clock was changed for this modification from 16 MHz to 18.432 MHz. In this special case changing the system clock did not require an additional clock for the communication rate and also increased the processing speed by 15%.

3.2.3. LTS Software

The host computer accesses LTS software through messages sent to the LTS. Messages are in ASCII and one or more bytes long. ASCII characters were originally chosen to operate the LTS from a terminal; this was not changed to binary since the control system

offers a similar terminal option. The first byte determines the intent of the message (command), the remaining bytes are parameters. The prompt from the LTS is '?', telling the host computer that the LTS is ready to receive a message. An error in a command returns a second prompt, a parameter error returns a backspace. All characters received from the host are returned immediately, the response from the LTS follows.

Some of the available commands are explained in the following:

picture (P), this command stores a complete picture from the CCD array to video memory. The routine handles all necessary timing, including the charging and firing of the strobe.

locate (Ln), locates landmark(s) in window n. The landmark area, peak value, row (Y) C.G. and column (X) C.G. are returned as an ASCII character string, the output can be modified using the threshold command. This algorithm scans the specified window for a bright pixel (threshold 0), investigating only every other row and every other pixel. When a bright pixel is found the landmark is systematically expanded by comparing the intensity of adjacent pixels to threshold 1.

picture & locate (Kn), takes a picture and locates landmark(s) in window n. This command combines the picture and locate commands.

show (Sn), creates a crude image display on screen, using window n. The image in the video buffer is displayed on 24 rows and 78 columns, the intensity is displayed as a hexadecimal number.

window (wnlrlchrhc), this command defines window n. The four bytes low row (lr), low column (lc), high row (hr) and high column (hc) define the boundaries of window n.

threshold (tnxx), the byte xx is the new value of threshold n, the available thresholds are explained in Table 3.1.

flag (fnx), sets the flag n to the value x, where x=0 is false and x=1-F is true. Available flags are explained in Table 3.2.

3.3. Performance of the Landmark Tracking System

The performance will be evaluated in this chapter, using as parameters sampling time, resolution and range. These three parameters are equivalent to speed, accuracy and area of the measurement and are not independent. Resolution and range as well as sampling rate and range are inversely proportional. The chosen configuration has to balance those three performance measures.

Resolution and range are more important for the static positioning system than sampling time. Range is particularly important since the investigated manipulator has a very large work space. For these reasons it was decided to chose a range of six feet between camera and manipulator, which results in an observed area of 43 by 43 inches.

3.3.1. Sampling Time

This section investigates the parameters determining the sampling time of the vision-based position measurement and describes modifications that improved the sampling rate. Some future improvements are suggested.

The sampling time is composed of six segments due to the sequential operation of the LTS: 1. the communication time to send a command to the LTS. 2. the charge time of the strobe unit. 3. the time for shifting the vision data out of the CCD array and storing them to memory. 4. the computation time of the landmark tracking algorithm. 5. the communication time to send the landmark data to the host computer. 6. the time to process the raw character string in the control system. Only the first five periods are influenced by the LTS, the last time is not further investigated.

Values for the first five periods and the resulting sampling time of the LTS are given in Table 3.3. for three variations of the LTS: the original system, the currently used system and a fictitious future system. The four components (the two communications are combined) of the sampling time are discussed separately in the following sections:

First, the charge time of the strobe unit was measured with an oscilloscope. The charge time depends on the flash energy necessary for a sufficient illumination of the landmark, therefore on the range

of the position measurement, and is relatively long for the investigated case. The charge time was decreased in the current system by hardware changes in the charge circuit, and can be software controlled through the setting of threshold 4. The relation between threshold setting, charge time and flash energy for the currently used strobe unit is displayed in Table 3.4. For a further decrease of the charge time it is suggested to operate the strobe unit in parallel with the other LTS functions.

Second, the time for the data shifting can be calculated from the number of pixels (165 row pixels x 192 column pixels) and the shifting rate. This time was decreased by using a faster system clock since the shifting rate is proportional to the system clock. This time is independent from other parameters discussed in this paragraph.

Third, the time required for the processing was estimated from the difference between total time and the remaining components. The processing time depends on the size of the landmarks used (in terms of pixels in the image) and on the area scanned for landmarks (window). The time in Table 3.3. was determined for scanning the whole image and tracking two landmarks. The processing time was decreased by using a faster clock and reducing computations, since second moments of the landmarks are not calculated. A future development should decrease the processing time by using a windowing algorithm that predicts the next landmark location and scans only a fraction of the image. Nam and Dickerson (1991) for example have decreased the

processing time to approximately 8 ms by scanning 1/50 of the image and using one landmark with a size of approximately 50 pixels.

Fourth, the communication time can be calculated from the number of transmitted bytes and the Baud rate. This time was identified as the major bottleneck and decreased by using a higher Baud rate and transmitting fewer bytes. A small hardware change, described in chapter 3.2.2. enabled Baud rates up to 57600 Baud. The experiments described in this thesis were performed using 19200 Baud. This has two reasons: First, at 57600 Baud a character is received at the host every 0.17 ms and this would slow down the current controller as explained in chapter 4.1. Second, 57600 Baud worked well with the laboratory setup but the RS-232 standard only supports Baud rates up to 20000, Lee, K.-M., et. al. (1991) The number of bytes transmitted from LTS to host computer depends on the number of data bytes and the number of echoed characters. Sending the data as digits or ASCII characters also influences the number of transmitted bytes. Table 3.5. lists the transmitted bytes. The original system transmitted 45 bytes (40 ASCII, data bytes for two landmarks and 5 echoed characters 'P?Ln?') at 2400 Baud. The current system transmits 27 bytes (24 ASCII, data bytes and 3 echoed characters 'Kn?').

Area and Peak are not necessary for the position measurement but are still transmitted for error checking. Sending those bytes could be eliminated if the error checking would be performed by the onboard software. Data is still transmitted in ASCII to support a

terminal option of the control system; modifying this option in the future could further reduce the number of transmitted bytes.

The minimum sampling time of the used system was measured to be 95 ms. However, this sampling time is not used for normal operations since the strobe frequency is currently limited by the available flash tubes.

3.3.2. Resolution of the Position Measurement

This section describes a repeatability test and discusses the resulting resolution of the position measurement. The range of the measurement ultimately determines the resolution.

The repeatability of the position measurement was first tested with an initial setup on a workbench. The most important result of those initial tests was that the prototype LTS should only be used for relative position measurement. The software controlled timing of the data shifting caused a random shift in the column value of the center of gravity of the landmark. This error is eliminated when the distance between two landmarks is measured, because the C.G.'s of both landmarks are shifted by the same amount. Newer versions of the LTS have a hardware controlled timing of the data shifting that eliminates the shifting error.

The repeatability of the relative position measurement was tested under setup conditions. The vision system was mounted on the camera frame, viewing the work space plane of the manipulator. Two

landmarks, 21" apart, were attached to a level and the level was placed in the manipulator plane. The repeatability was tested for x- and y-direction separately. The test results are summarized in Table 3.6.

The repeatability is better in x-direction than in y-direction, where the ratio of the repeatabilities is approximately equal the ratio of pixels in these directions (192 x-pixel, 165 y-pixel). The repeatability expressed in length instead of pixels is directly proportional to the range of the measurement.

Estimating that the accuracy is approximately 1/10 of the repeatability, a position measurement accuracy of 1.5 - 2.0 mm should be expected. This approximation seems reasonable since the worst case results of the repeatability test are approximately twice the size of the variation.

Table 3.1. Parameters used by the Landmark Tracking System (threshold command tn)

n	threshold description	LTS default (hex)	control system (hex)
0	minimum value for pixel to start growing a landmark	30	30
1	minimum value for pixel to be included in landmark	20	20
2	minimum area of blob to be reported as landmark	04	10
3	integration time for exposing CCD = 0.32 (t3 + 0.4) ms, with 16 MHZ clock	03	03
4	strobe charge time = 0.28 t4 ms, with currently used strobe	00	40
5	repeats of integration time (threshold 3)	00	00
7	configuration of landmark output, returned when bit cleared: bit 0 - landmark area bit 1 - peak value bit 2 - row (Y) C.G. bit 3 - column (X) C.G. bit 4-7 are not used currently	00 (area, peak, C.G's returned)	00 (area, peak, C.G's returned)

Table 3.2. Flags used by the Landmark Tracking System

n	flag description	default
0	all received characters are returned (echoed back) to the host computer	true
1	all landmarks in a window will be recognized, not just the first landmark	true

Table 3.3. Sampling time for three variations of the Landmark Tracking System

LTS sampling time (ms)	original system scanning whole image, 2 landmarks	current system scanning whole image, 2 landmarks (22 pixels)	future design
communication time, host to LTS	12.5 3 byte at 2400 Baud	1.0 2 byte at 19200	0.5 1 byte at 19200
strobe charge time, 0.5 J, 6'	41 sequential operation	18 sequential operation	- parallel operation
data shifting	7.9 4 MHz	6.9 4.6 MHz	6.9 4.6 MHz
processing time & overhead	?	55 4.6 MHz, estimated from total time	?
communication time, LTS to host	187.2 40 data bytes, ASCII and 5 echoed bytes at 2400 Baud	14.0 24 data bytes, ASCII and 3 echoed bytes at 19200 Baud	2.1 2 data bytes,hex and 2 echoed bytes at 19200 Baud
total time	> 311.9 estimated	95 timed with PC	< 64.6 estimated

Table 3.4. Relation between threshold setting, charge time and flash energy for currently used strobe unit

threshold t4 (hex)	charge time Δt (ms)	voltage diff. ΔV (V)	flash energy $E = 0.5*C*\Delta V^2$ (J)
08	2.22	4.6	.00033
10	4.55	48.4	.036
18	6.66	76.4	.091
20	8.9	110	.19
28	11.05	129	.26
30	13.25	148	.34
40	17.80	178	.49
50	22.15	199	.61
60	26.80	207	.66

Table 3.5. Bytes transmitted from Landmark Tracking System to host computer

transmitted bytes	send as digit	send as ASCII char
area of landmark	1	2
peak intensity	1	2
row center of gravity	2	4
column center of gravity	2	4
row 2nd moment	2	4
column 2nd moment	2	4
echoed characters	-	1 per character

Table 3.6. Repeatability of the relative position measurement

Repeatability	(pixel)	(inch)	(mm)
X-direction (col):			
variation	0.0287	0.00654	0.166
worst case (+)	+0.0667	+0.0152	+0.386
worst case (-)	-0.0633	-0.0144	-0.366
Y-direction (row):			
variation	0.0309	0.00807	0.205
worst case (+)	+0.07	+0.0183	+0.464
worst case (-)	-0.05	-0.0131	-0.332

conditions: 120 measurements for each direction, 6' range, 2 landmarks, 21" apart

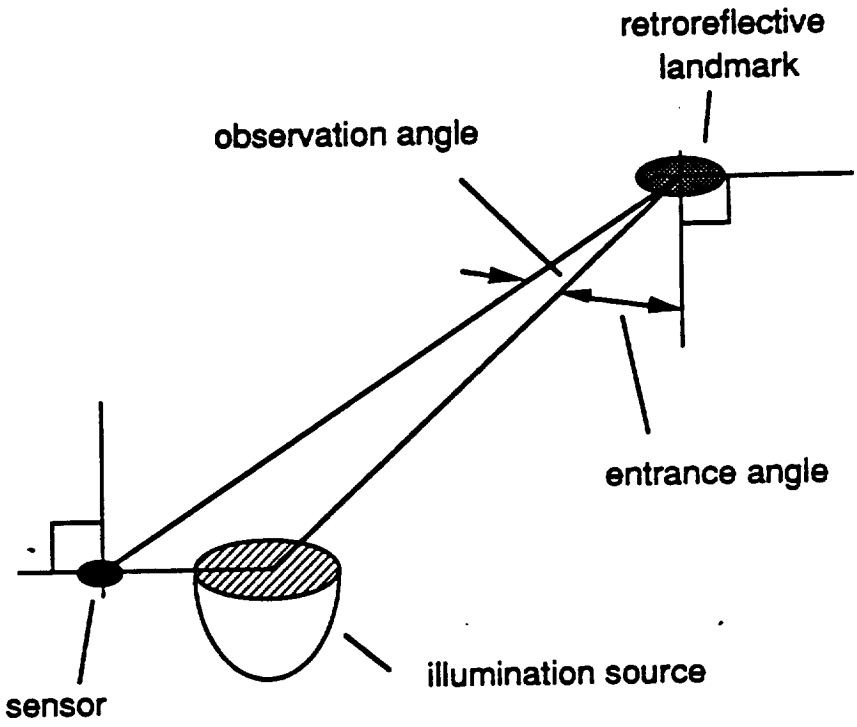
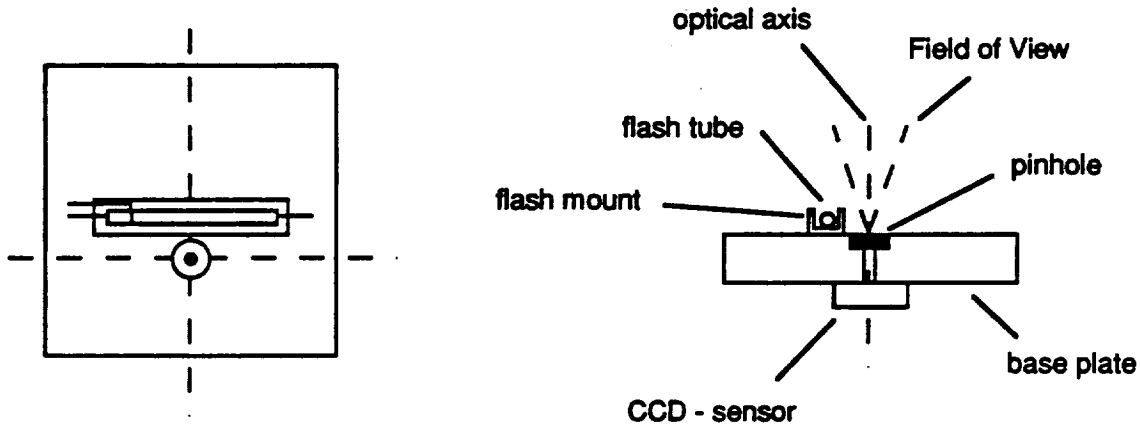


Figure 3.1. Optical elements in the Landmark Tracking System, according to Lee and Dickerson (1990)

Design A - straight flash tube



Design B - U-shaped flash tube

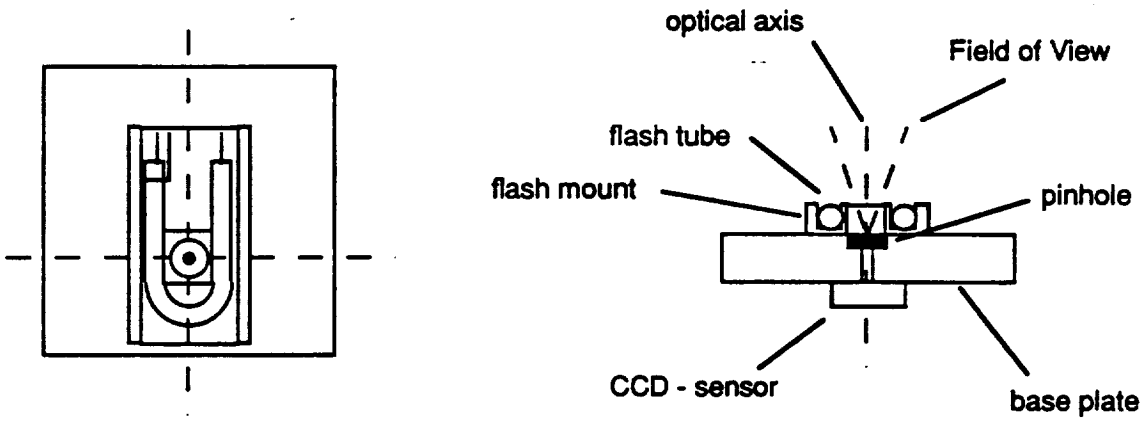


Figure 3.2. Illumination design using Xenon-strobes

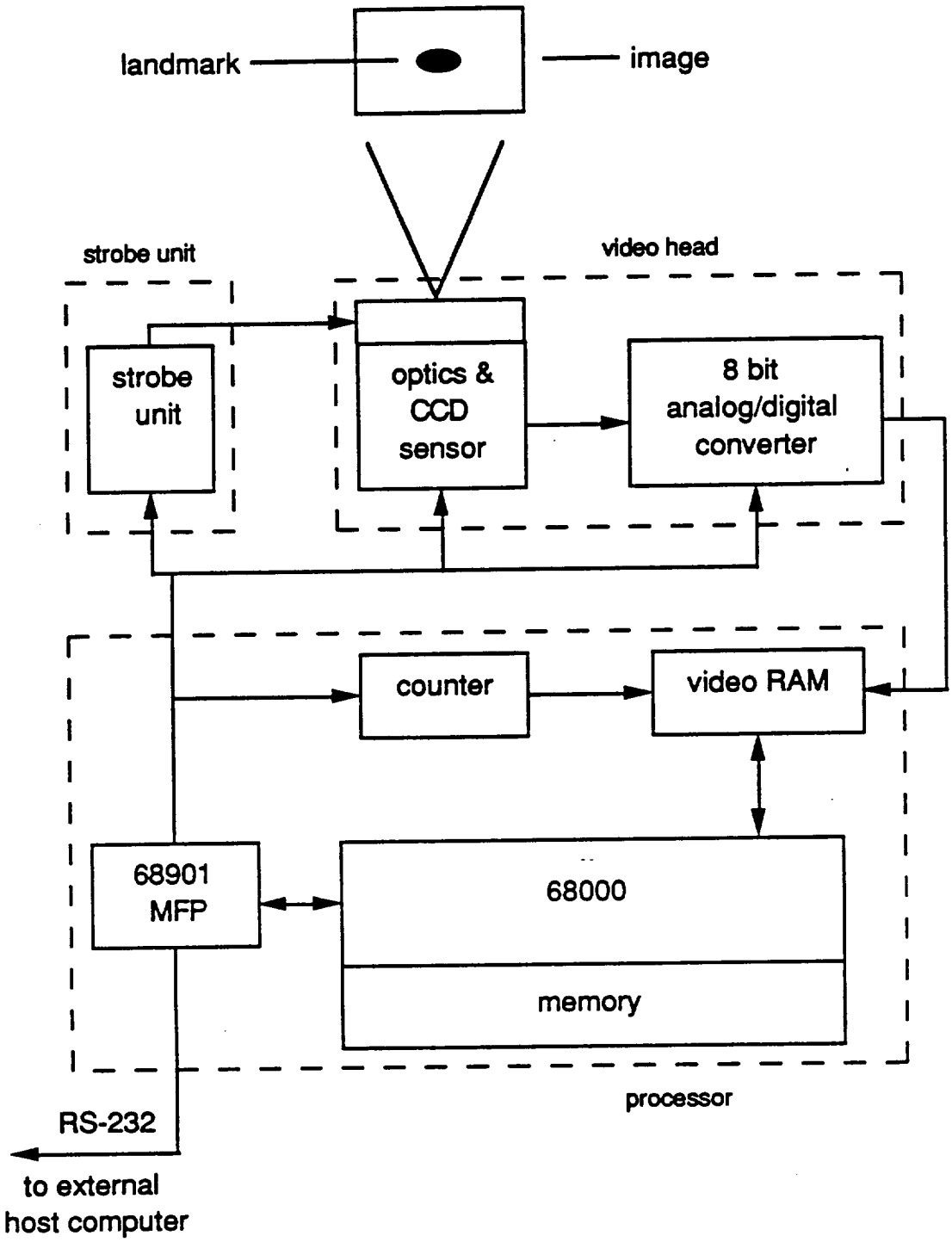


Figure 3.3. Components of the Landmark Tracking System, according to Dickerson et al (1990)

CHAPTER IV

CONTROLLER IMPLEMENTATION

The control system was implemented on a PC equipped with a data acquisition board. Time crucial operations of the system were implemented as interrupt service routines while all other operations were implemented in the background. The software was organized in modules and mostly written in C, some assembler code is used for interrupt functions.

4.1. Programming with Interrupts

Two operations of the control system were implemented as interrupt service routines: Joint control and the receive portion of the serial communication.

The joint control was implemented as a timer interrupt service routine to guarantee a constant sampling time. The signal for a timer interrupt is generated by a 8253 programmable interval timer that is on the PC system board and normally used for the

time-of-day interrupt. The interval of the timer is set to the sampling time of the joint controller by writing to the timer control port. (file alarm.c, taken from Auslander and Tham, 1989). The minimum sampling time of the joint controller is approximately 1.6 ms but the sampling time was set to 5.0 ms to give the computer enough time to process the other tasks.

The receive portion of the serial communication was implemented as a communication interrupt function (com1). The signal for a communication interrupt is generated by a 8250 UART (universal asynchronous receiver/transmitter) that implements the serial communication interface of the PC. This signal is generated every time a byte is received and stored in the UART data buffer. The communication interrupt service routine stores this byte to memory before the next byte is received since the UART data buffer can store only one byte. No handshaking is implemented for the communication between PC and LTS while at a communication rate of 19200 Baud a byte is received approximately every 0.5 ms. Therefore the communication interrupt needs a higher priority than the timer interrupt since bytes could be lost otherwise. The processing time of the communication interrupt service routine is short compared to the processing time of the timer interrupt service routine. Therefore the controller sampling time is not altered significantly when the communication interrupt temporarily suspends the processing of the timer interrupt.

Both interrupt signals are send to a 8259A programmable interrupt controller that handles hardware interrupt requests to the

processor of the PC. The interrupt controller can be programmed to assign priorities to external interrupt requests. The standard configuration assigns the highest priority to the timer interrupt and a lower priority to the com1 interrupt. However, assigning a higher priority to the com1 interrupt was not accomplished by just reprogramming the interrupt controller. Instead the timer interrupt service routine was programmed so that it can be interrupted by the communication interrupt. This is briefly explained in the next section, for a more thorough explanation of the interrupt processing sequence the reader is referred to the technical PC literature, e.g. Tomkins and Webster (1987), Eggebrecht (1990).

An interrupt request of the interrupt controller is only acknowledged by the central processing unit (CPU) when the interrupt flag is set (unmasked). During the interrupt sequence the interrupt flag is cleared (masked), so that no other interrupt requests are acknowledged while the current request is processed. The return-from-interrupt (IRET) command unmasks interrupts again after the interrupt service routine is completed. This can be altered by setting the interrupt flag during the processing of the interrupt service routine with the set-interrupt-flag (STI) command. This enables other interrupts to interrupt the currently processed interrupt service routine. In this case a flag should be used that checks if the routine is not self-interrupting.

A second issue is important for using the interrupt controller. The interrupt controller does not process further interrupt requests until it receives an end-of-interupt (EOI). This end-of-interrupt must be sent by the hardware interrupt service function.

Those two issues are considered in the following way: The communication interrupt service routine can't be interrupted. Therefore a specific end-of-interrupt (SEOI) is sent to the interrupt controller immediately before returning from the communication interrupt service routine. The specific end-of-interrupt resets the request for a com 1 interrupt. The timer interrupt service routine can be interrupted by the communication interrupt. Therefore a SEOI is send to the interrupt controller and a STI is send to the CPU before the actual control routine is called by the timer interrupt service routine. In this case the SEOI resets the request for a timer interrupt.

Some other issues should be mentioned that were considered for the interrupt implementation:

1. The math-coprocessor state is saved at the beginning of the timer interrupt service routine since joint controller and background routines use floating point calculations. This state is restored at the end of the service routine.

2. Non-reentrant functions can't be called by the foreground and background routines at the same time. Math functions, BIOS functions and functions interfacing the data acquisition board are non-reentrant functions. Those functions have in common that they address some special hardware components. Therefore the state of

those functions is determined by the processor state and the state of the hardware registers. Since the hardware registers are not saved and restored by an interrupt function, the processor could not continue a interrupted function correctly when a previous interrupt altered those registers. Therefore non-reentrant functions are not shared by foreground and background routines. The data acquisition board is only accessed from the foreground routines. The sensor readings are then stored in external memory for access by the background. Print statements that are supported by BIOS functions are only used in the background.

3. A program section that can't be interrupted is called a critical region. For example a critical region exists when a background routine writes shared data that a foreground routine should not access until the writing is completed. Such critical regions were implemented using control flags. Other critical regions exist at the beginning and end of the timer interrupt service routine. These regions include saving and restoring the coprocessor state, checking for self-interrupting and sending the SEOI to the interrupt controller. The interrupt flag of the CPU was masked in these critical regions, disabling further interrupts inside the regions.

4. No automatic variables are used in the foreground since this causes stack-overflows. Automatic variables are created on the stack after entering a function and exist only for the duration of the creating function call. The foreground accesses only static or external variables which are assigned to a particular memory location and therefore exist for the duration of the whole program

execution. Unfortunately static memory is limited to 64K, therefore the size of the arrays for data acquisition and trajectory data was limited. This is alleviated using a ring buffer for data acquisition and linear interpolation between trajectory points.

5. The DOS operating system of the PC normally uses the time-of-day interrupt to update the system clock and maintain other system functions. Therefore the "old" timer interrupt service routine is called by the "new" one every 55 ms to maintain those system functions.

4.2. Joint Controller Module

The joint controller module (file control2.c) consists of the timer interrupt service routine, control routine, trajectory data retrieving routine and data storing routine.

The timer interrupt service routine (tmr_isr) has several tasks: 1. it handles the correct interrupt processing described in the previous chapter by saving and restoring the math-coprocessor state, calling the DOS timer interrupts service routine every 55 ms, sending the SEOI to the interrupt controller and checking a flag to prohibit self interrupting. 2. it calls the control routine. 3. it keeps track of time by counting the calls to the timer interrupt service routine and 4. it calls the data storing routine.

The control routine (control) implements a PD controller that is based on Yuan's control algorithm (1989). Huggins's (1988) gain

scheduling algorithm is used to compute configuration dependent PD gains. Two proportional gains were added to the algorithm for controller tuning; the joint position error is multiplied by these gains. These gains are stored in an external gain structure (gain1) and can be modified from the background during operation. These gains were set to 1.1 and -1.4 for joint one and two respectively; for tuning the gains were varied from 1.1 to 1.4 and -1.1 to -1.6, but the steady state joint position error was not sensitive to these variations while the higher values caused more tip vibrations. Averaging multiple joint sensor readings reduced the steady state error by a factor of two to the value of 0.005 radians for both joints.

The current joint angles are retrieved from the joint position sensors by accessing the analog-to-digital converter and converting the digital values to angles. The joint angles are stored as external variables (th1actual, th2actual) that can be accessed by the background. Routines to access the analog-to-digital converter and for unit conversions are collected in separate files (adstuff.c and convers.c). The control outputs are digital values; these values are converted to voltages by the digital-to-analog converter and sent to the amplifiers that control the hydraulic valves.

The desired joint angles are retrieved by the trajectory data routine (get_traj). This routine reads the external trajectory data structure (traj1) and determines the desired joint angles depending on the current control status. The control status is indicated by the trajectory control flag: The flag is set when the controller holds the robot at the last desired position and the flag is cleared when the

controller follows a trajectory. In the first case the desired joint angles are stored as external variables (th1d_hold, th2d_hold). In the second case the desired joint angles are determined from the trajectory data array.

Depending on the duration of a trajectory the desired joint angles are directly taken from the data array or are computed by linear interpolation between trajectory points. This results from memory constraints, since for a long trajectory a desired pair of joint angles can not be stored for every control cycle. The number of interpolations is stored in a variable of the trajectory data structure (iterations). The trajectory data routine computes the desired joint angles by keeping track of the number of processed trajectory points and interpolations.

The data storing routine (store_data) stores time, actual and desired joint angles and control action to a ring buffer. The ring buffer is implemented as an external data structure (data1). A "full"-flag and a status-flag are used to control the buffer. The status flag can be set by the user and determines if data is stored or not. The "full"-flag indicates if space is available, no data is stored when the "full"-flag is set. This flag is set by the data storing routine when the buffer is full. A background routine writes the content of the ring buffer to a data file and clears the "full"-flag.

4.3. Motion Planning Module

The motion planning module (file move_rob.c) implements the trajectory planning algorithm and two motion control options.

Trajectory planning consists of two routines (plan_traj and move_robot_auto), the first routine implements the actual trajectory planner, the second routine determines parameters for the trajectory planning.

The trajectory planning algorithm is based on the previous implementation. Fourth order polynomials, divided into eight sections yield a smooth trajectory. The trajectory is planned for straight line tip motion and converted to joint angles using rigid kinematics. A work space boundary test algorithm keeps the entire trajectory within the manipulator work space. Initial and desired tip position, maximum tip velocity and duration of motion are parameters of the algorithm. The tip position, which is specified relative to the manipulator base, is an estimate of the exact tip position since it is related to joint angles by rigid manipulator kinematics.

The step size of the trajectory, i.e. the number of control cycles between trajectory points, depends on the duration of motion. For memory constraints, the static memory is limited to 64K, the trajectory is limited to 200 points. Linear interpolation between trajectory points is used if more than 200 control cycles are needed.

At a controller sampling rate of 200 Hz, this becomes the case when the duration is more than one second. The trajectory planning algorithm computes the necessary number of iterations and writes it to the trajectory data structure, while the linear interpolation is done during the control cycle by the trajectory data retrieving routine.

The second trajectory planning routine determines the duration of a motion when this time is not directly specified by a user input. A relation for the time of a motion was experimentally determined and is given in Table 4.1., this relation depends on the travelled distance.

The first motion control option is a pure joint control, the robot is moved to a position specified in joint coordinates. This option is implemented in the routine move_robot, consisting of user interface, motion planning and data saving. The user interface prompts the user for desired robot position and duration. The desired position can be specified in joint angles or as tip position. With this data the trajectory planning algorithm is called. Data stored in the ring buffer is written to a data file by the w_data_to_disk routine. This routine also clears the "full"-flag of the data buffer. The actual robot motion is started by clearing the trajectory control flag which indicates the joint controller that a new trajectory has arrived.

The second motion control option implements the static end-point positioning algorithm described in chapter 2.4. The routine vision_guided_pos moves the robot to an end-point position specified by the user. The algorithm first checks if the manipulator

tip is already in the area observed by the camera, this is simply done by taking a picture and searching for the landmarks. If the tip landmark is detected, the algorithm computes the first set of desired joint angles based on the tip position error. Otherwise the joint angles are estimated from the desired tip position. After this initial move end-point position feedback is utilized to update the reference input to the joint controller after every correction move. This updating continues until the end-point position is within a predefined tolerance of 0.06 inch. Both trajectory planning routines are utilized to move the robot to the final position without further user input.

4.4. Vision Module

The vision module (file vstuff.c) combines routines to retrieve the tip-position from the Landmark Tracking System. A position measurement command (get_position) consists of the following tasks: 1. sending the picture & locate command (Kn) to the LTS, 2. receiving the landmark data from the serial communication buffer, 3. checking the landmark data for errors and determining reference landmark and landmark at the robot tip, 4. compensating for the reference landmark position, 5. computing the position of the "moving" landmark relative to the reference landmark.

The communication interrupt service routine stores characters returned from the LTS in the external landmark data array (lm_data).

This data array is inspected for the landmarks. The reference landmark is detected by comparing the position of the returned landmarks to the expected position. The remaining landmark is the landmark at the robot tip or when more than one landmarks are remaining the landmark is detected by comparing area and peak intensity to expected values.

The reference landmark position is compensated by an experimentally determined factor since this landmark is not placed in the same plane as the tip landmark. With this compensation the computed tip-position is the relative distance between both landmarks, projected in the plane of the manipulator tip.

Conversion factors to convert pixels to length were determined experimentally, similar to the repeatability measurement described in chapter 3.3.2. A known distance is compared to the measured number of pixels to yield the conversion factor. This experimental method was used instead of computing the conversion factors from field of view and range since the distance between LTS and manipulator is not easy to measure.

4.5. Initialization Module

The initialization module (file setup.c) combines routines for default system initialization, parameter modification, Jacobian calibration and system shut down.

The default system initialization (def_init) performs the following tasks: 1. The ring buffer for data storing is initialized. 2. The manipulator Jacobian is retrieved from data file. 3. The data acquisition hardware is initialized. 4. The user is prompted for the initial communication rate and the serial communication is initialized. 5. The user is prompted to reset the LTS, new defaults are written to the LTS and a first picture is taken (since the first picture contains garbage). The new camera defaults are stored in an external data structure (v_config) that is used as an internal representation. 6. The controller gains are set to default values. 7. The trajectory data structure is initialized. The control flag is set for holding and the desired joint angles are set to the current joint angles. 8. The 8259 interval timer is set to the controller sampling time. 9. The DOS timer interrupt service routine is replaced by the new routine for the control system.

The changes to the DOS defaults are reversed before the control system shuts down (clean_up): 1. The old DOS timer interrupt service routine is restored. 2. The 8259 interval timer is set to the DOS default of 55 ms. 3. Zeros are written to the output ports of the digital-to-analog converter to drive the robot to it's rest position. 4. DOS defaults for serial communication are restored.

Modifications of the system defaults can be chosen from the initialization menu. This menu offers the options parameters, terminal program and Jacobian calibration.

The first option calls the parameter menu that offers the following choices: 1. Controller gains. The controller gains are

stored in an external data structure and can be modified during controller operation. 2. LTS thresholds. Thresholds 0-5 from Table 3.1. can be modified with this option. The system writes the new threshold to the LTS and keeps an internal record (v_config) for display purpose. 3. Baud rate and data format for serial communication.

The Jacobian calibration option determines a constant value estimation of the manipulator Jacobian for the work space area observed by the LTS: 1. data points are collected, by moving the manipulator through a series of points and recording tip position and joint angles. 2. the Jacobian is determined from the least square estimation (Searle, S.R. and Hausman, W.H., 1970):

$$J^{*T} = (\Delta\Theta^T \Delta\Theta)^{-1} \Delta\Theta^T \Delta x \qquad (4.1.)$$

where J^* is the estimate of the Jacobian, $\Delta\Theta$ is the matrix of joint angle changes between consecutive data points and Δx is the matrix of tip position changes between data points. 3., the Jacobian can be stored to a data file for later use.

Table 4.1. Duration for trajectory planning

max joint angle difference for desired motion (rad)	duration (seconds)
0 - 0.04	$0.8 + \Delta\Theta / 0.05$
0.04 - 0.1	$0.8 + \Delta\Theta / 0.08$
> 0.1	$0.8 + \Delta\Theta / 0.1$

CHAPTER V

VERIFICATION OF CONCEPT

The static positioning algorithm was tested with two experiments. The first experiment investigated the conversion of the algorithm and the second experiment investigated the positioning repeatability under payload variation. The tip position of the manipulator is in the following assumed to be identical with the position of the landmark located at the manipulator tip.

5.1. Convergence of the Algorithm

To test the convergence of the positioning algorithm the robot was started from several locations inside and outside the camera work space and commanded to a tip-position specified in camera-coordinates. The positioning process was continued until the tip position error was less than 0.06" or the number of positioning iterations exceeded a limit of 5 iterations. The robot was monitored with joint sensors and Landmark Tracking System.

The tip-position during a typical positioning sequence shows Fig. 5.1. for the case that the robot is started with the tip inside the camera work space. The robot is started from the position 25.9",22.1" (outside the figure) and reaches the proximity of the desired position at 14.5", -4.55" after three positioning motions (an initial move and two small corrections). Joint angle, joint angle error and control action during the positioning are shown in Figures 5.2. - 5.4. The control actions are digital values; these values are converted to voltages by the digital-to-analog converter and sent to the amplifiers that control the hydraulic valves. Those plots show how the desired joint angles are updated to reach the desired tip position in spite of a steady state error of the joint control.

The tip-position during a positioning sequence started outside the camera work space is shown in Fig. 5.5. In this case the algorithm needs four corrections after an initial move to an estimated position.

During all experiments the algorithm positioned the robot with less than 5 iterations after the initial move. When started inside the camera work space the robot was typically positioned after three moves. Starting the robot outside the camera work space resulted in one or two additional iterations since the estimation of the first desired position does not utilize end-point position feedback. The remaining tip position error, as measured by the LTS, was typically 0.03".

5.2. Positioning Experiment

A second experiment investigated the positioning repeatability under payload variation. The static end-point positioning algorithm was compared to pure joint control. The robot was initially positioned at a rack in the work space and the position of a pointer attached to the tip was manually measured with a scale. Landmark position and joint angles at this position were measured for later use as desired inputs. Then the robot was repeatedly commanded to the initial position, using one of the two controller options. Joint angles, landmark position and scale measurement were reported after each positioning. The scale measurement was used as a quasi-reference; the resolution of this measurement is less accurate than the LTS, but it can be used to confirm the qualitative trend of the positioning repeatability. The same procedure was repeated with a 55 lb payload attached to the manipulator tip.

Measurement results are summarized in Table 5.1. The second column displays an estimate of the tip position (converted from joint angles by rigid kinematics), the third column displays the measured landmark position and the fourth column displays the reference measurements from the scale. The fluctuation of the tip position due to payload variation was computed from this data and is shown in Table 5.2.

The payload causes the tip to deflect vertically approximately one inch when only the joints are controlled. The end-point positioning algorithm is able to compensate for this deflection.

Table 5.1. End-point positioning repeatability under payload variation

end-point position in inches: mean (variance)	estimation (robot coord.) x , y	measurement (camera coord.) x , y	measurement (scale) x , y
joint command, no payload	-139.62 , 47.68 (0.024 , 0.042)	14.65 , -4.81 (0.013 , 0.016)	7.56 , 6.63 (- , -)
joint command, 55 lb payload	-139.66 , 47.55 (0.015 , 0.020)	14.77 , -5.64 (0.011 , 0.007)	7.75 , 5.69 (- , -)
tip command, no payload	-139.81 , 47.91 (0.017 , 0.015)	14.49 , -4.52 (0.005 ,0.018)	7.38 , 6.94 (- , -)
tip command, 55 lb payload	-139.96 , 48.79 (0.025 , 0.049)	14.50 , -4.49 (0.017 , 0.072)	7.40 , 6.85 (0.045 , 0.045)

conditions: desired position for joint command: -140.0", 48"
desired position for tip command: 14.5", -4.55"
mean and variance based on 10 measurements

Table 5.2. End-point position fluctuation under payload variation

end-point position fluctuation in inches, due to 55lb payload variation	measurement (camera coordinates) Δx , Δy	measurement (scale) Δx , Δy
joint command	0.12 , 0.83	0.19 , 0.94
position command	0.01 , 0.07	0.02 , 0.09

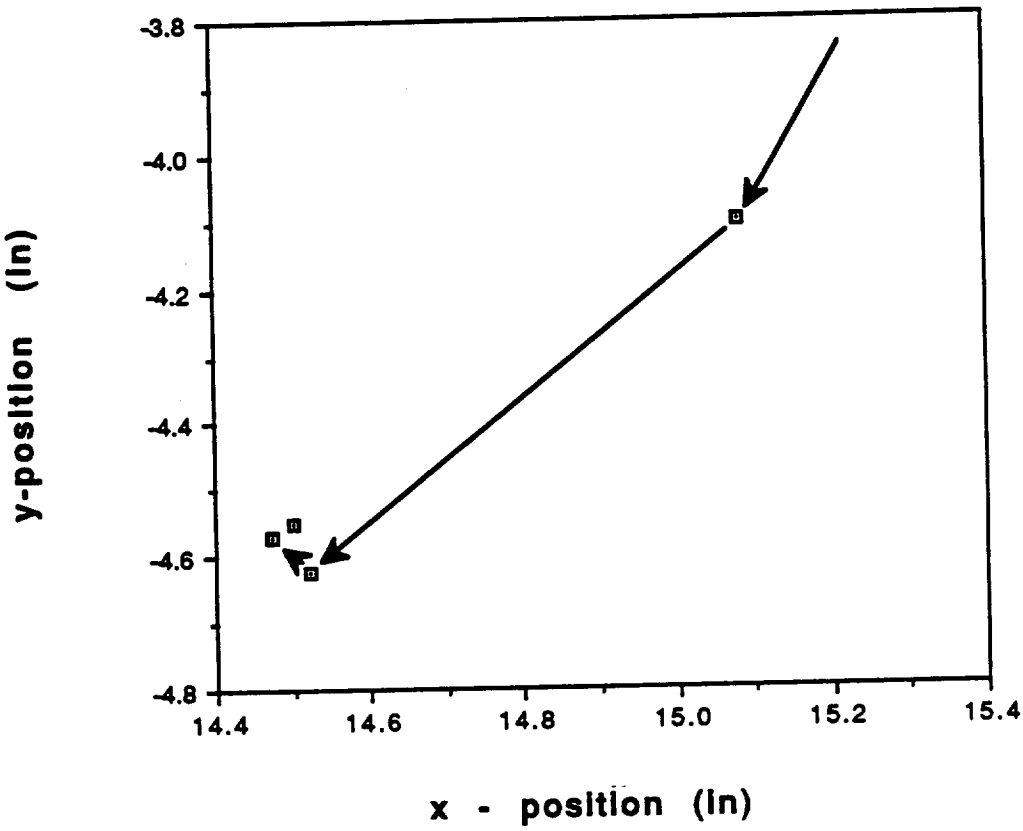


Figure 5.1. Tip-position during end-point positioning sequence (start inside camera work space)

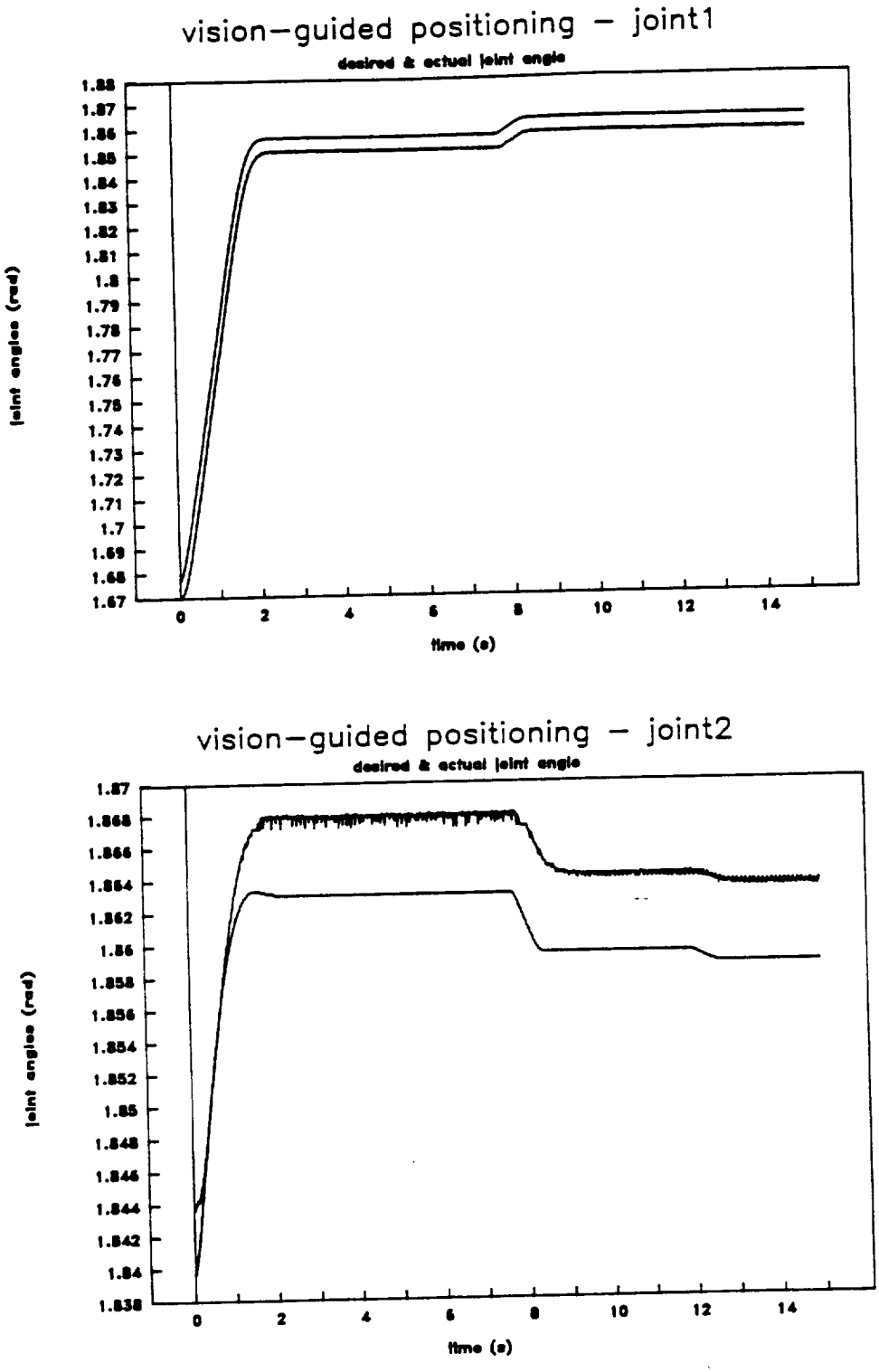


Figure 5.2. Desired and actual joint angles during end-point positioning sequence

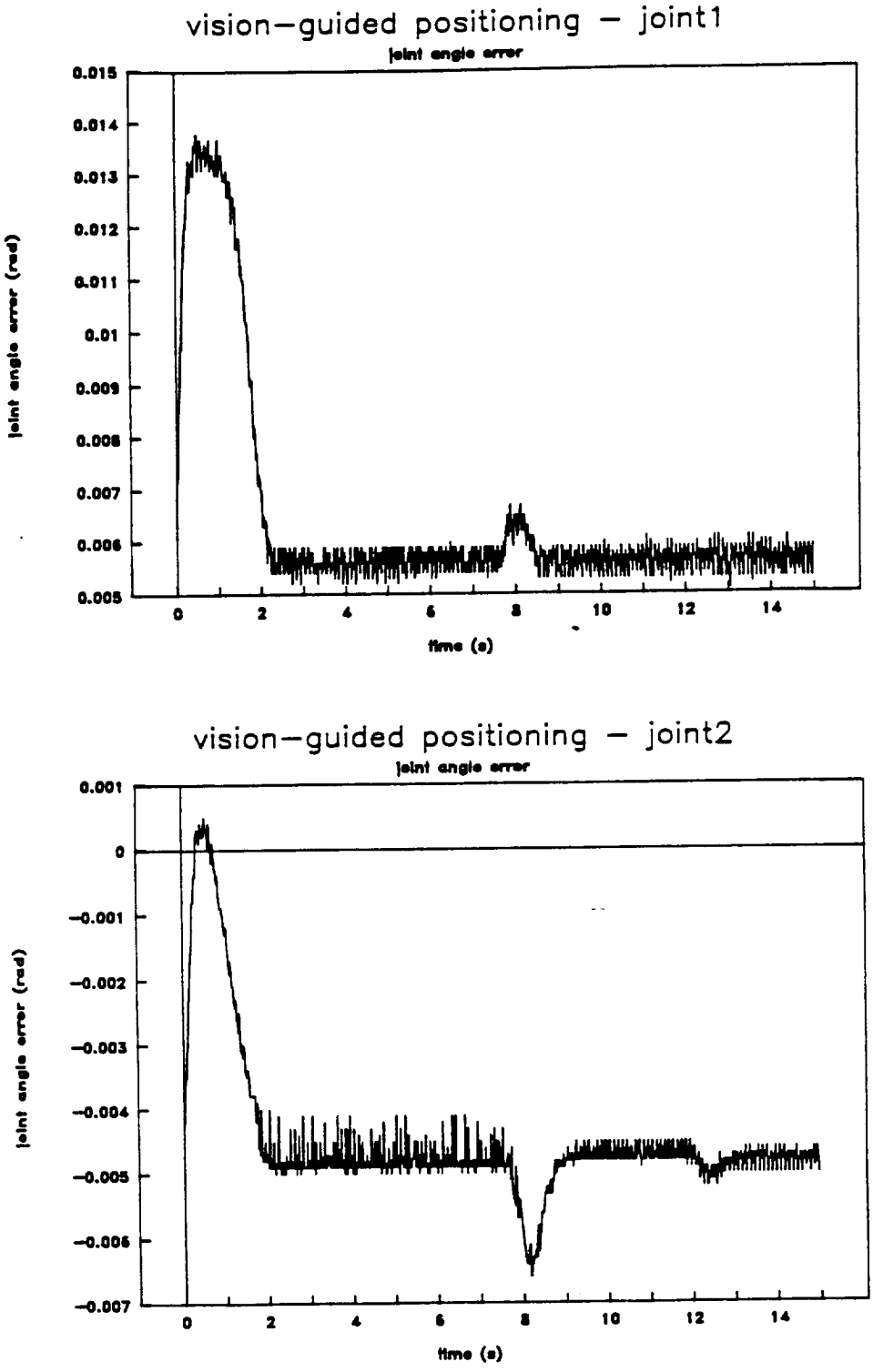


Figure 5.3. Joint angle error during end-point positioning sequence

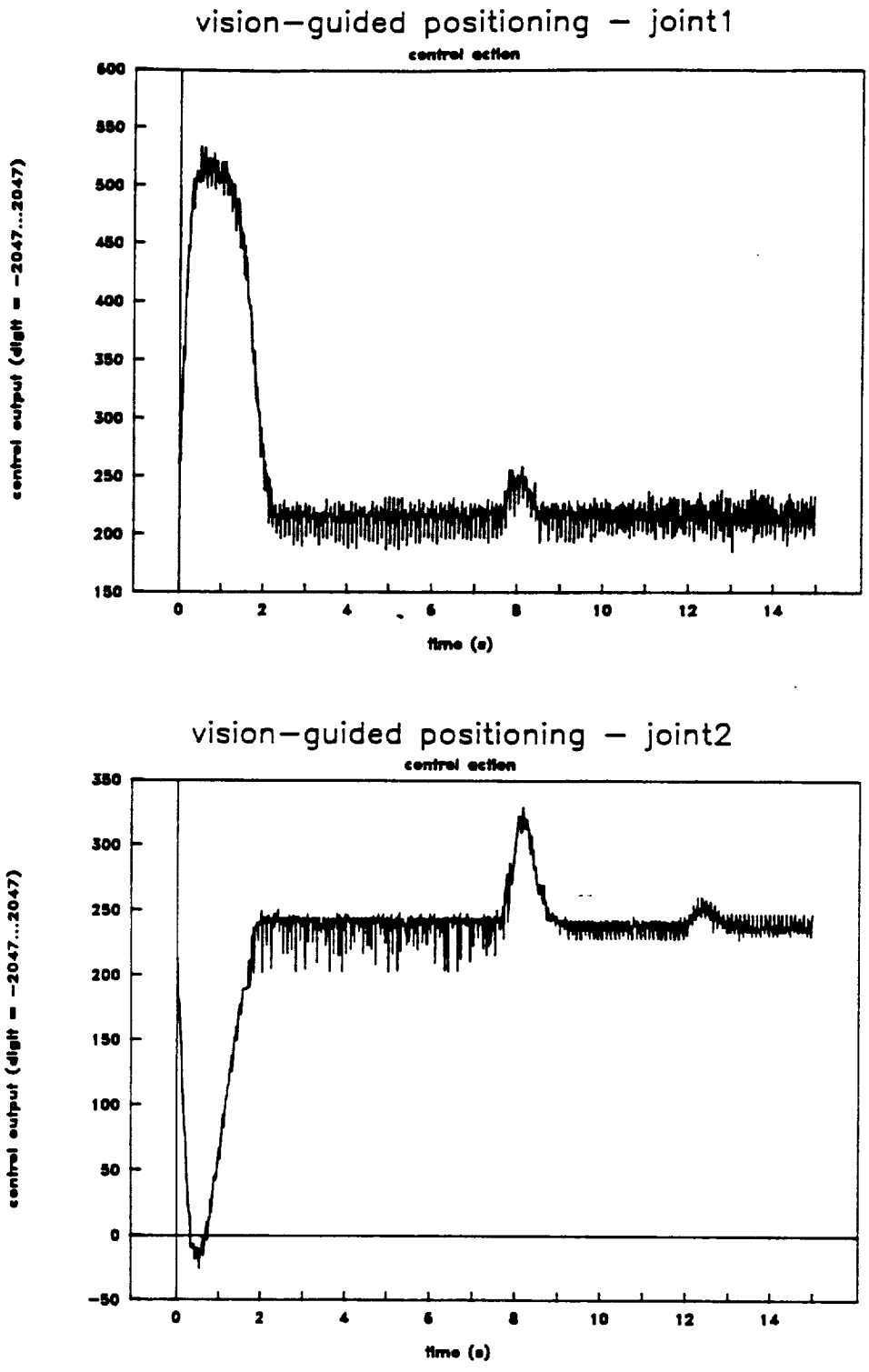


Figure 5.4. Control action during end-point positioning sequence

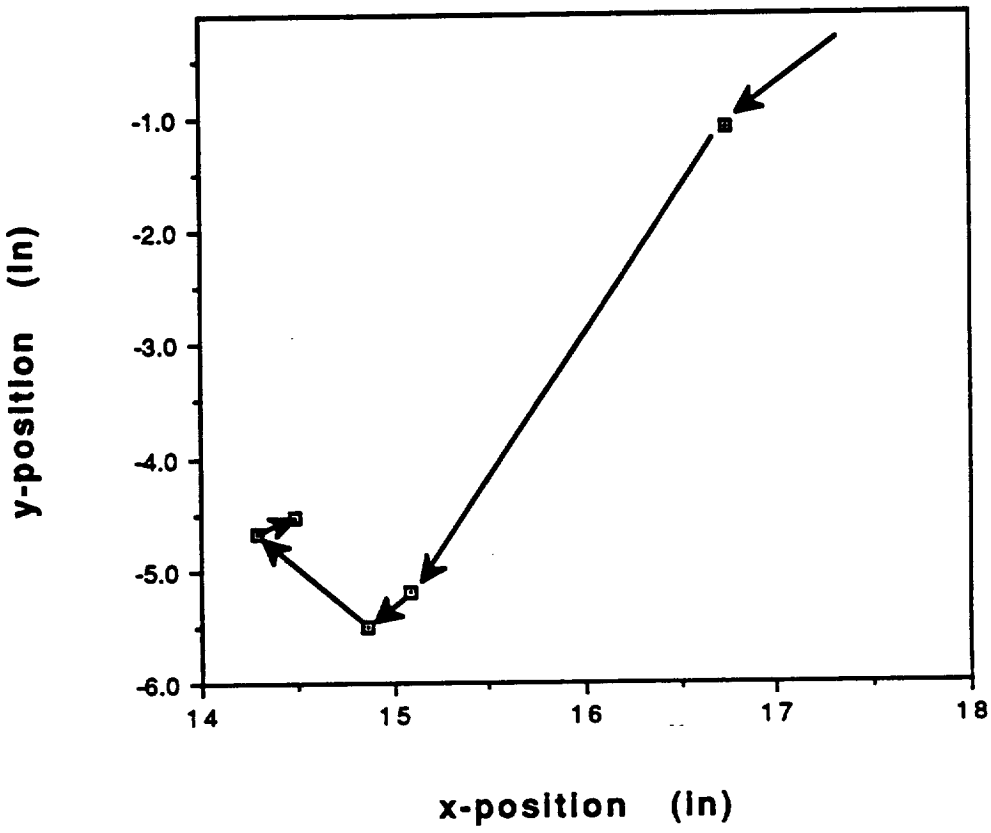


Figure 5.5. Tip-position during end-point positioning sequence (start outside camera work space)

CHAPTER VI

CONCLUSIONS AND RECOMMENDATIONS

A static end-point positioning control for a large two-link flexible manipulator was implemented and tested.

Feedback of the end-point position of the manipulator was provided from a Landmark Tracking System. Sampling time and repeatability of this sensor were investigated under setup conditions.

The sampling time was measured for illuminating a scene 6 feet from the LTS, scanning the whole image for two landmarks and returning area, peak and center of gravity of both landmarks as ASCII characters at a Baud rate of 19200. The minimum sampling time was 95 ms but for normal operation a sampling time of 300 ms was used due to power limitations of the strobe tubes used.

The repeatability was investigated by measuring the relative distance between two landmarks, 21 inches apart and 6 feet from the LTS. The variance of the measured distance was 0.166 mm in X-direction and 0.205 mm in Y-direction.

The controller was implemented using interrupts for joint control and communication with the LTS. The end-point positioning algorithm transforms the measured tip position error to joint angle error and utilizes this error to update the reference input to the joint controller. It was assumed that the tip-position is equivalent to the position of the landmark that is attached to the robot tip.

Convergence of the positioning algorithm was demonstrated for the manipulator work space observed by the LTS. The tip could be positioned with a positioning error of less than 1.5 mm after three iterations when the tip was initially in the area observed by the camera. One or two more iterations were necessary when the tip was initially outside this area.

The positioning repeatability was investigated under payload variation. The algorithm compensated for a deflection of approximately one inch, caused by attaching a 55 lb payload to the manipulator tip.

Several issues should be addressed in future extensions of this work:

The bottleneck of the current system is the landmark illumination. The strobe tube used is too weak for more rapid flashing and a more powerful tube did not produce sufficient intensity. Further testing of available flash tubes, different illumination techniques or the use of light sources instead of landmarks should be considered.

The communication between LTS and control system currently used disturbs the joint controller and limits the sampling rate of the position measurement. Handshaking should be implemented to assign different priorities to the interrupts. Assigning the highest priority to the timer interrupt results in a more constant joint controller sampling rate, since the timer interrupt would not be interrupted by the communication interrupt. No characters would be lost with handshaking since the communication would wait until the timer interrupt had finished.

The communication time should also be decreased by sending fewer characters. Three methods to achieve this are: 1. Use a new version of the LTS that times the data shifting with hardware and does not need the relative position measurement to yield a satisfying resolution. In this case only the data for one landmark is sent. 2. Perform more processing on the LTS board. Only the position would have to be send when error checking and computing the relative position could be performed by the LTS. 3. Send binary data instead of ASCII. The terminal option of the control system could be modified to interpret non ASCII data or multiple routine for position measurement and terminal operation could be stored on the LTS.

The onboard processing speed could be increased using a windowing algorithm that scans only a fraction of the image for a landmark or a faster processor could be used.

Implementing some of those recommendations should provide the necessary performance for future applications which could include: 1. measurement device for fast position measurements, 2. dynamic end-point position feedback, 3. extension to 3-dimensions and 4. increased work space.

APPENDIX A

REFERENCES

Auslander, D.M. and Tham, C.H. (1989): "Real Time Software for Control: Program Examples in C", Prentice-Hall, Inc., 1989.

Cannon, R.H., Jr. and Schmitz, E. (1984): "Initial Experiments on the End-Point Control of a Flexible One-Link Robot", International Journal of Robotics Research, Vol. 3, No. 3, Fall 1984.

Coulon, P.Y. and Nougaret, M. (1983): "Use of a TV Camera System in Closed-Loop Position Control of Mechanisms", in International Trends in Manufacturing Technology: Robot Vision, Pugh, A. Ed., IFS Publications Ltd, U.K., Springer-Verlag, 1983.

Dickerson, S.L. , Lee, E.H. , Ll, D.-R. and Single, T. (1990): "Landmark Tracking System for AS/RS", Material Handling Research Center Final Report, Georgia Institute of Technology, February 1990.

Dunbar, P. (1986): "Machine Vision - An Examination of what's new in Vision Hardware", BYTE, January 1986.

Eggebrecht, L.C. (1990): "Interfacing to the IBM Personal Computer - Second Edition", Howard W. Sams and Company, 1990.

Feddema, J.T. and Mitchell, O.R. (1989): "Vision-Guided Servoing with Feature-Based Trajectory Generation", IEEE Transactions on Robotics and Automation, Vol. 5, No. 5, October 1989.

Huggins, J.D. (1988): "Experimental Verification of a Model of a Two-Link Flexible, Lightweight Manipulator", M.S. Thesis, School of Mechanical Engineering, Georgia Institute of Technology, June 1988.

Kwon, D.-S. and Book, W.J. (1990): "An Inverse Dynamic Method Yielding Flexible Manipulator State Trajectories", Proc. of American Control Conference, (San Diego, CA), May 1990.

Lee, E.H. and Dickerson, S.L. (1990): "Pinhole Imaging for Industrial Vision System", Proc. of ASME Winter Annual Meeting, 1990.

Lee, H.G. , Kawamura, S. , Miyazaki, F. and Arimoto, S. (1990): "External Sensory Feedback Control for End-Effector of Flexible Multi-Link Manipulators", Proc of IEEE International Conference on Robotics and Automation, (Cincinnati, Ohio), May 1990.

Lee, K.-M. (1990): "A Low-Cost Flexible Part-Feeding System for Assembly Automation", Proc. of the 1990 Int. Conf. on Automation, Robotics and Computer Vision, Singapore, 19 - 21, September 1990.

Lee, K.-M., Blenis, R., Li, D.-R., Yutkowitz, S., Motaghedi P. (1991): "A Low-Cost Flexible Part-Feeding System for Machine Loading and Assembly", Material Handling Research Center Final Report (Draft), TR-9106, Georgia Institute of Technology,1991.

Nam, Y. and Dickerson, S.L. (1991): "Position Estimation with Accelerometer and Vision Measurement", to appear in Proc. of ASME Winter Annual Meeting, (Atlanta, GA), December 1991.

Oakley, C.M. and Cannon, R.H., Jr. (1988): "Initial Experiments on the Control of a Two-Link Manipulator with a very Flexible Forearm", Proc. of American Control Conference, (Atlanta, GA), June 1988.

Oakley, C.M. and Cannon, R.H., Jr. (1989): "End-Point Control of a Two-Link Manipulator with a very Flexible Forearm: Issues and Experiments", Proc. of American Control Conference, (Pittsburgh, PA), June 1989.

Oakley, C.M. and Barratt, C.H. (1990): "End-Point Controller Design for an Experimental Two-Link Flexible Manipulator Using Convex Optimization", Proc. of American Control Conference, (San Diego, CA), May 1990.

Rattan, K.S. , Fellu, V. and Brown, H.B., Jr. (1990): "Tip Position Control of Flexible Arms Using a Control Law Partitioning Scheme", Proc. of IEEE International Conference on Robotics and Automation, (Cincinnati, Ohio), May 1990.

Sanderson, A.C. and Weiss, L.E. (1983): "Adaptive Visual Servo Control of Robots", in International Trends in Manufacturing Technology: Robot Vision, Pugh, A. Ed., IFS Publications Ltd, U.K., Springer-Verlag, 1983.

Searle, S.R. and Hausman, W.H. (1970): "Matrix Algebra for Business and Economics", Wiley-Interscience, New York, 1970.

Smith, A.L. (1991): "Perception-Based Endpoint Sensing and Control", Ph.D. Thesis Proposal, Georgia Institute of Technology, May 1991.

Tompkins, W.J. and Webster, J.G. (1987): "Interfacing Sensors to the IBM PC", Prentice_Hall, Inc., 1987.

Weiss, L.E. , Sanderson, A.C. and Neuman, C.P. (1987): "Dynamic Sensor-Based Control of Robots with Visual Feedback", IEEE Journal of Robotics and Automation, Vol. RA-3, No. 5, October 1987.

Wilson, T.R. (1985): "The Design and Construction of a Flexible Manipulator", M.S. Thesis, School of Mechanical Engineering, Georgia Institute of Technology, December 1985.

Yuan, B.-S. (1989): "Adaptive Strategies for Position and Force Controls of Flexible Arms", Ph.D. Thesis, School of Mechanical Engineering, Georgia Institute of Technology, March 1989.

Zhang, D.B. , Van Gool, L. and Oosterlinck, A. (1990): "Stochastic Predictive Control of Robot Tracking Systems with Dynamic Visual Feedback", Proc. of IEEE International Conference on Robotics and Automation. (Cincinnati, Ohio), May 1990.

APPENDIX B

EQUIPMENT LIST

The following equipment and software products were added to the control environment for RALF, compare Huggins (1988) and Wilson (1985) for a listing of the original equipment.

- 386-AT, 25 MHz CPU and math-coprocessor, 4 MByte RAM
- MetraByte DAS-16 12 bit data acquisition board with control software (0.4 ms sampling time for a 8 channel scan)
- Interface for digital-to-analog conversion with power supply. This Op-Amp circuit (inside DAS-16 interface box) boosts the output voltage from (0 .. 5) V to (-10 .. +10) V to match the output from DAS-16 with the existing VAX-control-system output.
- Landmark Tracking System with power supply

- Illumination unit consisting of power unit (circuit diagram in Fig. B.1., technical data sheet for trigger transformer in Fig. B.2.) and strobe tube (technical data sheet in Fig. B.3.). Strobe tubes and trigger transformer were purchased from

  Shokai Far East LTD.

  9 Elena Court

  Peekskill, NY 10566

  (914) 736-3500

  The parts used have the following catalog numbers:

straight strobe tube	SFT 3534T
U-shaped strobe tube	SFT 112M
trigger transformer	STR 6KS/RH1

- Borland, Turbo C ++ and Turbo Assembler

- Magna Carta Software, C Communications Toolkit

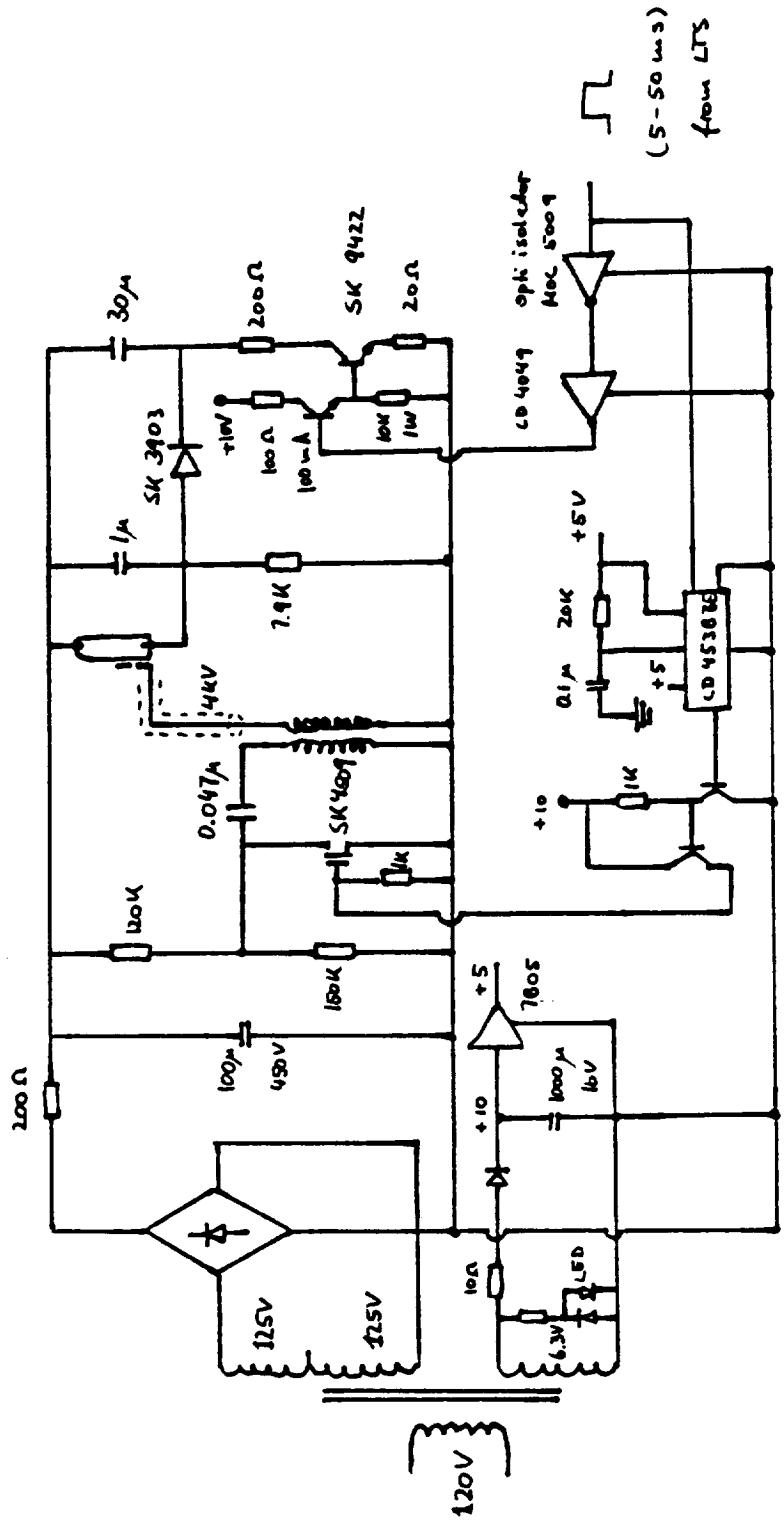
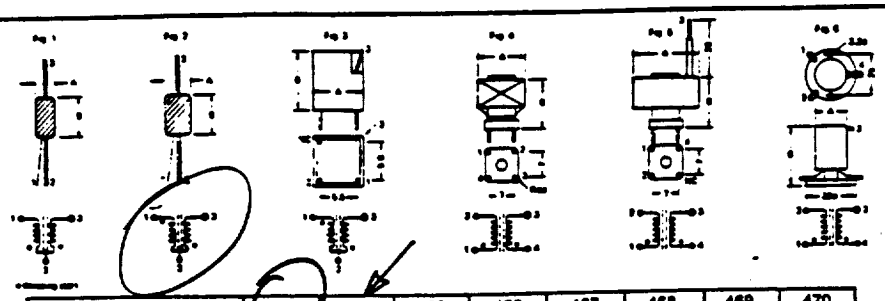


Figure B.1. Circuit diagram for strobe power unit, according to K.-M. Lee et al (1991)

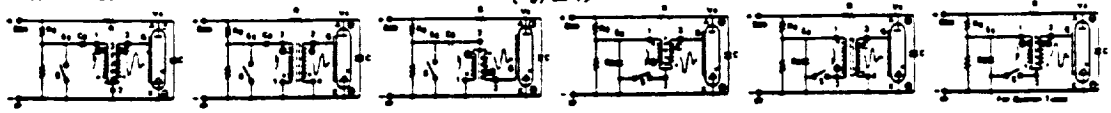
TRIGGER TRANSFORMERS (Old Type TR6KN is Superseded by TR6KS/RH1)



Line No	463	464	465	466	467	468	469	470
Shokai lamp No. / Specifications	STR 4KN	STR 6KS/RH1	STR 6KS TC8	STR 4KM	STR 6KM	STR 10KM	STR 15K	STR 20K
Max. Input Energy mJ (μF/Vac)	1.48 (0.033/300)	2.1 (0.047/300)	2.1 (0.047/300)	1.47 (0.047/280)	1.47 (0.047/280)	4.5 (0.1/300)	6.1 (0.1/280)	20 (0.2/450)
Normal Input Energy mJ (μF/Vac)	0.8 (0.033/220)	1.1 (0.047/220)	1.47 (0.047/280)	0.9 (0.047/200)	0.9 (0.047/200)	3.1 (0.1/250)	4.5 (0.1/300)	16 (0.2/400)
Peak Output Voltage of First Pulse at Normal Energy KVP	−4	−6	−8	−4	−6	−10	−15	−20
First Output Pulse Width (10) μS	0.7	0.9	1.0	0.7	0.8	1.4	3.0	4.6
Dimensions A mm	7φ	8φ	7.5	14φ	18φ	19φ	20φ	20φ
Dimensions B mm	16 Max	12 Max	8.5	15	15	15	35	35
Fig No	1	2	3	4	4	5	6	6

In Stock, 10$ each (5/LOT) → Not in stock, 10$ each.

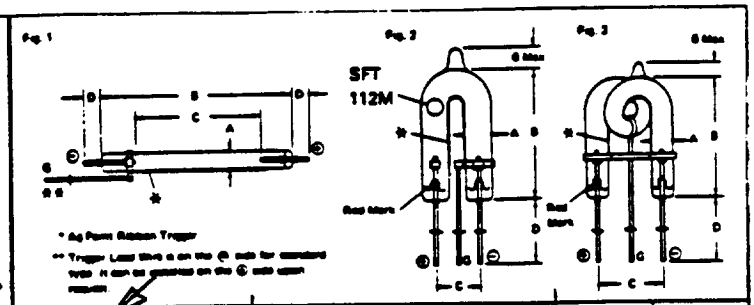
Typical Trigger Circuits



Note: Pulse Test Condition: Output impedance Z_L = 100Ω 15 PF (= typical trigger impedance of flashtubes)

Figure B.2. Trigger transformer, technical data sheet, Shokai Far East LTD.

HARD GLASS, STRAIGHT, U-SHAPED & HELICAL. FOR HIGH FPS TIMING & SIGNAL LIGHT.



Specifications	Line No.	420	421	422	423	424	425	426	427
	Shokai lamp No.	SFT 3522T	SFT 3534T	SFT 4044T	SFT 106MS	SFT 112M	SFT 1210	SFT 151M	SFT 151MH
Design Anode Voltage	Vdc	300	300	300	300	300	300	450	450
Min. Anode Voltage	Vdc	250	250	250	200	250	250	350	350
Max. Anode Voltage	Vdc	350	350	400	400	400	400	550	550
Max. Flash Energy For Single Shot	Joule	12	20	45	45	45	/	200	250
Normal Flash Energy For High FPS Shot	Joule	0.004	0.006	0.012	0.02	0.023	0.05	0.075	0.1
Max. Flash Rate at Normal Flash Energy	FPS	250	250	250	250	300	100	200	200
Ave. Power Input Max. (Joule x FPS)	W	1.0	1.5	3	5	10	5	15	20
Min. Trigger Pulse Voltage	KVP	−4	−4	−4	−4	−6	−4	−6	−6
Dimensions A	mm±	3.5	3.5	4.0	6.2	6.0	6.2	6.2	6.2
Dimensions B	mm	22	34	44	32	38	35	43	43
Dimensions C	mm	13	20	29	10	12	12	18	18
Dimensions D	mm	3	3	3	38	38	38	38	38
Dimensions E	mm	/	/	/	/	(Non polarity)		/	/
Fig. No.		1	1**	1	2	2	2 (No Trigger coating)	3	3
Ave. Flash Life at Normal Flash Energy	Flashes	100 Hrs	100 Hrs	100 Hrs	100 Hrs	100 Hrs	50 Hrs	100 Hrs	100 Hrs

Figure B.3. Strobe tubes, technical data sheet, Shokai Far East LTD.

APPENDIX C

USER MANUAL

C.1. Installation and System Start

The following procedure is recommended to start the control system:

1. Connect the blue DAS-16 interface box to the PC and turn on the power supply for the interface.

2. Connect manipulator sensors and actuators to DAS-16, i.e. connect cable from DAS-16 interface and cable from manipulator interface rack with 26 pin connectors.

3. Connect the serial line from LTS to com 1 port of PC.

4. Turn on power for LTS and strobe unit.

5. Turn on power for PC

6. Go to sub-directory "c:\klaus\control4"

7. Write zeros to the output ports of the digital-to-analog converter to set the voltage on the amplifiers for the hydraulic valves to zero (this prevents uncontrolled motion of the manipulator): execute program "d2atest" and type 0,0 on prompt.

8. Turn on power for manipulator interface rack and turn on valve amplifiers.

9. Turn on hydraulic unit.

10. Reset emergency switch, push button on interface rack (LED goes on).

11. Test connections by writing a voltage to the amplifiers and checking if manipulator moves. Use program "d2atest" to send voltages (recommended test values: 300,600 for lower and upper joint respectively)

12. If manipulator does not move, check 26 pin connector between DAS-16 interface and manipulator and switch settings of experimental setup on top of the manipulator interface rack ("shake"-test).

13. Start controller by executing program "main".

C.2. Operation

All control system options are chosen from menus. After start up the program prompts for communication rate and initial joint angles before displaying the main menu.

First, the user is prompted for the Baud rate and some options are listed, 19200 Baud is usually used. Check switches on the LTS board for the Baud rate used if problems are encountered, switch settings and corresponding Baud rates are listed in Table C.1.

Second, the user is prompted to reset the LTS. (reset button on LTS board or turn power off and on). This starts the controller with a known LTS configuration. New defaults are sent to the camera after reset. The LTS also takes a first picture, therefore the strobe should flash.

Third, the user is prompted to confirm the (desired) initial joint angles, the manipulator reads and displays the current position (which should be equivalent to the rest position if these instructions were followed). The displayed joint angles should be confirmed otherwise the program terminates.

The main menu is displayed after the initial user input, this menu offers the following options:

I **Initialization menu**, displays the initialization menu.

D **Debbugging menu**, displays the debugging menu.

M **Move robot**, this is the joint control option described in chapter 4.2, the robot is moved to a user specified position (joint angles or estimated tip-position) in a user specified time. Returns to main menu.

C **Camera assisted move**, this is the end-point positioning control option described in chapter 4.2, the robot tip is positioned at a user specified position (in camera coordinates). The camera work space boundaries are displayed to guide the user with the position input. The algorithm prompts the user for confirmation when a position outside the work space is chosen. Returns to main menu, an error message is displayed when the manipulator tip couldn't be positioned.

R **Read a2d-board, compute position**, this option read the joint position sensor and estimates the current manipulator tip-position (in robot coordinates) based on rigid kinematics. Returns to main menu.

P **get tip Position**, this option takes a picture and returns the measured tip-position (in camera coordinates). Returns to main menu.

F **repeated Fast position measurements**, this option measures the manipulator tip-position repeatedly and displays the values on the screen. The current sampling time is limited to 300 ms. Returns to main menu.

T **read Timer**, displays the time since program start. Returns to main menu.

Q **Quit**, returns to DOS prompt.

The initialization menu offers the following options:

P **Parameters**, displays the parameter menu.

T **Terminal program**, PC is used as a terminal to communicate with the LTS. The software commands described in chapter 3.2.3. can be used to test the camera setup. This option should be used carefully since the control system does not keep track of LTS configuration changes. The **SHOW (Sn)** command can not be used from this option because too much data is transferred and the program "crashes". Returns to initialization menu when "<alt> + <m>" is pressed.

to 12 calibration points (tip-positions) inside this area. The positions are compared to the boundary and the user is prompted until all points are inside the boundary. Then the manipulator is moved through all calibration points and the Jacobian is computed from the recorded tip-positions and joint angles according to eq. 4.1. This Jacobian can be stored to a data file for later use. Returns to initialization menu.

H **Help**, this option is currently not available.

Q **Quit**, returns to main menu.

The debugging menu combines several test options that were used during software development (all options return to debugging menu):

T **Trajectory computation test**, tests if interrupt functions create errors in data computed by background functions.

E **Error code**, prints out error code from timer interrupt service routine; this was used since "direct" debugging of interrupt functions is not possible.

F **read hold Flag**, this option prints the value of the trajectory control flag to the screen.

W **Write data to disk**, this option enables data storing to the ring buffer.

Q **Quit**, returns to main menu.

The parameter menu displays the current parameters and enables parameter changes:

A **Proportional gain joint 1**, sets the controller gains described in chapter 4.2.

B **Proportional gain joint 2**, sets the controller gains described in chapter 4.2.

C **Min pixel value to start growing landmark**, sets LTS threshold #0, compare Table 3.1.

D **Min pixel value to be included in landmark**, threshold #1

E **Min area of landmark to be reported**, threshold #2

F **Integration time**, threshold #3

G **Strobe charge time**, threshold #4

H **Repetition factor for integration**, threshold #5

J **(data) Format**, displays menu to chose data format for serial communication.

K **Speed**, displays menu to chose Baud rate for serial communication.

Q **Quit**, returns to initialization menu.

Table C.1. Switch setting for Baud rate selection

Baud Rate	S1	S2	S3	S4	S5	S6	S7	S8
57600				X				
38400	X		X	X				
28800		X			X			
23040	X	X	X		X			
19200			X	X	X			
9600		X		X	X	X		
4800		X	X		X	X	X	
2400		X	X	X		X	X	X

Note: Switch (S) ON = X

OFF = empty box

APPENDIX D

SOFTWARE LISTINGS

```
/*****************************************************************************
FILE
  main.h  - system parameter and data type declarations
  Klaus Obergfell, 05/02/91

REMARKS
  header file for RALF control system, contains general system parameter
  (constants) and data type declarations (structure declarations).
  Function prototypes and external data are included in file specific
  header files, not in this file.

UPDATES
  07/16/91  change struct traj, storing x,y only in debug mode
  07/14/91  change struct data to implement ring buffer
  07/13/91  big clean up
  06/25/91  merge control.h & vision.h
  06/17/91  include struct status
  05/30/91  change PERIOD to SAMPLE_T
  05/28/91  change traj data structure to array version
  05/27/91  only constants, definitions, data structures
            no more function prototypes, no includes
  05/26/91  include data structures for interrupt driven
            controller, change function prototypes
*****************************************************************************/

/*****************************************************************************
IMPORTS
*****************************************************************************/
#include "envir.h"      /* compiler options */

/*---------------------------------------------------------------------------
SYSTEM PARAMETER
---------------------------------------------------------------------------*/

#define SAMPLE_T        5.0  /* controller sampling time in ms */
#define SAMPLE_FREQ     200  /* controller sampling frequency in Hz */
#define DOS_TIMER_FREQ  11   /* multiplication factor for chaining of isr's
                                to old DOS timer isr:
                                SAMPLE_T * DOS_TIMER_FREQ = 55 ms ! */
#define DATA_T          4    /* multiplication factor for data saving:
                                data saving freq. = SAMPLE_FREQ / DATA_T */
#define DATAPOINTS      800  /* max. data points for data saving, capacity
                                of ring buffer depends on DATAPOINTS and
                                the number of parameters saved (2 int,
                                5 double on 07/13/91) */
#define MAXPOINTS       200  /* max. traj points */

/*
REMARK
  experienced memory limitations for DATAPOINTS. With MAXPOINTS = 200
  and 7 parameters saved:
  small memory model approx. 1000
  huge  memory model approx. 1600
  Currently use small model, for higher speed and safety (I am not
  shure about huge model and interrupts etc., but it worked fine in
  a first test on 06/21/91, with no serial communication included).

  The compact, medium and large memory modell don't support more
  static data than the small modell does. The only differences are
  in the memory size for code and automatic data.
*/


#define MENU          ALT_M    /* key to return to menu */
#define RXBUFLENGTH   2048     /* serial comm. buffer size */
#define LM_EXSPECTED  2        /* number of landmarks exspected in field of
                                  view
                                  !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
                                  procedures are not yet designed
                                  to allow LM_EXSPECTED other than 2
                                  !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!! */

#define LM_MAX        5        /* max number of landmarks processed in
                                  field of view */
#define LM_WAIT       0.3      /* max. waiting time for landmark data to return */
#define LM_REPETITION 5        /* max. number of attempts to detect exspected
                                  number of landmarks */

#define pi      3.141592654
#define True    1
#define False   0

/*---------------------------------------------------------------------------
DATA TYPE DECLARATIONS
---------------------------------------------------------------------------*/

struct traj
{
  int       flag;               /* status: 0 = follow traj */
                                /*         1 = done, hold at last traj
                                             point */
  int       points;             /* number of traj points */
  int       iterations;         /* number of iterations between points */
  int       holds;              /* number of holds after completing traj */

#if OPTION_DEBUG

  double    data[MAXPOINTS][4]; /* traj data array */
                                /* ..[0] = theta1 */
                                /* ..[1] = theta2 */
                                /* ..[2] = x      */
                                /* ..[3] = y      */

#else

  double    data[MAXPOINTS][2]; /* traj data array */
                                /* ..[0] = theta1 */
                                /* ..[1] = theta2 */

#endif

};

struct data
{
```

```
  int      s_flag;              /* status flag: 0 = do not save data
                                                1 = save data */
  int      f_flag;              /* full flag: 0 = buffer is not full
                                                1 = buffer is full */
  int      buffer_tail;         /* (one behind) next out of buffer location */
  int      buffer_head;         /* next in buffer location */
  double   ddata[DATAPOINTS][5]; /* time, actual & desired angles */
  int      idata[DATAPOINTS][2]; /* control output */
};

struct gains
{
  double    p1;            /* proportional gain link 1 */
  double    p2;            /* proportional gain link 2 */
};

struct camera
{
  char minval_grow[2];          /* min pixel value to start growing a blob */
  char minval_incl[2];          /* min pixel value to be included in blob */
  char minarea[2];              /* min area of blob to be reported */
  char integ_time[2];           /* integration time */
  char strobe_charge_time[2];   /* strobe charge time */
  char mult_integ[2];           /* repetition factor for integration time */
  char camera_output[2];        /* byte determines camera output */
   /*
   most significant nibble doesn't matter -> set to 'F'
   least significant nibble (output if bit cleared, no output otherwise):
   bit 4: x - center of gravity
   bit 3: y - center of gravity
   bit 2: peak value in blob (landmark)
   bit 1: area of blob (landmark)
   */
};

struct landmark
{
  int area;         /* area of landmark */
  int peak;         /* peak of landmark */
  double y_cgt;     /* y or row center of gravity */
  double x_cgt;     /* x or column center of gravity */
};

/*****************************************************************************
end of file main.h
*****************************************************************************/
```

```
/*****************************************************************************
FILE
  main.c - supervisor for RALF controller
  Klaus Obergfell 1991

ROUTINES
  main      - main program, choose from menu
  menu      - prints main menu

UPDATES
  07/20/91, transducer access only in tmr_isr
  07/19/91, debug menu
  07/12/91, big clean up
*****************************************************************************/

/*****************************************************************************
IMPORTS
*****************************************************************************/
#include "envir.h"        /* system environment, comiler options */

#include "main.h"         /* parameters */

#include "control2.h"     /* function prototypes, external data */
#include "setup.h"
#include "util.h"
#include "convers.h"
#include "move_rob.h"
#include "vstuff.h"
#include "debug.h"

#include <stdlib.h>       /* standard declarations */
#include <stdio.h>
#include <conio.h>
#include <ctype.h>
#include <bios.h>
#include <dos.h>

/*****************************************************************************
FUNCTION PROTOTYPES
*****************************************************************************/
static void menu (void);

/*****************************************************************************
VISIBLE ROUTINES
*****************************************************************************/

/*----------------------------------------------------------------------------
ROUTINE
  MAIN
  Klaus Obergfell

DESCRIPTION
  calls default initialization, waits till user choses option from
  supervisor menu, calls option procedure, loops thru until quit
  option is selected, calls clean up procedure, terminates program.
----------------------------------------------------------------------------*/
void main(void)
{
```

```
  short   key, DONE = false;

#if OPTION_DAS16
  double  temp1,temp2,temp3,temp4;
#elif OPTION_INT
  double  temp1;
#endif

  def_init();
  do
  {
  clrscr();
  menu();
  key = bioskey(0) & 0xff;
  switch (toupper(key))
    {
      case 'I':  /* Initialization menu */
        do_init();
        break;
      case 'D':  /* Debbugging menu */
        do_debug();
        break;

      case 'M':  /* Move robot */
        move_robot();
        wait_key_press();
        break;

#if (OPTION_INT && OPTION_DAS16 && OPTION_COMM)
      case 'C':  /* Camera assisted move */
        vision_guided_pos();
        wait_key_press();
        break;
#endif

#if OPTION_DAS16
      case 'R':  /* access angle data, compute position */
        disable();
        temp1 = th1actual;
        temp2 = th2actual;
        enable();
        forward(temp1,temp2,&temp3,&temp4);
        printf("\n\nCurrent x,y-position: x = %.3f inch  y = %.3f inch",
          temp3,temp4);
        printf("\n                      (th1 = %.1f deg  th2 = %.1f deg)",
          temp1*180./pi,temp2*180./pi);
        wait_key_press();
        break;
#endif

#if OPTION_COMM
      case 'P':  /* get tip Position */
        report_tip_position(&port1);
        wait_key_press();
        break;
      case 'F':  /* repeated, fast position measurements */
        report_tip_position_2(&port1);
        wait_key_press();
        break;
#endif

#if OPTION_INT
      case 'T':  /* read Timer */
        disable();   /* critical region */
        temp1 = timer;
        enable();
        printf("\n\ntime since program start = %.3f seconds",temp1);
        wait_key_press();
        break;
#endif

      case 'Q':  /* Quit */
        DONE = True;
        break;
    }
  } while(!DONE);

  clean_up();

  /* check if any unwritten data */
  if (data1.s_flag)       /* check status of data taking facility */
  {
    printf("\n\nwriting old data to file");
    w_data_to_disk(&data1);    /* write old data to disk */
  }

  return;
} /* end of routine main */

/*****************************************************************************
PRIVATE ROUTINES
*****************************************************************************/

/*---------------------------------------------------------------------------
ROUTINE
  MENU
  Klaus Obergfell

DESCRIPTION
  prints main menu
---------------------------------------------------------------------------*/
static void menu (void)
{
  printf("\nRALF supervisor menu, choose option:\n");
  printf("\nI.    Initialization menu");
  printf("\nD.    Debugging menu");
  printf("\n");
  printf("\nM.    Move robot");

#if (OPTION_INT && OPTION_DAS16 && OPTION_COMM)
  printf("\nC.    Camera assisted move");
#endif

  printf("\n");

#if OPTION_DAS16
```

```
  printf("\nR.    Read a/d - board, compute position");
#endif

#if OPTION_COMM
  printf("\nP.    get tip Position");
  printf("\nf.    repeated, fast position measurements");
#endif

#if OPTION_INT
  printf("\nT.    read Timer");
#endif

  printf("\n");
  printf("\nQ.    Quit (Exit program)\n");
  return;
} /* end of routine menu */

/*****************************************************************************
end of file main.c
*****************************************************************************/
```

```
/*****************************************************************************

FILE
    envir.h  -  defines program environment

LAST UPDATE
    07/11/91, Klaus Obergfell
      include program options, take out operating system and machine
      type, take out some compiler options
    05/30/91, Klaus Obergfell
      include compiler option TURBO C
    16 August 1987
        remove unnecessary clutter

    Copyright(c) 1985,1986,1987  D.M. Auslander and C.H. Tham

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

/*************************   Operating System   *************************/

/*#define UNIX     0*/        /* 4.2 BSD, implies UNIX C compiler */
/*#define PCDOS    1*/        /* includes generic MSDOS family */
/*#define CPM      0*/        /* the CP/M family, including MP/M */


/*********************   Hardware or Machine Type   *********************/

/*#define IBMPC        1*/       /* standard PC, PC/XT, PC/AT */
/*#define COMPUPRO     0*/       /* Compupro 8086 or Dual Processor */
/*#define INTEL310     0*/       /* Intel 310 development system */


/*****************************   Compilers   *****************************/

#define MICROSOFT   0        /* Microsft C ver. 4.00 */
#define TURBO_C     1        /* Turbo C ver. 2.0 and TC++ ver. 1.0 */

#define ANSI                 /* use ANSI C features */


/*************************   Program Options   *************************/

#define OPTION_INT       1   /* timer interrupt */
#define OPTION_ALARM     1   /* set timer with alarm */
#define OPTION_COMM      1   /* serial communication */
#define OPTION_DAS16     1   /* data acquisition board */
#define OPTION_DEBUG     0   /* debugging */
#define OPTION_DATA      1   /* store data to disk */
#define OPTION_SCR       1   /* print information to screen */
#define OPTION_VGPDATA   1   /* store vision guided pos. data */


/*****************************************************************************
end of file envir.h
```

```
/*****************************************************************************

FILE
    inout.h  -  maps compiler's 8-bit i/o function names

    revision 05/30/91 include TURBO C compiler

REMARKS
    Different compilers call their 8-bit port i/o routines by different
    names.  This file contains macros to map these names to the generic
    in() and out().  This makes programs much more portable across
    different compilers

    Note: compiler #define's must be placed before this file.

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


#if CIC86            /* Computer Innovations C86 */

#define in(port)              inportb((unsigned)(port))
#define out(port, value)      outportb((unsigned)(port), value)

#endif

#if DESMET           /* Desmet C  */

#define in(port)              _inb(port)
#define out(port, value)      _outb(port, value)

#endif

#if LATTICE          /* Lattice C (Version 2.15) */

#define in(port)              inp(port)
#define out(port, value)      outp(port, value)

#endif

#if MICROSOFT        /* Microsoft C Version 4.00 and 5.00 */

#include <conio.h>
#define in(port)              inp(port)
#define out(port, value)      outp(port, value)

#endif

#if TURBO_C          /* Turbo C Version 2.0 */

#include <dos.h>
#define in(port)              inportb(port)
#define out(port, value)      outportb((port),value)

#endif
```

```
/*****************************************************************************
FILE
  control2.h - function prototypes and external data of file control2.c
  Klaus Obergfell 1991
*****************************************************************************/

/*---------------------------------------------------------------------------
FUNCTION PROTOTYPES
---------------------------------------------------------------------------*/
extern void interrupt tmr_isr();

/*---------------------------------------------------------------------------
EXTERNAL DATA
---------------------------------------------------------------------------*/
extern struct traj traj1;           /* trajectory data = control input */
extern struct data data1;           /* data = control performance output */
extern struct gains gains1;         /* controller gains */
extern double th1actual,th2actual;  /* actual (measured) joint angles in rad */
extern double th1d_hold,th2d_hold;  /* desired joint angles during hold */
extern double timer;                /* keeps track of time in seconds */

extern int c_points;                /* counter for traj points */
extern int c_iterations;            /* counter for iterations */
extern int c_holds;                 /* counter for holds after traj */

#if OPTION_DEBUG
extern int error_a2d_1;           /* for debugging */
extern int error_a2d_count_max_1;
#endif


/*****************************************************************************
end of file control2.h
*****************************************************************************/
```

```
/*****************************************************************************
FILE
  control2.c - real time control module
  Klaus Obergfell 1991

ROUTINES
  tmr_isr      - timer interrupt service routine
  control      - control algorithm
  get_traj     - desired joint angles from trajectory structure
  store_data   - stores data to buffer
  multiply     - matrix multiplication

REMARKS
  this real time control module implements the manipulator joint controller.
  The desired joint angles are computed by a supervisor program. Those
  programms communicate thru the following external variables:

  traj1   a structure containing an array of desired joint angles and
          control flags. This is the reference input to the controller.
  data1   a structure containing a ring buffer and control flags to store
          data collected by the controller. This data is read by the
          supervisor and stored to file.
  gains1  a structure containing the controller gains. The gains can be
          modified by the supervisor.

  th1d_hold,th2d_hold  the desired joint angles during holding operation
  th1actual,th2actual  the actual (measured) joint angles. The transducers
                       are only read inside this module, when this data
                       is needed on the main program level, the corrosponding
                       global data variable is accessed.
  timer                time since program start

  the corrosponding data structures are declared in main.h

UPDATES
  07/20/91, transducer access only inside isr
  07/16/91, no strain feedback in control algorithm
  07/14/91, change in struct data to implement ring buffer
  07/11/91, clean up, conditional compilation
  06/17/91, used with TC++ interrupt keyword
  06/13/91, change interrupt stuff, all variables are external or
    static, no automatic variables, all functions that are called
    outside this  ile should have "_1" at the end of their function name.

NOTE
  variables declared as statics remain in existence when function
  is deactivated, statics are initialized to zero by default.

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

/*****************************************************************************
IMPORTS
*****************************************************************************/
#include "envir.h"      /* compiler options */

#include <stdlib.h>     /* standard libraries */
#include <stdio.h>
#include <dos.h>
```

```
#include "main.h"       /* system parameters, data types */

#include "adstuff.h"    /* function prototypes */
#include "convers.h"

/*****************************************************************************
FUNCTION PROTOTYPES
*****************************************************************************/
extern void interrupt (*oldtmr_isr)();  /* defined in setup.c */
static void control (void);
static void get_traj (void);
static int store_data(struct data *);
static void multiply (double [][8], double [], double []);

/*****************************************************************************
EXTERNAL DATA
*****************************************************************************/
struct traj traj1;              /* trajectory data = control input */
struct data data1;              /* data = data acquisition output */
struct gains gains1;            /* controller gains */
double th1d_hold,th2d_hold;     /* desired joint angles during hold */
double th1actual,th2actual;     /* actual (measured) joint angles in rad */
double timer;                   /* keeps track of time in seconds */

int c_points;                   /* counter for traj points */
int c_iterations;               /* counter for iterations */
int c_holds;                    /* counter for holds after traj */

#if OPTION_DEBUG
int error_a2d_i=0;              /* for debugging */
int error_a2d_count_max_i=0;
#endif


/*****************************************************************************
PRIVATE DATA
*****************************************************************************/
static double  th1d,th2d;       /* desired joint angles from traj data */
static int     out1,out2;       /* control outputs to actuators */

/*****************************************************************************
VISIBLE ROUTINES
*****************************************************************************/

/*---------------------------------------------------------------------------
ROUTINE
  TMR_ISR
  Klaus Obergfell

DESCRIPTION
  this is the interrupt service routine that is executed when a timer
  interrupt occurs (IRQ0 at 8259A interrupt controller) and interrupts
  are enabled. The frequency of occurence is set with setalarm (alarm.c),
  which configures the 8253-0 timer.

  The 386 registers are saved before entering this routine and are restored
  when returning from this routine (TC interrupt keyword). This routine
  saves the coprocessor state before any floating point operations are
  performed and restores the coprocessor state before returning.

  The execution time of this routine is critical!. If the execution
  takes longer then the time between two timer interrupts, an other timer
  interrupt would interrupt the service function already in operation.
  A flag is used to prevent this from happening. If the isf flag is
  set upon entering the routine, a message is print to the screen and then
  the routine exits without calling the controller. This message might
  appear at the screen right after enabling the interrupts. I assume
  that this results from the variable initialization.

  A specific end of interrupt (SEOI) is send to the 8259A interrupt
  controller, before the control routine is called. This has two reasons
  - an EOI has to be send in the isr anyway, to enable any further
    interrupts (the IRET command doesn't automatically do this)
  - other interrupts should be allowed to interrupt this routine
    (especially the communication interrupt).
  Therefore the sending of the SEOI enables the 8259A to process further
  interrupts, also interrupts with lower priorities than IRQ0.

  Since the interrupt flag is cleared by the CPU when an interrupt function
  is entered, this flag has also to be set before any other interrupts
  can be processed. The TC functions enable() and disable() set and clear
  this flag. The STI (set interrupt flag) command can also be used.
  The IRET (return from interrupt) also sets the interrupt flag.

  To summarize the last two paragraphs:
  A SEOI and a STI are necessary to enable other service routines to
  interrupt this routine.

  In critical regions interrupts are enabled.

  This routine also keeps track of time and stores data to the buffer.
  Every 55ms the old DOS interrupt function is called at the end of this
  routine.
---------------------------------------------------------------------------*/
void interrupt tmr_isr(void)
{
  static int isf;       /* interrupt in service flag */
  static int tick=0;
  static int time=0;
  static int dos_tick=0;
  static int count=0;

  disable();  /* critical region */

#if OPTION_DEBUG  /* debugging: write 'A' to screen */
  asm {
    push ds
    push ax
    push bx
    mov  ax,0B800h
    mov  ds,ax
    mov  bx,0
    mov  al,41h
    mov  [bx],al
    mov  bx,4
```

```
    mov  [bx],al
    pop  bx
    pop  ax
    pop  ds
  }
#endif

  if (isf == 0)    /* routine not serviced = normal operation */
  {
    isf = 1;       /* set routine in service flag */
    asm {
      push bp
      sub sp,94
      mov bp,sp
      fsave [bp]       /* saves current coprocessor state */
      fwait            /* lets processor wait until coprocessor save is done */

      push ax
      mov al,60h       /* send specific EOI to interrupt controller */
      out 20h,al       /*  to get ready for next interrupt */
      pop ax
    }
    enable();    /* end of critical region, enable interrupts */

    /* keep track of time. During abnormal program operation a tick is
       only lost when tick equals (SAMPLE_FREQ - 1) */
    if (++tick >= SAMPLE_FREQ)
    {
      tick = 0;
      time++;
    }
    timer = (double) time +  ((double) tick) * SAMPLE_T / 1000.;

    control();   /* call control routine */

    /* save data to memory */
    if (++count >= DATA_T)
    {
      if (data1.s_flag)             /* if data saving enabled */
      {
        if (!data1.f_flag)          /* if space available */
        {
          count = 0;
          store_data(&data1);
        }
      }
    }

    disable();   /* critical region */
    asm {
      mov bp,sp
      frstor [bp]      /* restores coprocessor state */
      add sp,94
      pop bp
    }

    /* keep track of interrupt chaining to old timer isr, miss one
       dos_tick during abnormal operation */
    if (++dos_tick >= DOS_TIMER_FREQ)
    {
      dos_tick = 0;
      oldtmr_isr();
    }

    isf = 0;  /* done processing */
  } /* end if (isf == 0) */

  else      /* timing problem - abnormal operation */
  {
    /* write 'Trouble' to screen (first line, right half) */
    asm {
      push ds
      push ax
      push bx
      mov  ax,0B800h
      mov  ds,ax
      mov  bx,100
      mov  al,54h
      mov  [bx],al
      mov  bx,102
      mov  al,72h
      mov  [bx],al
      mov  bx,104
      mov  al,6Fh
      mov  [bx],al
      mov  bx,106
      mov  al,75h
      mov  [bx],al
      mov  bx,108
      mov  al,62h
      mov  [bx],al
      mov  bx,110
      mov  al,6Ch
      mov  [bx],al
      mov  bx,112
      mov  al,65h
      mov  [bx],al
      pop  bx
      pop  ax
      pop  ds
    }

    tick++;  /* no timer tick is lost during abnormal operation */

    asm {
      push ax
      mov al,60h       /* send specific EOI to interrupt controller */
      out 20h,al       /*  to get ready for next interrupt */
      pop ax
    }
  } /* end of else */

#if OPTION_DEBUG  /* debugging: write 'B' to screen */
  asm {
    push ds
    push ax
```

```
    push bx
    mov  ax,0B800h
    mov  ds,ax
    mov  bx,0
    mov  al,42h
    mov  [bx],al
    mov  bx,4
    mov  [bx],al
    pop  bx
    pop  ax
    pop  ds
  }
#endif

  enable();   /* enable interrupts before return */

} /* end of routine tmr_isr */

/*****************************************************************************
PRIVATE ROUTINES
*****************************************************************************/

/*---------------------------------------------------------------------------
ROUTINE
  CONTROL
  Klaus Obergfell 05/01/91, based on B.S. Yuan's control algorithm.

DESCRIPTION
  this routine does the control work, called by tmr_isr.

REMARK
  current version doesn't use strain feedback
---------------------------------------------------------------------------*/
static void control (void)
{
#if OPTION_DAS16
  static double  addata[8];          /* arrray of a/d numbers */
/*  static double  st1,st2;*/          /* strains in link 1&2, not used */
#if OPTION_DEBUG
  static int     error,error_count; /* for debugging */
#endif
#endif
  static int     i,j;                /* multi purpose counters */
  static double  X[8];               /* state vector, used to compute control */
  static double  U[2];               /* computed control */
  static double  xa[4];              /* position and strain error vector */
  static double  xb[4];              /* lower half of state vector */
  static double  xb11,xb12,xb13;     /* values for control algorithm */
  static double  xb31,xb32,xb33;
  static double  vel1,vel11;
  static double  vel12,vel13;
  static double  vel2,vel21;
  static double  vel22,vel23;
  static double  ct[4][4];
  static double  K[2][8] = {         /* gain matrix */
    { -35000.0, 0., 0., 0., 775.0, 0., 0., 0. },
    { 0., 0., 0., 0., 0., 0., 0., -1. },
  };
```

```
#if OPTION_DAS16  /* a/d board used ? (= normal operation) */

  /* read a/d board, get actual angles and strains */

#if OPTION_DEBUG  /* debugging options included ? */

  error_count = 0;
  do    /* this type of error checking might be dangerous */
  {
    error = readad_i(addata);
    if (error != 0)
      error_a2d_i = 1;
    error_count++;
  } while(error != 0);
  if (error_count > error_a2d_count_max_i)
    error_a2d_count_max_i = error_count;

#else

  while(readad_i(addata) != 0);

#endif

  angles_i(addata,&th1actual,&th2actual);
/*  strains_i(addata,&st1,&st2);  no strain !!!!! */

  /* get desired path data */
  get_traj();

  /* xa vector is the error vector for joint 1 and joint 2 */
/*  xa[1] = st1;  no strain !!!!!
  xa[3] = st2;*/
  xa[1] = 0;
  xa[3] = 0;

#else  /* a/d board not used = special debugging session */

  /* during debugging: controller is not connected to output, set
  all variables to some dummy value, to avoid overflows */
  th1actual = .65;
  th2actual = 1.8;
  th1d = .65;
  th2d = 1.8;
  xa[1] = 0.;
  xa[3] = 0.;

#endif

  /* base some of the matrix gains on current position */
  K[1][6] = 260.0*th2actual-402.0;       /* upper joint velocity gain */
  K[1][2] = -14804.0*th2actual-8339.0;   /* upper joint position gain */

  /*
  create error term due to difference between desired angle
  and actual angle
  */
```

```
xa[0] = (th1d-th1actual)*gains1.p1;
xa[2] = (th2d-th2actual)*gains1.p2;

/* shift terms in ct matrix */
for(i=0; i<4; i++)
  for(j=1; j<4; j++)
    ct[i][j-1] = ct[i][j];

/* assign values to the ct matrix */
for(i=0; i<4; i++)
  ct[i][3] = xa[i];

/* assign values to the xb matrix */
for(i=0; i<4; i++)
  xb[i] = (ct[i][3]-ct[i][0]+3.0*ct[i][2]-3.0*ct[i][1])/0.048;

/*
calculate joint velocity for theta 1 and theta 2
using a smoothing function created by Yuan
*/
vel1=2.1291*vel11-1.783386*vel12+0.5434631*vel13
     +0.05634*(xb[0]-0.0166*xb11-0.0166*xb12+xb13);
vel2=2.1291*vel21-1.783386*vel22+0.5434631*vel23
     +0.05634*(xb[2]-0.0166*xb31-0.0166*xb32+xb33);

/* shift velocity values and xb values */
vel13=vel12;
vel12=vel11;
vel11=vel1;
vel23=vel22;
vel22=vel21;
vel21=vel2;
xb13=xb12;
xb12=xb11;
xb11=xb[0];
xb33=xb32;
xb32=xb31;
xb31=xb[2];

/*
assign xa and xb to the state matrix x
xa is the top of the state matrix, which is the error
xb is the bottom of the state matrix x
*/
for(i=0; i<4; i++)
  X[i] = xa[i];
X[4] = vel1;
X[5] = xb[1];
X[6] = vel2;
X[7] = xb[3];

/*
calculate the control law U = -K*X
multiply the K and X matrices together
*/
multiply(K,X,U);
/* change sign of the control law */
for(i=0; i<2; i++)
  U[i] = -U[i];

/*
converts the control output to an integer
D/A board only handles integers
*/
out1 = (int)U[0];
out2 = (int)U[1];

#if OPTION_DAS16  /* a/d board used ? (= normal operation) */

/* limit the range of output to the range of the D/A board */
d2a_limits_i(&out1,&out2);

/* send control signal to D/A board */
while (writeda_i(out1,out2) != 0);

#endif

  return;
} /* end of routine control */

/*---------------------------------------------------------------------------
ROUTINE
  GET_TRAJ
  Klaus Obergfell, 05/26/91

DESCRIPTION
  compute desired joint angle from trajectory data
---------------------------------------------------------------------------*/
static void get_traj (void)
{
  static double  temp1,temp2;       /* temp values for traj iteration */
  static double  slope1,slope2;     /* slope values for traj iteration */

/* get the desired position from trajectory */
  if(traj1.flag)  /* hold_flag set */
  {
    th1d = th1d_hold;
    th2d = th2d_hold;
  } /* end if hold_flag set */
  else if (c_points < traj1.points)  /* follow traj */
  {
    if (traj1.iterations == 0)  /* no iterations */
    {
      th1d = traj1.data[c_points][0];
      th2d = traj1.data[c_points][1];
      if (c_points == (traj1.points-1))  /* last traj point */
      {
        th1d_hold = traj1.data[c_points][0];
        th2d_hold = traj1.data[c_points][1];
      }
      c_points++;
    }
    else if (c_iterations == 0)  /* iterations, but at traj point */
    {
      temp1 = traj1.data[c_points][0];
      temp2 = traj1.data[c_points][1];
```

```
      th1d = temp1;
      th2d = temp2;
      if (c_points == (traj1.points-1))  /* last traj point */
      {
        th1d_hold = temp1;
        th2d_hold = temp2;
        c_points++;  /* no more iterations after last point */
      }
      else   /* compute slope for next iterations */
      {
        c_iterations++;
        slope1 = (traj1.data[c_points+1][0]-temp1)/(SAMPLE_T*(traj1.iterations+1));
        slope2 = (traj1.data[c_points+1][1]-temp2)/(SAMPLE_T*(traj1.iterations+1));
      }
    }
    else  /* iteration between two trajectory points */
    {
      th1d = temp1 + (slope1 * c_iterations * SAMPLE_T);
      th2d = temp2 + (slope2 * c_iterations * SAMPLE_T);
      c_iterations++;
      if (c_iterations > traj1.iterations)  /* done with iterations */
      {
        c_iterations = 0;
        c_points++;
      }
    }
  } /* end if follow traj */
  else  /* hold after traj completed */
  {
    th1d = th1d_hold;
    th2d = th2d_hold;
    c_holds++;
    if (c_holds > traj1.holds)   /* done with hold */
    {
      c_holds = 0;
      c_iterations = 0;
      c_points = 0;
      traj1.flag = 1;
    }
  } /* end else (hold after traj completed) */
  return;
} /* end of routine get_traj */

/*-------------------------------------------------------------------------
ROUTINE
  STORE_DATA
  Klaus Obergfell

DESCRIPTION
  stores data to data buffer. Updates buffer_head. Does not write data,
  when buffer is full.

PARAMETER
  struct data *databuf  - pointer to structure data

RETURN
  1 - error, buffer already full, didn't write data
  0 - successful
-------------------------------------------------------------------------*/
static int store_data(struct data *databuf)
{
  static int next;

  next = databuf->buffer_head + 1;
  if(next >= DATAPOINTS)  /* wrap around */
    next = 0;
  if(next == databuf->buffer_tail)  /* buffer full */
  {
    databuf->f_flag = 1;
    return 1;
  }

  /* store data */
  databuf->ddata[next][0] = timer;
  databuf->ddata[next][1] = th1actual;
  databuf->ddata[next][2] = th2actual;
  databuf->ddata[next][3] = th1d;
  databuf->ddata[next][4] = th2d;
  databuf->idata[next][0] = out1;
  databuf->idata[next][1] = out2;

  /* updata buffer_head */
  databuf->buffer_head = next;

  return 0;
} /* end of routine store_data */

/*-------------------------------------------------------------------------
ROUTINE
  MULTIPLY
  Klaus Obergfell, based on a routine by Dave Magee

DESCRIPTION
  multiplies matrix and vector: C = A * B

PARAMETER
  double A[][8] - pointer to matrix (input)
  double B[]    - pointer to vector (input)
  double C[]    - pointer to vector (output)
-------------------------------------------------------------------------*/
static void multiply (double A[][8], double B[], double C[])
{
  static int i,j,k;

  for(i=0; i<2; i++)
    for(j=0; j<1; j++)
    {
      C[i] = 0.0;
      for(k=0; k<8; k++)
        C[i] = C[i] + A[i][k]*B[k];
    }
  return;
} /* end of routine multiply */

/*************************************************************************************
end of file control2.c
```

```
/*****************************************************************************
FILE
  move_rob.h - function prototypes to file move_rob.c
  Klaus Obergfell 1991
*****************************************************************************/

/*---------------------------------------------------------------------------
FUNCTION PROTOTYPES
---------------------------------------------------------------------------*/
extern void move_robot(void);
extern int  vision_guided_pos(void);
extern void move_robot_auto(double, double, short, double);

/*****************************************************************************
end of file move_rob.h
```

```
/*****************************************************************************
FILE
  move_rob.c  - computes reference input data for robot motion

ROUTINES
  move_robot            - move robot to user specified position
  vision_guided_pos     - vision supported positioning
  move_robot_auto       - move robot, no user interface
  plan_traj             - plans a smooth, straight line (in x,y space)
                          trajectory

UPDATES
  07/21/91,  new routine move_robot_auto
  07/20/91,  transducer access only inside tmr_isr
  07/16/91,  some clean up
  07/14/91,  change struct data for implementation of ring buffer
  07/13/91,  big clean up
*****************************************************************************/

/*****************************************************************************
IMPORTS
*****************************************************************************/
#include "envir.h"      /* compiler options */

#include "main.h"       /* system parameter and data type declarations */

#include <stdlib.h>     /* standard libraries */
#include <stdio.h>
#include <math.h>
#include <dos.h>
#include <conio.h>
#include <ctype.h>
#include <bios.h>

#include "convers.h"    /* function prototypes and external data */
#include "control2.h"
#include "util.h"
#include "setup.h"
#include "vstuff.h"

/*****************************************************************************
DEFINES
*****************************************************************************/
/* for motion planing in routine move_robot_auto */
#define SMALL_MOTION_UB     0.04    /* rad */
#define MEDIAN_MOTION_UB    0.1
#define SMALL_MOTION_VEL    0.05    /* rad / sec. */
#define MEDIAN_MOTION_VEL   0.08
#define LARGE_MOTION_VEL    0.1
#define T_ACCEL             0.8     /* sec. */
#define MAX_VELOCITY        0.25

/* parameters for routine vision_guided_pos */
#define MAX_ITERATIONS  5
#define MIN_ERROR       0.06    /* inch ( 0.06 inch = 1.525 mm ) */

/* position of reference landmark in joint coordinates */
#define TH1_REF_LM      1.942
```

```
#define TH2_REF_LM      1.703

/*****************************************************************************
FUNCTION PROTOTYPES
*****************************************************************************/
void move_robot_auto(double, double, short, double);
static void plan_traj(double, double, double, double, double, double);

/*****************************************************************************
FILES
*****************************************************************************/
#if (OPTION_DEBUG || OPTION_DATA || OPTION_VGPDATA)
extern FILE *move_rob_out;
#endif

/*****************************************************************************
VISIBLE ROUTINES
*****************************************************************************/

/*---------------------------------------------------------------------------
ROUTINE
  MOVE_ROBOT

- reads transducers and outputs robot position
- prompts user for desired position and trajectory specification
- plans trajectory, writes traj data to data structure for controller,
   writes trajectory data to file
- sets flag to give interrupt driven controller the new traj.
- returns to main menu
---------------------------------------------------------------------------*/
void move_robot (void)
{
  /* variable declaration */
  double  thli,th2i;      /* initial joint angles */
  double  thld,th2d;      /* desired joint angles */
  double  xi,yi;          /* initial end effector x,y position */
  double  xd,yd;          /* desired end effector x,y position */
  double  velmax;         /* maximum velocity during motion */
  double  tmotion;        /* time for motion */
  double  thold;          /* time to hold manipulator at desired position */
  int     steph;          /* steps for hold */
  int     itemp;
  double  temp1,temp2;
#if OPTION_DEBUG
  double  temp[4];        /* temp array for printing to file */
  double  temp5,temp6;
  int     i,j;            /* counter */
#endif


  /* check if done with previous movement, ready to write new desired
     trajectory */
  if (!traj1.flag)
    printf("\n\nWait for previous motion to finish");
  while (!traj1.flag);


  /*
  user interface:
  read and print current position, prompt for trajectory data ( desired
  position, time for move, maximum velocity, holding time at desired
  position ). Note: time for move and maximum velocity are not independent,
  but this is not considered in the current program version.
  */

#if OPTION_DAS16

  disable();
  thli = thlactual;
  th2i = th2actual;
  enable();
  forward(thli,th2i,&xi,&yi);
  clrscr();
  printf("\nPlease enter data for trajectory computation:\n");
  printf("\nCurrent position: x = %.3f inch  y = %.3f inch",xi,yi);
  printf("\n                  thl = %.4f rad  th2 = %.4f rad",thli,th2i);
  printf("\n                 (thl = %.1f deg  th2 = %.1f deg)"
    ,thli*180./pi,th2i*180./pi);

#else  /* debugging session */

  clrscr();
  printf("\nPlease enter data for trajectory computation:\n");
  printf("\n\nEnter initial x,y position (x,y):");
  scanf("%lf,%lf",&xi,&yi);
  inverse(xi,yi,&thli,&th2i);

#endif

  do
  {
    printf("\n\nEnter mode for desired position:");
    printf("\n1.  abs. x,y coordinates in inch");
    printf("\n2.  joint angles th1,th2 in rad");
    scanf("%d",&itemp);
  } while(itemp != 1 && itemp != 2);
  switch(itemp)
  {
    case 1:  /* x,y coordinates */
      printf("\nEnter desired position: (x,y)");
      scanf("%lf,%lf",&xd,&yd);
      inverse(xd,yd,&thld,&th2d);
      break;
    case 2:  /* joint coordinates */
      printf("\nEnter desired position: (th1,th2)");
      scanf("%lf,%lf",&thld,&th2d);
      forward(thld,th2d,&xd,&yd);
      break;
  }
  printf("\nEnter time for move in seconds (12.0 in cut3)");
  scanf("%lf",&tmotion);
/*  printf("\nEnter maximum velocity during move (0.25 in cut3)");
  scanf("%lf",&velmax);*/
  velmax = 0.25;
/*  printf("\nEnter holding time at desired position (3.0 in cut3)");
  scanf("%lf",&thold);*/
```

```
  thold = 3.0;

  /* get trajectory for motion */
  printf("\n\ntrajectory calculation");
  /* make old final position the new starting position for trajectory */
  disable();
  th1i = th1d_hold;
  th2i = th2d_hold;
  enable();
  forward(th1i,th2i,&xi,&yi);
  plan_traj(xi,yi,xd,yd,velmax,tmotion);
  steph = ((int) (thold/(SAMPLE_T/1000))) + 1;
  traj1.holds = steph;


#if OPTION_DEBUG

  /* write trajectory data to file */
  printf("\n\nwriting trajectory data to file");
  move_rob_out = fopen("traj.d","a");
  fprintf(move_rob_out,"\n\nnext trajectory data:");
  fprintf(move_rob_out,"\ninitial position: th1 = %.3f, th2 = %.3f, x = %.3f,
    y =.%.3f",th1i,th2i,xi,yi);
  fprintf(move_rob_out,"\ndesired position: th1 = %.3f, th2 = %.3f, x = %.3f,
    y = %.3f\n",th1d,th2d,xd,yd);
  fprintf(move_rob_out,"\ntraj points = %d, iterations = %d, holds = %d\n"
    ,traj1.points,traj1.iterations,steph);
  for (i=0;i<traj1.points;i++)
  {
    for (j=0;j<4;j++)
      temp[j] = traj1.data[i][j];
    forward(temp[0],temp[1],&temp5,&temp6);
    fprintf(move_rob_out,"\n%d  %.5f  %.5f  %.5f  %.5f  %.5f  %.5f"
      ,i,temp[0],temp[1],temp[2],temp[3],temp5,temp6);
  }
  fclose(move_rob_out);

#endif


#if OPTION_DAS16

#if OPTION_DATA

  /* check status of data taking facility */
  if (data1.s_flag)
  {
    printf("\nwrite old data to file");
    w_data_to_disk(&data1);     /* write old data to disk */
  }
  else
  {
    disable();
    data1.s_flag = 1;       /* enable data taking during move */
    enable();
  }

#endif
```

```
  /* get controller data ready */
  printf("\nclear hold flag, execute motion");
  disable();            /* disable interrupts in critical region */
  traj1.flag = 0;
  enable();             /* enable interrupts after critical region */

#endif

  return;
} /* end of routine move_robot */

/*---------------------------------------------------------------------------
ROUTINE
  VISION_GUIDED_POS
  Klaus Obergfell 1991

DESCRIPTION
  routine for vision supported positioning. Robot is moved to default position
  in camera workspace and user is prompt for desired position. This Routine
  automatically positions robot until desired position is reached within some
  error. Algorithm stops after MAX_ITERATIONS.

RETURN
  0  - success
  1  - error
---------------------------------------------------------------------------*/
int vision_guided_pos(void)
{
  double th1s,th2s,th1f,th2f,oth1f,oth2f;
  double temp1,temp2;
  double xtip,ytip;
  double xtipd,ytipd;
  double deltaxtip,deltaytip,delta_tip;
  double deltath1,deltath2;
  int    count,itemp1;
  short  key,DONE = False;
  double start_time,end_time,pos_time;

#if OPTION_DATA
  /* check status of data taking facility */
  if (data1.s_flag)
  {
    printf("\nwrite old data to file");
    w_data_to_disk(&data1);     /* write old data to disk */
  }
#endif

  /* user interface, print info, enter desired position */
  clrscr();
  printf("\nVision supported positioning algorithm");
  printf("\npositions robot tip w.r.t. reference landmark\n");
  print_camera_ws_info('C');
  do
  {
    printf("\n\nPlease enter desired tip position in camera coordinates");
    scanf("%lf,%lf",&xtipd,&ytipd);
    itemp1 = camera_ws_boundary_test(xtipd,ytipd,'C');
```

```
  if (itempl == 0)  /* desired position inside workspace */
    DONE = True;
  else
  {
    printf("\nDesired position likely to be outside camera workspace");
    printf("\nContinue anyway (y) or enter new desired position (n) ?");
    key = bioskey(0) & 0xff;
    if (toupper(key) == 'Y')
      DONE = True;
  }
} while(!DONE);

/* read timer before starting positioning */
disable();
start_time = timer;
enable();

/*
reference joint angles for first positioning move:
1. if robot tip is in camera workspace use Jacobian based algorithm
   to compute reference joint angles
2. if robot tip is outside camera workspace estimate reference joint
   angles, using position of reference landmark
*/

itempl = tip_position(&xtip,&ytip);
if(itempl == 0)  /* inside workspace */
{
  /* case 1. */
  deltaxtip = xtipd - xtip;  /* compute error in camera coordinates */
  deltaytip = ytipd - ytip;
  /* transform to error in joint angles */
  deltath1 = jacobian[0] * deltaxtip + jacobian[1] * deltaytip;
  deltath2 = jacobian[2] * deltaxtip + jacobian[3] * deltaytip;
  disable();
  th1f = th1d_hold;          /* get current desired joint angles */
  th2f = th2d_hold;
  enable();
  th1f = th1f + deltath1;  /* compute new desired joint angles */
  th2f = th2f + deltath2;

  /* move robot */
  move_robot_auto(th1f,th2f,'A',3.0);
}
else /* outside workspace */
{
  /* case 2: estimate joint angles for desired position, using position
     of reference landmark in joint coordinates and transforming desired
     tip position to a difference in joint angles (inverse Jacobian) */
  deltath1 = jacobian[0] * xtipd + jacobian[1] * ytipd;
  deltath2 = jacobian[2] * xtipd + jacobian[3] * ytipd;
  th1f = TH1_REF_LM + deltath1;
  th2f = TH2_REF_LM + deltath2;

  /* move robot to estimated desired position */
  move_robot_auto(th1f,th2f,'A',3.0);
}
```

```
  /* print data while waiting for end of motion */
#if OPTION_VGPDATA
  move_rob_out = fopen("vgpos.d","a");
  fprintf(move_rob_out,"\n\nnext vision supported positioning, desired tip");
    fprintf(move_rob_out," position: %.2f, %.2f",xtipd,ytipd);
/*
  fprintf(move_rob_out,"\n\n(tip position, actual joint angle, (old) desired");
    fprintf(move_rob_out," joint angle)");
*/
  fprintf(move_rob_out,"\n(tip error, error in joint angles (Jacobian),");
    fprintf(move_rob_out," error in joint angles (joint control))");
  fclose(move_rob_out);
#endif

  while(!traj1.flag);

  /* get joint angles, get tip position */
  disable();
  th1s = th1actual;
  th2s = th2actual;
  enable();
  itempl = tip_position(&xtip,&ytip);
  if(itempl != 0)  /* valid tip position ? */
  {
    vision_error_message(itempl);
    return 1;
  }
  disable();
  temp1 = th1actual;
  temp2= th2actual;
  enable();

  /* compute tip position error, enter positioning loop */
  th1s = (th1s + temp1) / 2;  /* average joint angles before and after */
  th2s = (th2s + temp2) / 2;  /*   taking picture */
  deltaxtip = xtipd - xtip;  /* compute error in camera coordinates */
  deltaytip = ytipd - ytip;


  /************************* positioning loop ******************************/
  count = 0;
  do
  {
    count++;

    /* update previous (old) desired joint angle */
    oth1f = th1f;
    oth2f = th2f;

    /* transform to error in joint angles, compute new desired joint angles */
    deltath1 = jacobian[0] * deltaxtip + jacobian[1] * deltaytip;
    deltath2 = jacobian[2] * deltaxtip + jacobian[3] * deltaytip;
    th1f = th1f + deltath1;
    th2f = th2f + deltath2;

    /* move robot, print data and messages while waiting for end of motion */
    move_robot_auto(th1f,th2f,'A',3.0);
```

```
#if OPTION_SCR
    /* print info to screen */
    printf("\ntip error: %.2f, %.2f   new desired position: %.4f, %.4f",
      deltaxtip,deltaytip,thlf,th2f);
#endif

#if OPTION_VGPDATA
    /* write data to disk */
    move_rob_out = fopen("vgpos.d","a");
/*    fprintf(move_rob_out,"\n\n%.2f  %.2f    %.4f  %.4f    %.4f  %.4f",xtip,
      ytip,th1s,th2s,othlf,oth2f);*/
    fprintf(move_rob_out,"\n%.3f %.3f    %.5f  %.5f    %.5f  %.5f",deltaxtip,
      deltaytip,deltath1,deltath2,othlf-th1s,oth2f-th2s);
    fclose(move_rob_out);
#endif

    while(ltraj1.flag);

    /* take picture */
    disable();
    th1s = th1actual;
    th2s = th2actual;
    enable();
    itemp1 = tip_position(&xtip,&ytip);
    if(itemp1 != 0)  /* valid tip position ? */
    {
      vision_error_message(itemp1);
      return 1;      /* take care of special case later */
    }
    disable();
    temp1 = th1actual;
    temp2 = th2actual;
    enable();

    th1s = (th1s + temp1) / 2;  /* average joint angles before and after */
    th2s = (th2s + temp2) / 2;  /*   taking picture */
    deltaxtip = xtipd - xtip;  /* compute error in camera coordinates */
    deltaytip = ytipd - ytip;
    delta_tip = sqrt(deltaxtip * deltaxtip + deltaytip * deltaytip);

    /* check stop criteria */
  } while(delta_tip > MIN_ERROR && count < MAX_ITERATIONS);
  /************************ end of positioning loop ****************************/

  /* read timer at end of positioning */
  disable();
  end_time = timer;
  enable();
  pos_time = end_time - start_time;

#if OPTION_VGPDATA
    /* write data to disk */
    deltath1 = jacobian[0] * deltaxtip + jacobian[1] * deltaytip;
    deltath2 = jacobian[2] * deltaxtip + jacobian[3] * deltaytip;
    move_rob_out = fopen("vgpos.d","a");
/*    fprintf(move_rob_out,"\n\n%.2f  %.2f    %.4f  %.4f    %.4f  %.4f",xtip,
      ytip,th1s,th2s,othlf,oth2f);*/
    fprintf(move_rob_out,"\n%.3f %.3f    %.5f  %.5f    %.5f  %.5f",deltaxtip,
      deltaytip,deltath1,deltath2,thlf-th1s,th2f-th2s);
    fclose(move_rob_out);
#endif


  /* print message to screen and to data file */
  if (count < MAX_ITERATIONS)  /* successful */
  {
    printf("\nVision supported positioning successful completed");
    printf("\niterations = %d, tip error = %.3f, time = %.3f",count,
      delta_tip,pos_time);
#if OPTION_VGPDATA
    move_rob_out = fopen("vgpos.d","a");
    fprintf(move_rob_out,"\nVision supported positioning successful completed");
    fprintf(move_rob_out,"\niterations = %d, tip error = %.3f, time = %.3f\n",
      count,delta_tip,pos_time);
    fclose(move_rob_out);
#endif
  }
  else
  {
    printf("\nStopped vision supported positioning after %d iterations",count);
    printf("\ntip position error = %.3f",delta_tip);
#if OPTION_VGPDATA
    move_rob_out = fopen("vgpos.d","a");
    fprintf(move_rob_out,"\nStopped vision supported positioning after %d
      iterations",count);
    fprintf(move_rob_out,"\ntip position error = %.3f\n",delta_tip);
    fclose(move_rob_out);
#endif
    return 1;
  }

  return 0;
} /* end of routine vision_guided_pos */

/*---------------------------------------------------------------------------
ROUTINE
  MOVE_ROBOT_AUTO
  Klaus Obergfell 1991

DESCRIPTION
  this is a variation of routine move_robot. No user interfacing is necessary.
  Computes time for move depending on travel distance. Time for hold
  depends on parameter.

PARAMETER
  double d1          - desired value 1
  double d2          - desired value 2
  short  coord       - coordinates parameter:
                       determines if desired values are specified in rigid
                       endpoint x,y coordinates ('X') or in joint angles ('A')
  double hold_time   - holding time after move in seconds
---------------------------------------------------------------------------*/
void move_robot_auto(double d1,double d2,short coord,double hold_time)
{
  /* variable declaration */
  double  th1i,th2i;     /* initial joint angles */
```

```
  double  thld,th2d;      /* desired joint angles */
  double  xi,yi;          /* initial end effector x,y position */
  double  xd,yd;          /* desired end effector x,y position */
  double  deltath1,deltath2,deltamax;
  double  tmotion;        /* time for motion */
  int     steph;          /* steps for hold */

  /* make old final position the new starting position for trajectory,
  compute initial position in x,y coordinates */
  disable();
  thli = thld_hold;
  th2i = th2d_hold;
  enable();
  forward(thli,th2i,&xi,&yi);

  /* desired position */
  switch(toupper(coord))
  {
    case 'X':  /* desired values already in x,y coordinates */
      xd = d1;
      yd = d2;
      inverse(xd,yd,&thld,&th2d);
      break;
    case 'A':  /* desired values are in joint angles */
      thld = d1;
      th2d = d2;
      forward(thld,th2d,&xd,&yd);
      break;
  }

  /* time for move */
  deltath1 = fabs(thld - thli);
  deltath2 = fabs(th2d - th2i);
  if (deltath1 >= deltath2)
    deltamax = deltath1;
  else
    deltamax = deltath2;
  if (deltamax <= SMALL_MOTION_UB)  /* small motion */
    tmotion = T_ACCEL + deltamax / SMALL_MOTION_VEL;
  else if (deltamax <= MEDIAN_MOTION_UB)  /* median motion */
    tmotion = T_ACCEL + deltamax / MEDIAN_MOTION_VEL;
  else  /* large motion */
    tmotion = T_ACCEL + deltamax / LARGE_MOTION_VEL;

  /* steps for hold */
  steph = ((int) (hold_time/(SAMPLE_T/1000))) + 1;

  /* plan trajectory */
  plan_traj(xi,yi,xd,yd,((double) MAX_VELOCITY),tmotion);
  traj1.holds = steph;

  /* get controller data ready */
  disable();            /* disable interrupts in critical region */

#if OPTION_DATA
  data1.s_flag = 1;     /* enable data taking during move */
#endif

  traj1.flag = 0;       /* clear hold flag, start moving */
  enable();             /* enable interrupts after critical region */

#if OPTION_DATA
  /* write old data to disk */
  w_data_to_disk(&data1);
#endif

  return;
} /* end of routine move_robot_auto */

/*****************************************************************************
PRIVATE ROUTINES
*****************************************************************************/

/*---------------------------------------------------------------------------
ROUTINE
  PLAN_TRAJ

revision 05/27/91 Klaus Obergfell:
  allow linear approximation between traj points, less computation
  and data storage, number of approximations depends on time for
  move (maximum number of points).
Klaus Obergfell, 05/01/91
based on Dave Magees traj3.c

This subroutine uses fourth order polynomials divided into
eight sections to give a smooth displacement profile with
straight line tip motion
returns theta positions in radians


---------------------------------------------------------------------------*/
static void plan_traj(Xo,Yo,Xf,Yf,Vmax,Tf)
double Xo,Yo;     /* initial x,y position in inch */
double Xf,Yf;     /* final x,y position in inch */
double Vmax;      /* maximum velocity */
double Tf;        /* time for complete movement */

{
  int     steps;              /* number of traj points */
  double  IOfreq;             /* period between traj points */
  int     iterations;         /* number of iterations between points */
  double  delta;              /* max time interval with 0 iterations */
  int     i;
  int     region;
  double  x,y,th1,th2;
  double  DX,DY,DT,DTact,DTnorm,Tact,Tnorm;
  double  Qo,Qf;
  double  coeff,To;
  double  a[8][5];

  delta = MAXPOINTS * SAMPLE_T / 1000.;  /* in seconds */
  iterations = (int) (Tf/delta);
  IOfreq = SAMPLE_T * (iterations+1) / 1000; /* in seconds */
  steps = ((int) (Tf/IOfreq)) + 1;

  DX= Xf-Xo;
```

```
DY= Yf-Yo;

/* Setup time steps in normalized time and actual time */
Tact=0.0;
DTact=1Ofreq;
Tnorm=0.0;
DTnorm=6.5/steps;

/*
Scales displacement from 0.0 to 1.0 so that it can applied to x and
y motions simultaneously to generate a straight path in x,y space
*/
Qo= 0.0;
Qf= 1.0;

a[0][0]=  0.0;
a[0][1]=  0.0;
a[0][2]=  0.0;
a[0][3]=  0.0;
a[0][4]=  0.333333333333333*Vmax;

a[1][0]=  0.02083333333333*Vmax+Qo;
a[1][1]=  0.166666666666666*Vmax;
a[1][2]=  0.5*Vmax;
a[1][3]=  0.666666666666666*Vmax;
a[1][4]= -0.666666666666666*Vmax;

a[2][0]=  0.27083333333333*Vmax+Qo;
a[2][1]=  0.833333333333333*Vmax;
a[2][2]=  0.5*Vmax;
a[2][3]= -0.666666666666666*Vmax;
a[2][4]=  0.333333333333333*Vmax;

a[3][0]=  0.75*Vmax+Qo;
a[3][1]=  Vmax;
a[3][2]=  0.0;
a[3][3]=  0.0;
a[3][4]=  0.0;

a[4][0]=  1.75*Vmax+Qo;
a[4][1]=  Vmax;
a[4][2]=  0.0;
a[4][3]=  0.0;
a[4][4]= -0.17708333333333*Vmax+0.041666666666666*Qf-0.041666666666666*Qo;

a[5][0]=  2.57291666666666*Vmax+0.041666666666666*Qf+0.958333333333333*Qo;
a[5][1]=  0.29166666666666*Vmax+0.166666666666666*Qf-0.166666666666666*Qo;
a[5][2]= -1.0625*Vmax+0.25*Qf-0.25*Qo;
a[5][3]= -0.70833333333333*Vmax+0.166666666666666*Qf-0.166666666666666*Qo;
a[5][4]=  0.48958333333333*Vmax-0.125*Qf+0.125*Qo;

a[6][0]=  1.58333333333333*Vmax+0.5*Qf+0.5*Qo;
a[6][1]= -2.0*Vmax+0.666666666666666*Qf-0.666666666666666*Qo;
a[6][2]= -0.25*Vmax;
a[6][3]=  1.25*Vmax-0.333333333333333*Qf+0.333333333333333*Qo;
a[6][4]= -0.44791666666666*Vmax+0.125*Qf-0.125*Qo;

a[7][0]=  0.13541666666666*Vmax+0.958333333333333*Qf+0.041666666666666*Qo;
a[7][1]= -0.54166666666666*Vmax+0.166666666666666*Qf-0.166666666666666*Qo;
a[7][2]=  0.8125*Vmax-0.25*Qf+0.25*Qo;
a[7][3]= -0.54166666666666*Vmax+0.166666666666666*Qf-0.166666666666666*Qo;
a[7][4]=  0.13541666666666*Vmax-0.041666666666666*Qf+0.041666666666666*Qo;


for (i=0;i<steps;i++)
{
  if(Tnorm < 0.5)
  {
    region=0;
    To=0.0;
  }
  else if(Tnorm < 1.0)
  {
    region=1;
    To=0.5;
  }
  else if(Tnorm < 1.5)
  {
    region=2;
    To=1.0;
  }
  else if(Tnorm < 2.5)
  {
    region=3;
    To=1.5;
  }
  else if(Tnorm < 3.5)
  {
    region=4;
    To=2.5;
  }
  else if(Tnorm < 4.5)
  {
    region=5;
    To=3.5;
  }
  else if(Tnorm < 5.5)
  {
    region=6;
    To=4.5;
  }
  else
  {
    region=7;
    To=5.5;
  };

  DT= Tnorm-To;
  coeff=a[region][0]+a[region][1]*DT+a[region][2]*DT*DT+
        a[region][3]*DT*DT*DT+a[region][4]*DT*DT*DT*DT;
  x=coeff*DX + Xo;
  y=coeff*DY + Yo;

  /* find the corresponding angles  */
  inverse(x,y,&th1,&th2);
```

```
    /*
    check to see if desired angles are within the workspace
    if not it is assigned the closest angle
    */
    ws_limits(&th1,&th2);

    traj1.data[i][0] = th1;
    traj1.data[i][1] = th2;

#if OPTION_DEBUG
    traj1.data[i][2] = x;
    traj1.data[i][3] = y;
#endif

    Tact=Tact+DTact;
    Tnorm=Tnorm+DTnorm;

  }/* end of for loop */
  traj1.points = steps;
  traj1.iterations = iterations;

  return;
} /* end of routine plan_traj */

/*****************************************************************************
end of file move_rob.c
```

```
/*****************************************************************************
FILE
  vstuff.h  - function prototypes and external data for file vstuff.c
  Klaus Obergfell
*****************************************************************************/

/*****************************************************************************
IMPORTS
*****************************************************************************/
#include <comm.h>

/*---------------------------------------------------------------------------
FUNCTION PROTOTYPES
---------------------------------------------------------------------------*/
extern void report_tip_position (COMM_PORT *);
extern void report_tip_position_2 (COMM_PORT *);
extern int tip_position(double *, double *);
extern void print_camera_ws_info(short);
extern int camera_ws_boundary_test(double, double, short);
extern void vision_error_message(int);

/*---------------------------------------------------------------------------
EXTERNAL DATA
---------------------------------------------------------------------------*/
extern struct landmark lm_data[];

/*****************************************************************************
end of file vstuff.h
```

```
/*****************************************************************************
FILE
  vstuff.c  - vision supporting routines
  Klaus Obergfell 1991

ROUTINES
  report_tip_position       - prints tip position and joint angles to screen
  tip_position              - tip position, used by calib and vision_guided_pos
  print_camera_ws_info      - prints camera workspace info to screen
  camera_ws_boundary_test   - checks if x,y position is in camera workspace
  vision_error_message      - prints error message to screen
  get_landmark              - gets landmark pixel information from camera
  get_position              - investigates landmark data, computes position
  rxbuf_test                - checks number of char's in receive buffer
  convert_double            - converts ascii string to double
  convert_int               - converts ascii string to int
  digit                     - converts ascii to digit
  pixel2length_conv         - converts pixel to inch

UPDATES
  07/21/91, add routine tip_position, change in error handling in routies
            get_tip_position and get_position
  07/20/91, transducer access only inside tmr_isr
  07/13/91, big clean up
*****************************************************************************/

/*****************************************************************************
IMPORTS
*****************************************************************************/
#include "envir.h"      /* compiler options */

#include "main.h"       /* system parameter and data type declarations */

#include <stdlib.h>     /* standard libraries */
#include <stdio.h>
#include <ctype.h>
#include <dos.h>
#include <math.h>

#include "util.h"       /* function prototypes and external data */
#include "control2.h"
#include "convers.h"
#include "setup.h"

#include <comm.h>       /* serial comm. */

/*****************************************************************************
DEFINES
*****************************************************************************/
/* vision sampling time for report_tip_position_2 (limited by flash) */
#define VSAMPLE_TIME        0.35      /* in seconds */

/* pixel to length (inch) conversion, from tests */
#define INCH_PER_COL_PIXEL  0.227881  /* inch/pixel */
#define INCH_PER_ROW_PIXEL  0.261032

/* compensation for reference landmark out of plane error, from tests */
#define R_LM_ERROR_X        7.757215    /* pixel */
```

```
#define R_LM_ERROR_Y        9.060403

/* boundary pixel coordinates for reference landmark test, from tests */
#define RLM_YPIXEL_MIN      146
#define RLM_YPIXEL_MAX      150
#define RLM_XPIXEL_MIN      22.2
#define RLM_XPIXEL_MAX      27.3

/* thresholds for moving landmark test, from tests */
#define MLM_AREA_MIN        40
#define MLM_AREA_MAX        56
#define MLM_PEAK_MIN        40

/* camera workspace boundary, from tests */

/* corner points */
/* absolute x,y coordinates - rigid manipulator assumption */
#define cam_wsb_ur_x_abs        -119.0    /* upper right */
#define cam_wsb_ul_x_abs        -159.0    /* upper left */
#define cam_wsb_ll_x_abs        -143.8    /* lower left */
#define cam_wsb_lr_x_abs        -140.0    /* lower right */
#define cam_wsb_uy_abs          87.8      /* upper y */
#define cam_wsb_ly_abs          46.0      /* lower y */
/* camera x,y coordinates */
#define cam_wsb_ur_x_cam        36.1      /* upper right */
#define cam_wsb_ul_x_cam        -3.5      /* upper left */
#define cam_wsb_ll_x_cam        10.3      /* lower left */
#define cam_wsb_lr_x_cam        14.3      /* lower right */
#define cam_wsb_uy_cam          34.3      /* upper y */
#define cam_wsb_ly_cam          -5.1      /* lower y */

/* parameters for left and right straight line boundaries (y = ax + b) */
/* absolute x,y coordinates - rigid manipulator assumption */
#define cam_wsb_la_abs          -2.75     /* left boundary */
#define cam_wsb_lb_abs          -349.45
#define cam_wsb_ra_abs          1.9905    /* right boundary */
#define cam_wsb_rb_abs          324.6666
/* camera x,y coordinates */
#define cam_wsb_la_cam          -2.8551   /* left boundary */
#define cam_wsb_lb_cam          24.3072
#define cam_wsb_ra_cam          1.8073    /* right boundary */
#define cam_wsb_rb_cam          -30.945

/*****************************************************************************
FUNCTION PROTOTYPES
*****************************************************************************/
static int get_landmark (COMM_PORT *);
static int get_position (int, double *, double *);
static int rxbuf_test (COMM_PORT *);
static double convert_double(short *);
static int convert_int (short *);
static int digit(short);
static void pixel2length_conv (double , double , double *, double *);

/*****************************************************************************
EXTERNAL DATA
*****************************************************************************/
struct landmark lm_data[LM_MAX];
```

```
/*****************************************************************************
VISIBLE ROUTINES
*****************************************************************************/

/*---------------------------------------------------------------------------
ROUTINE
  REPORT_TIP_POSITION
  Klaus Obergfell

PARAMETER
  COMM_PORT *p_port  - pointer to COMM_PORT structure, for serial comm.
---------------------------------------------------------------------------*/
void report_tip_position (COMM_PORT *p_port)
{
#if OPTION_DAS16
  double  th1i,th2i;      /* initial joint angles */
  double  th1f,th2f;      /* final joint angles */
  double  xi,yi,xf,yf;    /* tip position in absolute x,y coordinates */
#endif
  double  tip_x,tip_y;    /* tip position in camera x,y coordinates */
  double  col_p,row_p;    /* tip position in row,col pixel */
  int     lm_number;      /* number of landmarks found */
  int     pos_return;     /* return value from get_position routine */

#if OPTION_DAS16

  /* read adboard, compute and store joint angles */
  disable();
  th1i = th1actual;
  th2i = th2actual;
  enable();
  forward(th1i,th2i,&xi,&yi);
  printf("\nDone reading A/D");

#endif

  /* take picture, by calling get_landmark */
  lm_number = get_landmark(p_port);
  printf("\nDone with get_landmark, ret_value = %d",lm_number);
  if (lm_number == 0)  /* fails repeated times to detect exspected
                          number of landmarks */
  {
    printf("\nTrouble detecting exspected number of landmarks, check
      parameters");
    return;
  }

  /* get tip position, by calling get_position */
  pos_return = get_position(lm_number,&col_p,&row_p);
  printf("\nDone with get_position, ret_value = %d",pos_return);
  if (pos_return != 0)  /* trouble validating landmarks */
  {
    printf("\nTrouble validating landmarks");
    switch (pos_return)
    {
      case 1:
        printf("\nError: More than one candidate for reference landmark");
```

```
        break;
      case 2:
        printf("\nError: No candidate for reference landmark");
        break;
      case 3:
        printf("\nError: More than one candidate for 'moving' landmark");
        break;
      case 4:
        printf("\nError: No candidate for 'moving' landmark");
        break;
    }
    return;
  }

  pixel2length_conv(col_p,row_p,&tip_x,&tip_y);

#if OPTION_DAS16

  /* read adboard again, average joint angles
     and compute x,y position of tip */
  disable();
  th1f = th1actual;
  th2f = th2actual;
  enable();
  forward(th1i,th2i,&xf,&yf);

#endif

  /* print tip position in camera coordinates (pixels and inch),
     tip position in x,y coordinates, joint angles */

#if OPTION_DAS16

  printf("\nJoint transducer based data:");
  printf("\nth1i = %.3f th2i = %.3f (rad)    xi = %.2f yi = %.2f (inch)",
    th1i,th2i,xi,yi);
  printf("\nth1f = %.3f th2f = %.3f (rad)    xf = %.2f yf = %.2f (inch)",
    th1f,th2f,xf,yf);

#endif

  printf("\n\nCamera based data:");
  printf("\ncol_p = %.2f row_p = %.2f (pixel)    x = %.2f y = %.2f (inch)",
    col_p,row_p,tip_x,tip_y);

    return;
} /* end of routine report_tip_position */

/*---------------------------------------------------------------------------
ROUTINE
  REPORT_TIP_POSITION_2
  Klaus Obergfell

DESCRIPTION
  repeated, fast position measurements

PARAMETER
  COMM_PORT *p_port  - pointer to COMM_PORT structure, for serial comm.
---------------------------------------------------------------------------*/
void report_tip_position_2 (COMM_PORT *p_port)
{
  double  tip_x[10],tip_y[10];   /* tip position in camera x,y coordinates */
  double  col_p[10],row_p[10];   /* tip position in row,col pixel */
  int     lm_number;     /* number of landmarks found */
  int     pos_return;    /* return value from get_position routine */
  int     i;             /* counter */
  double  start,end;     /* timing */

  for (i=0;i<10;i++)
  {
    /* get start time */
    disable();
    start = timer;
    enable();

    /* take picture, by calling get_landmark */
    lm_number = get_landmark(p_port);
    printf("\nDone with get_landmark, ret_value = %d",lm_number);
    if (lm_number == 0)  /* fails repeated times to detect exspected
                            number of landmarks */
    {
      printf("\nTrouble detecting exspected number of landmarks, check
        parameters");
      return;
    }

    /* get tip position, by calling get_position */
    pos_return = get_position(lm_number,&col_p[i],&row_p[i]);
    printf("\nDone with get_position, ret_value = %d",pos_return);
    if (pos_return != 0)  /* trouble validating landmarks */
    {
      printf("\nTrouble validating landmarks");
      switch (pos_return)
      {
        case 1:
          printf("\nError: More than one candidate for reference landmark");
          break;
        case 2:
          printf("\nError: No candidate for reference landmark");
          break;
        case 3:
          printf("\nError: More than one candidate for 'moving' landmark");
          break;
        case 4:
          printf("\nError: No candidate for 'moving' landmark");
          break;
      }
      return;
    }

    pixel2length_conv(col_p[i],row_p[i],&tip_x[i],&tip_y[i]);

    /* wait for rest of sampling period */
    do
    {
      disable();
```

```
      end = timer;
      enable();
    } while (end-start < VSAMPLE_TIME);
  }

  /* print tip position in camera coordinates (pixels and inch) */
  for (i=0;i<10;i++)
  printf("\ncol_p = %.2f row_p = %.2f (pixel)    x = %.2f y = %.2f (inch)",
    col_p[i],row_p[i],tip_x[i],tip_y[i]);

  return;
} /* end of routine report_tip_position_2 */


/*---------------------------------------------------------------------------
ROUTINE
  TIP_POSITION
  Klaus Obergfell

DESCRIPTION
  variation of routine get_tip_position. No transducer readings, no print
  statements

PARAMETER
  double *x  - pointer to variable for x coordinate of tip
  double *y  - pointer to variable for y coordinate of tip

RETURN
  0 - success
  1 - error: More than one candidate for reference landmark
  2 - error: No candidate for reference landmark
  3 - error: More than one candidate for 'moving' landmark
  4 - error: No candidate for 'moving' landmark
  5 - error: Not able to detect exspected number of landmarks
---------------------------------------------------------------------------*/
int tip_position(double *x, double *y)
{
  int    itemp;
  double temp1,temp2;

  /* take picture, by calling get_landmark */
  itemp = get_landmark(&port1);
  if (itemp == 0)  /* trouble detecting exspected number of landmarks */
    return 5;

  /* get tip position, by calling get_position */
  itemp = get_position(itemp,&temp1,&temp2);
  if (itemp != 0)  /* trouble validating landmarks */
    return itemp;

  /* convert pixel values to length */
  pixel2length_conv(temp1,temp2,&temp1,&temp2);
  *x = temp1;
  *y = temp2;

  return 0;
} /* end of routine tip_position */
```

```
/*---------------------------------------------------------------------------
ROUTINE
  PRINT_CAMERA_WS_INFO
  Klaus Obergfell 1991

DESCRIPTION
  prints information about the current camera workspace boundaries to screen

PARAMETER
  short coord  - determines coordinates of information:
                 A - absolute x,y coordinates
                 C - camera x,y coordinates
---------------------------------------------------------------------------*/
void print_camera_ws_info(short coord)
{
  switch(toupper(coord))
  {
    case 'A':  /* absolute x,y coordinates */
      printf("\ncamera workspace boundaries (in abs. x,y coord.)\n");
      printf("\n(%.1f,%.1f)-----------------------------------------(%.1f,%.1f)",
        cam_wsb_ul_x_abs,cam_wsb_uy_abs,cam_wsb_ur_x_abs,cam_wsb_uy_abs);
      printf("\n     |                                                    |");
      printf("\n          |                                                  |");
      printf("\n               |                                        |");
      printf("\n                    |                              |");
      printf("\n              (%.1f,%.1f)-----------(%.1f,%.1f)",cam_wsb_ll_x_abs,
        cam_wsb_ly_abs,cam_wsb_lr_x_abs,cam_wsb_ly_abs);
      break;
    case 'C':  /* camera x,y coordinates */
      printf("\ncamera workspace boundaries (in camera x,y coord.)\n");
      printf("\n(%.1f,%.1f)-----------------------------------------(%.1f,%.1f)",
        cam_wsb_ul_x_cam,cam_wsb_uy_cam,cam_wsb_ur_x_cam,cam_wsb_uy_cam);
      printf("\n     |                                                    |");
      printf("\n          |                                                  |");
      printf("\n               |                                        |");
      printf("\n                    |                              |");
      printf("\n              (%.1f,%.1f)-----------(%.1f,%.1f)",
        cam_wsb_ll_x_cam,cam_wsb_ly_cam,cam_wsb_lr_x_cam,cam_wsb_ly_cam);
      break;
  }
  return;
} /* end of routine print_camera_ws_info */

/*---------------------------------------------------------------------------
ROUTINE
  CAMERA_WS_BOUNDARY_TEST
  Klaus Obergfell 1991

DESCRIPTION
  determines if a tip position is within the camera workspace. The test
  approximates the workspace boundaries with straight lines. This is only
  an approximation, because the boundary is determined by manipulator and
  camera workspace boundaries. Manipulator boundaries are circular lines, but
  since the radius of those cirles is big compared to the camera workspace,
  it is sufficient for this test to be approximated by straight lines.

PARAMETER
  double x       - x coordinate
```

```
  double y      - y coordinate
  short  coord  - determines coordinates for test:
                  A - absolute x,y coordinates
                  C - camera x,y coordinates

RETURN
  0  - within workspace boundaries
  1  - outside workspace boundaries
-------------------------------------------------------------------------------*/
int camera_ws_boundary_test(double x, double y, short coord)
{
  switch(toupper(coord))
  {
    case 'A':  /* absolute x,y coordinates */
      if (!(cam_wsb_ly_abs <= y && y <= cam_wsb_uy_abs))
        return 1;
      if (cam_wsb_ll_x_abs <= x && x <= cam_wsb_lr_x_abs)
        return 0;
      if (x < cam_wsb_ul_x_abs || x > cam_wsb_ur_x_abs)
        return 1;
      if (x < cam_wsb_ll_x_abs)
      {
        if (y >= cam_wsb_la_abs * x + cam_wsb_lb_abs)
          return 0;
        else
          return 1;
      }
      else
      {
        if (y >= cam_wsb_ra_abs * x + cam_wsb_rb_abs)
          return 0;
        else
          return 1;
      }
      break;
    case 'C':  /* camera x,y coordinates */
      if (!(cam_wsb_ly_cam <= y && y <= cam_wsb_uy_cam))
        return 1;
      if (cam_wsb_ll_x_cam <= x && x <= cam_wsb_lr_x_cam)
        return 0;
      if (x < cam_wsb_ul_x_cam || x > cam_wsb_ur_x_cam)
        return 1;
      if (x < cam_wsb_ll_x_cam)
      {
        if (y >= cam_wsb_la_cam * x + cam_wsb_lb_cam)
          return 0;
        else
          return 1;
      }
      else
      {
        if (y >= cam_wsb_ra_cam * x + cam_wsb_rb_cam)
          return 0;
        else
          return 1;
      }
      break;
  }
} /* end of routine camera_ws_boundary_test */

/*-------------------------------------------------------------------------------
ROUTINE
  VISION_ERROR_MESSAGE
  Klaus Obergfell 1991

DESCRIPTION
  prints error message to screen

PARAMETER
  int error_number  - number of error (uses error coding of tip_position)
-------------------------------------------------------------------------------*/
void vision_error_message(int error_number)
{
  switch(error_number)
  {
    case 1:
      printf("\nProblem reading tip position: more than one candidate for");
        printf(" reference landmark");
      break;
    case 2:
      printf("\nProblem reading tip position: no candidate for reference");
        printf(" landmark");
      break;
    case 3:
      printf("\nProblem reading tip position: more than one candidate for");
        printf(" 'moving' landmark");
      break;
    case 4:
      printf("\nProblem reading tip position: no candidate for 'moving'");
        printf(" landmark");
      break;
    case 5:
      printf("\nProblem reading tip position: not able to detect exspected");
        printf(" number of landmarks");
      break;
  }
  return;
} /* end of routine vision_error_message */

/*******************************************************************************
PRIVATE ROUTINES
*******************************************************************************/

/*-------------------------------------------------------------------------------
ROUTINE
  GET_LANDMARK
  Klaus Obergfell

DESCRIPTION
  retrieves landmark data from camera:
  sends 'K'-command, waits till receive buffer fills up, converts
  character string to landmark data.

PARAMETER
  COMM_PORT *p_port  - pointer to COMM_PORT structure, for serial comm.
```

```
RETURN
  0    - didn't detect exspected number of landmarks
  >= 2 - number of detected landmarks
-------------------------------------------------------------------------------*/
static int get_landmark (COMM_PORT *p_port)
{
  int return_char;       /* number of exspected char from camera */
  int count=0;           /* counter for number of received landmarks */
  int i;
  short c,com_values[4];  /* array for returned camera data */
  double start,temp;      /* testing for not enough landmarks returned case */
  int    trouble=0;

  /* flush receive buffer completely */
  while( (c = c_flush(p_port,0)) != EOF);

  /* determine exspected number of receive characters,
     depending on camera output configuration and landmarks
     in field of view (output: 'K1', lm * (area,peak,x,y), '?') */
  return_char = 12 * LM_EXSPECTED + 3;

again:

#if OPTION_INT
  disable();
  start = timer;
  enable();
#endif

  c_putc(p_port,'K');  /* call "K"-routine to get position */
  c_putc(p_port,'1');

  /* wait untill receiving exspected number of characters
     in receive buffer, checks if less than exspected landmarks
     are returned: assumes that not enough landmarks are returned
     after waiting LM_WAIT seconds. In this case an other picture
     is taken, i.e. procedure is repeated. If this process fails
     LM_REPETITION times, the procedure assumes that no landmarks
     can be found with the current parameter settings and the
     procedure terminates, returning 0 */
  do
  {

#if OPTION_INT

    disable();
    temp = timer;
    enable();
    if (fabs(temp - start) > LM_WAIT)  /* not enough landmarks ? */
    {
      if (trouble >= LM_REPETITION)
      {
        /* flush receive buffer completely */
        xdelay(1); /* wait 2 ms */
        while( (c = c_flush(p_port,0)) != EOF);
        count = 0;
        return count;
      }
      else
      {
        /* flush receive buffer completely */
        xdelay(1); /* wait 2 ms */
        while( (c = c_flush(p_port,0)) != EOF);
        trouble++;
        goto again;
      }
    }

#endif

  } while ( (i = rxbuf_test(p_port)) < return_char);

  /* read serial buffer and convert character string */
  c_inchar(p_port);  /* throw away returned 'K' */
  c_inchar(p_port);  /* throw away returned '1' */
  for (count=0;count<LM_EXSPECTED;count++)
  {
    for (i=0;i<2;i++)  /* read area data from buffer */
      com_values[i] = c_inchar(p_port);
    lm_data[count].area = convert_int(com_values);
    for (i=0;i<2;i++)  /* read peak data from buffer */
      com_values[i] = c_inchar(p_port);
    lm_data[count].peak = convert_int(com_values);
    for (i=0;i<4;i++)  /* read y-cgt data from buffer */
      com_values[i] = c_inchar(p_port);
    lm_data[count].y_cgt = convert_double(com_values);
    for (i=0;i<4;i++)  /* read x-cgt data from buffer */
      com_values[i] = c_inchar(p_port);
    lm_data[count].x_cgt = convert_double(com_values);
  }

  /* check if more than exspected number of landmarks were
     reported. Doesn't include the case lm > LM_MAX */
  while( ((c = c_inchar(p_port)) != '?') && count < (LM_MAX -1))
  {
    /*-more landmarks than exspected are detected, wait until
       receive character buffer has filled up again. */

#if OPTION_INT
    disable();
    start = timer;
    enable();
#endif

    do
    {

#if OPTION_INT
      disable();
      temp = timer;
      enable();
      if (fabs(temp - start) > LM_WAIT)  /* trouble */
      {
        /* flush receive buffer completely */
        xdelay(1); /* wait 2 ms */
        while( (c = c_flush(p_port,0)) != EOF);
```

```
        count = 0;
        return count;
      }
#endif

    } while ( (i = rxbuf_test(p_port)) < 12);

    com_values[0] = c;  /* read area data from buffer */
    com_values[1] = c_inchar(p_port);
    lm_data[count].area = convert_int(com_values);
    for (i=0;i<2;i++)  /* read peak data from buffer */
      com_values[i] = c_inchar(p_port);
    lm_data[count].peak = convert_int(com_values);
    for (i=0;i<4;i++)  /* read y-cgt data from buffer */
      com_values[i] = c_inchar(p_port);
    lm_data[count].y_cgt = convert_double(com_values);
    for (i=0;i<4;i++)  /* read x-cgt data from buffer */
      com_values[i] = c_inchar(p_port);
    lm_data[count].x_cgt = convert_double(com_values);
    count++;
  }

  return count;
} /* end of routine get_landmark */

/*-----------------------------------------------------------------------------
ROUTINE
  GET_POSITION
  Klaus Obergfell

DESCRIPTION
  investigates landmark data and determines reference landmark and valid
  "moving" landmark by checking position, peak and area. Computes pixel
  coordinates of "moving" landmark relative to reference landmark,
  algorithm considers that reference landmark is not in "moving" landmark
  plane.

PARAMETER
  int number   - number of landmarks detected
  double *colp - pointer to relative column pixel coordinate
  double *rowp - pointer to relative row pixel coordinate

RETURN
  0 - error, too many landmarks and criteria is not sufficient to
      determine valid landmarks
  0 - success
  1 - error: More than one candidate for reference landmark
  2 - error: No candidate for reference landmark
  3 - error: More than one candidate for 'moving' landmark
  4 - error: No candidate for 'moving' landmark
-----------------------------------------------------------------------------*/
static int get_position (int number, double *colp, double *rowp)
{
  int reference=-1,moving=-1;  /* number of corosponding lm's
                                  in lm-data-array */
  int i;

  /* check for reference landmark */
  for (i=0;i<number;i++)
  {
    /* criteria for reference landmark */
    if(lm_data[i].y_cgt >= RLM_YPIXEL_MIN && lm_data[i].y_cgt <= RLM_YPIXEL_MAX &&
       lm_data[i].x_cgt <= RLM_XPIXEL_MAX && lm_data[i].x_cgt >= RLM_XPIXEL_MIN)
    {
      if(reference == -1)  /* first to meet criteria */
        reference = i;
      else                 /* more than one lm meets criteria */
        return 1;
    }
  }
  if(reference == -1)  /* couldn't find reference landmark */
    return 2;

  /* check for "moving" landmark */
  if(number == LM_EXSPECTED)  /* correct number of lm's */
  {
    if(reference == 1)
      moving = 0;
    else
      moving = 1;
  }
  else  /* more than exspected number of landmarks */
  {
    for (i=0;i<number;i++)
    {
      /* criteria for "moving" landmark */
      if(i != reference && lm_data[i].area >= MLM_AREA_MIN &&
         lm_data[i].area <= MLM_AREA_MAX && lm_data[i].peak > MLM_PEAK_MIN)
      {
        if(moving == -1)  /* first to meet criteria */
          moving = i;
        else              /* more than one lm meets criteria */
          return 3;
      }
    }
    if(moving == -1)  /* couldn't find moving landmark */
      return 4;
  }

  /* compute position */
  *colp = lm_data[moving].x_cgt - lm_data[reference].x_cgt - R_LM_ERROR_X;
  *rowp = - lm_data[moving].y_cgt + lm_data[reference].y_cgt - R_LM_ERROR_Y;

  return 0;
} /* end of routine get_position */

/*-----------------------------------------------------------------------------
FILE
  RXBUF_TEST
  Klaus Obergfell

DESCRIPTION
  checks number of bytes in the receive buffer for serial communication

PARAMETER
  COMM_PORT *p_port  - pointer to COMM_PORT structure, for serial comm.
```

```
-------------------------------------------------------------------------------*/
static int rxbuf_test (COMM_PORT *p_port)
{
  BYTE temp;
  int  result;

  temp = p_port->rxbufhead - p_port->rxbuftail;
  if (temp < 0)
    temp += RXBUFLENGTH;
  result = (int) temp;
  return result;
} /* end of routine rxbuf_test */

/*-------------------------------------------------------------------------------
ROUTINE
  CONVERT_DOUBLE
  Klaus Obergfell & Tony Smith

DESCRIPTION
  converts four digit hex number string to its decimal value

PARAMETER
  short *data  - pointer to hex number string

RETURN
  double       - decimal value (floating point number)
-------------------------------------------------------------------------------*/
static double convert_double (short *data)
{
  double result;

  result = 16.0 * digit(*data);
  result += digit(*(data+1));
  result += digit(*(data+2)) / 16.0;
  result += digit(*(data+3)) / 256.0;
  return result;
} /* end of routine convert_double */

/*-------------------------------------------------------------------------------
ROUTINE
  CONVERT_INT
  Klaus Obergfell & Tony Smith

DESCRIPTION
  converts two digit hex number string to its decimal value

PARAMETER
  short *data  - pointer to hex number string

RETURN
  int          - decimal value (integer number)
-------------------------------------------------------------------------------*/
static int convert_int (short *data)
{
  int result;

  result = 16 * digit(*data);
  result += digit(*(data+1));
  return result;
} /* end of routine convert_int */

/*-------------------------------------------------------------------------------
ROUTINE
  DIGIT
  Klaus Obergfell & Tony Smith

DESCRIPTION
  routine that returns the decimal value for a hex (ASCII) digit;

PARAMETER
  short ch  - hex (ASCII) digit

RETURN
  0     - error, not a hex digit
  != 0  - decimal value
-------------------------------------------------------------------------------*/
static int digit (short ch)
{
  if (isxdigit(ch))
  {
    if (isdigit(ch))
      return (ch - 48);
    else
      return (toupper(ch) - 55);
  }
  else
    return 0;
} /* end of routine digit */

/*-------------------------------------------------------------------------------
ROUTINE
  PIXEL2LENGTH_CONV
  Klaus Obergfell

DESCRIPTION
  converts relative landmark position from pixel unit to length in inch

PARAMETERS
  double colp  - relative column position in pixel
  double rowp  - relative row position in pixel
  double *x    - pointer to relative x position in inch
  double *y    - pointer to relative y position in inch
-------------------------------------------------------------------------------*/
static void pixel2length_conv (double colp, double rowp, double *x, double *y)
{
  *x = colp * INCH_PER_COL_PIXEL;
  *y = rowp * INCH_PER_ROW_PIXEL;
  return;
} /* end of routine pixel2length_conv */

/*******************************************************************************
end of file vstuff.c
*******************************************************************************/
```

```
/*****************************************************************************
FILE
  setup.h - function prototypes and external data of file setup.c
  Klaus Obergfell 1991
*****************************************************************************/

#include <comm.h>  /* since data type BYTE is declared in this file */

/*---------------------------------------------------------------------------
FUNCTION PROTOTYPES
---------------------------------------------------------------------------*/
extern int def_init (void);
extern void clean_up (void);
extern void do_init (void);

/*---------------------------------------------------------------------------
EXTERNAL DATA
---------------------------------------------------------------------------*/
extern COMM_PORT  port1;          /* port for serial comm. */
extern BYTE  rxbuf[];             /* receive buffer for serial comm. */
extern struct camera v_config;    /* camera/vision configuration */

extern double jacobian[];         /* Jacobian matrix for transformation from
                                     camera coordinates to joint coordinates */

/*****************************************************************************
end of file setup.h
*****************************************************************************/
```

```
/*****************************************************************************
FILE
  setup.c - system parameters and initialization
  Klaus Obergfell 1991

ROUTINES
  def_init          - default initialization at program start
  clean_up          - clean up at program termination
  do_init           - initialization menu
  init_menu         - prints initialization menu
  parameters        - parameter menu
  parameter_list    - prints parameter list
  set_gains         - choose controller gains
  set_threshold     - set camera threshold
  set_camera_output - set camera output configuration
  write_threshold   - write threshold data to serial interface
  do_data_menu      - choose data format for serial communication
  do_speed_menu     - choose baud rate for serial communication
  jacob_calib       - computes Jacobian for camera workspace
  store_jac         - stores Jacobian matrix to data file
  replace_jac       - replaces Jacobian matrix and stores to data file

REMARKS
  this file replaces paramet.c, setcom.c and parts of main.c. All
  system parameters and initialization procedures are collected in this
  file.

UPDATES
  07/20/91, transducer access only inside tmr_isr. Also transducer access
            inside this module, before and after setting up controller.
            (no change necessary inside this module)
  07/14/91, change in struct data to implement ring buffer
*****************************************************************************/

/*****************************************************************************
IMPORTS
*****************************************************************************/
#include "envir.h"        /* system environment, compile options */

#include "main.h"         /* system parameter, data types */

#include "util.h"         /* function prototypes, external data */
#include "alarm.h"
#include "control2.h"
#include "adstuff.h"
#include "convers.h"
#include "move_rob.h"
#include "vstuff.h"
#include "m_math.h"

#include <stdlib.h>       /* standard libraries */
#include <stdio.h>
#include <conio.h>
#include <ctype.h>
#include <bios.h>
#include <dos.h>

#include <comm.h>         /* serial communications */
```

```
/*****************************************************************************
DEFINES
*****************************************************************************/
#define MAX_CALIB_POINTS     12
#define DEF_JACOBIAN_FILE    "jacobian.def"

/*****************************************************************************
FILES
  pointers to data files are defined here, use extern statement in other
  files
*****************************************************************************/
FILE *setup_inout;
FILE *debug_out1;
FILE *debug_out2;
FILE *move_rob_out;
FILE *util_out;

/*****************************************************************************
FUNCTION PROTOTYPES
*****************************************************************************/
static void init_menu(void);
static void parameters(COMM_PORT *);
static void parameter_list(COMM_PORT *);
static void set_gains(void);
static void set_threshold(COMM_PORT *, char, char);
static void set_camera_output(COMM_PORT *);
static void write_threshold(COMM_PORT *, char, char, char);
static void do_data_menu(COMM_PORT *);
static long do_speed_menu(COMM_PORT *);
static int jacob_calib(void);
static void store_jac(void);
static void replace_jac(void);

/*****************************************************************************
EXTERNAL DATA
*****************************************************************************/
COMM_PORT  port1;                /* port for serial comm. */
BYTE  rxbuf[RXBUFLENGTH];        /* receive buffer for serial comm. */
/* camera configuration: set to board default */
struct camera v_config = { '3','0','2','0','0','4','0','3','0','0','0','0',
  'F','0'};

void interrupt (*oldtmr_isr)(); /* pointer to old timer isr */

double jacobian[4];  /* Jacobian matrix for transformation from camera
                        coordinates to joint coordinates */

/*****************************************************************************
PRIVATE DATA
*****************************************************************************/
static char *parity_strings[] = {"NONE","ODD","EVEN","MARK","SPACE"};
static double new_jacobian[2][2];  /* Jacobian matrix */
#if OPTION_DEBUG
static double abs_jacobian[2][2];
#endif

/*****************************************************************************
VISIBLE ROUTINES
*****************************************************************************/

/*-----------------------------------------------------------------------------
ROUTINE
  DEF_INIT
  Klaus Obergfell

DESCRIPTION
  performs default initialization after program start

RETURN
  0 - successful
  1 - error
-----------------------------------------------------------------------------*/
int def_init (void)
{
  short   key;
  int     i;
#if OPTION_DAS16
  double  addata[8];     /* array of a/d numbers */
  double  th1,th2;       /* joint angles */
#endif

#if OPTION_COMM
  long    baud_rate;
#endif

  /* write start message to screen */
  message();

  /* set up data saving facility */
  data1.s_flag = 0;        /* no data saving */
  data1.f_flag = 0;        /* memory is empty */
  data1.buffer_head = 0;
  data1.buffer_tail = 0;

  /* retrieve default Jacobian from datafile */
  setup_inout = fopen(DEF_JACOBIAN_FILE,"r");
  for (i=0;i<4;i++)
      fscanf(setup_inout,"%lf",&jacobian[i]);
  fclose(setup_inout);

#if OPTION_DAS16
  /* initialize a/d board */
  printf("\n\nInitialize A/D board");
  if (setup_das16() == 1)
    return 1;
#endif

#if OPTION_COMM
  /* initialize serial communication */
  printf("\nInitialize serial communication");
  printf("\nPlease enter baud rate (2400,9600,19200,etc..)");
  scanf("\n%ld",&baud_rate);
  if(init_port(&port1, COM1, baud_rate, DATABITS8, PARITY_NONE, STOPBITS1)!=0)
    return 1;
  install_ipr(&port1, RECEIVE, NULL, rxbuf, sizeof(rxbuf));
```

```
  install_isr(&port1, 4, NULL);

  /* reset camera message, wait 'till key hit, write new
     defaults to camera 1 */
  printf("\nPlease reset camera, press ALT-M when done !");
  while ( (key = inkey()) != MENU);
  printf("\nWriting new Camera configuration");

  /* internal camera configuration */
  v_config.minarea[0] = '1';
  v_config.minarea[1] = '0';
  v_config.strobe_charge_time[0] = '4';
  v_config.strobe_charge_time[1] = '0';

  /* write thresholds to camera */
  write_threshold(&port1,'2','1','0');
  write_threshold(&port1,'4','4','0');

  /* take initial picture, since first picture contains garbage */
  c_putc(&port1,'P');

  /* flush receive buffer completely */
  xdelay(10); /* wait 20 ms for char to return */
  while( (key = c_flush(&port1,0)) != EOF);
  /* wait for '?' returned from picture routine */
  while ( (key = c_inchar(&port1)) != '?');
#endif

  /*
  initialize data structure and desired joint angles for
  controller
  */
  printf("\ninitialize controller data and parameters");
  traj1.flag = 1;   /* set hold flag */
  gains1.p1 = 1.1;    /* default gains */
  gains1.p2 = -1.4;

#if OPTION_DAS16
  /*
  get current joint angles from a/d board and ask user to confirm
  as initial desired joint angles. If data acquisition board is not used,
  (debugging) some default values for desired and actual joint angles are
  defined within the control algorithm.
  */
  readad(addata);
  angles(addata,&th1,&th2);
  printf("\n\nCurrent joint angles: th1 = %.2f, th2 = %.2f",th1*180/pi,
    th2*180/pi);
  printf("\nPlease confirm as desired joint angles (y/n):");
  key = bioskey(0) & 0xff;
  if(toupper(key) != 'Y')
    return 1;
  ws_limits(&th1,&th2);
  th1d_hold = th1;
  th2d_hold = th2;
#endif

#if OPTION_INT
  /* initialize interrupt driven controller */
  printf("\nStart interrupt driven control action");
#if OPTION_ALARM
  setalarm(SAMPLE_T);  /* set 8259 timer to tick every SAMPLE_T ms */
#endif
  oldtmr_isr = getvect(0x08);
  setvect(0x08,tmr_isr);  /* set up timer interrupt service routine */
#endif

  xdelay(500); /* wait 2 seconds */
  return 0;
} /* end of routine def_init */

/*---------------------------------------------------------------------------
ROUTINE
  CLEAN_UP
  Klaus Obergfell

DESCRIPTION
  sets everything back to DOS defaults before regular program execution
---------------------------------------------------------------------------*/
void clean_up (void)
{
#if OPTION_INT
  /* stop controller, clean up timer and isr */
  setvect(0x08,oldtmr_isr);  /* restore old timer isr */
#if OPTION_ALARM
  setalarm(-1.0);            /* restore timer to DOS default */
#endif
#endif

#if OPTION_DAS16
  /* write zeros to d/a converter */
  writeda_i(0,0);     /* can use "interrupt driven" routine, since done */
#endif

#if OPTION_COMM
  /* clean up serial communication */
  deinit_port(&port1);
#endif

  return;
} /* end of routine clean_up */

/*---------------------------------------------------------------------------
ROUTINE
  DO_INIT
  Klaus Obergfell

DESCRIPTION
  loops thru initialization menu and performs call to chosen subroutine
  until 'exit option is selected
---------------------------------------------------------------------------*/
void do_init (void)
{
  short key, DONE = false;

  do
```

```
  {
    clrscr();
    init_menu();
    key = bioskey(0) & 0xFF;
    switch (toupper(key))
    {
      case 'P':  /* Parameters */
        parameters(&port1);
        break;
#if OPTION_COMM
      case 'T':  /* Terminal program (communicate with camera) */
        term(&port1);
        break;
#endif
      case 'J':  /* Jacobian calibration */
#if (OPTION_INT && OPTION_DAS16 && OPTION_COMM)
        jacob_calib();
        wait_key_press();
#endif
        break;
      case 'H':  /* Help */
        break;
      case 'Q':  /* Quit (exit this menu) */
        DONE = TRUE;
        break;
    }
  } while(!DONE);
  return;
} /* end of routine do_init */

/*****************************************************************************
PRIVATE ROUTINES
*****************************************************************************/

/*---------------------------------------------------------------------------
ROUTINE
  INIT_MENU
  KLAUS OBERGFELL

DESCRIPTION
  prints initialization menu
---------------------------------------------------------------------------*/
static void init_menu (void)
{
  printf("\nRALF initialization menu, choose option:\n");
  printf("\nP.    Parameters");

#if OPTION_COMM
  printf("\nT.    Terminal Program (communicate with camera)");
#endif

  printf("\nJ.    Jacobian calibration");
  printf("\nH.    Help (*)");
  printf("\nQ.    Quit (exit this menu)\n");
  printf("\n      (*) = not available in current installation");
  return;
} /* end of routine init_menu */
```

```
/*---------------------------------------------------------------------------
ROUTINE
  PARAMETERS

DESCRIPTION
  loops thru parameter menu: sets parameters or quit.
  The following system parameters can be modified:
  - controller gains
  - camera threshold and flags:
    - min pixel value to start growing a blob
    - min pixel value to be included in blob
    - min area of blob to be reported
    - integration time
    - strobe charge time
    - repetition factor for integration
    ( - configuration for landmark output, future option)

PARAMETERS
  COMM_PORT *p_port  - pointer to COMM_PORT structure for serial comm.
---------------------------------------------------------------------------*/
static void parameters (COMM_PORT *p_port)
{
  short key, DONE = FALSE;

  do
  {
    clrscr();
    parameter_list(p_port);
    key = bioskey(0) & 0xFF;
    switch (toupper(key))
    {
      case 'A':  /* Proportional gain link 1, sets also p-gain 2 */
        set_gains();
        break;
      case 'B':  /* Proportional gain link 2, sets also p-gain 1 */
        set_gains();
        break;

#if OPTION_COMM
      case 'C':  /* Min pixel value to start growing landmark */
        set_threshold(p_port, '0', 'C');
        break;
      case 'D':  /* Min pixel value to be included in landmark */
        set_threshold(p_port, '1', 'D');
        break;
      case 'E':  /* Min area of blob to be reported */
        set_threshold(p_port, '2', 'E');
        break;
      case 'F':  /* Integration time */
        set_threshold(p_port, '3', 'F');
        break;
      case 'G':  /* Strobe charge time */
        set_threshold(p_port, '4', 'G');
        break;
      case 'H':  /* Repetition factor for integration time */
        set_threshold(p_port, '5', 'H');
        break;
/*
```

```
      future option: Camera output

      case 'I':
        set_camera_output(p_port, 'I');
        break;
*/
      case 'J':  /* Data format for camera communication */
        do_data_menu(p_port);
        break;
      case 'K':  /* Baud rate for camera communication */
        do_speed_menu(p_port);
        break;
#endif

      case 'Q':
        DONE = TRUE;
        break;
    }
  } while(!DONE);

#if OPTION_COMM
  /* flush receive buffer completely */
  while( (key = c_flush(&port1,0)) != EOF);
#endif

  return;
} /* end of routine parameters */

/*---------------------------------------------------------------------------
ROUTINE
  PARAMETER_LIST
  Klaus Obergfell

DESCRIPTION
  prints parameter list

PARAMETER
  COMM_PORT *p_port  - pointer to COMM_PORT structure, for serial comm.
---------------------------------------------------------------------------*/
static void parameter_list (COMM_PORT *p_port)
{
  printf("\nA.  Proportional gain joint 1 = %.2f",gains1.p1);
  printf("\nB.  Proportional gain joint 2 = %.2f",gains1.p2);

#if OPTION_COMM
  printf("\n");
  printf("\nC.  Min pixel value to start growing landmark   = %c%c  (def=30)",
    v_config.minval_grow[0],v_config.minval_grow[1]);
  printf("\nD.  Min pixel value to be included in landmark  = %c%c  (def=20)",
    v_config.minval_incl[0],v_config.minval_incl[1]);
  printf("\nE.  Min area of landmark to be reported         = %c%c  (def=04)",
    v_config.minarea[0],v_config.minarea[1]);
  printf("\nF.  Integration time                            = %c%c  (def=03)",
    v_config.integ_time[0],v_config.integ_time[1]);
  printf("\nG.  Strobe charge time                          = %c%c  (def=00)",
    v_config.strobe_charge_time[0],v_config.strobe_charge_time[1]);
  printf("\nH.  Repetition factor for integration           = %c%c  (def=00)",
    v_config.mult_integ[0],v_config.mult_integ[1]);
```

```
/*
  future option:

  printf("\nI.  Nibble that determines camera output        = %c  (def=0)",
    v_config.camera_output[1]);
  printf("\n       0:  output = area, peak, y - cgt, x - cgt");
  printf("\n       1:  output = peak, y - cgt, x - cgt");
  printf("\n       2:  output = area, y - cgt, x - cgt");
  printf("\n       3:  output = y - cgt, x - cgt");
*/
  printf("\n");
  printf("\nJ.  Format: %d Data bits, %s Parity, %d Stopbits",
    p_port->rxdatabits,
    parity_strings[p_port->parity],
    p_port->stopbits
  );
  printf("\nK.  Speed:  %ld bits-per-second", p_port->speed);
#endif

  printf("\n\nQ.  Quit (exit this menu)");
  return;
} /* end of routine parameter_list */

/*---------------------------------------------------------------------------
ROUTINE
  SET_GAINS
  Klaus Obergfell

DESCRIPTION
  prompts user for new gain values, disables interrupts, sets gains
  and enables interrupts again. This assures that gains are changed
  at the same time.
---------------------------------------------------------------------------*/
static void set_gains (void)
{
  double temp1,temp2;

  printf("\n\nEnter new values for gains (p1,p2)");
  scanf("%lf,%lf",&temp1,&temp2);
  disable();
  gains1.p1 = temp1;
  gains1.p2 = temp2;
  enable();
  return;
} /* end of routine set_gains */

/*---------------------------------------------------------------------------
ROUTINE
  SET_THRESHOLD
  Klaus Obergfell

DESCRIPTION
  sets camera thresholds

PARAMETER
  COMM_PORT *p_port  - pointer to COMM_PORT structure, for serial comm.
  char number        - flag number
  char letter        - letter of chosen option
```

```
------------------------------------------------------------------------*/
static void set_threshold (COMM_PORT *p_port, char number, char letter)
{
  char nv1,nv2;
  short nv1_valid,nv2_valid;

  /* input and verification */
  do
  {
    printf("\nEnter new value for %c.:",letter);
    scanf("\n%c%c",&nv1,&nv2);
    nv1_valid = ((nv1 > 47 && nv1 < 58) || (nv1 > 64 && nv1 < 71));
    nv2_valid = ((nv2 > 47 && nv2 < 58) || (nv2 > 64 && nv2 < 71));
  } while ( (!nv1_valid) || (!nv2_valid) );

  /* update internal camera configuration representation */
  switch (letter)
  {
    case 'C':
      v_config.minval_grow[0] = nv1;
      v_config.minval_grow[1] = nv2;
      break;
    case 'D':
      v_config.minval_incl[0] = nv1;
      v_config.minval_incl[1] = nv2;
      break;
    case 'E':
      v_config.minarea[0] = nv1;
      v_config.minarea[1] = nv2;
      break;
    case 'F':
      v_config.integ_time[0] = nv1;
      v_config.integ_time[1] = nv2;
      break;
    case 'G':
      v_config.strobe_charge_time[0] = nv1;
      v_config.strobe_charge_time[1] = nv2;
      break;
    case 'H':
      v_config.mult_integ[0] = nv1;
      v_config.mult_integ[1] = nv2;
      break;
  }

  /* change camera configuration */
  write_threshold(p_port,number,nv1,nv2);

  return;
} /* end of routine set_threshold */

/*------------------------------------------------------------------------
ROUTINE
  SET_CAMERA_OUTPUT
  Klaus Obergfell

DESCRIPTION
  sets camera output

PARAMETER
  COMM_PORT *p_port  - pointer to COMM_PORT structure, for serial comm.

NOTE
  this is a future option, the existing code only works when all
  landmark data (area, peak, x- and y-cgt) is send from camera.
------------------------------------------------------------------------*/
static void set_camera_output (COMM_PORT *p_port)
{
  char new_value;

  /* input and verification */
  do
  {
    printf("\nEnter new value for 1.:");
    scanf("\n%c",&new_value);
  } while (new_value < 48 || new_value > 51);

  /* update internal camera configuration representation */
  v_config.camera_output[1] = new_value;

  /* change camera configuration */
  write_threshold(p_port,'7','F',new_value);

  return;
} /* end of routine set_camera_output */

/*------------------------------------------------------------------------
ROUTINE
  WRITE_THRESHOLD
  Klaus Obergfell

DESCRIPTION
  writes new threshold value to camera

PARAMETER
  COMM_PORT *p_port  - pointer to COMM_PORT structure, for serial comm.
  char number        - threshold number
  char nv1,nv2       - new threshold values
------------------------------------------------------------------------*/
static void write_threshold (COMM_PORT *p_port, char number, char nv1, char nv2)
{
    c_putc(p_port,'t');
    c_putc(p_port,number);
    c_putc(p_port,nv1);
    c_putc(p_port,nv2);
    c_flush(p_port,0);  /* flush receive buffer completely */
    return;
} /* end of routine write_threshold */

/*------------------------------------------------------------------------
ROUTINE
  DO_DATA_MENU

DESCRIPTION
  Show a menu for setting the data tranfer rate and format.

PARAMETER
```

```
  COMM_PORT *p_port  - pointer to COMM_PORT structure, for serial comm.
-------------------------------------------------------------------------------*/
static void do_data_menu(COMM_PORT *p_port)
{
  short key, DONE = FALSE;

  do
  {
    clrscr();
    puts("\nSET THE DATA FORMAT");
    printf("\nFormat: %d Data bits, %s Parity, %d Stopbits\n",
      p_port->rxdatabits,
      parity_strings[p_port->parity],
      p_port->stopbits
    );
    puts("A. Set number of databits");
    puts("B. Set number of stopbits");
    puts("C. Set parity");
    puts("\nAny other key exits\n");
    key = bios_key(0) & 0XFF;
    switch (toupper(key))
    {
      case 'A':
        puts("\nEnter databits (5, 6, 7 or 8): ");
        key = bios_key(0) & 0XF;
        set_databits(p_port, key);
        break;
      case 'B':
        puts("\nEnter stopbits (1 or 2): ");
        key = bios_key(0) & 0XF;
        set_stopbits(p_port, key);
        break;
      case 'C':
        puts("\nEnter parity ('N' for none, 'E' for even, etc.): ");
        key = bios_key(0) & 0XFF;
        key = toupper(key);
        if (key == 'N') set_parity(p_port, PARITY_NONE);
          else if (key == 'E') set_parity(p_port, PARITY_EVEN);
          else if (key == 'O') set_parity(p_port, PARITY_ODD);
          else if (key == 'M') set_parity(p_port, PARITY_MARK);
          else if (key == 'S') set_parity(p_port, PARITY_SPACE);
          else puts("\nNot recognized.  Use 'N', 'E', 'O', 'M' or 'S'");
        break;
      default:
        DONE = TRUE;
        break;
    }
  } while (!DONE);
  return;
} /* end of routine do_data_menu */

/*-------------------------------------------------------------------------------
ROUTINE
  DO_SPEED_MENU

DESCRIPTION
  Show a menu for setting the data tranfer rate.

PARAMETER
  COMM_PORT *p_port  - pointer to COMM_PORT structure, for serial comm.
-------------------------------------------------------------------------------*/
static long do_speed_menu(COMM_PORT *p_port)
{
  short key;
  long ret;

  do
  {
    clrscr();
    puts("\nSET THE DATA TRANSFER SPEED");
    printf("\nSpeed:  %ld bits-per-second\n", p_port->speed);
    puts("A. Data transfer rate     50    bits-per-second");
    puts("B. Data transfer rate     75    bits-per-second");
    puts("C. Data transfer rate    110    bits-per-second");
    puts("D. Data transfer rate    134.5  bits-per-second");
    puts("E. Data transfer rate    150    bits-per-second");
    puts("F. Data transfer rate    300    bits-per-second");
    puts("G. Data transfer rate    600    bits-per-second");
    puts("H. Data transfer rate   1200    bits-per-second");
    puts("I. Data transfer rate   2400    bits-per-second");
    puts("J. Data transfer rate   4800    bits-per-second");
    puts("K. Data transfer rate   9600    bits-per-second");
    puts("L. Data transfer rate  19200    bits-per-second");
    puts("M. Data transfer rate  38400    bits-per-second");
    puts("N. Data transfer rate  57600    bits-per-second");
    puts("O. Data transfer rate 115200    bits-per-second");
    printf("\n\tNew data transfer rate: ");
    fflush(stdout);
    key = bios_key(0) & 0XFF;
    key = toupper(key);
  } while (key < 'A' || key > 'O');

  switch (key)
  {
    case 'A':
      ret = 50;
      break;
    case 'B':
      ret = 75;
      break;
    case 'C':
      ret = 110;
      break;
    case 'D':
      puts("\nSorry, that data transfer rate not supported");
      ret = EOF;
      break;
    case 'E':
      ret = 150;
      break;
    case 'F':
      ret = 300;
      break;
    case 'G':
      ret = 600;
      break;
```

```
    case 'H':
      ret = 1200;
      break;
    case 'I':
      ret = 2400;
      break;
    case 'J':
      ret = 4800;
      break;
    case 'K':
      ret = 9600;
      break;
    case 'L':
      ret = 19200;
      break;
    case 'M':
      ret = 38400L;
      break;
    case 'N':
      ret = 57600L;
      break;
    case 'O':
      ret = 115200L;
      break;
    default:
      ret = EOF;
  }
  if (ret != EOF)
  {
    set_speed(p_port, ret);
  }
  return ((long) ret);
} /* end of routinee do_speed_menu */

/*---------------------------------------------------------------------------
ROUTINE
  JACOB_CALIB
  Klaus Obergfell 1991

DESCRIPTION
  this routine computes a constant Jacobian for the camera workspace.

RETURN
  0  - success
  1  - error
---------------------------------------------------------------------------*/
int jacob_calib(void)
{
                                        /* desired coordinates: */
  double xyabs_d[MAX_CALIB_POINTS][2];  /* abs. tip coord. (w.r.t. base) */
                                        /* measured coordinates: */
  double theta_m[MAX_CALIB_POINTS][2];  /* joint angles */
  double xycam_m[MAX_CALIB_POINTS][2];  /* rel. tip coord. (w.r.t. camera) */
  double delta_th[MAX_CALIB_POINTS-1][2]; /* differences in joint angles */
  double delta_xy[MAX_CALIB_POINTS-1][2]; /* differences in rel. tip coord. */

#if OPTION_DEBUG
  double xyabs_m[MAX_CALIB_POINTS][2];  /* abs. tip coord. (w.r.t. base) */
  double delta_xyabs[MAX_CALIB_POINTS-1][2]; /* diff. in abs. tip coord. */
#endif

  double *mtemp1,*mtemp2,*mtemp3,*mtemp4; /* pointers to matricies for
                                             Jacobian calculation */
  double temp1,temp2;
  double th1s,th1f,th2s,th2f;
  int    i,count,count_max,itemp1;
  short  key,DONE=False,DONE2=False;

  /*
  print info to screen, enter number of calibration points, enter
  coordinates for all calibration points
  */
  clrscr();
  printf("Jacobian calibration routine");
  printf("\ncomputes a constant value Jacobian for the camera workspace\n");
  print_camera_ws_info('A');  /* abs. coordinates */
  itemp1 = MAX_CALIB_POINTS;
  do
  {
    printf("\n\nPlease enter number of calibration points (max = %d)",itemp1);
    scanf("%d",&count_max);
  } while(count_max <= 1 || count_max > MAX_CALIB_POINTS);
  for(i=0;i<count_max;i++)
  {
    DONE2 = False;
    do
    {
      printf("Please enter abs. x,y coordinates for %d. calibration point",i);
      scanf("%lf,%lf",&temp1,&temp2);
      itemp1 = camera_ws_boundary_test(temp1,temp2,'A');
      if (itemp1 == 0)
        DONE2 = True;
      else
        printf("\nCalibration point outside camera workspace\n");
    } while(!DONE2);
    xyabs_d[i][0] = temp1;
    xyabs_d[i][1] = temp2;
  }
  do
  {
    printf("\nPlease confirm calibration points: (y/n)\n");
    for(i=0;i<count_max;i++)
      printf("\n%d.  %.1f , %.1f",i,xyabs_d[i][0],xyabs_d[i][1]);
    key = bioskey(0) & 0xff;
    if(toupper(key) == 'Y')
      DONE = True;
    else
    {
      DONE2 = False;
      do
      {
        printf("\n\nPlease enter number of bad calibration point, -1 terminates");
        scanf("%d",&i);
        if(i == -1)  /* terminate */
          return 1;
        else if(i >= 0 && i < count_max)  /* valid */
```

```
        DONE2 = True;
      } while(!DONE2);
      DONE2 = false;
      do
      {
        printf("\nPlease enter abs. x,y coordinates for %d. calibration point",
          i);
        scanf("%lf,%lf",&temp1,&temp2);
        itemp1 = camera_ws_boundary_test(temp1,temp2,'A');
        if (itemp1 == 0)
          DONE2 = True;
        else
          printf("\nCalibration point outside camera workspace");
      } while(!DONE2);
      xyabs_d[i][0] = temp1;
      xyabs_d[i][1] = temp2;
    }
  } while(!DONE);

  /****************** loop until all data points finished ******************/
  count = 0;
  do
  {
    /* move robot, wait until done */
    move_robot_auto(xyabs_d[count][0],xyabs_d[count][1],'X',3.);
    while(!traj.flag);

    /* take picture, get tip coordinates, get joint coordinates */
    disable();
    th1s = th1actual;
    th2s = th2actual;
    enable();
    itemp1 = tip_position(&temp1,&temp2);
    if(itemp1 != 0)  /* valid tip position ? */
    {
      vision_error_message(itemp1);
      printf("\nJacobian calibration terminated at %d calibration point",count);
      return 1;
    }
    xycam_m[count][0] = temp1;
    xycam_m[count][1] = temp2;
    disable();
    th1f = th1actual;
    th2f = th2actual;
    enable();
    theta_m[count][0] = (th1s + th1f) / 2.;
    theta_m[count][1] = (th2s + th2f) / 2.;

#if OPTION_DEBUG
    forward(theta_m[count][0],theta_m[count][1],&temp1,&temp2);
    xyabs_m[count][0] = temp1;
    xyabs_m[count][1] = temp2;
#endif

    count++;
  } while(count < count_max);
  /*********************** end of loop ****************************/
```

```
  /* compute differences */
  for(i=0;i<count_max-1;i++)
  {
    delta_th[i][0] = theta_m[i+1][0] - theta_m[i][0];
    delta_th[i][1] = theta_m[i+1][1] - theta_m[i][1];
    delta_xy[i][0] = xycam_m[i+1][0] - xycam_m[i][0];
    delta_xy[i][1] = xycam_m[i+1][1] - xycam_m[i][1];
#if OPTION_DEBUG
    delta_xyabs[i][0] = xyabs_m[i+1][0] - xyabs_m[i][0];
    delta_xyabs[i][1] = xyabs_m[i+1][1] - xyabs_m[i][1];
#endif
  }

#if OPTION_DEBUG
  /* for debugging: print all data to file */
  setup_inout = fopen("jacocalc.d","a");
  fprintf(setup_inout,"\n\nnext:\n");
  for (i=0;i<count_max;i++)
    fprintf(setup_inout,"\n%.3f  %.3f    %.2f  %.2f    %.2f  %.2f",
      theta_m[i][0],theta_m[i][1],xycam_m[i][0],xycam_m[i][1],xyabs_m[i][0],
      xyabs_m[i][1]);
  fprintf(setup_inout,"\n");
  for (i=0;i<count_max-1;i++)
    fprintf(setup_inout,"\n%.3f  %.3f    %.2f  %.2f    %.2f  %.2f",
      delta_th[i][0],delta_th[i][1],delta_xy[i][0],delta_xy[i][1],
      delta_xyabs[i][0],delta_xyabs[i][1]);
  fclose(setup_inout);
#endif

  /* allocate memory for matrix computation */
  mtemp1 = (double*)malloc(2*(count_max-1)*sizeof(double));
  mtemp2 = (double*)malloc(2*2*sizeof(double));
  mtemp3 = (double*)malloc(2*2*sizeof(double));
  mtemp4 = (double*)malloc(2*(count_max-1)*sizeof(double));

  /* matrix calculation for new jacobian */
  mtrans(delta_th,mtemp1,count_max-1,2);
  mmmult(mtemp1,delta_th,mtemp2,2,count_max-1,2);
  minvert(mtemp2,mtemp3,2);
  mmmult(mtemp3,mtemp1,mtemp4,2,2,count_max-1);
  mmmult(mtemp4,delta_xy,mtemp3,2,count_max-1,2);
  mtrans(mtemp3,mtemp2,2,2);

  /* invert Jacobian */
  minvert(mtemp2,mtemp3,2);

  /* new Jacobian */
  for (count=0;count<2;count++)
  {
    for (i=0;i<2;i++)
      new_jacobian[count][i] = mtemp3[count*2+i];
  }

#if OPTION_DEBUG
  /* matrix calculation for absolute jacobian */
  mtrans(delta_th,mtemp1,count_max-1,2);
  mmmult(mtemp1,delta_th,mtemp2,2,count_max-1,2);
  minvert(mtemp2,mtemp3,2);
```

```
  mmmult(mtemp3,mtemp1,mtemp4,2,2,count_max-1);
  mmmult(mtemp4,delta_xyabs,mtemp3,2,count_max-1,2);
  mtrans(mtemp3,mtemp2,2,2);

  /* absolute Jacobian */
  for (count=0;count<2;count++)
  {
    for (i=0;i<2;i++)
      abs_jacobian[count][i] = mtemp2[count*2+i];
  }

  /* print absolute Jacobian to datafile */
  setup_inout = fopen("absjaco.d","a");
  fprintf(setup_inout,"\nnext:\n");
  for (count=0;count<2;count++)
  {
    fprintf(setup_inout,"\n");
    for (i=0;i<2;i++)
      fprintf(setup_inout,"  %.8f",abs_jacobian[count][i]);
  }
  fclose(setup_inout);

#endif

  /* free memory */
  free(mtemp1);
  mtemp1 = NULL;
  free(mtemp2);
  mtemp2 = NULL;
  free(mtemp3);
  mtemp3 = NULL;
  free(mtemp4);
  mtemp4 = NULL;

  /*
  print Jacobian to screen, prompt user for usage of computed Jacobian
  */
  printf("\n\nthe resulting Jacobian is:");
  for (count=0;count<2;count++)
  {
    printf("\n");
    for (i=0;i<2;i++)
      printf("%.5f",new_jacobian[count][i]);
  }
  DONE = false;
  do
  {
    printf("\n\nYou have the following choices for using this Jacobian:");
    printf("\n1.  replace default Jacobian, start using new Jacobian
      immediately");
    printf("\n2.  replace default Jacobian, keep using old Jacobian");
    printf("\n3.  store Jacobian with diffrent name, start using");
    printf("\n4.  store Jacobian with diffrent name, keep using old Jacobian");
    printf("\n5.  ignore calibration, i.e. don't store, don't use");
    key = bioskey(0) & 0xff;
    switch(toupper(key))
    {
      case '1':
        replace_jac();
        for (count=0;count<2;count++)
          for (i=0;i<2;i++)
            jacobian[2*count+i] = new_jacobian[count][i];
        DONE = True;
        break;
      case '2':
        replace_jac();
        DONE = True;
        break;
      case '3':
        store_jac();
        for (count=0;count<2;count++)
          for (i=0;i<2;i++)
            jacobian[2*count+i] = new_jacobian[count][i];
        DONE = True;
        break;
      case '4':
        store_jac();
        DONE = True;
        break;
      case '5':
        DONE = True;
        break;
    }
  } while(!DONE);

  return 0;
} /* end of routine jacob_calib */

/*---------------------------------------------------------------------------
ROUTINE
  STORE_JAC
  Klaus Obergfell 1991

DESCRIPTION
  stores Jacobian to user specified datafile
---------------------------------------------------------------------------*/
static void store_jac(void)
{
  char   datafile[12];
  int    count,i;

  printf("\nPlease enter name of datafile, for storing Jacobian");
  scanf("%s",datafile);

  setup_inout = fopen(datafile,"w");
  fprintf(setup_inout,"\n");
  for (count=0;count<2;count++)
  {
    fprintf(setup_inout,"\n");
    for (i=0;i<2;i++)
      fprintf(setup_inout,"  %.8f",new_jacobian[count][i]);
  }
  fclose(setup_inout);

  return;
} /* end of routine store_jac */
```

```
/*-------------------------------------------------------------------------
ROUTINE
  REPLACE_JAC
  Klaus Obergfell 1991

DESCRIPTION
  stores Jacobian to default datafile
-------------------------------------------------------------------------*/
static void replace_jac(void)
{
  int count,i;

  setup_inout = fopen(DEF_JACOBIAN_FILE,"w");
  fprintf(setup_inout,"\n");
  for (count=0;count<2;count++)
  {
    fprintf(setup_inout,"\n");
    for (i=0;i<2;i++)
      fprintf(setup_inout,"  %.8f",new_jacobian[count][i]);
  }
  fclose(setup_inout);

  return;
} /* end of routine replace_jac */


/*************************************************************************************
end of file setup.c
```

```
/*****************************************************************************
FILE
  adstuff.h - function prototypes to file adstuff.c
  Klaus Obergfell 1991
*****************************************************************************/

extern int  setup_das16(void);
extern int  readad(double []);
extern int  readad_i(double []);
extern int  writeda_i(int, int);
extern void d2a_limits_i(int *, int *);

/*****************************************************************************
end of file adstuff.h
*****************************************************************************/
```

```
/*****************************************************************************
FILE
  adstuff.c - routines for das16 data acquisition board
  Klaus Obergfell 1991

ROUTINES
  setupad       - initialize board and set scan limits
  readad        - read a2d channels, called from outside interrupt routines
  readad_i      - read a2d channels, called from interrupt routines
  writeda_i     - write d2a ports, called from interrupt routines
  d2a_limits_i  - check d2a limits, called from interrupt routines

REMARKS
  link with library DAS16C<x>.LIB in c:\klaus\lib. <x> = S,C,M,L

UPDATES
  07/16/91, changed channel configurations and channel scan, averaged
    readings in readad and readad_i.
  07/11/91, clean up
  06/20/91, don't initialize das16 timer (not in use anyway),
    clean up error handling of isr called functions (no printf),
    make variables in isr called functions static.
  06/08/91, introduce two sets of identical functions, one set
    is used by interrupt driven functions, the other set is used by
    foreground functions. The functions that are called from interrupts
    have the ending "_i" in their function name.
    When the das16 board is read in main, interrupts are enabled.
*****************************************************************************/

/*****************************************************************************
IMPORTS
*****************************************************************************/
#include "envir.h"      /* compiler options */

#include <stdio.h>      /* standard libraries */
#include <stdlib.h>

/*****************************************************************************
IMPORTS
*****************************************************************************/
#define CHANNEL_SCANS  4    /* number of channel scans for averaging */

/*****************************************************************************
FUNCTION PROTOTYPES
*****************************************************************************/
int DAS16(int mode ,int *data);

/*****************************************************************************
VISIBLE ROUTINES
*****************************************************************************/

/*---------------------------------------------------------------------------
ROUTINE
  SETUP_DAS16
  Klaus Obergfell 04/29/91

DESCRIPTION
  initializes das16 board, sets scan limits (2 channel scan, channels 0 - 1)
```

```
RETURN
  flag = 0 - successful
  flag = 1 - error
-------------------------------------------------------------------------------*/
int setup_das16(void)
{
  int error, mode, data[5], flag=0;

  /* initialize das16 board */
  mode = 0;
  data[0] = 0x330;  /* base address */
  data[1] = 2;
  data[2] = 1;
  if ((error = DAS16(mode,data)) != 0) {
    printf("DAS16 not initialized, error number %d!\n",error);
    flag = 1;
    return flag;
  }

  /* set up timer */
/*  out(0x330+10, 2);
  mode = 10;
  data[0] = 0;
  if ((error = DAS16(mode,data)) != 0) {
    printf("DAS16 timer set up error, error number %d!\n",error);
    flag = 1;
    return flag;
  }*/

  /* set scan limits */
  mode = 1;
  data[0] = 0;
  data[1] = 1;
  if ((error = DAS16(mode,data)) != 0) {
    printf("DAS16 set scan limits error,error number %d!\n",error);
    flag = 1;
    return flag;
  }

  return flag;
} /* end of routine setup_das16 */

/*-------------------------------------------------------------------------
ROUTINE
  READAD
  Klaus Obergfell

DISCRIPTION
  reads das16 channels 0 - 1 repeated times and averages, remaining
  channels 2 - 7 are set to zero.

PARAMETER
  double channel[] - pointer to returned data

RETURN
  flag = 0 - successful
  flag = 1 - error
-------------------------------------------------------------------------------*/
int readad (double channel[])
{
  int error,mode,data[5],flag,k,l;
  double a2dvalue_sum[2];

  mode = 3;  /* single channel read operation */
  flag = 0;
  for(k=0;k<2;k++)
    a2dvalue_sum[k] = 0.0;  /* initialize sum of a2d values */

  /* do repeated scans */
  for (k=0;k<CHANNEL_SCANS;k++)
  {
    /* sample two channels */
    for (l=0; l < 2; l++)
    {
      if ((error = DAS16(mode,data)) != 0)
      {
        printf("Error reading das16 board, error number %d!\n",error);
        flag = 1;
        return flag;
      }
      a2dvalue_sum[l] += ((double) data[0]);
    }
  }

  for(k=0;k<2;k++)
    channel[k] = a2dvalue_sum[k]/CHANNEL_SCANS;  /* average readings */
  for(k=2;k<8;k++)
    channel[k] = 0.0;  /* set remaining channels to zero */

  return flag;
} /* end of routine readad */

/*-------------------------------------------------------------------------
ROUTINE
  READAD_I
  Klaus Obergfell

DISCRIPTION
  reads das16 channels 0 - 1 repeated times and averages, remaining
  channels 2 - 7 are set to zero. Called from interrupt routine.

PARAMETER
  double channel[] - pointer to returned data

RETURN
  flag = 0 - successful
  flag = 1 - error
-------------------------------------------------------------------------------*/
int readad_i (double channel[])
{
  static int error,mode,data[5],flag,k,l;
  static double a2dvalue_sum[2];

  mode = 3;  /* single channel read operation */
  flag = 0;
```

```
  for(k=0;k<2;k++)
    a2dvalue_sum[k] = 0.0;  /* initialize sum of a2d values */

  /* do repeated scans */
  for (k=0;k<CHANNEL_SCANS;k++)
  {
    /* sample two channels */
    for (i=0; i < 2; i++)
    {
      if ((error = DAS16(mode,data)) != 0)
      {
        flag = 1;
        return flag;
      }
      a2dvalue_sum[i] += ((double) data[0]);
    }
  }

  for(k=0;k<2;k++)
    channel[k] = a2dvalue_sum[k]/CHANNEL_SCANS;  /* average readings */
  for(k=2;k<8;k++)
    channel[k] = 0.0;  /* set remaining channels to zero */

  return flag;
} /* end of routine readad_i */

/*---------------------------------------------------------------------------
ROUTINE
  WRITEDA_I
  Klaus Obergfell 04/29/91

DESCRIPTION
  writes to das16 digital to analog ports

  some conversion is done with d/a values:
  das16 software exspects value between 0 and 4095, corrosponding to
  0V .. +5V. This routine exspectes value between -2047 and 2047,
  corrosponding to -10V .. +10V.
  This routine converts output to 0 .. 4095, hardware interface boosts
  0 .. +5V to -10V .. +10V.

PARAMETER
  int out1,out2 - digital output values (-2047 .. +2047)
                  corrosponds to output voltage -10V .. +10 V

RETURN
  flag = 0 - successful
  flag = 1 - error
---------------------------------------------------------------------------*/
int writeda_i(int out1, int out2)
{
  static int error, mode, data[5], flag;

  /* output both d/a channels */
  flag = 0;
  mode = 16;
  data[0] = 2047 - out1;
  data[1] = 2047 - out2;

  if ((error = DAS16(mode,data)) != 0) {
    flag = 1;
    return flag;
  }
  return flag;
} /* end of routine writeda_i */

/*---------------------------------------------------------------------------
ROUTINE
  D2A_LIMITS_I
  Klaus Obergfell

DESCRIPTION
  checks if d/a values are within limits and sets to limit if not

PARAMETER
  int *out1,*out2 - pointers to d2a values
---------------------------------------------------------------------------*/
void d2a_limits_i (int *out1, int *out2)
{
  static int temp1,temp2;  /* local variables */
  temp1 = *out1;
  temp2 = *out2;
  if(temp1 > 2047)
    *out1 = 2047;
  else if(temp1 < -2047)
    *out1 = -2047;
  if(temp2 > 2047)
    *out2 = 2047;
  else if(temp2 < -2047)
    *out2 = -2047;
  return;
} /* end of routine d2a_limits_i */

/*****************************************************************************
end of file adstuff.c
*****************************************************************************/
```

```
/*****************************************************************************

FILE
    alarm.h  -  all the declarations needed for alarm.c

LAST UPDATE
    25 August 1985

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

#define CLOCKFREQ    1193180L          /* clock frequency */

#define DOS_COUNT    0xFFFF            /* DOS timer count */
#define MAX_MS       55L               /* max interval in milliseconds */

#define TIMER_CTL    0x43              /* 8253 timer control port */
#define TIMER        0x40              /* timer 0 port */

#ifdef ANSI

extern int      setalarm(double);
extern int      rsetalarm(long);
extern long     alarm_count(void);
extern double   alarm_time(void);

#else

extern int      setalarm();
extern int      rsetalarm();
extern long     alarm_count();
extern double   alarm_time();

#endif
```

```
/*****************************************************************************

FILE
    alarm.c -  set hardware timer (8253 timer 0) - IBM-PC version

ROUTINES
    setalarm     -  set hardware timer alarm in milliseconds (double).
    rsetalarm    -  a "raw" form of setalarm(), using ticks
    alarm_count  -  returns the current count (long).
    alarm_time   -  returns the current time in millisec (double).

REMARKS
    These procedures are obviously hardware dependent. The 8253 timer 0
    is set to mode 3, 16-bit binary count.  The maximum period on the
    IBM PC is limited to about 54.9 milliseconds, which is the default
    tick frequency used by DOS.

    To reduce overheads for very time critical applications, rsetalarm()
    allows the period to be specified in clock ticks to eliminate the
    millisecond to tick conversion overhead.

    If the interval argument is zero or negative, the timer is reset to
    the DOS default state.

LAST UPDATE
    3 May 1985
        add rsetalarm().
    12 January 1988
        change all floats to doubles

    Copyright (c) 1985-1988  D.M. Auslander and C.H. Tham

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

/*****************************************************************************
                            I M P O R T S
*****************************************************************************/

#include <stdio.h>

#include "envir.h"      /* environment declarations */
#include "inout.h"      /* in() and out() function mapping macros */
#include "alarm.h"      /* address and other declarations */


/*****************************************************************************
                    P R I V A T E    D A T A
*****************************************************************************/

static long count = 0xFFFFL;    /* current count, init to DOS default */


/*****************************************************************************
                    E N T R Y    R O U T I N E S
*****************************************************************************/

/*---------------------------------------------------------------------------
PROCEDURE
```

```
    SETALARM  -  set timer interrupt period in milliseconds

SYNOPSIS
    int setalarm(ms)
    double ms;

PARAMETERS
    ms  -  interval in milliseconds

RETURNS
    0  normal return
    1  error return

REMARKS
    Since the max. interval for the IBM PC corresponds to a maximum
    count of 65535, the maximum value of ms is limited to 54.9 ms.

    If ms is zero or negative, timer will be reset to DOS default.

LAST UPDATE
    23 August 1985 by D. M. Auslander
        change to float input format
    12 January 1988  by  Haam
        change floats to doubles
------------------------------------------------------------------------*/

int setalarm(ms)
double ms;
{

    if ((ms > 0.0) && (ms <= MAX_MS))   /* argument is valid */
    {
        count = (CLOCKFREQ * ms) / 1000.0 + 0.5;

        out(TIMER_CTL, 0x36);       /* mode 3, 16-bit binary count */

        out(TIMER, count & 0xFF);   /* send least sig. byte first */
        out(TIMER, (count >> 8) & 0xFF);
    }
    else if (ms <= 0.0)             /* reset to DOS clock rate */
    {
        out(TIMER_CTL, 0x36);

        out(TIMER, DOS_COUNT & 0xFF);
        out(TIMER, (DOS_COUNT >> 8) & 0xFF);

        count = DOS_COUNT;
    }
    else    /* bad input - do nothing */
    {
        return(1);      /* error return */
    }

    return(0);          /* normal return */
}
```

```
/*------------------------------------------------------------------------
PROCEDURE
    RSETALARM  -  raw form of setalarm using clock ticks instead of ms.

SYNOPSIS
    rsetalarm(ticks)
    long ticks;

PARAMETERS
    ticks  -  interval in clock ticks (1,193,180 Hz raw clock)

RETURNS
    0  normal return
    1  error return

REMARKS
    This raw form is intended for time critical programs that
    cannot afford the overhead of converting milliseconds to ticks.
    For even more time critical applications, the range check
    can be omitted.

    The clock frequency is defined in alarm.h as CLOCKFREQ.

LAST UPDATE
    3 May 1985
------------------------------------------------------------------------*/

int rsetalarm(ticks)
long ticks;
{

    if ((ticks > 0L) && (ticks < 65536L))
    {
        out(TIMER_CTL, 0x36);

        out(TIMER, ticks & 0xFF);   /* send least sig. byte first */
        out(TIMER, (ticks >> 8) & 0xFF);

        count = ticks;
    }
    else if (ticks <= 0L)           /* reset to DOS default */
    {
        out(TIMER_CTL, 0x36);

        out(TIMER, DOS_COUNT & 0xFF);
        out(TIMER, (DOS_COUNT >> 8) & 0xFF);

        count = DOS_COUNT;
    }
    else
        return(1);  /* error return */

    return(0);      /* normal return */
}


/*------------------------------------------------------------------------
```

```
PROCEDURE
    ALARM_COUNT  -  get the current count value

SYNOPSIS
    long alarm_count()

RETURNS
    the current count used by the timer

LAST UPDATE
    21-Aug-85 by D. M. Auslander
-------------------------------------------------------------------------*/

long alarm_count()
{

    return(count);
}


/*-----------------------------------------------------------------
PROCEDURE
    ALARM_TIME  -  get the current timer interval in millisec

SYNOPSIS
    double alarm_time()

RETURNS
    count * 1000 / CLOCKFREQ

LAST UPDATE
    23-Aug-85 by D. M. Auslander
-------------------------------------------------------------------------*/

double alarm_time()
{

    return((double)count * 1000.0 / CLOCKFREQ);
}
```

```
/*****************************************************************************
FILE
  convers.h - function prototypes to file convers.c
  Klaus Obergfell 1991
*****************************************************************************/

extern void angles(double [], double *, double *);
extern void angles_i(double [], double *, double *);
extern void strains_i(double [], double *, double *);
extern void forward(double, double, double *, double *);
extern void inverse(double, double, double *, double *);
extern int  ws_limits(double *, double *);

/*****************************************************************************
end of file convers.h
*****************************************************************************/
```

```
/*****************************************************************************
FILE
  convers.c - conversion routines for RALF, used by controller and
              trajectory planner
  Klaus Obergfell 1991

ROUTINES
  angles     - converts a2d value to joint angle
  angles_i   - converts a2d value to joint angle, called from controller
  strains_i  - converts a2d value to strain, called from controller
  forward    - forward kinematics routine
  inverse    - inverse kinematics routine
  ws_limits  - checks work space boundary of robot
  arccos_i   - arccos routine, called from angles_i

UPDATES
  07/16/91, change in channel configuration of das16 board
  06/20/91, isr called functions have only static variables
  06/08/91, introduce two sets of identical functions, one set is used by
    interrupt driven functions, the other set is used by foreground functions.
    The functions that are called from interrupts have the ending "_i" in
    their function name.
  06/07/91, take out power function

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

/*****************************************************************************
IMPORTS
*****************************************************************************/
#include "envir.h"      /* compiler options */

#include <stdlib.h>     /* standard libraries */
#include <math.h>

/*****************************************************************************
DEFINES
*****************************************************************************/
/*
factors for linear curve fit (y = m*x + b) for a2d value to length
conversion. m from calibration, b from calibration + minimum actuator
length (when transducer has length zero)
*/
#define m1  0.00490918
#define b1  39.57590820
#define m2  0.004923568
#define b2  38.46912407

/* factors for length to angle conversion */
#define l5    35.20853057   /* l5 = sqrt(pow(20.5,2)+pow(28.625,2)) */
#define l6    18.55397532   /* l6 = sqrt(pow(18.0,2)+pow(4.5,2)) */
#define l3    18.356
#define l7    33.00852163   /* l7 = sqrt(pow(33.0,2)+pow(0.75,2)) */
#define ang5  0.949309      /* ang5 = atan(28.625/20.5) */
#define ang6  0.244979      /* ang6 = atan(4.5/18.0) */
#define ang7  0.0227234     /* ang7 = atan(0.75/33.0) */
#define ang8  0.197396      /* ang8 = atan(3.6/18.0) */
#define pi    3.141592654
#define pi_2  1.570796327
```

```
/*
coefficients for linear curve fit (y = m*x + b) for a2d value to
strain conversion
*/
#define s1b  2.6694703767e-5
#define s1m  -1.00687194e-7
#define s2b  -0.1742427206e-5
#define s2m  -1.08871578e-7

/* constants for forward and inverse */
#define L1   120.00     /* length link 1 in inch */
#define L2   120.00     /* length link 2 in inch */
#define L3   5.75       /* offset between both links in inch */
#define L12  14400.0    /* L1 * L1 */
#define L22  14400.0    /* L2 * L2 */
#define L32  33.0625    /* L3 * L3 */

/*
workspace boundaries:
convert angle arguments from degrees to radians, shrink rank by
0.5 degree to insure within range
*/
#define arg1   0.628318531  /* 36.0*pi/180.0 */
#define arg2   1.902408885  /* 109.0*pi/180.0 */
#define arg3   0.977384381  /* 56.0*pi/180.0 */
#define arg4   1.867502300  /* 107.0*pi/180.0 */

/*****************************************************************************
FUNCTION PROTOTYPES
*****************************************************************************/
static double arccos_i(double);

/*****************************************************************************
VISIBLE ROUTINES
*****************************************************************************/

/*-----------------------------------------------------------------------
ROUTINE
  ANGLES
  Klaus Obergfell, 04/30/91
  based on act_lengths.c and convert_lengths.c by Dave Magee

DESCRIPTION
  converts a2d value first to actuator length and then to joint angle.
  Conversion from a2d value to actuator length assumes linear transducer
  reading. Conversion also assums no bending in structure.

PARAMETER
  double channel[]  - array of d2a values
  double *th1       - pointer to angle theta1 (in radians)
  double *th2       - pointer to angle theta2 (in radians)
-----------------------------------------------------------------------*/
void angles(double channel[], double *th1, double *th2)
{
  double l1,l2;
  double ang3,ang4;

  /* calculate actuator length */
  l1 = m1 * channel[0] + b1;
  l2 = m2 * channel[1] + b2;

  /*
  calculates theta 1:
  ang3 = acos((pow(15,2)+pow(16,2)-pow(11,2))/(2.0*15*16))
  */
  ang3 = acos((1583.890625-11*11)/1306.516414);
  *th1 = ang3 + ang5 + ang6 - pi/2.0;

  /*
  calculates theta 2:
  ang4 = acos((pow(13,2)+pow(17,2)-pow(12,2))/(2.0*13*17))
  */
  ang4 = acos((1426.505236-12*12)/1211.808846);
  *th2 = pi - ang4 + ang7 - ang8;

  return;
} /* end of routine angles */

/*-----------------------------------------------------------------------
ROUTINE
  ANGLES_I
  Klaus Obergfell, 04/30/91
  based on act_lengths.c and convert_lengths.c by Dave Magee


DESCRIPTION
  identical to routine angles, but called from interrupt driven controller.
  The routine arccos_i is used instead of the acos routine, since math
  functions are not re-entrant and acos is used by foreground routines.
-----------------------------------------------------------------------*/
void angles_i(double channel[], double *th1, double *th2)
{
  static double l1,l2;
  static double ang3,ang4;

  /* calculate actuator length */
  l1 = m1 * channel[0] + b1;
  l2 = m2 * channel[1] + b2;

  /*
  calculates theta 1:
  ang3 = acos((pow(15,2)+pow(16,2)-pow(11,2))/(2.0*15*16))
  */
  ang3 = arccos_i((1583.890625-11*11)/1306.516414);
  *th1 = ang3 + ang5 + ang6 - pi/2.0;

  /*
  calculates theta 2:
  ang4 = acos((pow(13,2)+pow(17,2)-pow(12,2))/(2.0*13*17))
  */
  ang4 = arccos_i((1426.505236-12*12)/1211.808846);
  *th2 = pi - ang4 + ang7 - ang8;

  return;
} /* end of routine angles_i */
```

```
/*-------------------------------------------------------------------------
ROUTINE
  STRAINS_I
  Klaus Obergfell, 05/01/91
  based on act_strains by Dave Magee

DESCRIPTION
  converts a/d value to strain in upper and lower link, uses linear
  curve fit of form y = m*x + b. Called from controller.

PARAMETER
  double channel[]  - array of a2d values
  double *st1       - pointer to strain in link1
  double *st2       - pointer to strain in link2

-------------------------------------------------------------------------*/
void strains_i(double channel[], double *st1, double *st2)
{
  static double s1,s2;

  s1 = s1m * channel[2] + s1b;
  /* s1 = 2.0*(s1/8.8282e-5)/2.7815 */
  *st1 = 0.719036491 * (s1 / 8.8282e-5);

  s2 = s2m * channel[3] + s2b;
  /* s2 = 10.0*(s2/1.377512e-4)/2.25 */
  *st2 = 4.444444444 * (s2 / 1.377512e-4);

  return;
}/* end of routine strains */

/*-------------------------------------------------------------------------
ROUTINE
  FORWARD
  Klaus Obergfell
  based on routine by David Magee

DESCRIPTION
  calculates the forward kinematics, i.e. x,y position based on the
  angles of joint one and two.

PARAMETER
  double th1  - joint angle theta1 (in radians)
  double th2  - joint angle theta2 (in radians)
  double *x   - pointer to x position (in inch)
  double *y   - pointer to y position (in inch)
-------------------------------------------------------------------------*/
void forward(double th1, double th2, double *x, double *y)
{
  *x = L2*cos(th1+th2)+L1*cos(th1)-L3*sin(th1);
  *y = L2*sin(th1+th2)+L1*sin(th1)+L3*cos(th1);
  return;
} /* end of routine forward */

/*-------------------------------------------------------------------------
ROUTINE
  INVERSE
  Klaus Obergfell, 05/01/91
  based on routine by Dave Magee

DESCRIPTION
  calculates the inverse kinematics i.e. joint angles based on the tip
  position x,y.

PARAMETER
  double x     - x position (in inch)
  double y     - y position (in inch)
  double *th1  - pointer to joint angle theta1 (in radians)
  double *th2  - pointer to joint angle theta2 (in radians)
-------------------------------------------------------------------------*/
void inverse(double x, double y, double *th1, double *th2)
{
  double k;
  double k2;
  double theta1,theta2;
  double s1,c1;
  double s2,c2;

  k = x * x + y * y - L12 - L22 - L32;
  k2 = k * k;

  /* calculates theta2 in radians */
  s2 = k * L3 + sqrt(4.0 * L12 *L22 * (L12+L32) -k2 * L12);
  c2 = k * L1 - L3 * sqrt(4.0 * L22 * (L12+L32) - k2);
  theta2 = atan2(s2,c2);

  /* calculates theta1 in radians */
  s1 = -x*(L2*sin(theta2)+L3)+y*(L2*cos(theta2)+L1);
  c1 =  x*(L2*cos(theta2)+L1)+y*(L2*sin(theta2)+L3);
  theta1 = atan2(s1,c1);

  *th1 = theta1;
  *th2 = theta2;
  return;
}/* end of routinee inverse */

/*-------------------------------------------------------------------------
ROUTINE
  WS_LIMITS
  Klaus Obergfell, 05/01/91
  based on Dave Magees limits.c

DESCRIPTION
  tests if desired joint angle pair is within workspace. If desired
  angle is out of the workspace, closest angle is assigned.

RETURN
  flag = 0 - joint angle pair was within workspace
  flag = 1 - joint angle pair was outside workspace

PARAMETER
  double *th1  - pointer to joint angle theta1 (in radians)
  double *th2  - pointer to joint angle theta2 (in radians)
-------------------------------------------------------------------------*/
int ws_limits(double *th1, double *th2)
```

```
{
  double ang1,ang2;
  int flag=0;

  ang1 = *th1;
  ang2 = *th2;

  if(ang1 < arg1) {
    ang1 = arg1;
    flag = 1;
  }
  if(ang1 > arg2) {
    ang1 = arg2;
    flag = 1;
  }
  if(ang2 < arg3) {
    ang2 = arg3;
    flag = 1;
  }
  if(ang2 > arg4) {
    ang2 = arg4;
    flag = 1;
  }

  *th1 = ang1;
  *th2 = ang2;
  return flag;
} /* end of routine ws_limits */

/*****************************************************************************
PRIVATE ROUTINES
*****************************************************************************/

/*---------------------------------------------------------------------------
ROUTINE
  ARCCOS_I
  Klaus Obergfell

DESCRIPTION
  acos routine, called from controller. The precision of this function
  is sufficient for fabs(arg) < 0.70, which is bigger than the exspected
  values.

RETURN
  double result = acos(argument)  (in radians)

PARAMETER
  double argument  - function argument
---------------------------------------------------------------------------*/
static double arccos_i(double argument)
{
  static double  n_old,n_new,d_old,d_new,fraction,result,argument2;
  static int     i;

  if (argument > 0.96)          /* special cases */
    result = 0.0;
  else if (argument < -0.96)
    result = pi_2;
  else                  /* standard cases */
  {
    argument2 = argument * argument;
    i = 0;
    n_new = argument;
    d_new = 1.;
    result = pi_2 - argument;
    do
    {
      n_old = n_new;
      d_old = d_new;
      i++;
      n_new = n_old * i * argument2;
      i++;
      d_new = d_old * i;
      fraction = n_new / (d_new * (i + 1));
      result = result - fraction;
    } while (i < 40);     /* 20 loops */
  }
  return result;
} /* end of routine arccos */

/*****************************************************************************
end of file convers.c
*****************************************************************************/
```

```
/*****************************************************************************
FILE
  debug.h  - function prototypes for file debug.c
  Klaus Obergfell 1991
*****************************************************************************/

extern void do_debug(void);

/*****************************************************************************
end of file debug.h
```

```
/*****************************************************************************
FILE
  debug.c  - debugging options for ralf control software
  Klaus Obergfell 1991

ROUTINES
  do_debug     - debugging menu
  debug_menu   - prints debugging menu
  test         - test trajectory calculation
*****************************************************************************/

/*****************************************************************************
IMPORTS
*****************************************************************************/
#include "envir.h"       /* compiler options */

#include "main.h"        /* system parameters and data types */

#include "util.h"        /* function prototypes */
#include "control2.h"

#include <stdlib.h>      /* standard libraries */
#include <stdio.h>
#include <conio.h>
#include <bios.h>
#include <ctype.h>
#include <dos.h>
#include <math.h>

/*****************************************************************************
DEFINES
*****************************************************************************/
/* constants for forward and inverse, used in test */
#define L1          120.00          /* length link 1 in inch */
#define L2          120.00          /* length link 2 in inch */
#define L3          5.75            /* offset between both links in inch */
#define L12         14400.0         /* L1 * L1 */
#define L22         14400.0         /* L2 * L2 */
#define L32         33.0625         /* L3 * L3 */

#define DIAG_SLOPE  0.13     /* spacing of traj points */
#define MARGIN      0.005    /* used to detect error */

/*****************************************************************************
FILES
*****************************************************************************/
extern FILE *debug_out1;
extern FILE *debug_out2;


/*****************************************************************************
FUNCTION PROTOTYPES
*****************************************************************************/
static void debug_menu(void);
static int test(void);

/*****************************************************************************
VISIBLE ROUTINES
```

```
*****************************************************************************/

/*---------------------------------------------------------------------------
ROUTINE
  DO_DEBUG
  Klaus Obergfell 1991

DESCRIPTION
  loops thru debbugging menu and performs chosen tasks until terminated
---------------------------------------------------------------------------*/
void do_debug(void)
{
  short   key, DONE = false;
  int     itemp1;
#if OPTION_INT
#if OPTION_DEBUG
  int     itemp2,itemp3,itemp4;
#endif
#endif

  do
  {
  clrscr();
  debug_menu();
  key = bioskey(0) & 0xFF;
  switch (toupper(key))
    {
      case 'T':  /* Trajectory computation test */
        itemp1 = test();
        printf("\nreturn = %d (0 = Okay; 1 = ???)",itemp1);
        wait_key_press();
        break;

#if OPTION_INT
#if OPTION_DEBUG

      case 'E':  /* Error code */
        disable();
        itemp1 = error_a2d_i;
        itemp2 = error_a2d_count_max_i;
        enable();
        printf("\n\nerror from read_ad = %d",itemp1);
        printf("\ncount from read_ad = %d",itemp2);
        wait_key_press();
        break;
      case 'F':  /* read hold flag */
        disable();
        itemp1 = traj1.flag;
        itemp2 = c_points;
        itemp3 = c_iterations;
        itemp4 = c_holds;
        enable();
        printf("\n\nhold flag = %d",itemp1);
        printf("\ncounter traj points = %d",itemp2);
        printf("\ncounter traj iterations = %d",itemp3);
        printf("\ncounter traj holds = %d",itemp4);
        wait_key_press();
        break;

#endif
#endif

      case 'W':  /* Write data to disk, writes only if the status flag
                    was enabled, waites until full flag is set. Clears
                    full flag and sets counter to zero when done. */
/*
        if (data1.s_flag)
        {
          if (!data1.f_flag)
            printf("\nWait until data buffer is full");
          while (!data1.f_flag);
          w_data_to_disk(&data1);
        }
*/
        break;
      case 'Q':  /* Quit */
        DONE = True;
        break;
    }
  } while(!DONE);
  return;
} /* end of routine do_debug */

/*****************************************************************************
PRIVATE ROUTINES
*****************************************************************************/

/*---------------------------------------------------------------------------
ROUTINE
  DEBUG_MENU
  Klaus Obergfell 1991

DESCRIPTION
  prints debug menu
---------------------------------------------------------------------------*/
static void debug_menu(void)
{
  printf("\nT.    Trajectory computation test");
  printf("\nE.    Error code");
  printf("\nF.    read hold flag");
  printf("\nW.    Write data to disk (*)");
  printf("\nQ.    Quit (exit this menu)");
  printf("\n      (*) = not available in current installation");

  return;
} /* end of routine debug_menu */

/*---------------------------------------------------------------------------
ROUTINE
  TEST
  Klaus Obergfell

DESCRIPTION
  tests if trajectory computations are performed correctly when
  interrupt controller is operating in the background.
```

```
RETURN
  0  - no error detected
  1  - some error in trajectory calculation
-------------------------------------------------------------------------------*/
static int test(void)
{
  int    error=0;           /* return value */
  int    i,steps;
  double x,y,xi,yi,xd,yd,mx,my,diag,delta_x,delta_y;

  double k;                 /* for inverse */
  double k2;
  double theta1,theta2,th1_o,th2_o,marg1,marg2;
  double s1,c1;
  double s2,c2;
  double norm;

  double temp1,temp2;   /* for debugging */

  /* input section */
  printf("\nEnter initial and desired position (xi,yi,xd,yd)");
  scanf("%lf,%lf,%lf,%lf",&xi,&yi,&xd,&yd);
  delta_x = xd - xi;
  delta_y = yd - yi;
  diag = sqrt(delta_x * delta_x + delta_y * delta_y);
  steps = diag / DIAG_SLOPE;
  mx = delta_x / (steps - 1);
  my = delta_y / (steps - 1);
  x = xi;
  y = yi;

  /* pseudo traj calculation */
  printf("\nstart calculation");
  debug_out1 = fopen("test.d","a");
  debug_out2 = fopen("debug.d","a");
  for(i=0;i<steps;i++)
  {
    k = x * x + y * y - L12 - L22 - L32;
    k2 = k * k;

    /* calculates theta2 in radians */
    temp1 = 4.0 * L12 * L22 * (L12+L32) - k2 * L12;
    s2 = (k * L3 + sqrt(temp1)) / 100000000.;
    temp2 = 4.0 * L22 * (L12+L32) - k2;
    c2 = (k * L1 - L3 * sqrt(temp2)) / 100000000.;
    norm = sqrt(s2 * s2 + c2 * c2);
    s2 = s2 / norm;
    c2 = c2 / norm;
    theta2 = atan2(s2,c2);

    /* calculates theta1 in radians */
    s1 = -x*(L2*sin(theta2)+L3)+y*(L2*cos(theta2)+L1);
    c1 =  x*(L2*cos(theta2)+L1)+y*(L2*sin(theta2)+L3);
    norm = sqrt(s1 * s1 + c1 * c1);
    s1 = s1 / norm;
    c1 = c1 / norm;
    theta1 = atan2(s1,c1);

    fprintf(debug_out1,"\n%d  %.5f  %.5f    %.5f  %.5f",i,theta1,theta2,x,y);
    if (i > 0)
    {
      marg1 = fabs(theta1 - th1_o);
      marg2 = fabs(theta2 - th2_o);
      if (marg1 > MARGIN || marg2 > MARGIN)
      {
        error = 1;
        fprintf(debug_out2,"\n%d  %.3f  %.3f  %.5f    %.3f  %.3f  %.5f    %.2f
          %.2f",i,s2,c2,theta2,s1,c1,theta1,temp1,temp2);
      }
    }
    th1_o = theta1;
    th2_o = theta2;
    x = x + mx;
    y = y + my;
  }/* end of for loop */
  fprintf(debug_out1,"\n\nnext run:");
  fprintf(debug_out2,"\n\nnext run:");
  fclose(debug_out1);
  fclose(debug_out2);
  printf("\ndone with calculation");

  return error;
} /* end of routine test */

/*******************************************************************************
end of file debug.c
*******************************************************************************/
```

```
/*****************************************************************************
FILE
  m_math.h  - function prototypes for file m_math.h
  Klaus Obergfell
*****************************************************************************/

extern void mmmult(double *, double *, double *, int, int, int);
extern void minvert(double *, double *, int);
extern void mtrans(double *, double *, int, int);

/*****************************************************************************
end of file m_math.h
```

```
/*****************************************************************************
FILE
  m_math.c  - matrix operations
  Klaus Obergfell

Routines
  mmmult   - matrix multiplication
  minvert  - matrix inversion
  mtrans   - matrix transpose

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

/*****************************************************************************
IMPORTS
*****************************************************************************/
#include "envir.h"      /* compiler options */

#include <stdlib.h>     /* standard libraries */
#include <stdio.h>


/*****************************************************************************
VISIBLE FUNCTIONS
*****************************************************************************/

/*----------------------------------------------------------------------------
ROUTINE
  MMMULT
  Klaus Obergfell

DESCRIPTION
  multiplies mxn-matrix A by nxp-matrix B, the result is the mxp-matrix C.

PARAMETER
  double *A  - pointer to A matrix
  double *B  - pointer to B matrix
  double *C  - pointer to C matrix
  int m,n,p  - matrix dimensions
----------------------------------------------------------------------------*/
void mmmult(double *A, double *B, double *C, int m, int n, int p)
{
  int r,c,i;  /* counter for row,col,row and col */

  /* initialisation of C */
  for (i=0;i<(m*p);i++)
    C[i]=0;

  /* matrix-multiplication */
  for (c=0;c<p;c++)
  {
    for (r=0;r<m;r++)
    {
      for (i=0;i<n;i++)
        C[r*p+c]=C[r*p+c]+A[r*n+i]*B[i*p+c];
    }
  }

  return ;
```

```
} /* end of routine mmmult */

/*---------------------------------------------------------------------------
ROUTINE
  MINVERT
  Klaus Obergfell

DESCRIPTION
  inverts a nxn-matrix, using gaussian elimination. This subroutine
  does not include the case that there are zeros on the main-diagonal.

PARAMETERS
  double *A  - pointer to matrix A that has to be inverted
  double *B  - pointer to inverted matrix B
  int n      - size of (square) matricies A,B
---------------------------------------------------------------------------*/
void minvert(double *A, double *B, int n)
{
  double *AI;        /* pointer to augmented matrix */
  int r,c,d;         /* counter for row,col,diag */
  double temp;       /* temporary variable */

  /* memory allocation */
  AI = (double*)malloc(2*n*n*sizeof(double));

  /* initialization of augmented matrix */
  for (r=0;r<n;r++)
  {
    for (c=0;c<n;c++)
      AI[r*(2*n)+c]=A[r*n+c];
    for (c=n;c<(2*n);c++)
      AI[r*(2*n)+c]=0;
    AI[r*(2*n)+n+r]=1;
  }

  /* gaussian elimination */
  for (d=0;d<n;d++)
  {
    /* normalize d-th row */
    temp=AI[d*(2*n)+d];
    for (c=0;c<(2*n);c++)
      AI[d*(2*n)+c]=AI[d*(2*n)+c]/temp;

    /* produce zeros in d-th col, below d-th row */
    for (r=d+1;r<n;r++)
    {
      temp=AI[r*(2*n)+d];
      for (c=0;c<(2*n);c++)
        AI[r*(2*n)+c]=AI[r*(2*n)+c]-AI[d*(2*n)+c]*temp;
    }

    /* produce zeros in d-th col, above d-th row */
    for (r=0;r<d;r++)
    {
      temp=AI[r*(2*n)+d];
      for (c=0;c<(2*n);c++)
        AI[r*(2*n)+c]=AI[r*(2*n)+c]-AI[d*(2*n)+c]*temp;
    }
  }

  /* B = inverse (A) */
  for (r=0;r<n;r++)
  {
    for (c=0;c<n;c++)
    B[r*n+c]=AI[r*(2*n)+n+c];
  }

  /* free memory allocation for AI */
  free(AI);
  AI=NULL;

  return;
} /* end of routine minvert.c */

/*---------------------------------------------------------------------------
ROUTINE
  MTRANS
  Klaus Obergfell

DESCRIPTION
  transposes the mxn matrix A, the result is the nxm matrix B.

PARAMETER
  double *A  - pointer to matrix A
  double *B  - pointer to matrix B
  int m,n    - matrix dimensions
---------------------------------------------------------------------------*/
void mtrans(double *A, double *B, int m, int n)
{
  int r,c;   /* counter for rows, columns */

  /* initialize B */
  for(r=0;r<m*n;r++)
    B[r] = 0;

  /* matrix transpose */
  for(r=0;r<m;r++)
  {
    for(c=0;c<n;c++)
      B[c*m+r] = A[r*n + c];
  }

  return;
} /* end of routine mtrans */

/*****************************************************************************
end of file m_math.c
```

```
/*****************************************************************************
FILE
  util.h - function prototypes to file util.c
*****************************************************************************/

#include <comm.h>  /* since COMM_PORT is declared in this file */

extern void term(COMM_PORT *);
extern void w_data_to_disk(struct data *);
extern void message(void);
extern void xdelay(int);
extern void wait_key_press(void);

/*****************************************************************************
end of file util.h
*****************************************************************************/
```

```
/*****************************************************************************
FILE
  util.c - some utility functions for RALF controller system
  Klaus Obergfell 1991

ROUTINES
  term              - terminal emulation routine
  w_data_to_disk    - writes data to disk
  message           - prints program start message
  xdelay            - implements a delay loop
  wait_key_press    - waits until MENU - key is pressed
  get_data          - reads data from buffer
  check_data        - checks status of data buffer

UPDATES
  07/20/91,  transducer access only inside tmr_isr
             (no changes in this module)
  07/14/91,  change in struct data to implement ring buffer
  07/13/91,  big clean up
*****************************************************************************/

/*****************************************************************************
IMPORTS
*****************************************************************************/
#include "envir.h"     /* compiler options */

#include "main.h"      /* parameters, data types */

#include <stdlib.h>    /* standard libraries */
#include <stdio.h>
#include <dos.h>
#include <conio.h>

#include "control2.h" /* function prototypes and external data */

#include <comm.h>      /* serial comm. */

/*****************************************************************************
FILES
*****************************************************************************/
extern FILE *util_out;

/*****************************************************************************
FUNCTION PROTOTYPES
*****************************************************************************/
static int get_data(struct data *, FILE *);
static int check_data(struct data *);

/*****************************************************************************
VISIBLE ROUTINES
*****************************************************************************/

/*---------------------------------------------------------------------------
ROUTINE
  TERM
  Klaus Obergfell, based on example from communication software

DESCRIPTION
```

```
  terminal emulation routine. Simply polls the COM port and the
  keyboard alternately for characters.

PARAMETER
  COMM_PORT *p_port  - pointer to COMM_PORT structure, for serial comm.
-------------------------------------------------------------------------*/
void term(COMM_PORT *p_port)
{
  short c;              /* must be int to detect a -1 return */
  clrscr();
  puts("Terminal Program, press ALT-M to return to menu");
  for (;;)
  {
    /* CHECK SERIAL PORT FOR BYTE */
    if ( (c = c_inchar(p_port)) != EOF)
    {
      putchar(c);
      fflush(stdout);
    }

    /* CHECK KEYBOARD FOR A KEY PRESS */
    if ( (c = inkey()) != EOF)
    {
      if (c == MENU)
        return;
      else
        c_putc(p_port, c);
    }
  }
} /* end of routine term */

/*-------------------------------------------------------------------------
ROUTINE
  W_DATA_TO_DISK
  Klaus Obergfell

DESCRIPTION
  writes data from memory to hard disk. Clears full flag after writing data
  of 10 datapoints, so that buffer can fill up again, while still writing
  data to disk
-------------------------------------------------------------------------*/
void w_data_to_disk (struct data *databuf)
{
  int i,count;

  count = check_data(databuf);
  if(count > 10)
  {
    util_out = fopen("data.d","a");
    fprintf(util_out,"\n\nnext data:\n");
    for(i=0;i<10;i++)
      get_data(databuf,util_out);
    disable();    /* critical region */
    databuf->f_flag = 0;
    enable();
    for(i=10;i<count;i++)
      get_data(databuf,util_out);
    fclose(util_out);
    disable();    /* critical region */
    databuf->f_flag = 0;
    enable();
  }
  else if(count > 0)
  {
    util_out = fopen("data.d","a");
    fprintf(util_out,"\n\nnext data:\n");
    for(i=0;i<count;i++)
      get_data(databuf,util_out);
    fclose(util_out);
    disable();    /* critical region */
    databuf->f_flag = 0;
    enable();
  }

  return;
} /* end of routine w_data_to_file */

/*-------------------------------------------------------------------------
ROUTINE
  MESSAGE
  Klaus Obergfell

DESCRIPTION
  prints program start message
-------------------------------------------------------------------------*/
void message(void)
{
  printf("\nRALF supervisor program\n");
  printf("\nVersion 1.0");
  return;
} /* end of routine message */

/*-------------------------------------------------------------------------
ROUTINE
  XDELAY
  Klaus Obergfell

DESCRIPTION
  a very crude delay, implements a delay loop

PARAMETER
  int t  - delay in ms (timing without interrupt driven controller, delay
           increases 2-3 times with controller operating, depending on
           controller sampling time)
-------------------------------------------------------------------------*/
void xdelay (int t)
{
  static int i,k;

  for (k=0;k<t;k++)
  {
    for (i=0;i<668;i++);
  }
  return;
} /* end of routine xdelay */
```

```
/*-------------------------------------------------------------------------
ROUTINE
  WAIT_KEY_PRESS

DESCRIPTION
  waits until MENU - key is pressed
-------------------------------------------------------------------------*/
void wait_key_press (void)
{
  short c;

  printf("\npress ALT-M to return to menu");
  while ( (c = inkey()) != MENU);
  return;
} /* end of routine wait_key_press */

/*************************************************************************
PRIVATE ROUTINES
*************************************************************************/

/*-------------------------------------------------------------------------
ROUTINE
  GET_DATA
  Klaus Obergfell

DESCRIPTION
  reads data buffer and writes data to disk.

PARAMETER
  struct data *databuf   - pointer to structure data
  FILE *out_stream       - pointer to structure FILE

RETURN
  0 - successful
  1 - buffer was empty
-------------------------------------------------------------------------*/
static int get_data(struct data *databuf, FILE *out_stream)
{
  int next,temp6,temp7;
  double temp1,temp2,temp3,temp4,temp5;

  if(databuf->buffer_head == databuf->buffer_tail)  /* buffer empty */
    return 1;

  next = databuf->buffer_tail + 1;
  if(next >= DATAPOINTS)  /* wrap around */
    next = 0;

  /* write data to disk */
  temp1 = databuf->ddata[next][0];
  temp2 = databuf->ddata[next][1];
  temp3 = databuf->ddata[next][2];
  temp4 = databuf->ddata[next][3];
  temp5 = databuf->ddata[next][4];
  temp6 = databuf->idata[next][0];
  temp7 = databuf->idata[next][1];
  fprintf(out_stream,"\n%.3f   %.4f  %.4f   %.4f  %.4f   %d  %d",
    temp1,temp2,temp3,temp4,temp5,temp6,temp7);

  /* update buffer_tail */
  disable();
  databuf->buffer_tail = next;
  enable();

  return 0;
} /* end of routine get_data */

/*-------------------------------------------------------------------------
ROUTINE
  CHECK_DATA
  Klaus Obergfell

DESCRIPTION
  checks number of data points stored in buffer

PARAMETER
  struct data *databuf   - pointer to structure data

RETURN
  number of data points stored in buffer
-------------------------------------------------------------------------*/
static int check_data(struct data *databuf)
{
  int ret_value;

  ret_value = databuf->buffer_head - databuf->buffer_tail;
  if(ret_value < 0)
    ret_value += DATAPOINTS;

  return ret_value;
} /* end of routine check_data */


/*************************************************************************
end of file util.c
*************************************************************************/
```