IMPLEMENTATION OF A VISUAL SERVOING SYSTEM FOR EVALUATION OF ROBOTIC REFUELING APPLICATIONS

THESIS

Mikel M. Miller
Captain, USAF

AFIT/GE/ENG/87D-45

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY
AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

(1)

AFIT/GE/ENG/87D-45

IMPLEMENTATION OF A VISUAL SERVOING SYSTEM FOR EVALUATION OF ROBOTIC REFUELING APPLICATIONS

THESIS

Mikel M. Miller
Captain, USAF

AFIT/GE/ENG/87D-45

Approved for public release; distribution unlimited

AFIT/GE/ENG/87D-45

IMPLEMENTATION OF A VISUAL SERVOING SYSTEM FOR EVALUATION OF ROBOTIC REFUELING APPLICATIONS

THESIS

Presented to the Faculty of the School of Engineering
of the Air Force Institute of Technology
Air University
in Partial Fulfillment of the
Requirements of the Degree of
Master of Science

Mikel M. Miller, B.S.
Captain, USAF

December 1987

Preface

The purpose of this research was to design and integrate a visual servo control scheme for a PUMA 560 robot arm that derives its position feedback information from a Machine Intelligence Corporation (MIC) vision system. The vision system's television camera was mounted to the PUMA's third joint. The integrated vision-robot system (VRS) successfully employed closed loop, static and dynamic, visual servo control techniques for demonstrating the feasibility of using a VRS to ground refuel an aircraft.

I wish to express my deepest gratitude to a number of people who helped me complete this exciting research project. In particular Dr. Matthew Kabrisky, my advisor, and Dr. Michael Leahy both of whom provided valuable suggestions and recommendations. Dr. Leahy's expertise in robot control and the PUMA 560 was invaluable. This research effort would not have been a "success" without his help and assistance. I am forever grateful to Dr. Kabrisky who was always excited to come "underground" to view my progress ("YOU HAVE 20 SECONDS TO DROP YOUR WEAPON", to quote the well-renowned movie production "RoboCop"). Dr. "K" provided unmeasurable guidance and direction, especially in my write-up. I would also like to express by deepest thanks to Mr. Bob Ewing (THANKS BOB) for his encouragement and excitement, especially at the start of this project. Thanks also go out to Lt Col Zdzislaw H. Lewantowicz for his efforts in reviewing my work.

Thanks also go out to Capt Dewayne Davis from the Flight Dynamics Laboratory and Mr. Ed Horan from Unimation. Capt Davis provided good information on current robotic issues facing the Air Force. A special thanks go to Mr. Horan, for without Ed Horan and "Ma Bell", I'd still have two pieces of equipment that "didn't want to talk to each other".

I also owe my most sincere appreciation to two fellow students and friends, Captains Peter Van Wirt and Robert Lashlee. These individuals helped make the AFIT experience a highlight in my life - THANKS GUYS.

Finally, last - but by far not least, I thank my blessed, loving wife, Colleen; my wonderful children, Casey and Krista; and God. Colleen's enthusiasm and encouragement are inspirational and they make our relationship a special gift. Her ability to make me work and have fun are uncanny (Thanks for all the fun on the golf course).

Table of Contents

List of Figures

Abstract

The threats of dangerous environments and projected cuts in military personnel, combined with advances in robotics and sensors, have caught the attention of the United States Air Force. The Flight Dynamics Laboratory at Wright-Patterson Air Force Base has been conducting research into concepts for performing aircraft ground refueling using robotic systems.

The main sensor for a robotic refueler application is vision. Visual information received from a TV camera mounted to the refueling boom, provides the feedback data necessary for employing visual servo control techniques. This feedback data, the refueling port's centroid and depth, is used to visually guide the robot refueler to the refueling port.

This research effort designs and integrates a visual servo control scheme for a PUMA 560 robot arm that derives its control information from a Machine Intelligence Corporation vision system. The vision system's TV camera is mounted to the PUMA's third joint. The integrated vision-robot system (VRS) uses closed loop, static and dynamic, visual servo control techniques to demonstrate the feasibility of ground refueling.

The results successfully demonstrate the VRS's ability to search for the receptacle, and use visual servo control techniques to guide the VRS to it.

IMPLEMENTATION OF A VISUAL SERVOING SYSTEM FOR EVALUATION OF ROBOTIC REFUELING APPLICATIONS

I. Introduction

General Issue

Advances in robotic technology have helped to revolutionize industry worldwide. The major reason for this revolution lies in a robot's ability to repeat a preprogrammed sequence of tasks. Robots have freed human beings from boring, repetitive tasks. Their strength and endurance allows safe, accurate operations, especially in hazardous conditions which exist in various industrial and military situations. A robot required to operate effectively in changing, hazardous environments must be equipped with sensors (vision, acoustic, etc.) and artificial intelligence (AI). AI algorithms process data received from the sensors enabling real-time decisions to be made affecting the preprogrammed operation of the robot. A robot equipped with sensors and AI is considered an intelligent robot (8:229-230).

One potentially dangerous environment for personnel exists in a chemical/biological (CB) contaminated area. A CB

contaminated environment on the runway of a military installation is possible during a hostile situation. Exposure to this environment can result in serious injury or death (3).

One solution to the problem of exposing personnel to a CB environment is to have an intelligent robotic refueler accomplish the task of aircraft ground refueling. A brief, simplified scenario of a possible refueling operation follows:

1) The aircraft parks.

2) The robotic refueler pulls up to the aircraft. This could be accomplished autonomously or by using a human driver.

3) A refueling boom swings up and over the aircraft near the air refueling port.

4) A vision system finds the refueling port and vision servo control techniques guide the robot boom into the port.

5) Once the aircraft has been refueled, the boom is removed from the port and stowed.

6) The robotic refueler pulls away from the aircraft.

7) The aircraft departs.

An artist's conception of a robotic refueler with a human driver refueling an aircraft through the air refueling port is shown in Figure 1.

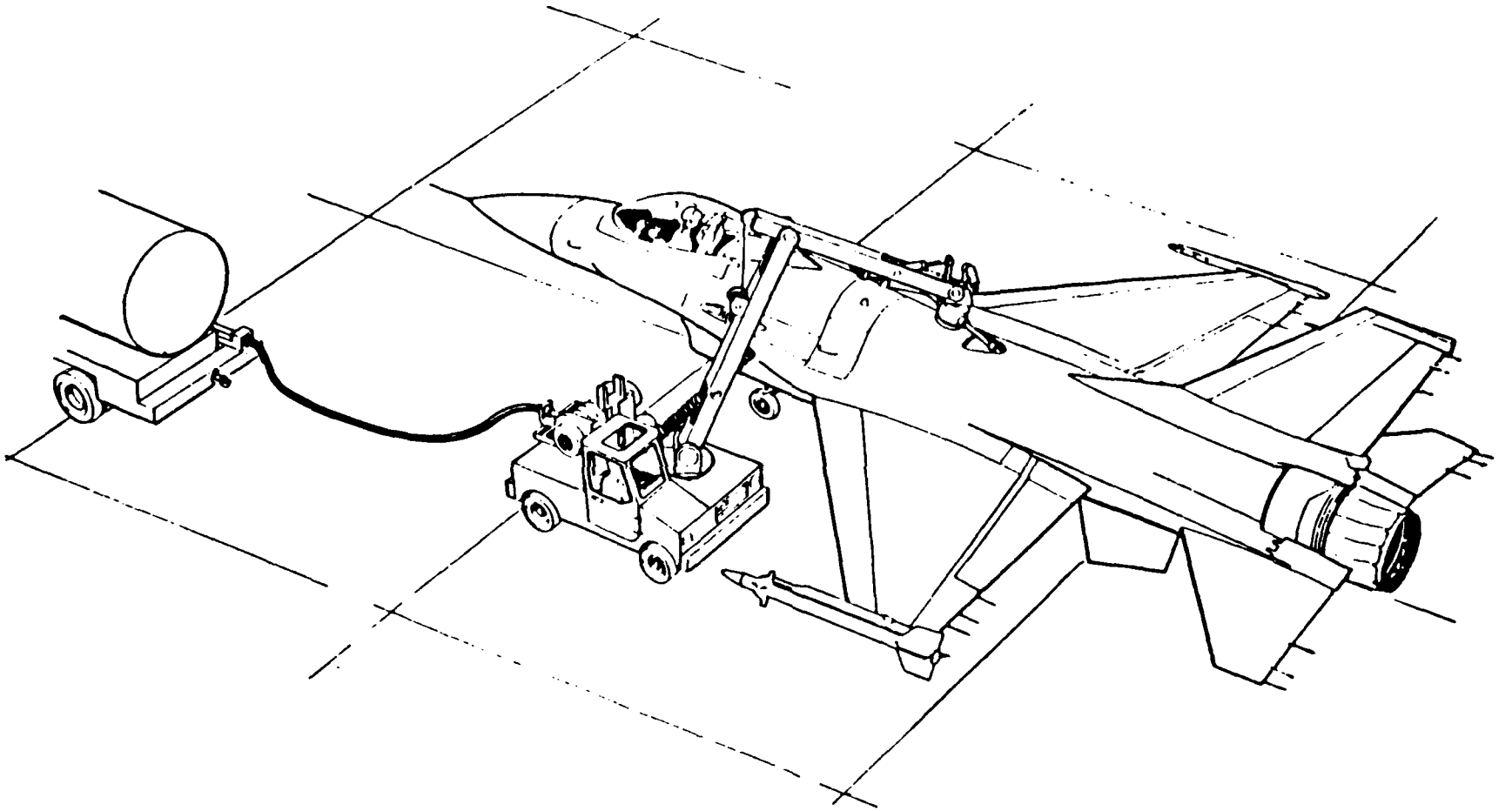


Figure 1. Robotic Refueler (13:53)

Background

The threats of dangerous environments and projected cuts in military personnel have raised concern in several operational aspects of the United States Air Force (USAF). The Flight Dynamics Laboratory at Wright Patterson Air Force Base has been conducting research into concepts for performing aircraft ground refueling using robotic systems. To enable a robotic refueler to operate autonomously, intelligence must be integrated into the system. The main sensor for the application of a robot refueler is vision. Visual information received from a television camera provides the data necessary for vision servo control. Vision servo control provides feedback information on the relative position of the robot to a target. This feedback data is used to guide the robot to the desired target (12:107-116).

Safety requirements in current ground refueling procedures require a minimum of three military members to refuel an aircraft. With predicted cuts in military manpower, any safe refueling method which could reduce personnel helps solve the manning problem. It is also desirable to have a minimum number of human beings exposed in a hostile environment during the refueling.

The Flight Dynamics Laboratory's initial research determined the air refueling port to be the best place to refuel an aircraft using a robotic refueler. The air refueling port has many advantages:

1) It is standard on all Air Force fighter aircraft.

2) It is compliant within a 30 degree cone of insertion.

3) It is a slipway for teleoperated insertion of a boom nozzle.

4) It has connections for audio or other data transfer information.

Using the air refueling port eliminates the need for robot end-effectors to simulate complicated human dexterous manipulations required by current ground refueling procedures, such as opening and closing an access panel, removing and replacing the fuel cap, and twisting on and off the fuel nozzle (3).

Statement of Problem

The purpose of this research is to design and integrate a visual servo control scheme for a PUMA 560 robot arm that derives its control information from a Machine Intelligence Corporation (MIC) vision system. The aim of this vision-robot system (VRS) is to demonstrate the ability of a ground refueling robot to acquire the refueling port of an aircraft in real-time. Guidance of the refueling boom into the port is based on visual feedback received from a television camera rigidly attached to the robot arm.

Scope

This research concentrates on visual servoing. The MIC vision system takes a picture of a well defined scene and forms a video grey-scale binary image which is then processed to determine the position and orientation of a target in the scene. Binary images are needed for the fast computer computations required during efficient, real-time visual servo control.

This investigation focuses on three separate areas to solve the problem of ground refueling an aircraft with a visually equipped robot:

1) Teach the vision system to recognize the refueling port from any viewing angle.

2) Design a visual servo controller for the real-time guidance of the robot refueler to the air refueling port.

3) Develop a scanning algorithm that moves the robot arm in a pre-programmed pattern so its vision system can acquire the refueling port.

Assumptions

Since the cost of developing a full-scale robot refueler is not practical for this project, a simple simulation of the robot refueler will demonstrate the theory developed during this research. To simulate video acquisition of the refueling port, an artificial, well-defined, high contrast target-background scene was constructed; the target, a white ball, represents the refueling port and a black background represents the surrounding area. The vision-robot system

(VRS) will scan an area until the vision system acquires the target. Once located, the visual servo controller will guide the VRS to the target.

Computer processing time is of the utmost importance for effective, real-time visual servo control. Determining the location and orientation of a ball requires the least amount of computer processing in the MIC image processor, because the two dimensional image of a ball is a circle from any viewing angle. The high scene contrast assists in providing the fast processing time required during real-time control operation. White against black also eliminates the chance of shadows distorting the shape of the ball. A lack of shadows removes the requirement for any special lighting normally found in a grey-scale image system.

Another reason for using a ball as the target is the camera's fixed focus lens. Even though the ball becomes blurry as the camera approaches the target, the ball's digital image will remain circular, thus allowing simple target identification.

Additionally, it is assumed that the path between the ball and the VRS has no obstacles. Therefore, obstacle avoidance techniques will not be researched or employed.

Finally, it is noted that an added capability of the MIC vision system enables it to determine the centroid and

orientation of any target placed in its camera's field-of-view (FOV). This enables the vision system to provide some information about a target even though it may not be able to identify the target. The vision system's television camera will be mounted rigidly on the third joint of the PUMA 560 robot arm. Specifications concerning the PUMA 560 and the MIC vision system are in Appendix A.

General Approach

This research effort will be separated into two tasks based on two different types of visual servo control schemes. The first task implements closed-loop, static look-and-move visual servo control techniques to simulate the acquisition and guidance of the robot refueler boom into the refueling port. This task verifies the interface and correct operation of the PUMA 560 and the MIC vision system. It also develops a system that searches for a target, and once acquired, implements static visual servo control techniques to guide the VRS towards the target. The second task implements closed-loop, dynamic look-and-move visual servo control techniques to simulate the guidance of the robot refueler boom into the refueling port.

Presentation

This thesis is composed of five chapters. Chapter II summarizes current research in the field of visual servo control. Chapter III presents the system setup and research methodology used in completing this research effort. Chapter IV discusses the results. Finally, Chapter V presents the conclusions and recommendations for further work.

II. Review of the Literature

Introduction

The use of robots, especially in industrial assembly, has grown at a remarkable rate during the last two decades. The term "robot" was coined by the Czech writer Karel Capek and means forced labor. Robots do the work of humans although they do not look or act like humans (8:2-3). The major advantage of robots lies in their ability to repeat a preprogrammed sequence of tasks in a constant environment with great accuracy. The major disadvantage of robots lies in their inability to sense and respond to a changing environment (8:229).

The integration of intelligence into a robot is needed for it to effectively operate in a changing environment. Intelligence can be installed in a robot by equipping it with appropriate sensors and artificial intelligence (AI). Figure 2 shows a block diagram of a possible intelligent robot. Data from the sensors provide the necessary information required by AI algorithms to enable real-time decisions that may affect the preprogrammed operation of the robot (8:229-230).

This chapter briefly introduces the various robotic sensors and control schemes currently in use or in development. A summary of the various applications of robot manipulator systems integrated with visual sensors and visual servo control schemes will also be presented.

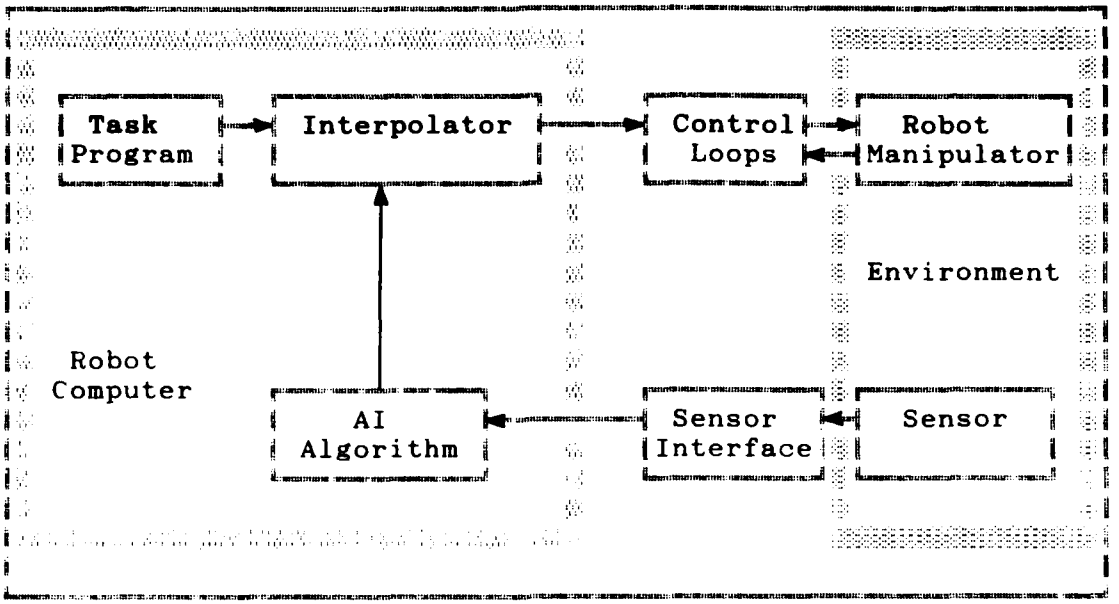


Figure 2. Block Diagram of an Intelligent Robot System (8:230)

Sensors

Sensor development and integration has been an area of active research. "The interaction of robots with sensors has always been an important goal in the development of robotic systems. Such sensor-based systems would have increased functional capabilities as well as flexibility in the execution of tasks" (12:107). Robotic sensors are classified as either contact or noncontact. Contact sensors include tactile and force-torque sensors. Noncontact sensors include proximity, acoustic, range, and visual sensors (8:230).

The visual sensor of interest to this research uses an image processor to digitize a scene viewed through a television (TV) camera. The vision system provides the position and orientation of a target. This information is fed back to the robot controller. The controller directs the robot to the target, enabling the robot to complete its programmed tasks.

Control

The two basic types of control schemes employed, when using sensor data for robot control, are static sensing and dynamic sensing. The static sensing method maintains the robot in a stationary position while sensing takes place. Most control systems which implement this method use open loop control. Once all processing of sensor data has been completed, the robot accomplishes its required tasks without any further information received from the sensing device. By contrast, the dynamic method controls real-time robot motion based on informational updates acquired from the sensor in a closed loop configuration (8:231).

Most modern vision systems employ a static closed-loop mode of operation. The vision system takes a target's picture and feedbacks its position with respect to the robot's position. The difference between the two positions is considered a sensed error. The sensed error actuates the necessary robot movements required to eliminate the error. The vision system continually monitors the error between the

robot's position relative to the target's after each robot motion until the error no longer exists (8:231).

Visual Servo Control

A limited amount of research has been done in the development of control structures for visual servoing. This section discusses current techniques and approaches used in visual servo control.

"The function of machine vision in servoing is to determine the spatial relationship that exists between the camera, tool, and the workpiece" (1:943).

> Visual servo robot control systems provide feedback on the relative end-effector position of a robot. They offer an interactive positioning mechanism which depends upon extraction and interpretation of visual information from the environment (12:107).

The accuracy of visual feedback depends on the distance from the camera to the end-effector and the distortion of the image plane due to the camera lens (1:944-945).

Visual servo control systems are characterized by either a feedback representation or a joint control mode. Image-based and position-based visual feedback systems comprise the feedback representation. Image-based visual feedback control parameters are based on an image's features, where as position-based visual feedback control parameters are based on an object's geometry, position and orientation (12:108).

The two basic joint control modes are closed-loop and open-loop. "The 'look-and-move' structures utilize inner closed-loop joint control. The 'visual-tracking' structures

have no closed-loop joint control and rely only on sensory feedback to drive the manipulator" (12:108).

Practical, real-time operation of sensor servo systems has yet to be achieved. Sanderson and Weiss believe:

> A number of factors have delayed the practical development of sensor-based servo systems. Robot positioning systems and robot control systems are difficult to analyze and design in themselves, and there are a large number of practical applications of non-sensing robots in highly structured industrial and other environments. Sensing systems, particularly vision, are often slow relative to manipulator dynamics, and practical applications or machine vision are currently also limited to highly constrained situations. The analytical complexity of both manipulator control and sensory data interpretation make general formulation of the sensor-based control problem challenging (12:107).

In visual control systems, the video camera provides the required image information for determining camera position relative to object position. Computer processing times (including interprocessor communications) under 1/10 of a second (14:214) are required during scene analysis to achieve real-time visual servoing for some typical engineering applications. The computer processing includes image preprocessing, feature extraction, and interpretation. The computer processing adds unwanted noise and time delays to the system which, when taken into account, adds complexity to the system's operation.

A majority of the control work using vision systems has been with the open loop, static look-and-move type discussed earlier. In these applications the camera is mounted

separately from the robot. The vision system takes a picture, processes the scene information, and outputs the object's position and orientation to the robot control system so the robot manipulator can accomplish its predetermined task. This technique suffers since the above operations do not occur in the real-time requirement for visual servo control.

Research

Numerous authors have researched and/or developed a variety of vision-robot systems using some form of visual servoing. There are two basic methods used in vision servoing. One involved the camera mounted over the workspace while the other has the camera mounted directly on the robot manipulator. The first method usually works on systems concerned with the servoing of a robot to grasp an unorientated object. Ward et al., developed CONSIGHT: an industrial vision-based robot system which uses a linear array camera (mounted overhead) and structured light. As an object passes below the camera on a moving belt, CONSIGHT produces a two-dimensional (2-D) image of the object in order to determine its position and orientation with respect to the robot (15:195-211). This system, as well as others using structured light, only work effectively in controlled, relatively stable environments like those found in many industrial applications.

Another example using a camera mounted separately from the robot arm was done by Palm et al. They have demonstrated how a visual system comprised of a single camera and two light sources used visual servoing techniques to enable the accurate robotic assembly of pins into an electrical connector. The vision algorithm relied upon the relative distance between the pin's actual position and its desired position (10:221-236).

The second method used in visual servoing has the camera mounted on the robot manipulator. "This technique is desirable for accuracy due to the close proximity of the camera to the end-effector and the workpiece" (1:943). Research in this area involves the extraction of 3-D information from a 2-D image using triangulation, structured lighting , and perspective cue depth measurement (1:943). This method has been proven successful by Hill and Park, and VanderBrug (6;14). VanderBrug et al. developed a system utilizing a miniature solid state camera and a strobographic light source mounted directly onto the robot manipulator. The light source flashes a plane of light onto an object in the camera's field of view. When the plane of light strikes the object, line segment images are viewed by the camera. Computer algorithms utilizing triangulation principles interpret the image segments to provide the range and orientation of the object (14:213-231).

Bamba et al. incorporated a system based on the optical pattern projection method used for recognition of 3-D objects. Their system uses a small visual sensor which has a photo sensitive position detector and a light emitting diode (LED) for its main components. This system can be used for path correction during arc-welding. Light emitted from the LED is projected onto the surface of the object to be welded and deflected back to the position detector, providing 3-D information about the object (2:169-177).

The systems mentioned above all suffer from the same problem; the requirement for some form of structured light and controlled environments. These systems would not be practical for a robotic ground refueler which would have to operate in a changing environment. The next three developments more closing relate to the scope of this research effort.

An example using a small, solid state camera attached to a robot end-effector was demonstrated by Hill and Park. This system can operate by either conventional lighting or structured lighting, therefore, making it more versatile. The image processor converts a scene into binary images to achieve fast and reliable processing. Visual feedback enables the robot to guide the manipulator to a desired target (6:233-246).

Harrell et al., developed a visual servo system for a three degree-of-freedom harvest robot to pick fruit from a

tree. The system received its visual feedback information from a CCD camera mounted in the robot's third prismatic joint. "Simple proportional gain control laws utilized this vision feedback to independently servo the two revolute joints" (5:537) to the desired piece of fruit. The system used a strobe light to determine the distance to the fruit. The system successfully tracked fruit motions during a harvest cycle (5:537-545).

A system that did not need to compute the coordinate relationship between the robot end-effector and the camera was developed by Kim et al. The system used a miniature camera mounted to the sixth joint of the PUMA 560 robot arm. The system visually servoed itself to the correct orientation and position to grasp a target. The system obtained 3-D information about the target from a single camera which used an iterative focusing method and a distance formula which derived distance from two different images along the same line of sight of the camera (7:417-422).

Summary

"Visual servo robot control systems provide feedback on the relative end-effector position of a robot. They offer an interactive positioning mechanism which depends upon extraction and interpretation of visual information from the environment" (12:115). Most current systems operate using the basic static look-and-move sequence of operations. These systems usually incorporate a camera mounted over the

workspace. However, there is increasing development in the areas of integrated vision-robot systems using dynamic look-and-move and visual-tracking techniques. These new developments usually have the camera mounted directly on the robot manipulator. An intelligent robot is "visionable" by equipping it with the ability to sense and respond to a changing environment.

This completes the Review of the Literature; the system setup and methodology can now be presented.

III. System Setup and Methodology

Introduction

The goal of this research is to design and implement a visual servo control scheme to simulate the ground refueling of an aircraft using a visually guided robot. An integrated vision-robot system (VRS) composed of a PUMA 560 robot arm and MIC vision system are used to accomplish the research.

This chapter describes the system setup and methodology involved in applying static and dynamic, look-and-move visual servo control schemes to an integrated robot-vision system.

Static Look-And-Move

To accomplish the task of developing a system that searches for and acquires a target using static look-and-move visual servo control, the task was separated into the following four subtasks:

1. Develop an open loop, static look-and-move system with the camera mounted above the robot work space.
2. Develop a closed loop, static look-and-move visual servo control system with the camera mounted to the third joint of the PUMA 560 that tracks a target (white ball) against a black background.
3. Using the results from Subtask 2, develop a visual servo control system that tracks and moves towards the target.
4. Add to Subtask 3 by developing a scanning algorithm to search for and acquire the target.

The accomplishment of each subtask serves as a building block

in achieving the overall task. The following subsections describe each subtask's system setup and methodology.

Subtask 1: Camera Mounted Above the Work Space.

This subtask verifies the interface between the MIC system and the PUMA 560 robot arm. It introduces the following: the PUMA 560 robot arm and its associated peripherals; the VAL II programming language; the MIC vision system; the interface between the vision system and the PUMA 560; and the relationship between locations in the camera's frame and the robot's frame. A description of the PUMA 560, the MIC vision system, and the interface between the two systems is in Appendix A.

The camera is mounted to a camera stand and positioned above a table in the robot's work space as shown in Figure 3. The vision system's effective work area is established by the camera's field-of-view (FOV). The camera's FOV is determined by the type of lens used and its focal length, and is described by the height and width of the visual field. The lens used in this research is a Fujinon TV 1:1.4/25 and has a FOV with a height of 300 mm and width of 365 mm.

The image processor in the vision system transforms the grey-scale camera image into a binary image by means of a programmable threshold in order to generate its silhouette. To simplify the threshold operation, images should be selected with maximum contrast between the target and its background. White targets against a black background are

used in this subtask. A white target against a black background also eliminates shadows, thereby abolishing any requirement for special lighting.

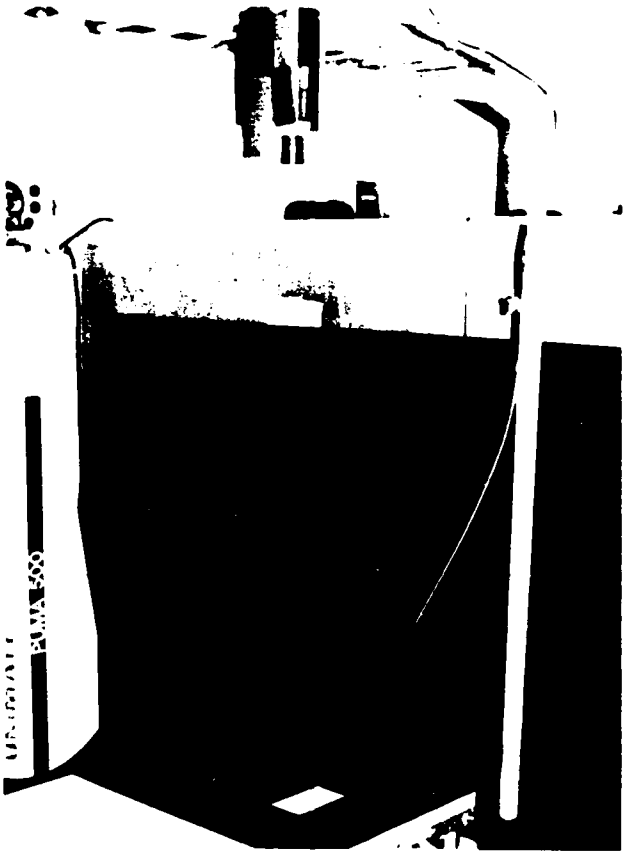


Figure 3. Camera Mounted Above the Work Space

An open loop, static look-and-move system is implemented to verify the interface between the vision system and the robot. The vision system identifies a target and provides the two-dimensional location of its centroid. This information is transmitted, open loop, to the robot

controller enabling the robot to move to the desired location. A functional block diagram of the open loop vision-robot control system is shown in Figure 4.

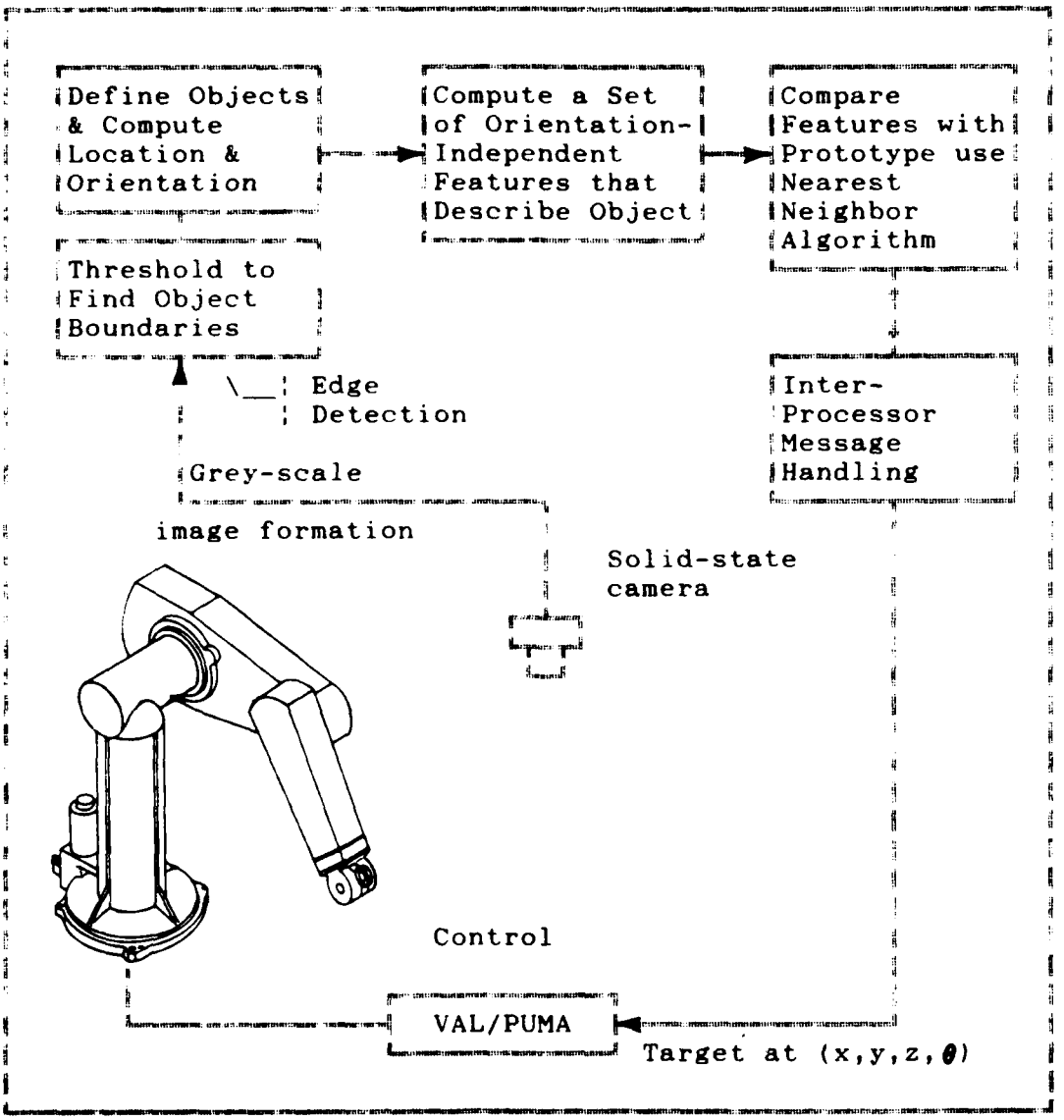


Figure 4. Functional Block Diagram of The MIC/PUMA Control System (18:3)

A menu driven software package accomplishes the tasks shown in Figure 5. Appendix B lists the various programs used by this package (NOTE: listings of the system programs are proprietary information and not provided. However, brief descriptions of the software are in Appendix A). Appendix C contains a user's manual. The 'WARNING' is required to prevent unpredicted movements of the arm due to inaccurate calibration. Figure 6 shows the software structure of the various user selectable options. Option 1 initializes communication between the vision system and the robot.

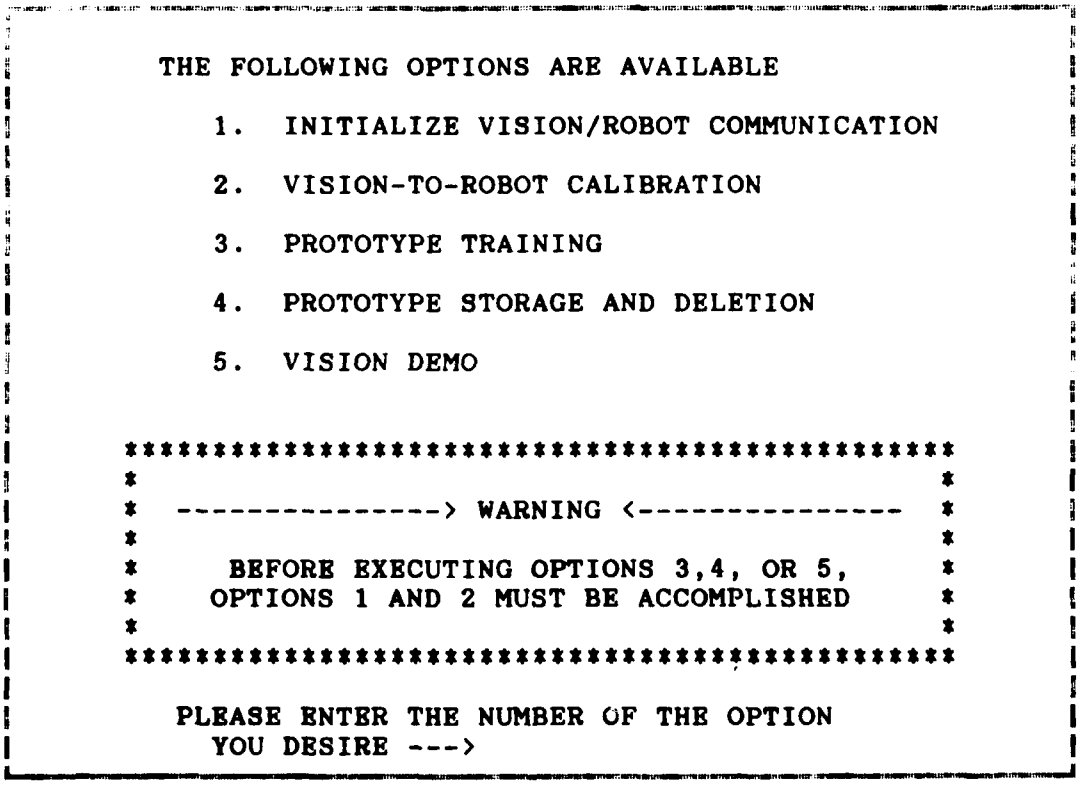


Figure 5. Vision Menu

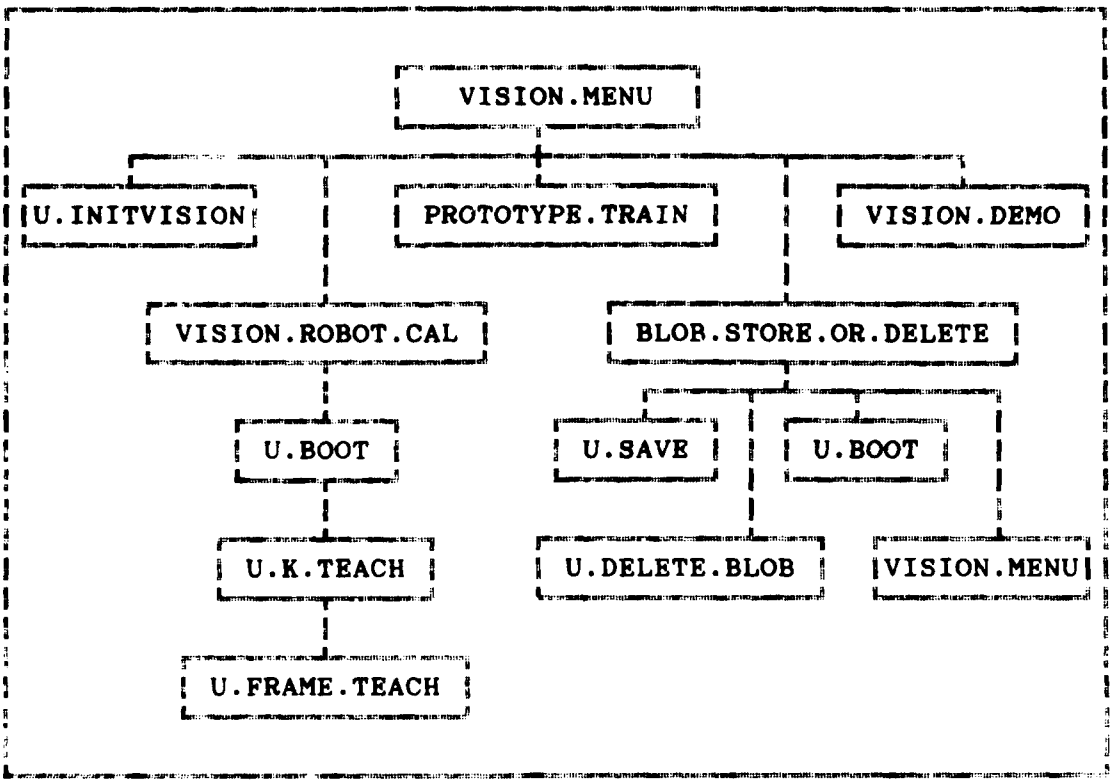


Figure 6. Software Structure Chart

Option 2 calibrates the vision-robot system. This option determines the scale factor defining the ratio of the distance between two locations in the robot's frame to the distance in the camera's frame. Finally, the vision-robot work frame is established to determine the relative transformation between the camera and the robot.

Options 3 and 4 deal with prototyping. Option 3 allows the user to teach the vision system various targets, such as a circle or rectangle. Option 4 enables the user to store, recall, or delete the various targets trained.

Option 5 calls another menu which runs a demo permitting the user to run various programs which identify trained targets and/or directs the robot to point to the centroid of a target in the camera's FOV. Figure 7 displays the various user selectable options in the demo. Appendix B lists the programs used in the demo options. Figures 8 displays the software structure chart. Demo 1 instructs the vision system to take a picture to determine the number of targets, and the identification and centroid position of the largest target in the camera's FOV. Demos 2 and 3 accomplish the same tasks as Demo 1 with the addition of instructing the robot to point to the desired target's centroid.

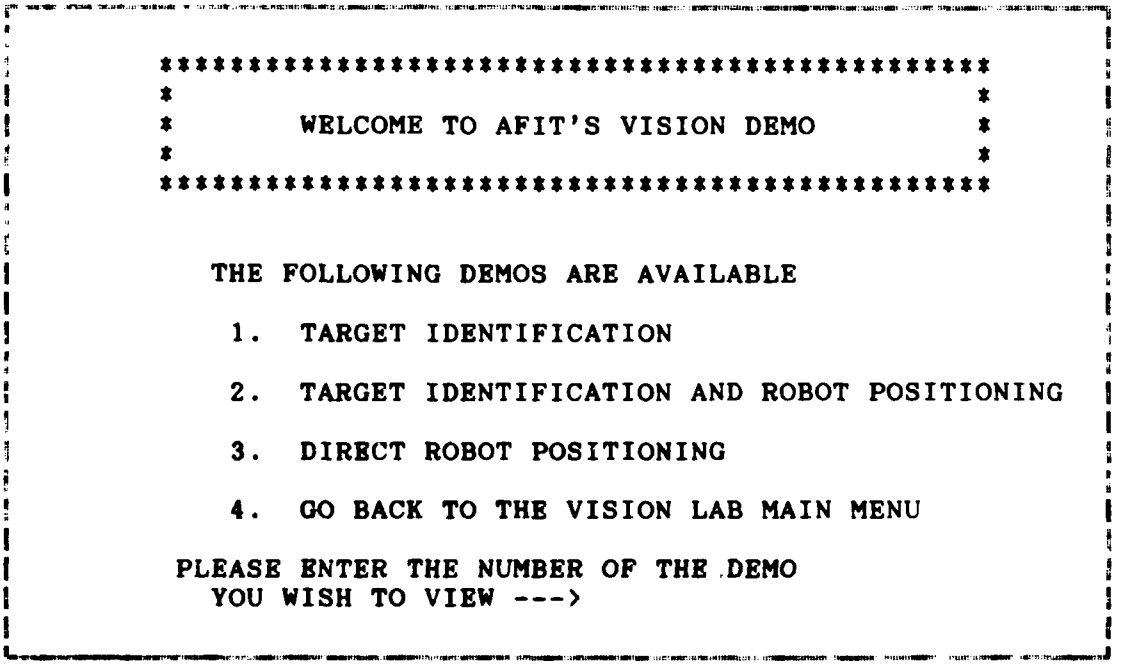


Figure 7. Vision Demo Menu

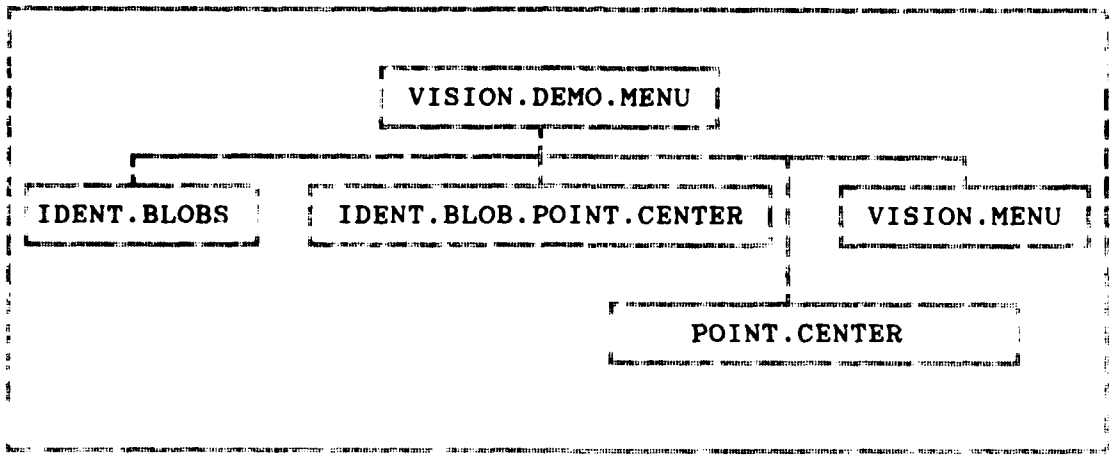


Figure 8. Software Structure Chart

With the robot and vision systems introduced and the interface verified, the remaining three subtasks will describe the setup and methodology for developing a visually guided robot refueler.

Subtask 2: Camera Mounted on Arm; Tracking

This subtask mounts the camera onto the third joint of the PUMA 560 and implements a closed loop, static look-and-move visual servo control scheme. The control scheme utilizes information from the vision system to provide the location of a target's centroid to the robot controller. The controller uses the location information to control the motion of the robot in the PUMA's world coordinate frame.

The PUMA 560 manipulator is controlled point-to-point. Motion between points can be controlled by a proportional (P) or proportional and derivative (PD) feedback loop. The P

feedback loop uses high gain position feedback to bring the difference between the robot's desired position and the actual position to zero. However, a P feedback loop controller cannot guarantee an overdamped response, i.e. no overshoot (4:89-92). This visual servo control scheme cannot afford a problem with overshoot, because the VRS may strike the target or oscillate around it endlessly. To overcome this problem, the PD feedback loop controller, which feeds back velocity as well as position is used. This system is tuned by the manufacturer to provide an overdamped response; guaranteeing no overshoot. PD loop control is desired for this research effort, however, the PD loop is only active for 28 millisecond increments. Therefore, robot movement commands must be divided into motions that can be completed during each time increment, otherwise P loop control takes over and an overdamped response is not guaranteed.

It is desired to have the camera centered on the target at all times. Therefore, as the target moves, the robot will track it, attempting to keep the target centered in the camera's FOV. The vision system takes a picture and estimates the error between the target's position and the center of the camera's FOV. The block diagram of the visual servo control scheme is shown in Figure 9. This subtask simulates the visually guided robot refueler's responsibility to maintain camera view on the aircraft refueling port.

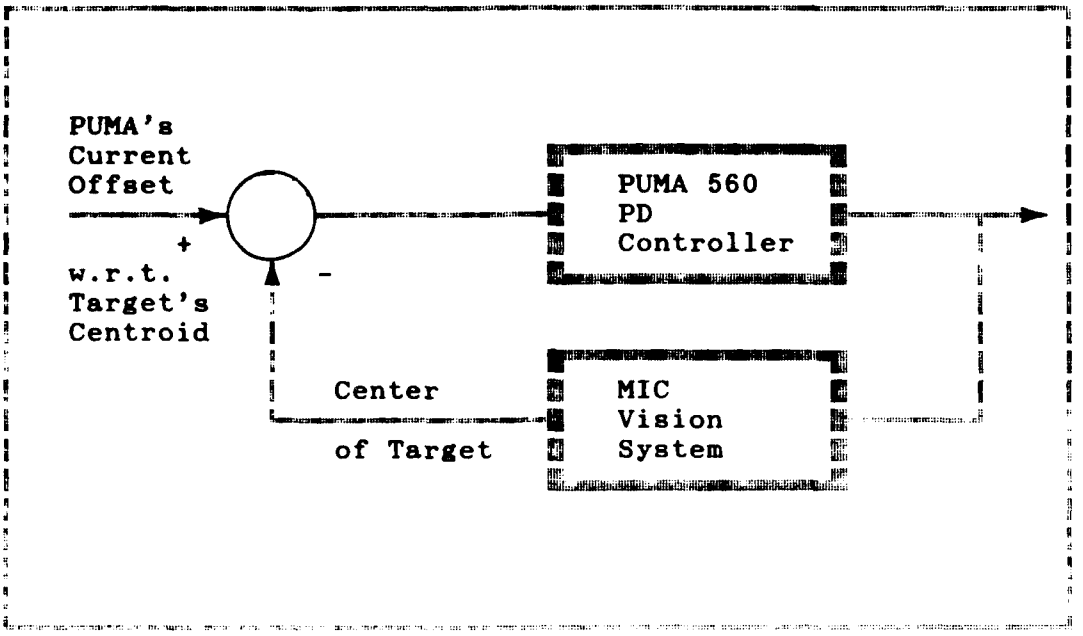


Figure 9. Block Diagram of Visual Servo Control Scheme

The camera is mounted to the robot's third joint using Velcro strips. The camera lens is pointed perpendicularly to the robot's World Coordinate System (WCS) x-y plane, see Figure 10. In this configuration, an offset exists between the location of the camera and the robot's end-effector. To account for the offset, the center of the camera's FOV is considered to be the robot end-effector's current location. This is a translation along the camera's viewing axis. An x and y correction is not necessary since it is desired to keep the target centered in the camera's FOV, not centered below the robot end-effector. Therefore, if the target is not centered in the camera's FOV when a picture is taken, the

target's x and y centroid position is considered an offset to be corrected by the controller.

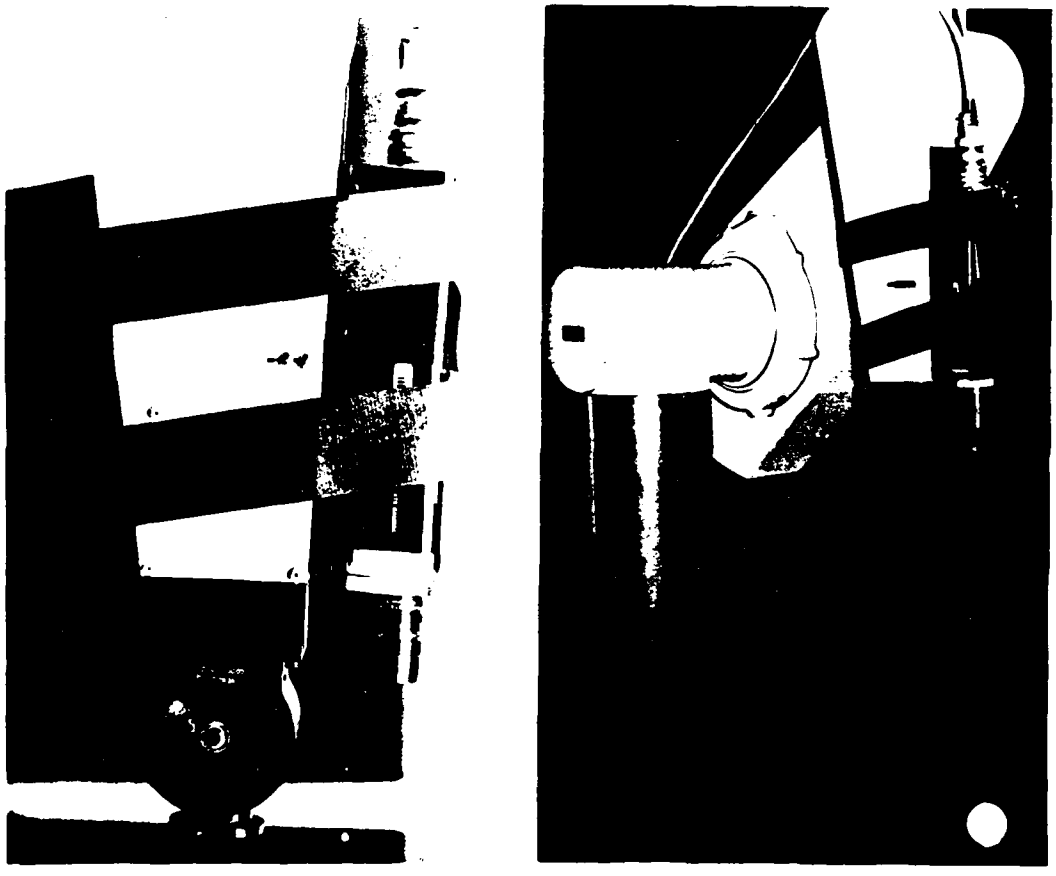


Figure 10. Camera Mount and Starting Position

The vision-robot system is calibrated to relate camera pixel positions in the camera's x-y plane to robot positions in the robot's WCS x-y plane. Since this work is two-dimensional and the x-y plane of the camera's FOV is parallel to the robot's x-y plane, complicated transformations relating the position of a target in the camera's FOV to the location of the robot's end-effector are not required. Thus, during Subtask 2, the system is not interested in the coordinate relationship between the robot's location with respect to a target placed in the camera's FOV. The vision system simply provides the x and y offset position of the target with respect to the robot's current x and y position. Therefore, a simple one-to-one relationship exists between the location of the target in the camera's FOV and the robot's work space.

The computer program, **track.targ**, (whose flow diagram is shown in Figure 11 and program listing is in Appendix B) uses information from the vision system to correct any offsets between the centroid position of the target and the center of the camera's FOV. Once the target is placed anywhere in the camera's FOV, its centroid position is transmitted to the robot controller. This provides the information necessary to command the robot to position the arm, such that the center of the camera's FOV is directly over the target's centroid.

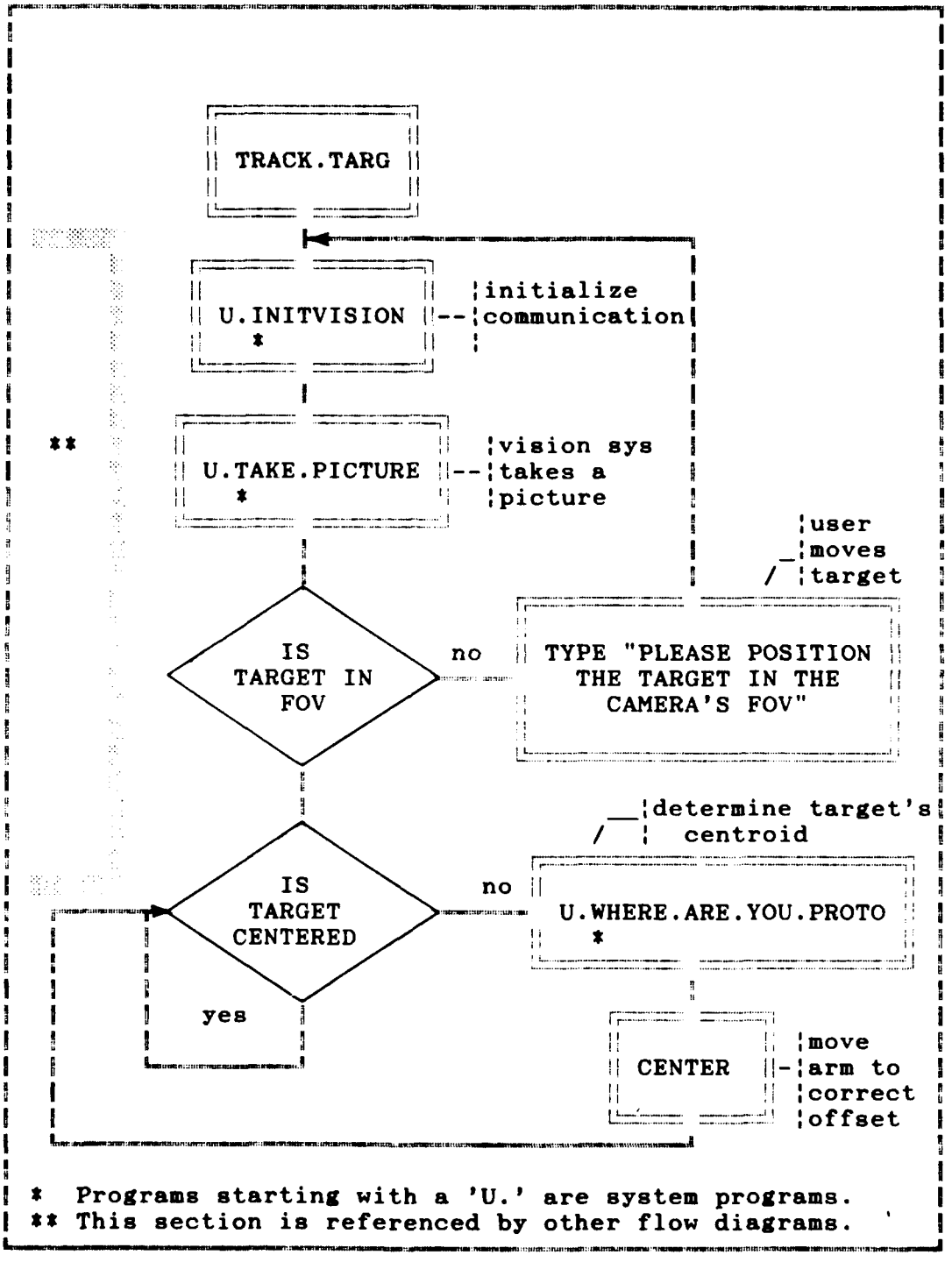


Figure 11. Software Flow Diagram of track.targ

Subtask 3: Camera on Arm; Tracking and Acquiring

This subtask builds on the second subtask by instructing the vision-robot system to move towards the target a predetermined distance while it is tracking. This simulates the refueling nozzle approaching the refueling port. The same closed loop, static look-and-move visual control scheme is employed, with the additional capability of enabling the VRS to move towards the target.

To provide the VRS with the distance of the target to the camera, the vision system is trained with the target at nine different locations, 25 mm apart, along the camera's pointing axis. When the target is at different distances from the camera, the image size varies. The larger the image, the closer the camera is to the target. During training of the vision system, the target is placed in the first position at the maximum distance from the camera lens. In the ninth position, the target is trained at the closest position to the camera. Thus, the larger the target in the camera's FOV, the closer the target is to the vision-robot end-effector.

A white ball is used as the target to alleviate one problem encountered with a fixed focus camera lens. As the camera moves closer to the ball, the camera image becomes out of focus. However, the image still remains circular in the camera's FOV and the binary threshold computation produces sharp boundaries for the centroid computation.

The computer program, **track.targ.scat**, (whose partial flow diagram is shown in Figure 12 and program listing is in Appendix B) instructs the vision system to take a picture and recognize one of the trained targets. Depending on which of the nine targets is identified, the VRS knows the height of the target from the camera. This method works fine for a well defined target like a ball, however, an improved system is required for a more complicated image.

The procedure described above, enables the VRS to obtain three dimensional (x, y, z) information about the target placed in the camera's FOV. The vision system takes a picture to determine the x and y centroid position of the target. The z position is determined once one of the nine target images has been identified.

When a picture is taken, the VRS approaches the target. To account for positional errors inherent in the vision-robot system controller, the system converges towards the target in increments, allowing the system to keep the target centered in the camera's FOV. This convergence method minimizes the tracking error. The target identified determines the incremental distance the system will move towards the target. Another picture is taken and the process is repeated until the VRS is at the desired distance and position from the target.

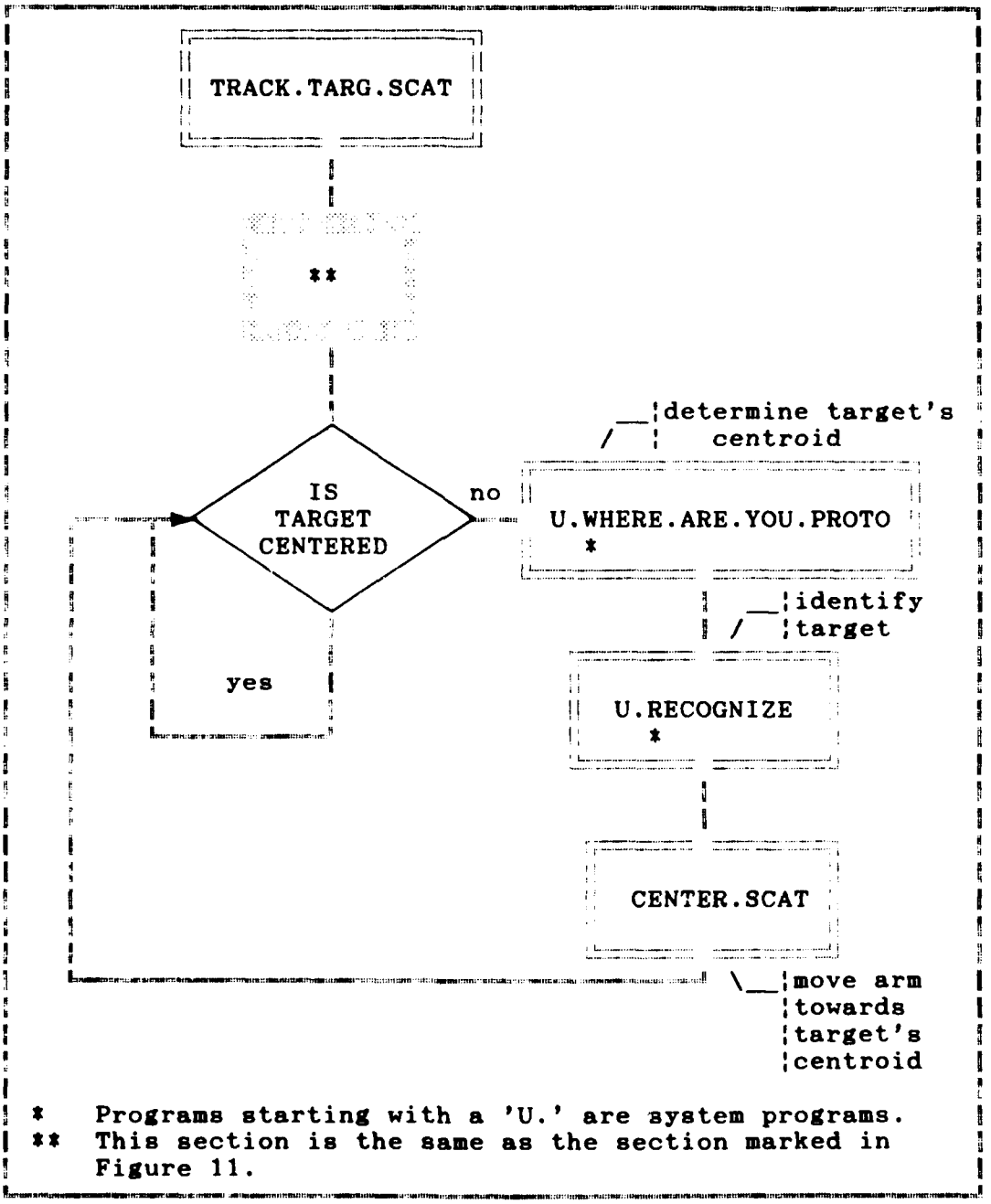


Figure 12. Software Flow Diagram of track.targ.scat

Subtask 4: Camera on Arm: Search

This subtask completes the overall task of designing and implementing a vision-robot system that searches for and acquires a ball.

The computer program, **search**, (whose partial flow diagram is shown in Figure 13 and program listing is in Appendix B) instructs the VRS to go through a square conical search, see Figure 14, covering the perimeter of the search area first, then moving towards the center of the area. The VRS moves in increments approximately equivalent to half the camera's FOV. Moving in this fashion provides an overlap to ensure the search covers the entire search area. The search is broken up into 16 increments, as shown in Figure 14. Each time the robot moves in the search pattern, a picture is taken to check if the ball is in the camera's FOV. Once the target has been acquired by the VRS, the procedures from Subtask 3 are executed and the VRS visually servos itself towards the target.

This completes the description of the system setup and methodology for the closed loop, static look-and-move visual servo control scheme. The dynamic look-and-move system is presented in the next section.

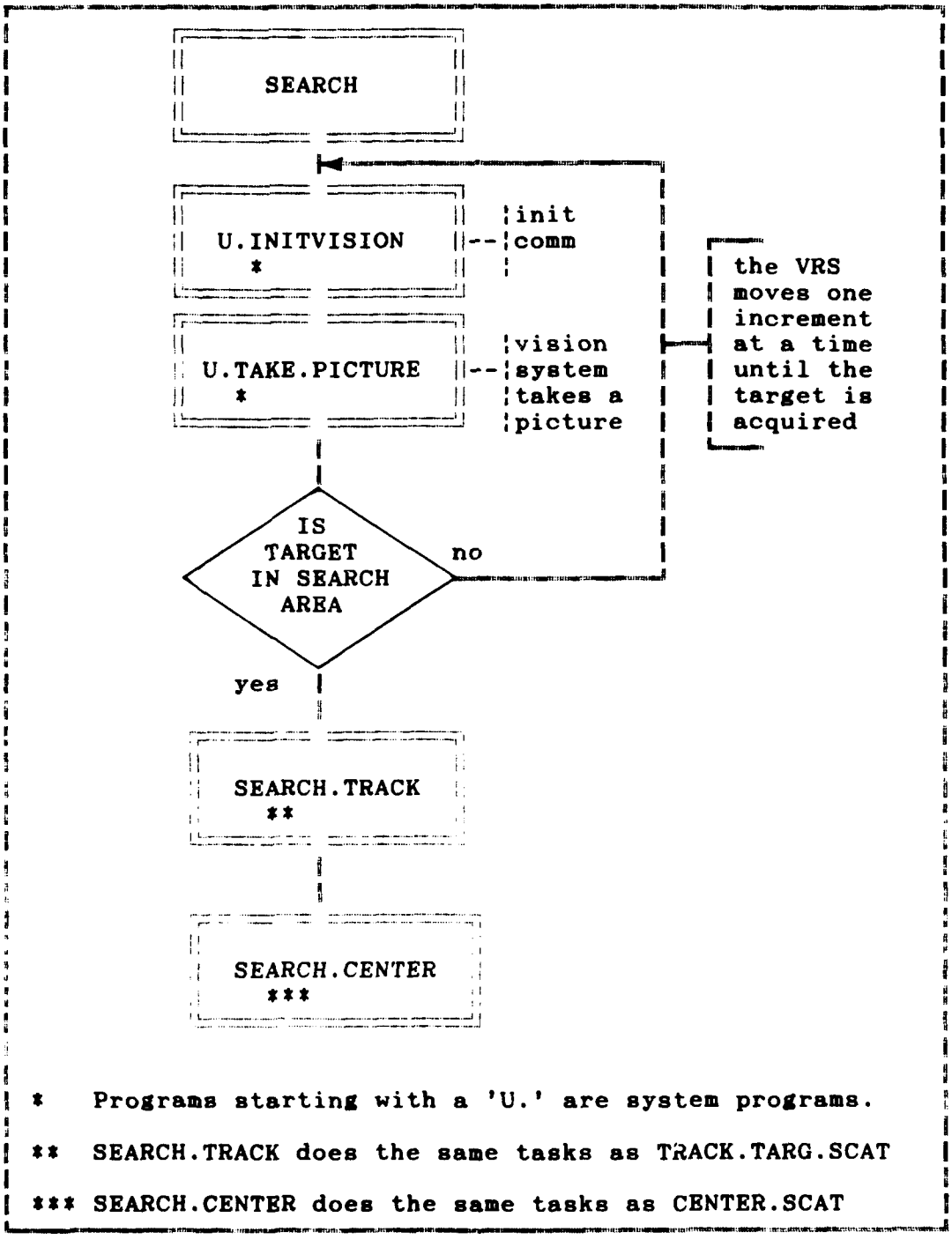


Figure 13. Software Flow Diagram of search

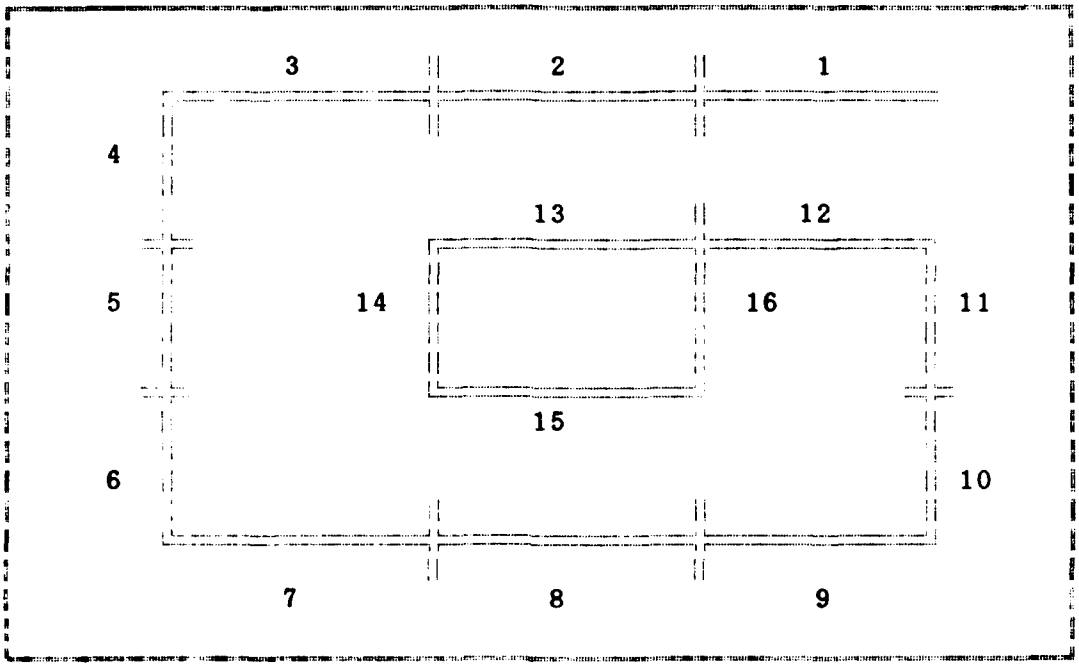


Figure 14. Square Conical Search

Dynamic Look-And-Move System

In this task, a closed loop, dynamic look-and-move visual servo control system is designed and implemented. This task repeats much of the work accomplished in Subtask 3. The difference being in the approach-to-target motion of the robot arm. The controller implemented in this task uses dynamic visual servoing to guide the arm towards the ball.

A system with parallel processing is required to achieve true closed-loop, dynamic visual servo control. Because of computer speed limitations, this capability is currently not possible for this integrated vision-robot configuration. However, a form of dynamic control using serial program execution is possible with the present configuration. In VAL II program execution, once a command has been executed, the processor immediately executes the next instruction, even if the last instruction has not been completed. Unimation refers to the procedure as "procedural motion" (17). Thus, when a command to process a picture is executed before arm motion has stopped, the requirements for dynamic, visual servo control are met. The situation exists when the arm moves towards the target.

As the arm moves towards the ball, the computer program, **track.targ.dcat**, (whose flow diagram is shown in Figure 15 and program listing is in Appendix B) instructs the vision system to take a picture and provide real time corrections for any offsets occurring as the arm moves towards the ball.

This completes the description of the system setup and methodology for this thesis; the results are now presented.

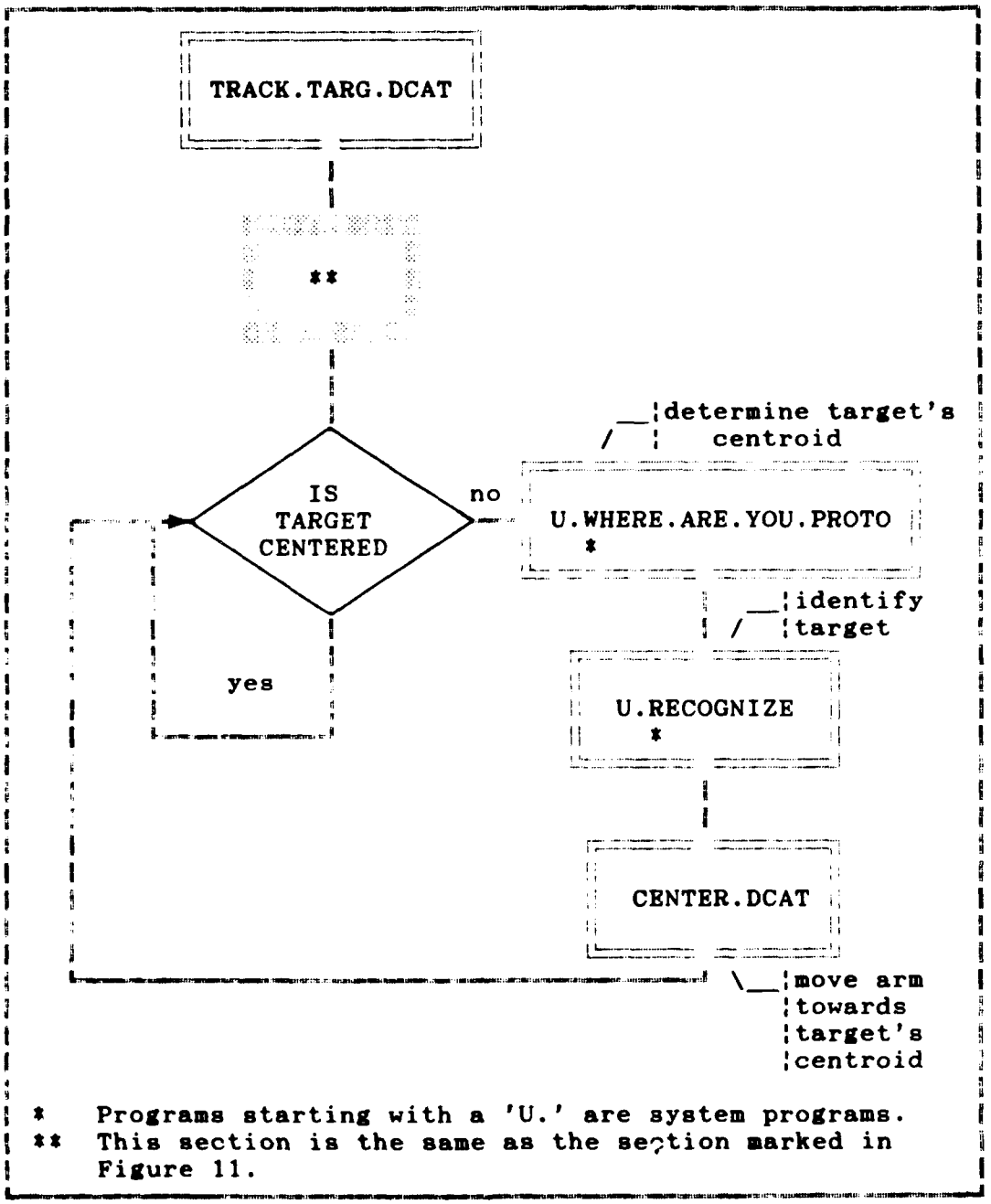


Figure 15. Software Flow Diagram of track.targ.dcat

IV. Results and Discussion

Introduction

The goal of this thesis was to develop an integrated vision-robot system (VRS) to visually acquire a target. The VRS implemented two visual servo control techniques to guide the system to the target. These two techniques included static and dynamic, look-and-move visual servo control. Each control technique was considered a separate task during development and testing. The testing of each task of the integrated VRS occurred simultaneously during development. The results from each task were directly applied to subsequent tasks. This chapter describes the results from each task in the following sections.

Static Look-And-Move

The task of developing a static look-and-move visual servo control system was divided into the following four subtasks:

1. Develop an open loop, static look-and-move system with the camera mounted above the robot.
2. Develop a closed loop, static look-and-move visual servo control system with the camera mounted to the third joint of the PUMA 560 that tracks a target (white ball) against a black background.
3. From Subtask 2, develop a visual servo control system to track and approach the target.
4. Add to Subtask 3 by developing a scanning algorithm to search for and acquire the target.

These subtasks are described in the following subsections.

Subtask 1: Camera Mounted Above the Work Space.

The first subtask was to develop an open loop, static look-and-move scheme which has the video camera mounted above the workspace. The purpose was to introduce the equipment and verify the software and hardware interfaces between the vision system and the robot. In this scheme, the camera was mounted to a stand and positioned above the robot workspace.

The hardware and software interface was verified, the vision-robot system calibrated, and the vision-robot work frame established by following the procedures in the Univision User's Manual. In addition, two targets were trained to the vision system; a white 2 inch diameter circle and a white 3 by 5 inch rectangle, see Figure 16.

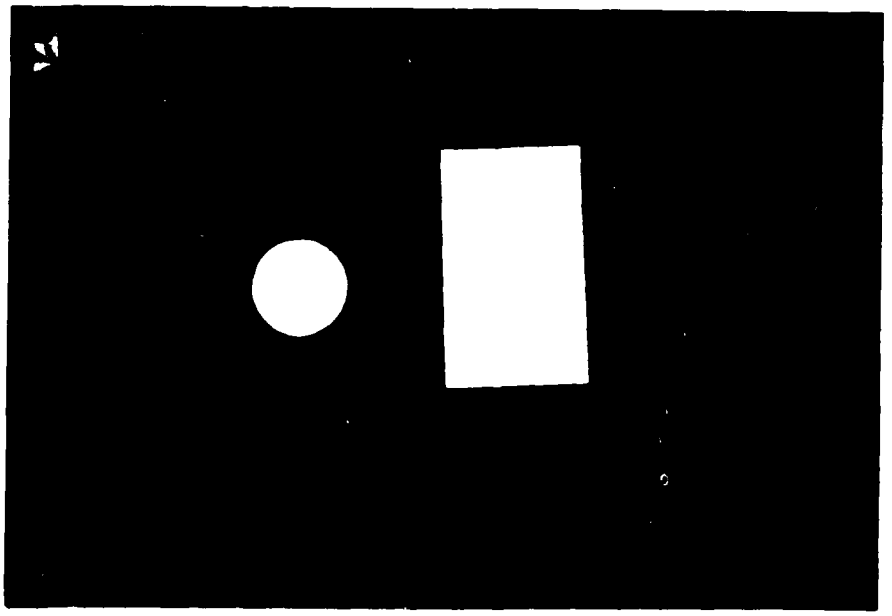


Figure 16. Targets Used in Subtask 1.

Once the targets were trained to the vision system, the program vision.demo, see Appendix B, was executed. The programs executed in this demonstration successfully achieved the open loop, static look-and-move configuration. Each time a program was executed, the vision system identified the target and transmitted its orientation and two-dimensional centroid position to the robot controller.

The first program in the demonstration, ident.blobs, see Appendix B, determined the number of targets in the camera's FOV, identified the largest target, and provided the orientation and two-dimensional centroid position of the largest target in the FOV. This program also instructed the vision system to highlight each target on the video monitor with a white border. It was observed, for the processing of the rectangular target, that the inter-processor communication and image processing took approximately 2.3 seconds during program execution. This time varies depending on how many targets are processed. Figures 17, 18, and 19 show the rectangular target placed in the work frame, the highlighted digitized image displayed on the vision system's video monitor, and the output to the user's terminal.

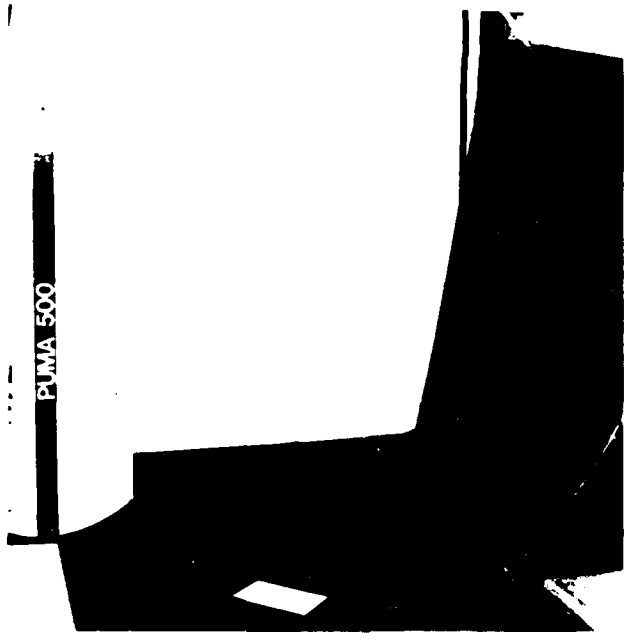


Figure 17. Rectangular Target Placed in Work Frame

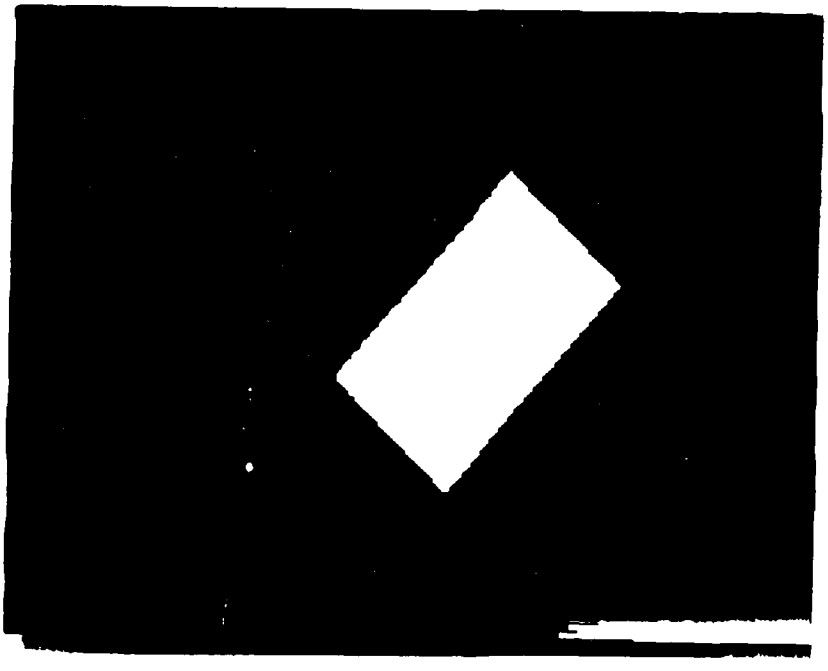


Figure 18. Highlighted Digitized Image on Video Monitor

```
GOOD PICTURE -  THERE ARE   1.
BLOB(S) IN THE CAMERA'S VIEW!


****************************************
* THE LARGEST OBJECT IS A RECTANGLE  *
****************************************


THE X COORDINATE OF THE CENTROID (in mms) IS 32.3818

THE Y COORDINATE OF THE CENTROID (in mms) IS -1.655741

THE ORIENTATION IS -58.64832


====== PRESS <RETURN> TO CONTINUE ======
```

Figure 19. User Terminal Output

The second program, ident.blob.point.center, see Appendix B, accomplished the same tasks as ident.blobs, with the addition of instructing the robot controller to position the robot end-effector above the center of the target. Figures 20, 21, and 22 show the circular target placed in the work frame, the digitized image displayed on the vision system's monitor once the image was processed, and the final position of the robot's end-effector after program execution. Figure 23 shows the results displayed on the robot terminal.

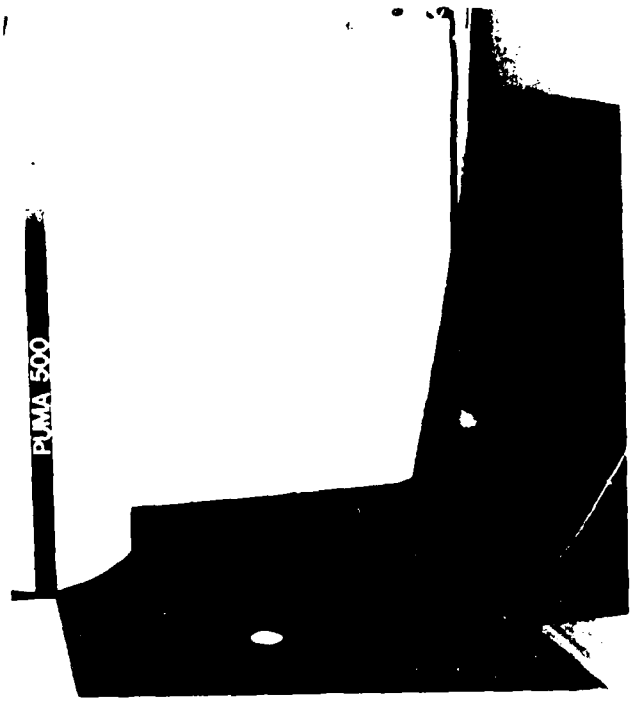


Figure 20. Circular Target Placed in Work Frame

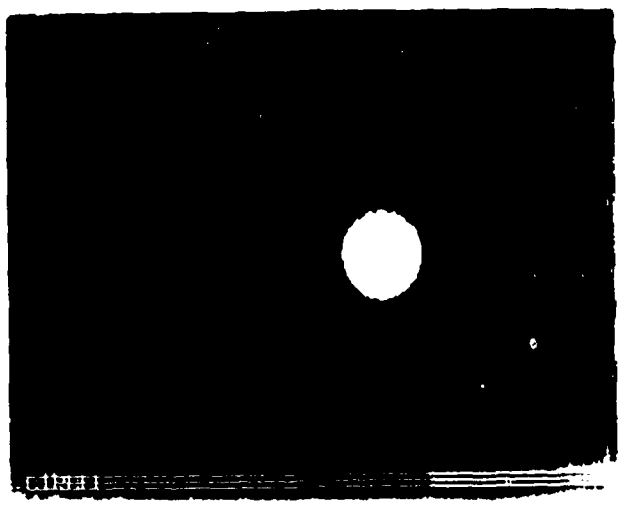


Figure 21. Highlighted Digitized Image on Video Monitor

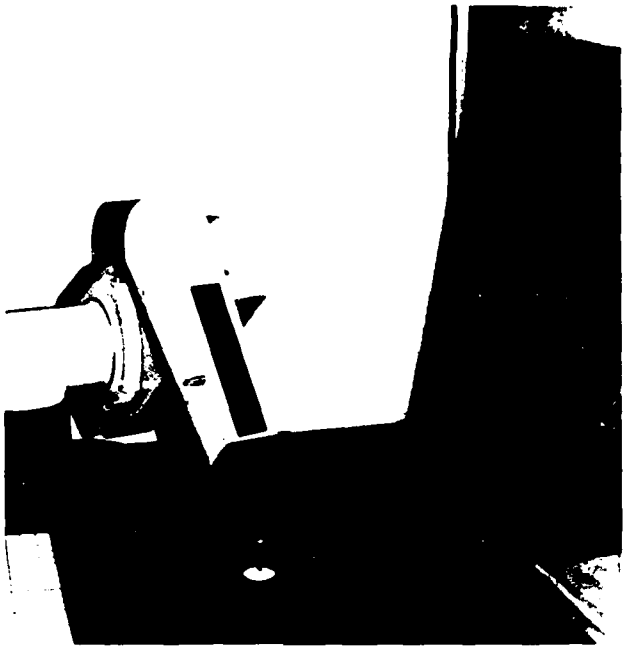


Figure 22. Final Position of Robot End-Effector

```
GOOD PICTURE -  THERE ARE   1.
BLOB(S) IN THE CAMERA'S VIEW!


****************************************
* THE LARGEST OBJECT IS A CIRCLE       *
****************************************


THE X COORDINATE OF THE CENTROID (in mms) IS 45.65081

THE Y COORDINATE OF THE CENTROID (in mms) IS -1.617786

THE ORIENTATION IS -89.2925


====== PRESS <RETURN> TO CONTINUE ======
```

Figure 23. User Terminal Output

The third program, point.center, see Appendix B, accomplished the same tasks as ident.blob.point.center, except it did not highlight or identify the targets. This minimized image processing thereby allowing faster operation.

During the testing of the second and third programs, several observations indicated parallax induced errors in the end-effector's final position over the target when the target was placed towards the edges of the work frame (see Figure 24). However, this thesis was focused on visual servo control, therefore, these errors were judged as insignificant and disregarded since they did not have any bearing towards the success or failure of the research. This subtask's purpose was to introduce the various systems and verify their correct operation.

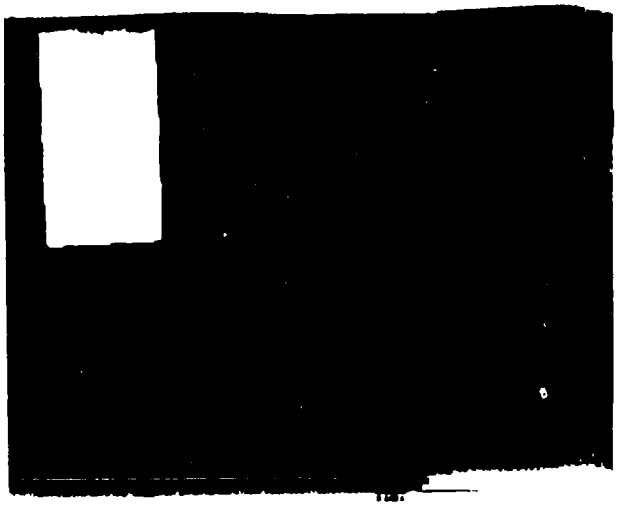


Figure 24. Digitized Image of Target Towards Edge of Work Frame

In summary, this subtask introduced and demonstrated the various hardware and software used throughout the research effort. The results demonstrated the ability to successfully implement an open loop, static look-and-move configuration to point to the centroid of a target placed anywhere in the work frame.

The results presented next address the essence of the research effort.

Subtask 2: Camera Mounted on Arm; Tracking

Subtask 2 initiated the first step towards developing an integrated, closed loop, static look-and-move visual control system. This subtask enabled the VRS to track any target placed anywhere in the camera's FOV. The camera was mounted to the third joint of the PUMA 560 and software was written to guide the movement of the arm based on visual feedback received from the vision system.

The vision system was calibrated (using the procedures outlined by the Univision User's Manual) from an initial starting position based on robot joint positions, #start.position. The initial starting position selected for this research, aligned the camera perpendicularly to the floor. The starting position was stored in software, allowing the VRS to start from the same point throughout testing and development. A vision-robot frame did not have to be established as in Subtask 1 because the camera was mounted parallel to the robot's end-effector. This

configuration considered the robot end-effector's current location to be the center of the camera's FOV translated along the camera's viewing axis.

Once a target (white ball) was placed in the camera's FOV, the program track.targ was executed. The program instructed the vision system to take a picture and provide the two-dimensional position (considered an offset) of the target's centroid. The program corrected the offset by commanding the robot to move by an amount equal to the offset. The program repeated itself until the user aborted it, therefore, enabling the system to maintain track on the target as the target moved. It should be noted that the use of a high contrast target, i.e., a white ball against a black background always guaranteed easy, unambiguous "target" location. The general task of target location (finding an arbitrary target in an arbitrary scene) was not considered part of this research.

Robot motion continued until the vision system determined that the target was within a window of $\pm$ 10 mm from the center of the camera's FOV. Ten millimeters was selected to reduce the amount of time required to center the target in the camera's FOV. Without any window, however, the VRS oscillated continuously around the target, primarily because of jitter in the video quantizer and robot positioning system. Since the application for this thesis involved the refueling of an aircraft through a refueling

port that is compliant within a 30 degree cone of insertion, the use of a tolerance window should not be a problem. It was observed that for this task, using the previously described equipment, the time for the system to center itself on a stationary target (white ball), place on one corner of the camera's FOV, took approximately 2.0 seconds ($\approx$ 0.7 seconds was used in the image processing). Figure 25 shows the sequential motion of the digital images on the video monitor as the VRS visually servos itself to the target.

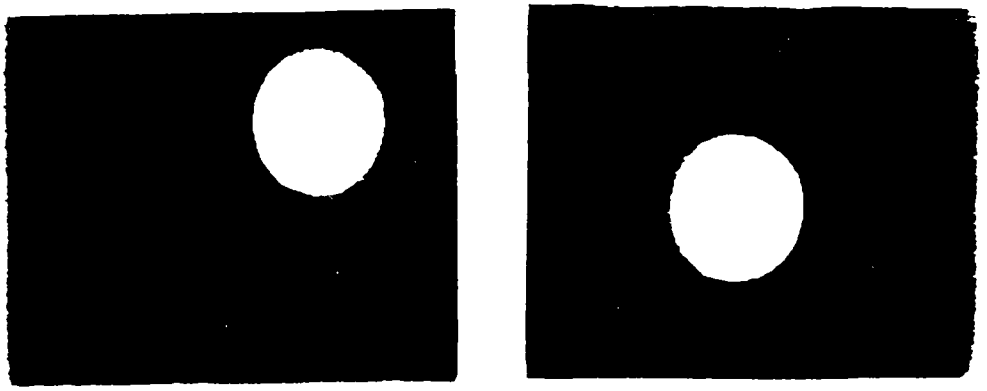


Figure 25. Sequential Motion During Visual Servoing

The oscillation mentioned above was due to errors associated with the calculation of robot transformations and with the image processing of the target. The errors associated with the calculation of the desired final location in the robot

controller occurred "whenever a transformation is used to define the destination of a robot motion" (17). The transformation must be converted into individual joint positions, inevitably inducing (small) errors (17).

Factors that can affect accurate image processing and thus cause errors in measurements include:

1) Image contrast, intensity, and threshold
2) Focus and parallax
3) Vision calibration
4) Video noise/spatial quantization.

Most of the errors introduced by these factors were minimized by using a white ball for the target and a black background. The voltage threshold for the video system was set to the middle of the threshold scale, which was the optimal value as discussed in the Univision User's Manual (18). This provided the maximum contrast and intensity during the research. A ball also provided a circular image from any viewing angle even if the lens was not in perfect focus. Parallax induced errors were ignored since the VRS eventually aligned the camera above the center of the target. As stated earlier, the VRS was calibrated at an initial start position, thus minimizing any calibration errors. The video noise/spatial quantization errors are random and related to target size and the related image boundary perturbations and can be ignored due to the tolerance window.

Additionally, the vision system coordinates were scaled

by the scale factor determined during vision calibration. With the camera mounted to the arm, it was observed that the positive y direction of the camera's FOV was the negative y direction of the robot's coordinate system. It was also observed that the y component provided by the vision system had to be further scaled by a factor of 2/3's to enable accurate positioning of the vision-robot system. This additional scale factor was necessary to account for the difference in the x and y dimensions of each pixel element, which is dependent on camera type.

Finally, to maintain active PD control, the robot motion instructions for this task were broken up into no less than 10 continuous "back-to-back" move commands as opposed to 1 or 2 move commands. With less than 10 move commands, it was observed that the VRS would continuously overshoot the target. The overshoot problem resulted from the same errors as those mentioned above for the oscillations and also during the deceleration of the arm as it moved towards its desired final location. The deceleration induced errors are due to the P feedback control loop taking over control of the PUMA. Breaking up the move commands allowed smooth robot motion with minimal overshoot of the target.

In summary, the results of this subtask demonstrated the ability of the VRS to successfully implement closed loop, static look-and-move visual servo control techniques to track a white ball placed anywhere in the camera's FOV.

Subtask 3: Camera on Arm; Tracking and Acquiring

Subtask 3 continued the work accomplished in Subtask 2 by enabling the VRS to move towards its target as well as track it.

To provide the VRS with the knowledge of how far to move towards the target, the vision system was trained to recognize a target at a predetermined distance. To enable quick image processing, only the area of the target was used for identification (another reason for using a ball, because its two-dimensional image is always a circle). From the starting position defined in Subtask 2, it was determined that joint 6 of the PUMA 560 was approximately 350 mm from the floor. From this initial position, the VRS was trained to the white ball target. This initial target, referred to as Target 1, was assigned to the first of nine locations in the vision system's memory. Each subsequent location in memory was trained with the target 25 mm closer than the previous target. Therefore, if Target 1 was identified by the vision system, joint 6 would be 350 mm from the target and 150 mm from the target if Target 9 was identified.

Once the target was placed in the camera's FOV, the program **track.ball.scat** was executed. The program accomplished the same tasks as **track.targ** with the addition of having the vision system identify the target (based on the target's area). The program used this additional information to instruct the VRS to move towards the target. Robot motion

continued until the target was at the desired distance away from the VRS and within the 10 mm window discussed in Subtask 2.

As stated previously, the camera was mounted on the third joint of the PUMA 560. The PUMA's controller defined the robot's end position at its sixth joint with respect to the robot base. It also doesn't use just the first three joints to specify position when solving the inverse kinematic solution for the final joint locations. This was not a problem in Subtask 2, where only tracking of a target was required. However, in the subtask described here, it was found that the VRS lost camera sight (two-dimensional) of the target as the VRS approached the target. Therefore, an offset had to be programmed to account for the position of the camera as the camera approached the target. This offset correction enabled the VRS to successfully track and approach the target.

To further minimize the positional errors encountered during the VRS's approach to the target, the program instructed the VRS to converge towards the target in increments. This distance convergence approach proved successful and it was observed that the time for the VRS to approach and center itself above a stationary target, placed at one corner of the camera's FOV, took approximately 5.6 seconds. Figures 26, 27, and 28 show the sequential motion of the VRS and of the digital images on the video monitor as the VRS visually servos itself to the target.

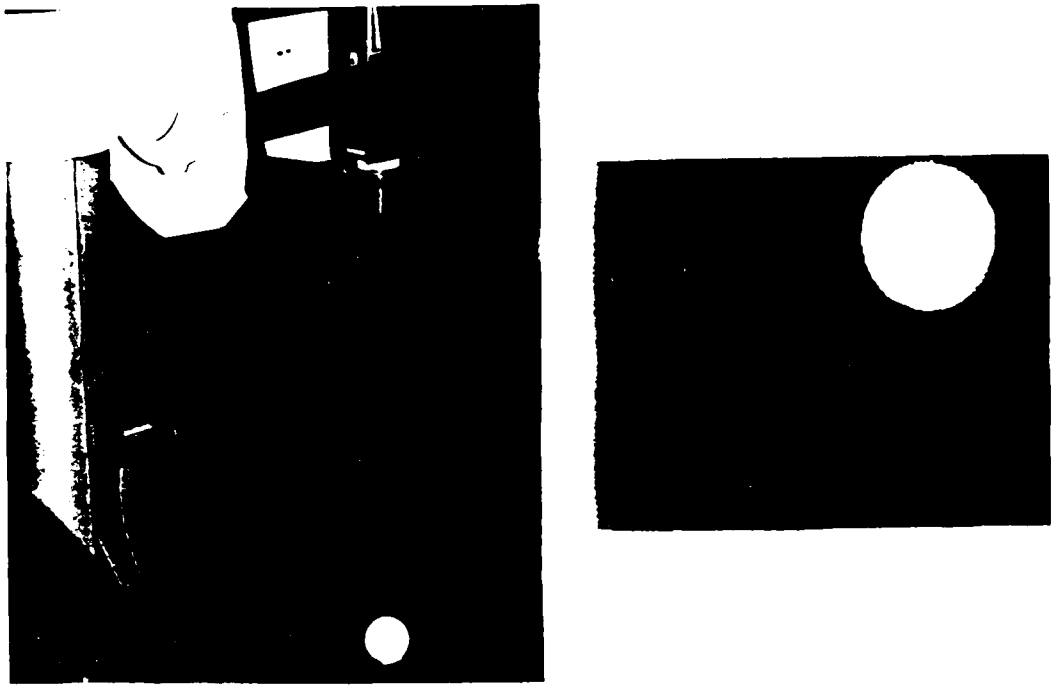


Figure 26. Starting Position of VRS

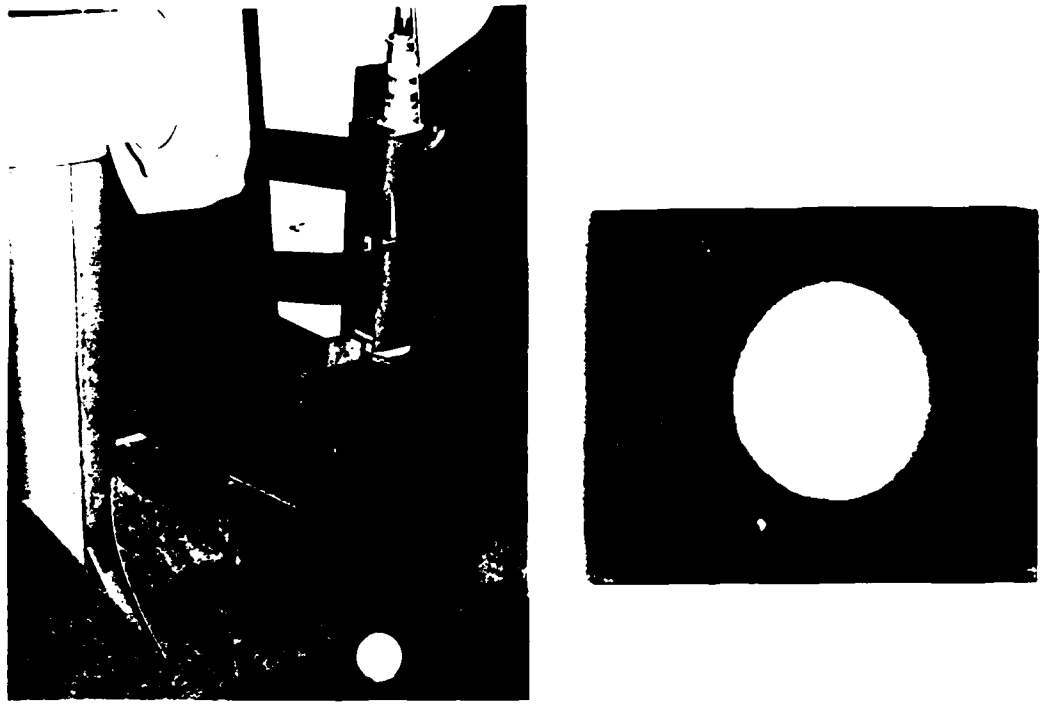


Figure 27. Intermediate Position of VRS

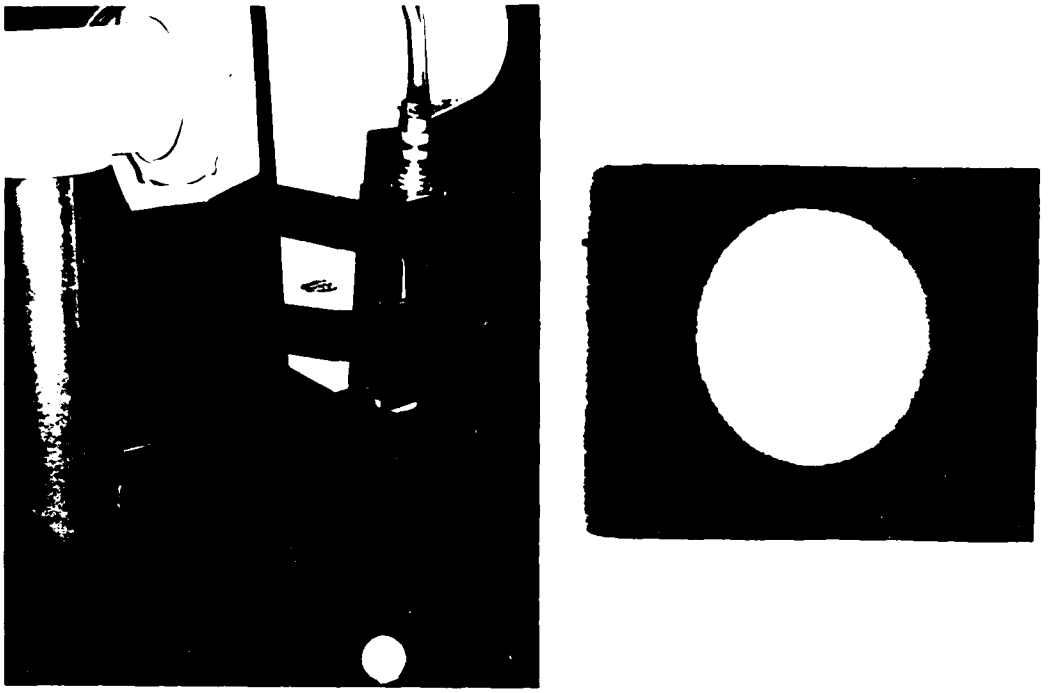


Figure 28. Final Position of VRS

Originally, the VRS was to move within 50 mm of the target. However, oscillations occurred whenever the target's image in the camera's FOV encompassed an area greater than or equal to half the viewing area of the camera. The oscillations were due to the video noise/spatial quantization and the transformation conversion errors discussed in Sub-task 2. The situation was resolved by allowing the VRS to approach the target no closer than 200 mm. Since the application of this thesis involves the refueling of an aircraft, the limited approach does not pose a problem. As shown in the artist's conception of the robotic refueler in

Figure 29, the camera was mounted to the refueler's third joint, with the final location of the camera positioned above and away from the refueling port, during the refueling operation.

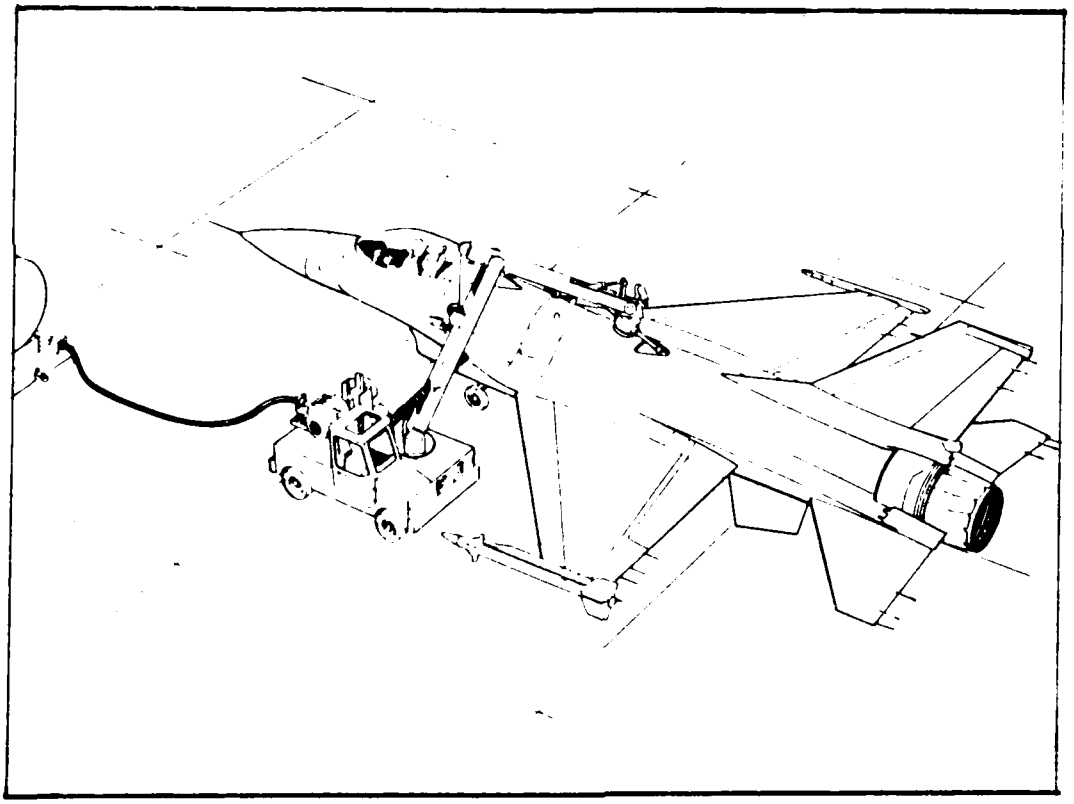


Figure 29. Robot Refueler (13:53)

Additional work accomplished in Subtask 3 enabled the VRS to become a tracker if the target started to move around in the camera's FOV. This was an unlikely event for an application involved with refueling a stationary aircraft, however, it could be useful for other applications where the target is allowed to move.

In summary, the results of this subtask demonstrated the ability of the VRS to successfully implement closed loop, static look-and-move visual servo control techniques to track and approach a white ball placed anywhere in the camera's FOV.

Subtask 4: Camera on Arm; Search

Subtask 4 completed the overall task of developing an integrated VRS that searched for and acquired the desired target.

Once the target was placed anywhere in a predetermined "search area", the program search was executed. The program instructed the VRS to search for the target. Once the target was found, Subtask 3 took over and visually servoed the VRS to the target.

The search pattern for this subtask was divided into 16 increments in the robot's WCS x-y plane. The increments were determined by measuring the window (perimeter) visible in the camera's FOV. For the specific CRT display used in this project, the size of the window was approximately 165 mm in the x direction by 135 mm in the y direction. An overlap

between the increments in the search pattern was added to the program, **search**, to ensure the entire search area was scanned. Increments of 80 mm were used when the VRS moved in the positive or negative x direction and in increments of 65 mm in the positive or negative y direction. The entire search covered an area of approximately 400 mm by 380 mm, this was limited by the work envelope of the PUMA. The time required to find the target depended on where the target was placed in the search area. The search time ranged from 0.2 seconds directly below the VRS to 22.0 seconds when the target was placed in the center of the search area. Figures 30, 31, and 32 show only three of the sequential motions of the VRS as it searched for the target and then servoed towards it.

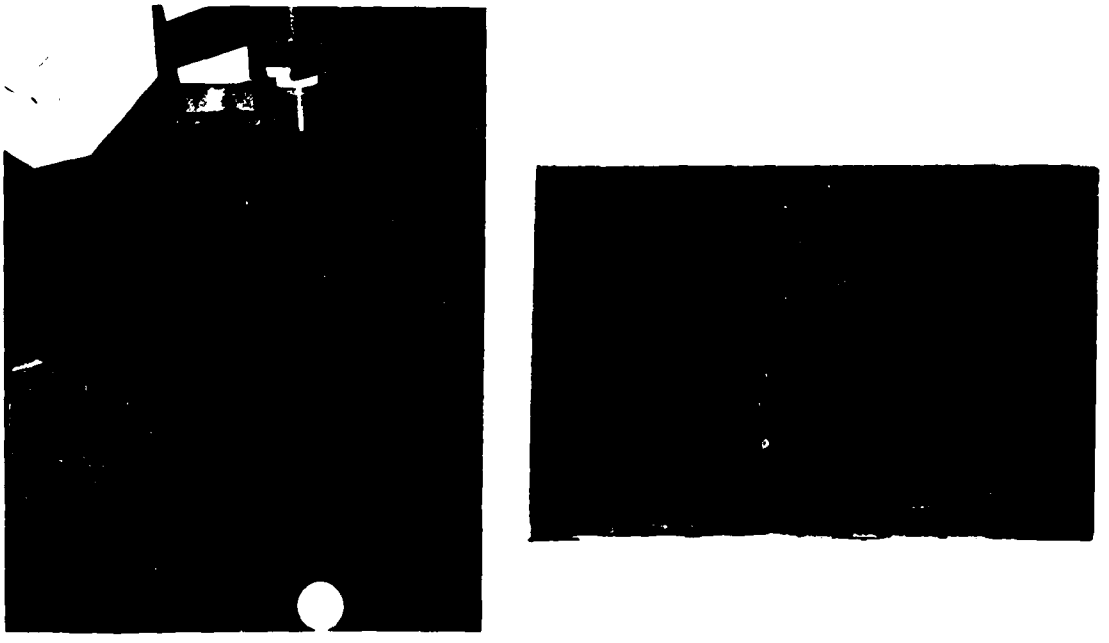


Figure 30. Starting Position of VRS

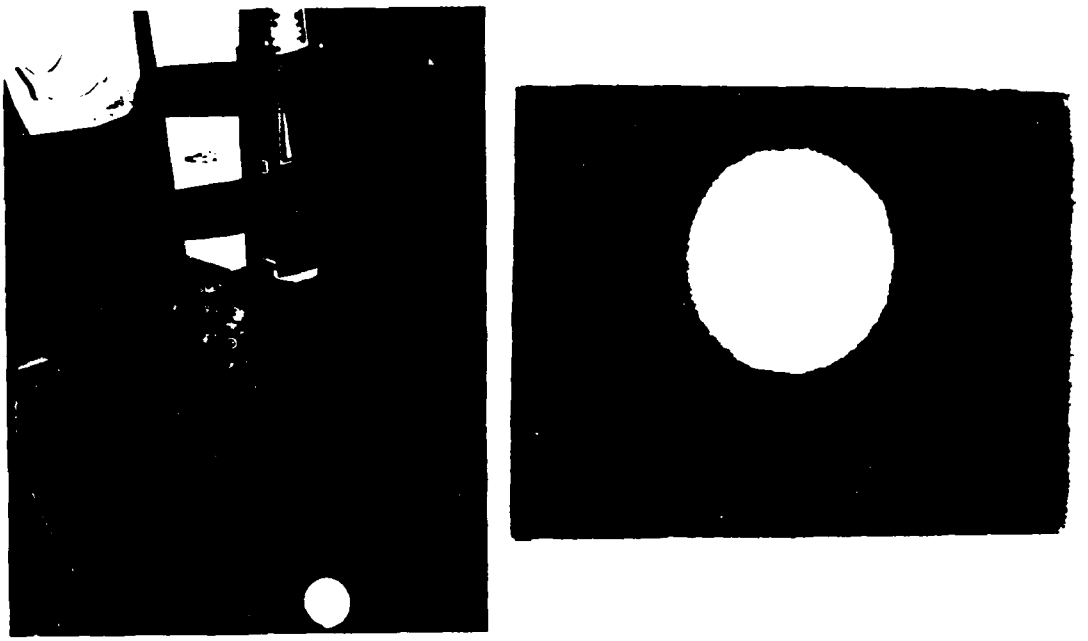


Figure 31. Intermediate Position of VRS

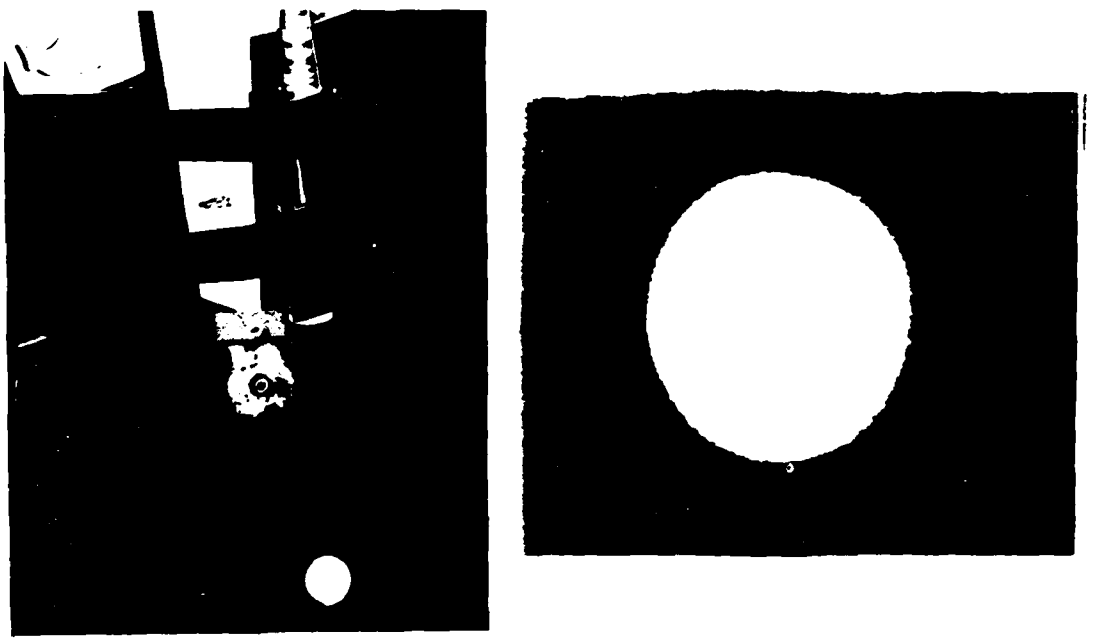


Figure 32. Final Position of VRS

During testing of this subtask, it was discovered if the target's initial position in the search area was at a distance greater than that for which the vision system was trained, the target could not be identified. This did not prevent the vision system from acquiring the target since acquisition was effected by previously existing detection and centroid computation inherent to the vision system. When the search mode was initiated, the vision system decided that a target was in the camera's FOV when its area (in pixels) was greater than the smallest acceptable blob area set by the user. For this research, the minimum area was arbitrarily set to 10 pixels. To enable target verification following acquisition, the VRS approached the target in 25 mm increments until the vision system could identify the target. Once the target was successfully identified, the VRS visually servoed itself to the target as in Subtask 3.

Additional work accomplished in Subtask 4 enabled the VRS to automatically restart the search if the target was ever removed from the camera's FOV. This was an unlikely event for an application involved with refueling a stationary aircraft, however, it could be useful for other applications where the target is allowed to move.

In summary, the results of this subtask demonstrated the ability of the VRS to search for a target and successfully implement closed loop, static look-and-move visual servo control techniques to track and approach the target.

The combined results of the four subtasks described above achieved the desired goal of this research. The next section discusses the results from additional research which implemented a closed loop, dynamic look-and-move visual control scheme in a tightly controlled environment.

Dynamic Look-And-Move System

The task of developing a closed loop, dynamic look-and-move visual servo control system was an extension of the work accomplished in Subtask 3 of the Static Look-and-Move Task. However, dynamic visual servo control was implemented when the VRS moved towards the target.

Once the target was placed near the center of the camera's FOV, the program **track.targ.dcat** was executed. From all indications, the VRS performed in the same manner as in Subtask 3. However, a difference existed in the movement of the VRS towards the target. In this task, the robot motion instructions were broken up into only 2 continuous "back-to-back" move commands as opposed to the 10 move commands in Subtask 3. Also robot motions were instructed to occur at half the speed of those in Subtask 3. Under the concept of procedural motion, a picture was taken during the second move command in the program. The data from the image provided an in-course correction to the VRS, thus allowing a smoother approach to the target.

A limitation existed in this task because of the oscillation problem discussed in Subtask 2. Therefore, the

target had to be placed near the center of the camera's FOV for the VRS to work correctly. It was observed for this task, that the time for the VRS to approach and center itself above a stationary target took approximately 6.1 seconds. This is a slower time than Subtask's 3, however, as stated above, robot motions were at half the speed. When Subtask 3 was executed at the slower speed, it took approximately 7.8 seconds for the VRS to approach and center itself above the stationary target. Figures 33, 34, and 35 show the sequential motions of the VRS as it dynamically servoed itself towards the target.

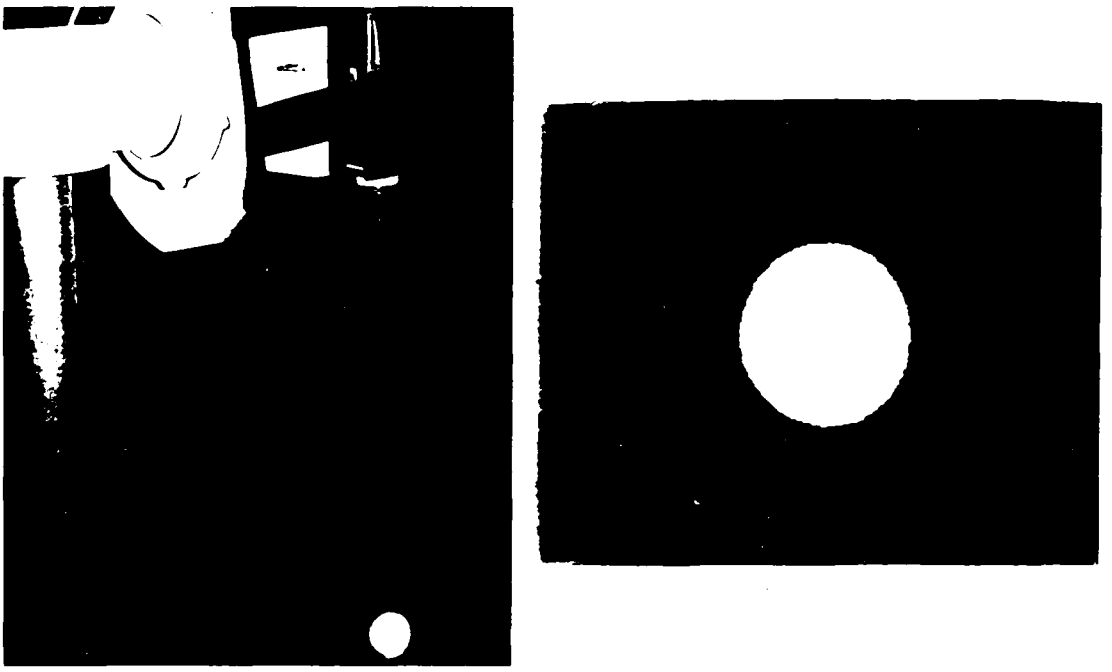


Figure 33. Starting Position of VRS

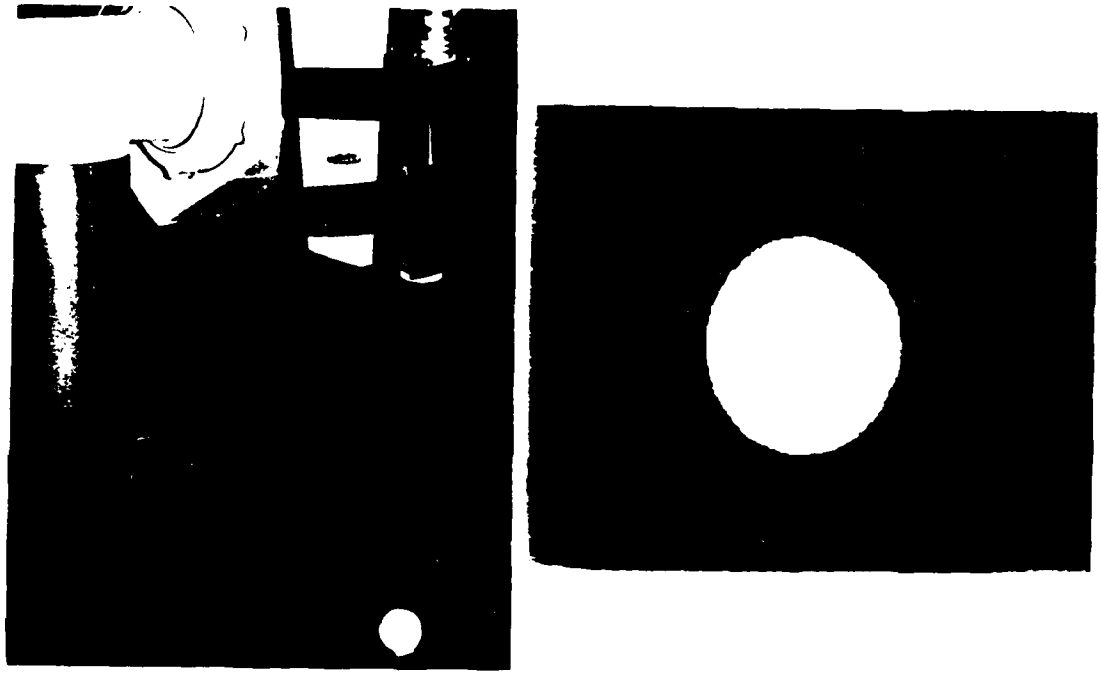


Figure 34. Intermediate Position of VRS

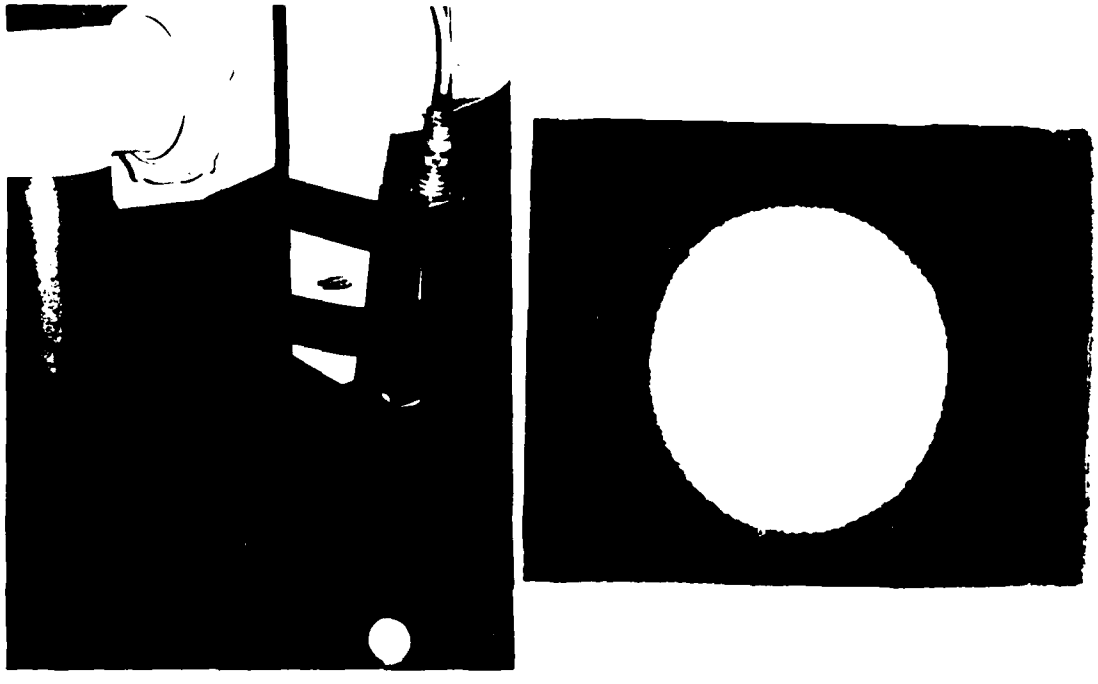


Figure 35. Final Position of VRS

This completes the presentation of the results of the research effort, the conclusions and recommendations are now presented.

V. Conclusions and Recommendations

In this research, visual servo control schemes were designed and integrated for a PUMA 560 robot arm which derived its visual information from a MIC vision system. The vision system's camera, with fixed focused lens, was rigidly mounted to the third joint of the PUMA. The two control schemes researched involved static and dynamic, look-and-move visual control techniques. These techniques successfully implemented and demonstrated the concept of using a robot, equipped with a vision sensor system, to ground refuel an aircraft.

Conclusions

Two conclusions were drawn from the success of this research effort. First, the results presented in Chapter IV proved that the closed loop, static look-and-move visual servo control vision-robot system (VRS) could successfully find and move towards a well defined target, a white ball, placed in a designated search area. The vision system was used to identify the target and determine its two-dimensional centroid position with respect to the robot's end-effector. Because one camera with a fixed focus lens was used, a crude method was developed to determine the distance from the target to the camera. As the camera approached the target, the target's image grew larger with respect to the camera's

field of view (FOV). Therefore, by training the target to the vision system at different distances, the vision system could determine, by which target was identified, the distance from the target to the VRS. This distance measurement along with the centroid position measurement provided three-dimensional information about the target. This method worked adequately for a well defined target like a ball, however, an improved method would be required for a more complex target.

Finally, the results from Chapter IV also proved closed loop, dynamic look-and-move visual servo control techniques could be successfully implemented. However, due to slow computer processing and the inability to implement parallel processing, limitations existed with this control technique. These limitations were overcome by using procedural motion. During movements of the VRS over large distances, dynamic visual control could only be successfully implemented. This situation only occurs when the VRS moves towards the target. Thus, although limitations existed in the closed loop, dynamic look-and-move visual control technique, excellent results were achieved when the target was initially placed near the center of the camera's FOV.

Recommendations

The work accomplished in this research effort provides a solid foundation on which to continue work in areas of robotics and vision. The following are recommendations for continued research in areas related to this thesis:

- Implement the visual servo control techniques presented in this thesis on the Robotics and Automation Laboratory (RAL) Hierarchical Control System (RHCS) (9). The RHCS will be the primary system used for research with the PUMA 560 at the Air Force Institute of Technology (AFIT). The RHCS also allows the user to specify which joints are controlled. This alleviates the problem associated with solving for position using inverse kinematics.

- Using the RHCS, develop a parallel processing capability, to enable real-time, closed loop, dynamic look-and-move visual servo control for all robot motions.

- Decrease the computer processing time involved during inter-computer communication by developing a quicker, more efficient method for transferring messages between the two computers. Almost twice as much time was spent on communication than image processing (about 0.5 sec. for communication and 0.3 sec. for image processing of the ball target). A total processing time of less than .1 seconds is required for real-time visual servo control (14:214).

- Implement various image processing techniques currently in existence at AFIT to enable the acquisition of more visually complex targets against an arbitrary background.

- Implement a more efficient method to determine the distance to a target. Various techniques include using the following: auto focus camera; stereo vision; or sonar.

- Remount the camera on joint three at an angle, such that the center of the camera's FOV will be the at the center of the robot's sixth joint, only translated along the z axis. This will alleviate the requirement for programming an offset for the camera location.

- Obtain a camera which weighs less then 5 kgs and mount it to the sixth joint. This would be valuable to applications where the target's orientation is important.

Appendix A

Equipment and Interface Descriptions

This Appendix contains brief descriptions of the hardware and software used in this research effort. Part 1 contains sections of Chapter 1 from (16) which is an introduction and description of the PUMA Mark II, 500 Series, Robot System (in particular the PUMA 560). Part 2 is a copy of the prefatory information from (18) which provides an introduction and description of the Machine Intelligence Corporation's (MIC) Vision System. Finally, Part 3 is a copy of (19) which contains descriptions of the interface software between the MIC Vision System and Unimations' VAL II programming language.

A miniature Table of Contents follows:

Table of Contents

Part 1: Description of PUMA 560 (16)

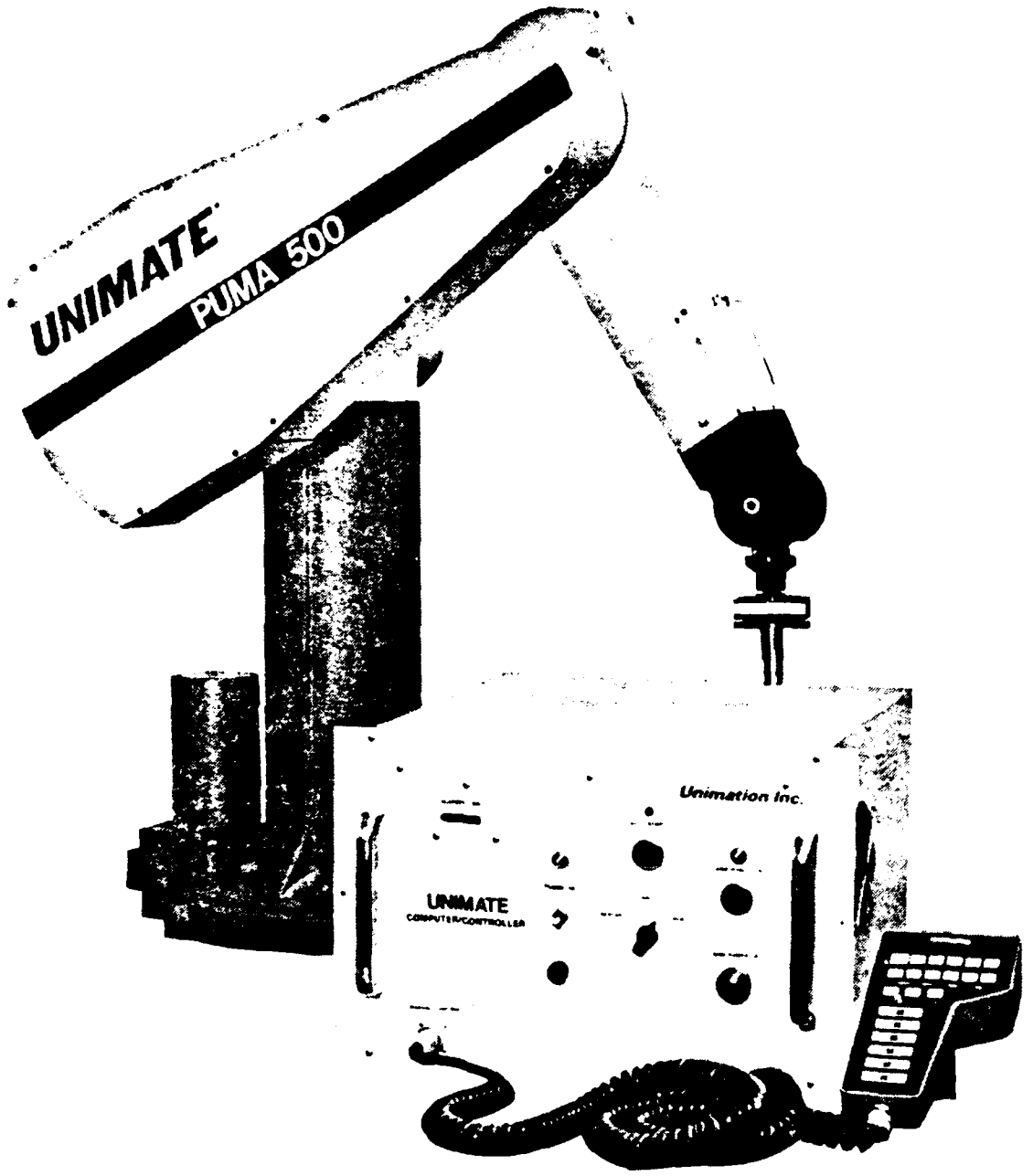


Figure 1-i. PUMA Mark II, 500 Series, Robot System

CHAPTER 1

INTRODUCTION AND DESCRIPTION

1-1. INTRODUCTION

The Unimate PUMA Mark II, 500 Series Robot (Figure 1-1), is an advanced computer controlled robot arm system, manufactured by Unimation Incorporated, A Westinghouse Company, of Danbury, Connecticut.

This equipment manual is to be used with VAL II and VAL PLUS operating systems. It contains the introduction and description, installation, operation, maintenance, troubleshooting, spare parts list, appendixes, and a glossary.

Instructions on how to write and execute programs for the PUMA Mark II system is given in the "User's Guide To VAL II," Part No. 398T1, or "User's Guide to VAL PLUS," Part No. 398AC1.

Note

Appropriate programming manuals will be sent to users of VAL II or VAL PLUS operating systems.

The mechanical and electrical drawings are contained in two separate manuals as follows:

PUMA Mark II, 500 Series, Mechanical Drawing Set, 394AB1 (for VAL II or VAL PLUS operating systems).

PUMA Mark II, 500 Series, Electrical Drawing Set, 394AC1 (for VAL II or VAL PLUS operating systems).

Please note that personnel responsible for programming and operating the PUMA are expected to attend the training course given by the Unimation Technical Training Department. They must also have complete understanding of all information given in this manual and the User's Guide. The combined information from the training program and manuals will ensure safe and efficient operation of the PUMA robot.

1-2. DESCRIPTION - PUMA SYSTEM AND SOFTWARE

1-2-1. PUMA SYSTEM. The PUMA robot system is designed to adapt to a wide range of applications. The basic units are the teach pendant, software, controller, peripherals, and robot arm.

The system software that controls the robot arm is stored in the computer memory located in the controller, which also houses the operating controls for the system.

To teach the robot arm, either of two procedures can be used. The teach pendant may be used to manually direct the movements of the robot arm through each step of the task. These steps are recorded and then stored in the computer memory. The second method is to write a program using software instructions. Position data and software programs are entered into the computer memory through the peripheral terminal keyboard or through the teach pendant.

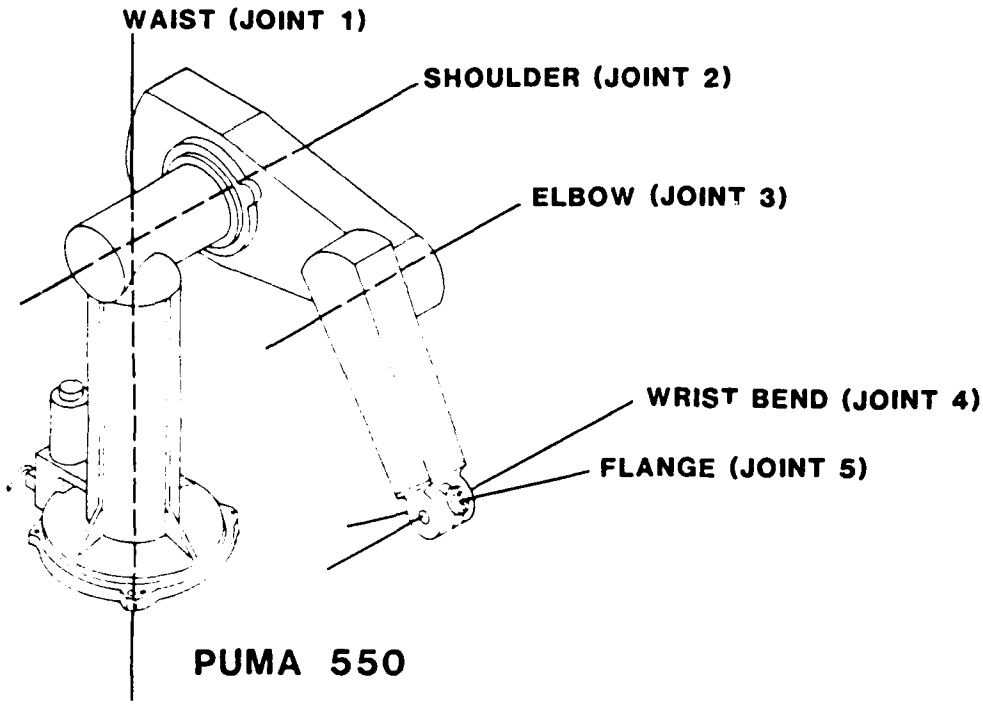
In either case, the controller transmits the instructions from the computer memory to the arm. Position data obtained from incremental encoders and potentiometers in the robot arm are transmitted back to the controller/computer to provide closed-loop control of the arm motions.

The programs may also be stored external to the controller on a floppy disk, a small flexible disk coated with a magnetic medium that provides a permanent record of the program. The floppy disk unit is an optional peripheral.

Additionally, the PUMA can be programmed to interact with its environment by using external input and output signals. External input (WX) signals can be used to halt a program, branch to another program step, or branch to another subroutine. For example, an external input signal can stop the program when safe operation is impaired. External output (OX) signals allow the PUMA system to control other equipment related to its work environment. If the WX/OX features are used, at least one optional I/O module is required.

The robot arm executes the instructions transmitted to it by the controller. The arm assembly is capable of rotational movement around six axes (five axes for the PUMA 550). The axes of rotation are shown in Figure 1-2.

1-2-2. SOFTWARE. The PUMA system operates on a high level language called VAL II or VAL PLUS. In addition to being a sophisticated programming language developed for assembly, it is a complete robot control system.



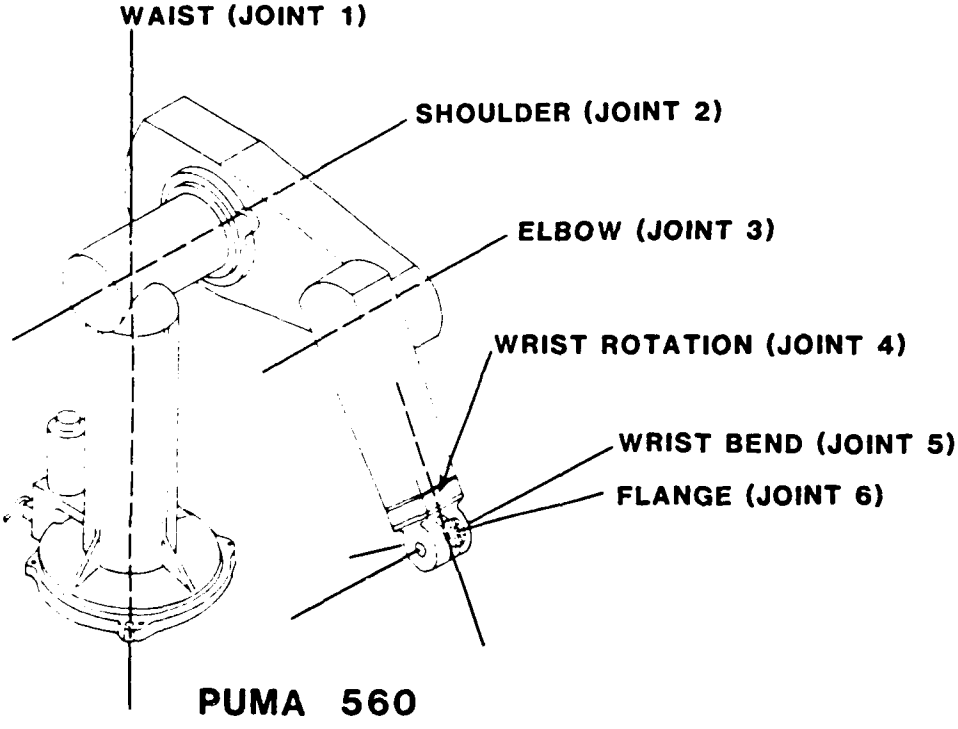


Figure 1-2. Robot Arm: Joint Identification

The VAL II or VAL PLUS programming language consists of a full set of English language instructions for teaching and editing. However, the VAL II control system has additional capabilities that readily communicate with other computer based systems such as vision and force sensors, as well as with supervisory computer systems.

Robot programming can be accomplished by the teach-by-showing method using the teach pendant or by keyboard entry. The full programming versatility can be realized only through the keyboard.

In either programming method, all taught points are stored as transformations (referenced to a coordinate system fixed relative to the stationary robot base), as precision points (position information stored in the form of joint angles), or as compound transformations (point locations referenced to previous locations as a measurement from a Cartesian coordinate system fixed relative to the tool mounting surface).

1-3. DESCRIPTION - CONTROLLER

The controller is the master component of the electrical system. All signals to and from the robot pass through the controller and are used by it to perform real-time calculations to control arm movement and position (Figure 1-3). (Peripheral components are discussed in paragraph 1-4.)

Operating controls and indicators are located on the front and top panels of the controller. Connections for the robot arm, terminal, floppy disk drive, I/O modules, and accessory are located on the controller rear panel. Software is stored in the computer memory, located in the controller. The software interprets the operating instructions for the robot arm, and the controller transmits these instructions from the computer memory to the arm. From incremental encoders and potentiometers in the robot arm, the controller/computer receives data about arm position. This provides a closed loop control of arm motions.

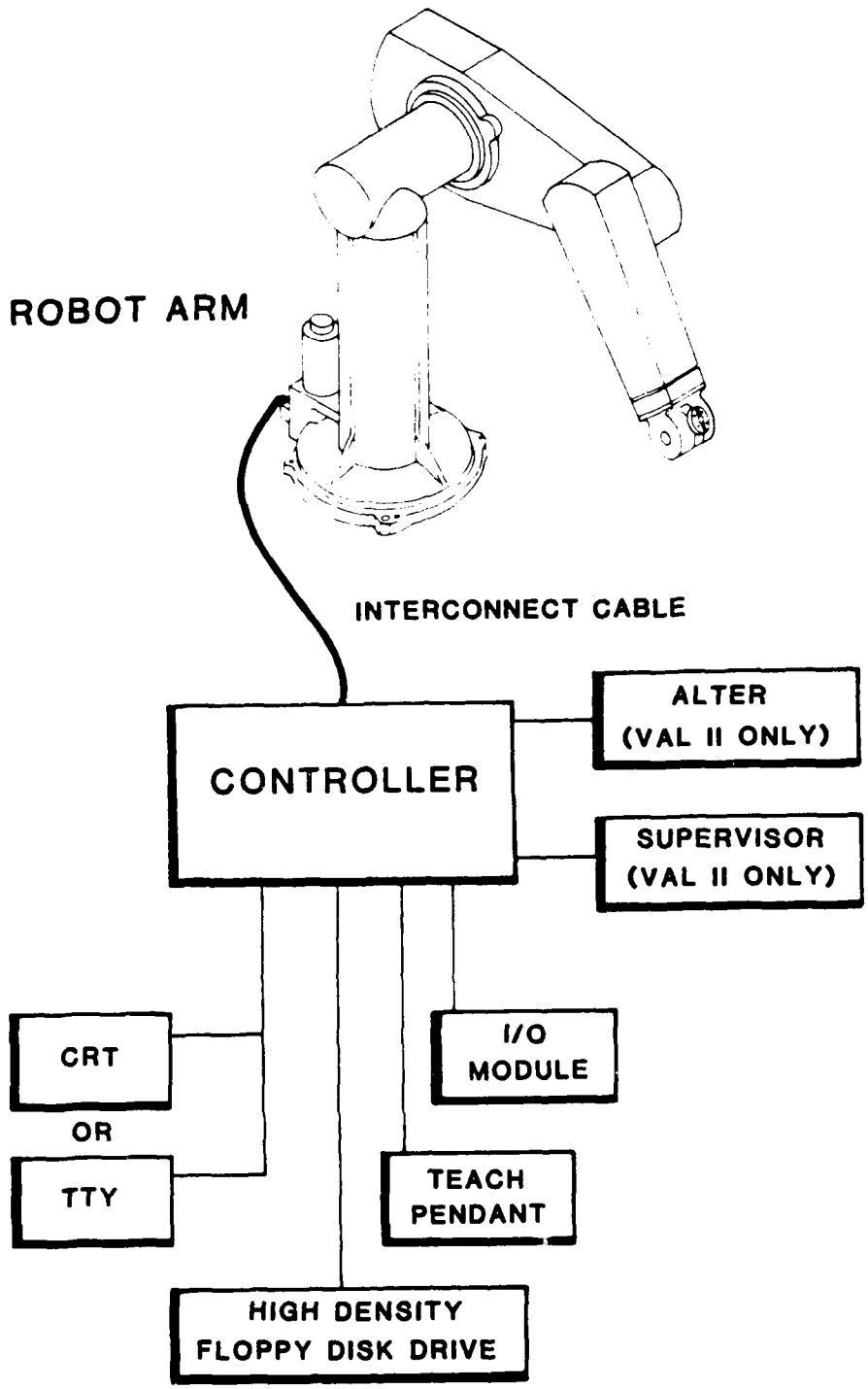


Figure 1-3. PUMA System: Information Flow

The main internal components of the controller are listed below. Their locations are shown in Figures 1-4 and 1-4A, and described in paragraphs 1-3-1 through 1-3-9.

a. DEC LSI-11 computer (DEC and LSI-11 are trademards of Digital Equipment Corporation). (See Note.)

b. DLV11-J quad serial interface boards (two). (See Note.)

c. CMOS board. (See Note.)

d. "A" interface board (with boot chips)

e. "B" interface board (with clock)

f. Digital servo boards (six)

g. Power amplifier assemblies (one major and one minor)

h. Power amplifier control board

i. Input/output interface board (not shown)

j. Power supplies (two)

k. High power function board

l. Arm cable board

Note

1. The VAL PLUS operating system contains an LSI-11/23 processor, 48K CMOS, (additional memory optional), and one quad serial board.

2. The VAL II operating system contains an LSI-11/73 processor, 64K CMOS (additional memory optional), and two quad serial boards.

1-3-1. LSI-11 COMPUTER SYSTEM. The LSI-11 system is a standard DEC unit containing a processor, memory and communication boards. System software and user programs are stored in a Complimentary Metal Oxide Semiconductor (CMOS) nonvolatile memory. Communication between processor and other components is accomplished as follows:

a. A four-port asynchronous serial interface board (DLV11-J) links the processor, terminal, teach pendant, and high-density floppy disk. A second DLVII-J board is available to serve as the link between the system processor and a Supervisory processor system. This same board provides communications along the real-time trajectory modifications link known as ALTER.

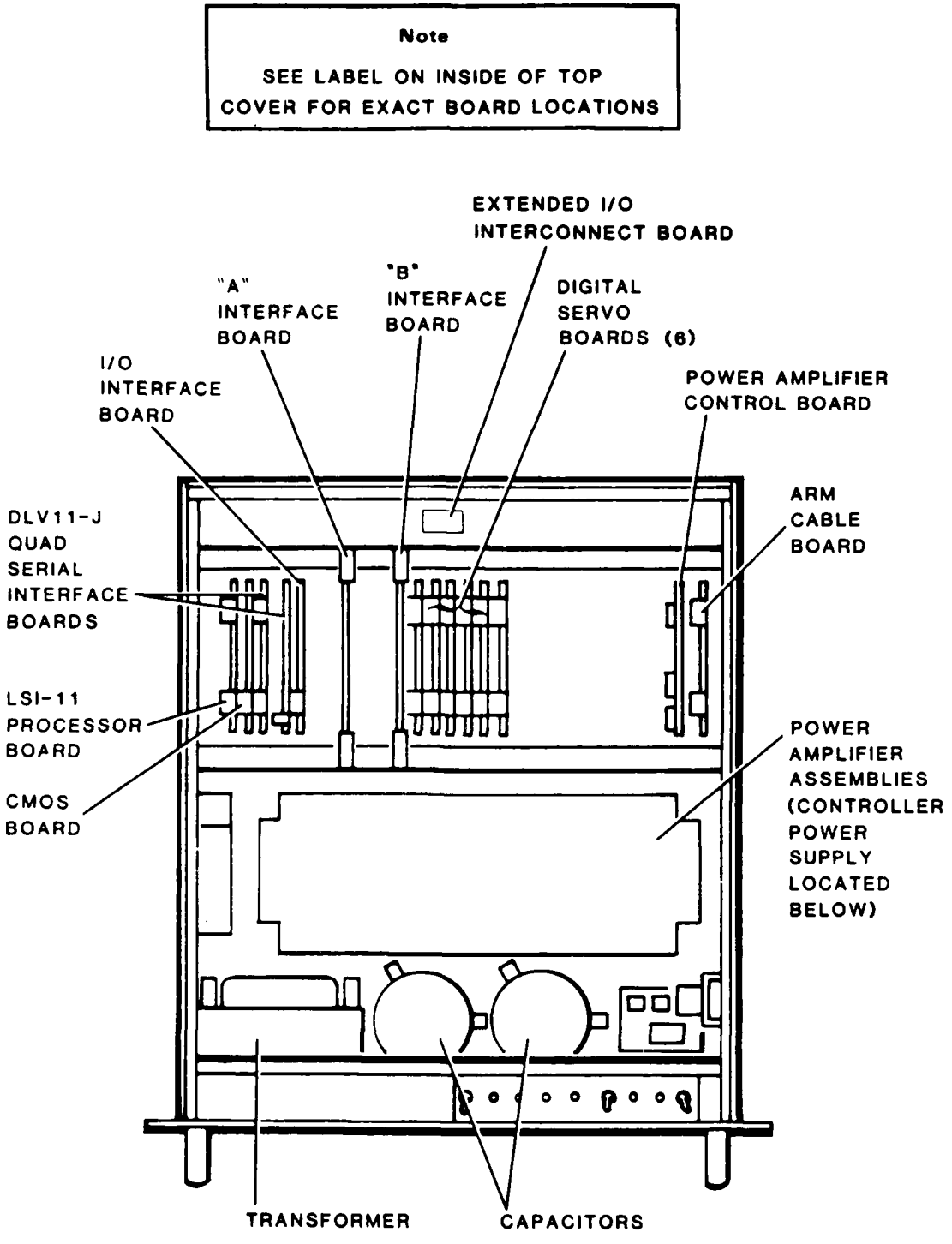


Figure 1-4. Controller: Board Location (Top)

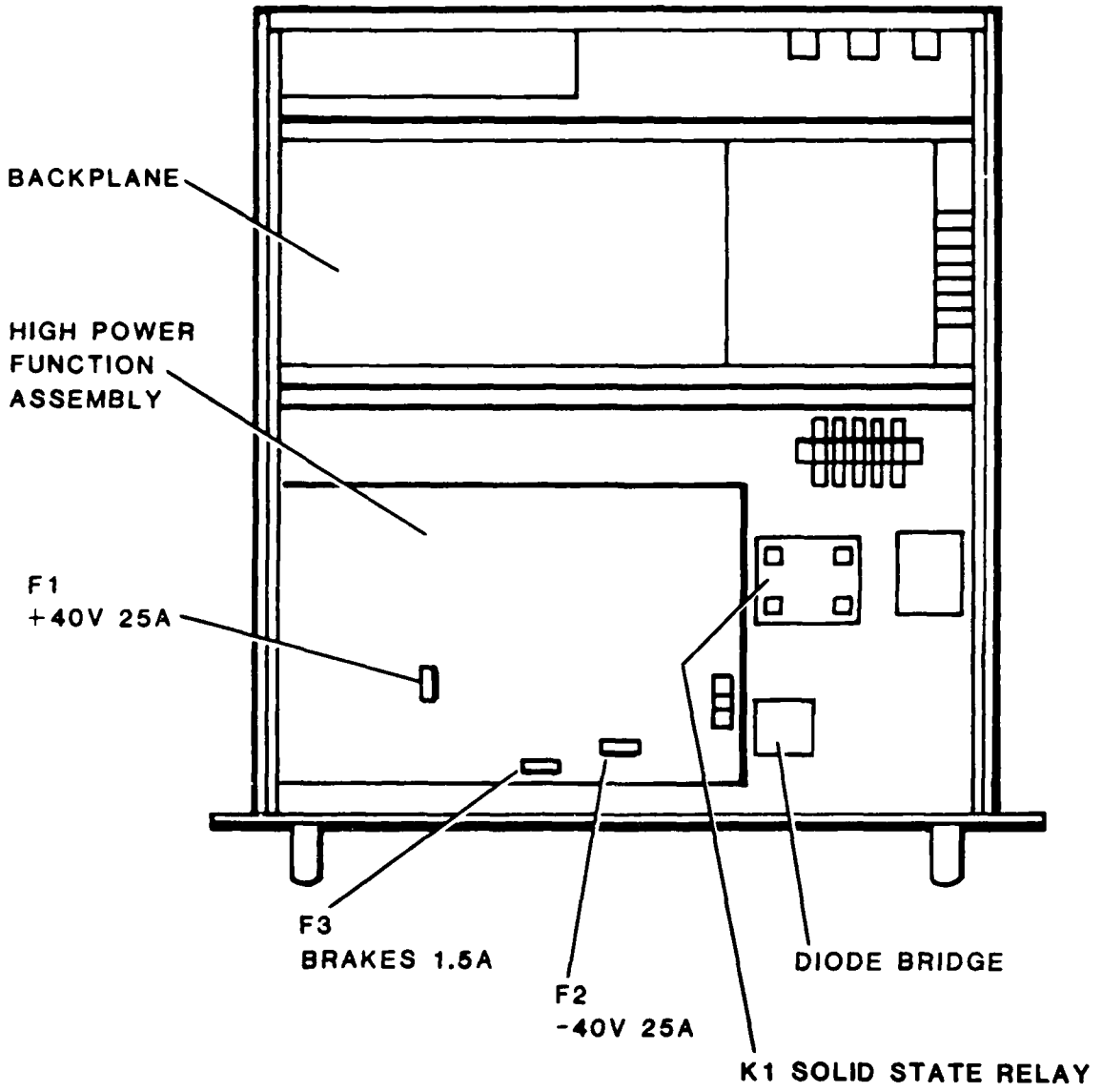


Figure 1-4A. - Controller: Board Location (Bottom)

b. The "A" interface board provides a communications link to the "B" interface board (paragraph 1-3-2). The analog-to-digital converter that reads the potentiometers is located on the "A" interface board.

1-3-2. "B" INTERFACE BOARD. The "B" interface board links the LSI-11 system through the "A" interface board, to the servo drive side of the control system. Command signals sent by the LSI/11 are interfaced to a servo bus by this board. Once the commands or data have been interpreted and acted upon, return signals are sent to the LSI-11 through the "A" interface board. Multiplexer circuitry on this board provides analog-to-digital conversion of motor potentiometer feedback and clock/terminator functions.

1-3-3. DIGITAL SERVO BOARDS. There are six digital servo boards, one for each joint. Each joint is controlled by a separate microprocessor. Position inputs from the computer, as dictated by the LSI-11 calculations, are fed into the digital servo board every 28 milliseconds. Digital information is fed to a digital-to-analog convertor to generate the analog signal required to drive the DC servomotors. Communications from each digital servo board through the "A" and "B" interface boards, to the LSI-11, are controlled by a protocol using interrupt servicing routines.

1-3-4. POWER AMPLIFIER ASSEMBLIES. For each joint, the output from the digital servo board is fed to a power amplifier board, where it is amplified to voltage and current levels high enough to drive the servomotors. Outputs from the power amplifier boards are fed to the arm through the connecting cable via the arm cable board assembly (paragraph 1-3-10).

1-3-5. POWER AMPLIFIER CONTROL BOARD. The power amplifier control board contains the following:

a. Six joint error indicator lights. When lit, these lights indicate an overcurrent hardware failure.

b. A joint reset pushbutton. Pressing pushbutton resets joint after error has been corrected.

c. Two high temperature indicator lights, Th1 and Th2. When lit, these lights indicate the presence of high temperature at the power amplifier assemblies.

1-5. DESCRIPTION - ROBOT ARM

The robot arm is the mechanical component of the system incorporating 6 degrees of freedom (5 degrees on PUMA 550), each controlled by a DC servomotor. It is sufficiently flexible to be taught a wide variety of tasks. Each member of the robot arm is connected to another member at a joint, much like a human arm and torso. Through each joint passes one or more axes around which the members of the arm rotate.

The members of the robot arm are shown in Figure 1-15; they are the trunk, shoulder, upper arm, forearm, wrist, and gripper. The robot arm members contain the various servomotors and gear trains.

To achieve maximum strength with minimum weight, the upper arm and forearm are of monocoque contruction. Monocoque is a method of construction that uses the covering plates or "skin" of an assembly to carry all or part of the stresses.

The axes of rotation and the ranges of rotation are shown in Figure 1-16 and described in Table 1-3.

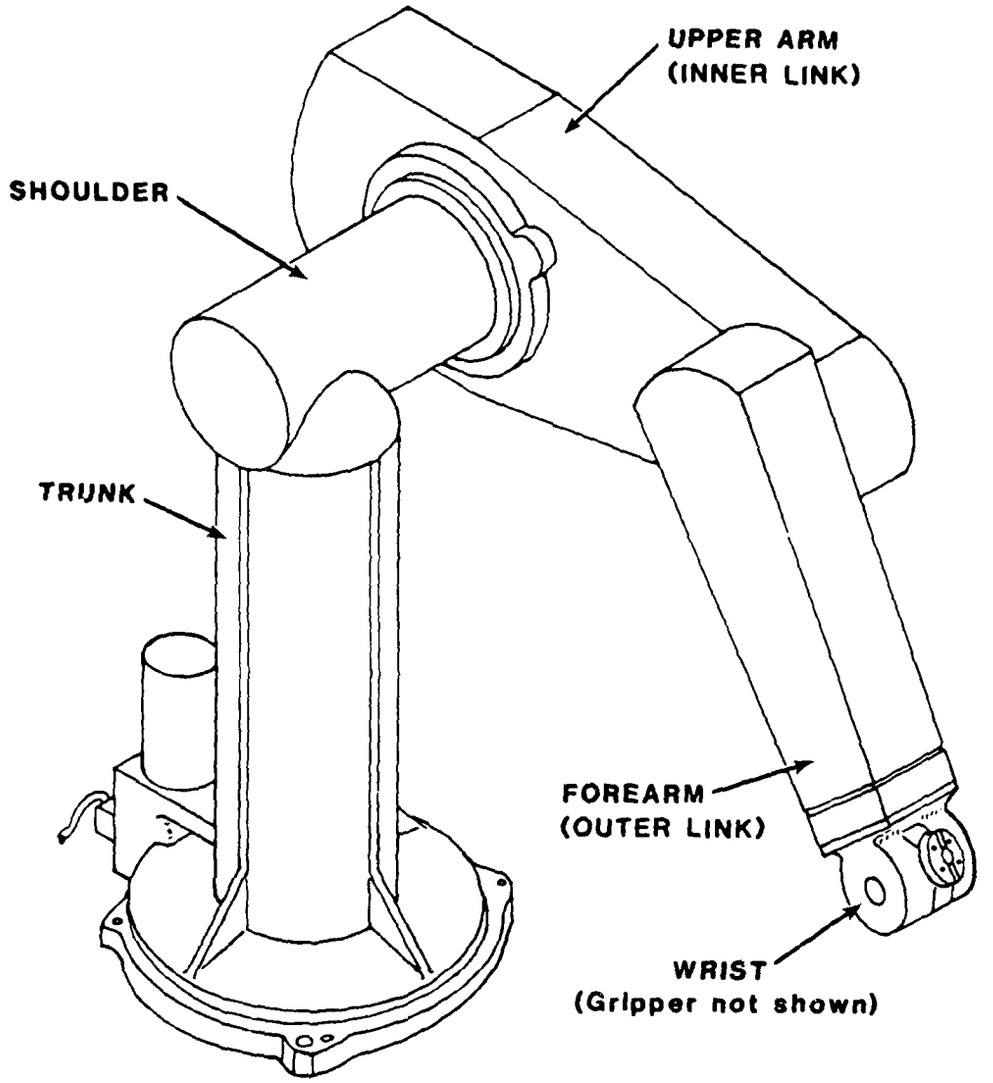
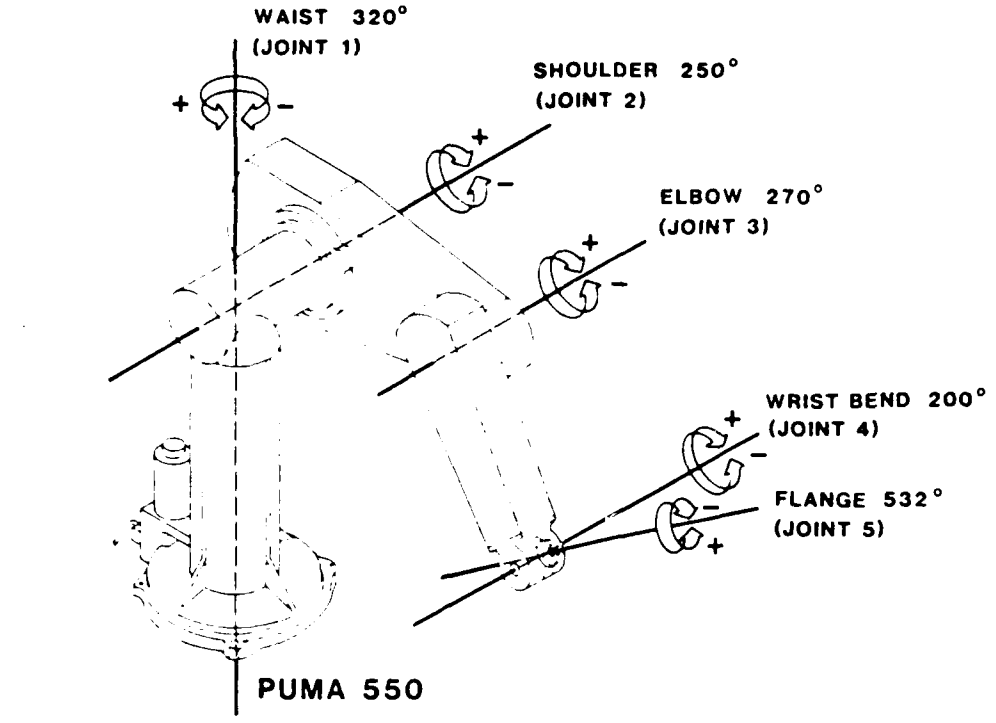


Figure 1-15. Robot Arm: Member Identification



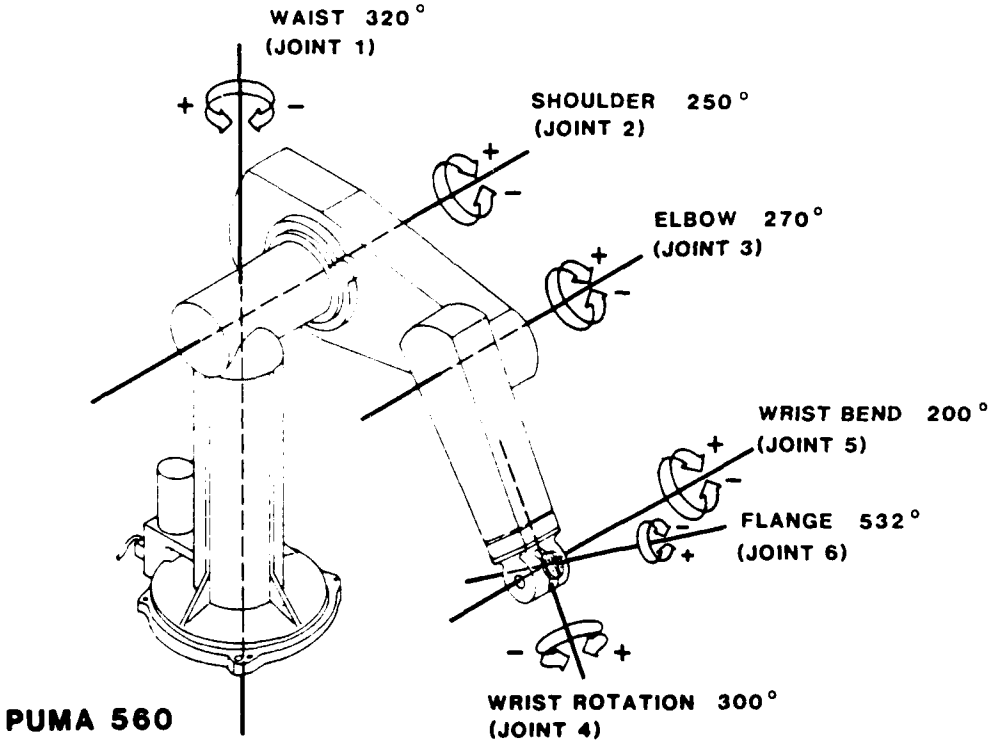


Figure 1-16. Robot Arm: Degrees of Joint Rotation

Table 1-3. Robot Arm Axes

JOINT	DESCRIPTION
Waist - Joint 1	Joint 1 axis is perpendicular to the mounting plane of the PUMA and coincident with the centerline of the trunk.
Shoulder - Joint 2	Joint 2 axis is perpendicular to and intersects Joint 1 axis; it is coincident with the centerline of the shoulder.
Elbow - Joint 3	Joint 3 axis is parallel to the Joint 2 axis.
Wrist - Joint 4 (PUMA 560 only)	Joint 4 axis is perpendicular to and intersects Joint 5 axis.
Wrist - Joint 5 (Joint 4 on PUMA 550)	Joint 5 axis (Joint 4 on PUMA 550) is parallel to the Joint 2 and 3 axes
Wrist - Joint 6 (Joint 5 on PUMA 550)	Joint 6 axis (Joint 5 on PUMA 550) is perpendicular to and intersects Joint 5 axis (Joint 4 on PUMA 550); it is coincident with the centerline of the gripper mounting flange.

Each member of the arm assembly is driven by a permanent-magnet DC servomotor driving through its associated gear train. Each motor in the PUMA robot arm contains an incremental encoder and a potentiometer driven through a 116 to 1 gear reduction. The proper functioning of the PUMA requires control of the position and the velocity of each joint of the robot arm.

For a servo-controlled robot system, position must be measured relative to a known initial, absolute position. The potentiometers, incorporated in the motor, are used to initialize the PUMA; that is, to establish its absolute position. The initializing procedure must be done each time the PUMA system is powered up. (Refer to paragraph 3-4.)

The incremental encoders are mounted on the shaft of each motor and provide position change and velocity signals for the servo system. Position change signals are read from the encoders, and velocity signals are calculated. Approximately 32 times during each 28-millisecond window of the digital servo system, the signals from the encoders are compared to the calculated position and any necessary correction signals are generated.

The servomotors for the major axes (Joints 1, 2, and 3) are equipped with electromagnetic brakes. These brakes are activated when power is removed from the motors, thereby locking the robot arm in a fixed position. This safety feature removes the risk of injury or damage that could result from the arm collapsing if power is accidentally removed.

Power for the motors is supplied through the cable connecting the robot arm and the controller. Feedback signals from the incremental encoders and potentiometers are also carried by this cable.

1-5-1. WAIST - JOINT 1. The motor for Joint 1 is located outside the trunk, on the base casting. The gear train is shown in Figure 1-17. For clarity of description, the gear train is divided into three sections (Table 1-4).

Table 1-4 Joint 1 - Gear Train

COMPONENT	DESCRIPTION
Motor shaft spur pinion and two gears	A spur pinion cut into the end of the motor shaft drives two spur gears simultaneously. **Note** Pinion: Of the two gears that run together, the pinion has the smaller number of teeth. Gear: Of two gears that run together, the gear has the greater number of teeth.
Idler shafts	The two spur gears transmit power through idler shafts that have different torsional rigidities. In one, torsional rigidity is high. This prevents any appreciable twisting of the shaft around its long axis (called windup). The other shaft has a lower torsional rigidity designed for a predetermined amount of windup to preload the entire gear train eliminating backlash.
Two pinions and the bull gear	Two more pinions, at the end of the idler shafts, drive the bull gear.

1-6. SPECIFICATIONS

Table 1-5 contains the specifications for the PUMA system.

Table 1-5. Specifications

ITEM	SPECIFICATIONS
Robot Arm	
Axes	Six revolute axes (5 axes for PUMA 550)
Clearance Required	Spherical volume with shoulder at center: 0.92 m (36.2 in.) radius
Mounting Orientation	Must be mounted vertically. Base must be level within 1 degree. Vertical can mean right side up or upside down --- NOT horizontal.
Weight	534 N (120 lb)
Drive	Electric DC servomotor
Mounting Surface for Gripper	Four 10-24 holes on a 0.041 m (1.625 in.) diameter bolt circle
Maximum Inertia Load (Including Standard Gripper: Toggle Type - P/N 510-9013; Parallel Type - P/N 510-0100)	
Wrist Rotation - Joint 4 (PUMA 560)	NOT to exceed 5.7 in.-oz-sec^2 (e.g., 5.5 lb steel, 4 in. dia. disk mounted 5 in. from wrist rotation axis).
Wrist Bend - Joint 4 (PUMA 550) Joint 5 (PUMA 560)	NOT to exceed 5.7 in.-oz-sec^2 (e.g., 5.5 lb steel, 4 in. dia. disk mounted 5 in. from wrist bend axis).
Flange Rotation Joint 5 (PUMA 550) Joint 6 (PUMA 560)	NOT to exceed 0.5 in.-oz-sec^2 (e.g., 5.5 lb steel, 4 in. dia. disk centered on axis of rotation).
Static Force at Tool	58 N (13.0 lb) maximum

Table 1-5. Specifications (Cont)

ITEM	SPECIFICATIONS
Position Repeatability	±0.1 mm (±0.004 inches) within primary work envelope (as measured to center of the tool mounting flange).
Gripper Control	Computer controlled, pneumatic 0.003 m^3/s at 710 KPa (6 ft^3/min at 100 psi)
Tool Acceleration	1 g maximum (with maximum load)
Tool Velocity	1.0 m/s (3.3 fps) maximum (with maximum load within primary work envelope)
Software Movement Limits	
Waist - Joint 1	5.59 r (320 deg)
Shoulder - Joint 2	4.36 r (250 deg)
Elbow - Joint 3	4.72 r (270 deg)
Wrist - Joint 4 (PUMA 560)	5.24 r (300 deg)
Wrist - Joint 5 (PUMA 560) Wrist - Joint 4 (PUMA 550)	3.49 r (200 deg)
Wrist - Joint 6 (PUMA 560) Wrist - Joint 5 (PUMA 550)	9.29 r (532 deg)
Joint Angular Resolution	
PUMA 550/560	
Waist - Joint 1	5.7557×10^{-3} deg/bit
Shoulder - Joint 2	4.17408×10^{-3} deg/bit
Elbow - Joint 3	6.7098×10^{-3} deg/bit
PUMA 550	
Wrist - Joint 4	5.00625×10^{-3} deg/bit
Wrist - Joint 5	4.69177×10^{-3} deg/bit

Table 1-5. Specifications (Cont)

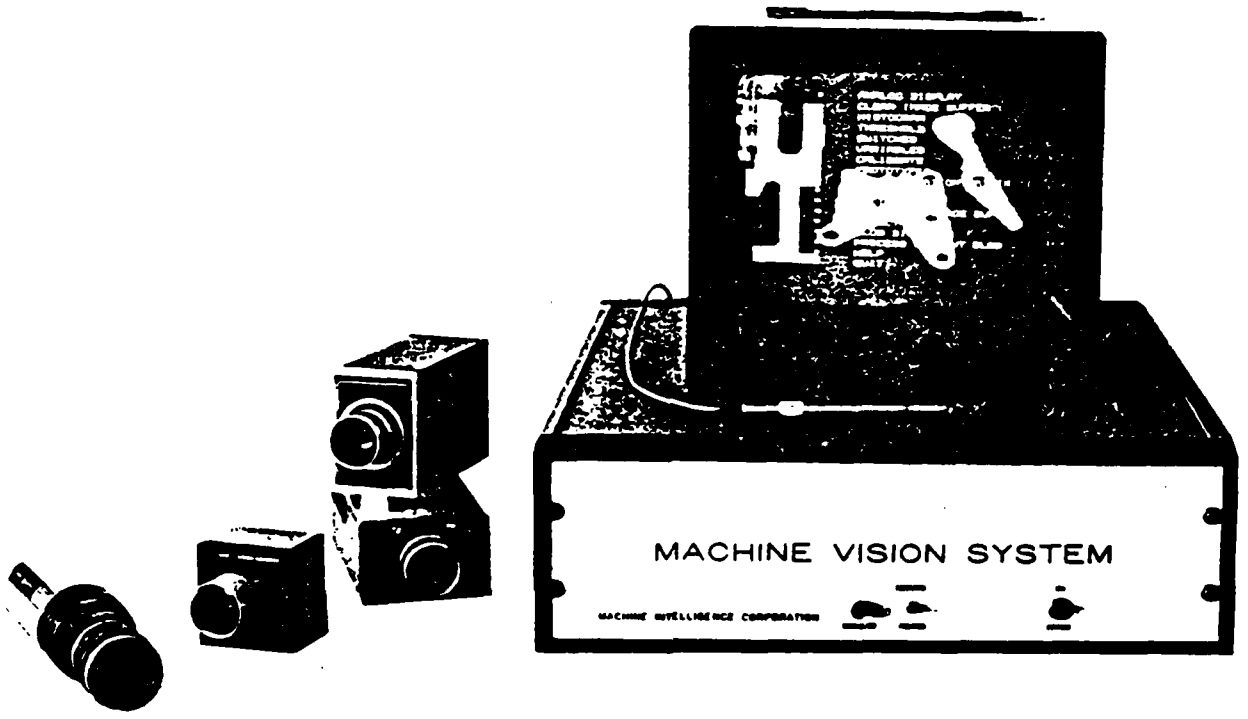
ITEM	SPECIFICATIONS
PUMA 560	
Wrist - Joint 4	4.7361×10^{-3} deg/bit
Wrist - Joint 5	5.00625×10^{-3} deg/bit
Wrist - Joint 6	4.69177×10^{-3} deg/bit
Encoder Index Resolution (One Motor Revolution)	
PUMA 550/560	
Waist - Joint 1	5.7557 deg
Shoulder - Joint 2	3.3392 deg
Elbow - Joint 3	6.7098 deg
PUMA 550	
Wrist - Joint 4	5.0062 deg
Wrist - Joint 5	4.6918 deg
PUMA 560	
Wrist - Joint 4	4.7358 deg
Wrist - Joint 5	5.0062 deg
Wrist - Joint 6	4.6918 deg
Velocity at SPEED 100	
PUMA 550/560	
Waist - Joint 1	82.1 deg/sec
Shoulder - Joint 2	53.5 deg/sec
Elbow - Joint 3	122.1 deg/sec
PUMA 550	
Wrist - Joint 4	241.4 deg/sec
Wrist - Joint 5	227.8 deg/sec
PUMA 560	
Wrist - Joint 4	227.8 deg/sec
Wrist - Joint 5	241.4 deg/sec
Wrist - Joint 6	227.8 deg/sec

Table 1-5. Specifications (Cont)

ITEM	SPECIFICATIONS
Maximum Linear Velocity at Speed 100	468 mm/sec
Maximum Cartesian and Joint Acceleration (= Deceleration) Time. Rest to Maximum Velocity	
Cartesian	0.112 sec
Waist - Joint 1	0.112 sec
Shoulder - Joint 2	0.112 sec
Elbow - Joint 3	0.056 sec
Wrist - Joint 4	0.056 sec
Wrist - Joint 5	0.056 sec
Wrist - Joint 6	0.056 sec
Maximum Envelope Error (Joint Excursion to Fatal Error)	
PUMA 550/560	
Waist - Joint 1	11.51 deg
Shoulder - Joint 2	8.34 deg
Elbow - Joint 3	13.41 deg
PUMA 550	
Wrist - Joint 4	10.01 deg
Wrist - Joint 5	9.38 deg
PUMA 560	
Wrist - Joint 4	9.47 deg
Wrist - Joint 5	10.01 deg
Wrist - Joint 6	9.38 deg
Controller	
Dimensions	0.48 m (19 inches) W x 0.311 m (12.25 in.) H x 0.050 m (19.7 in.) D
Weight	356 N (80 lb)

Part 2: Description of MIC Vision System (18)

Model VS-100
MACHINE VISION SYSTEM

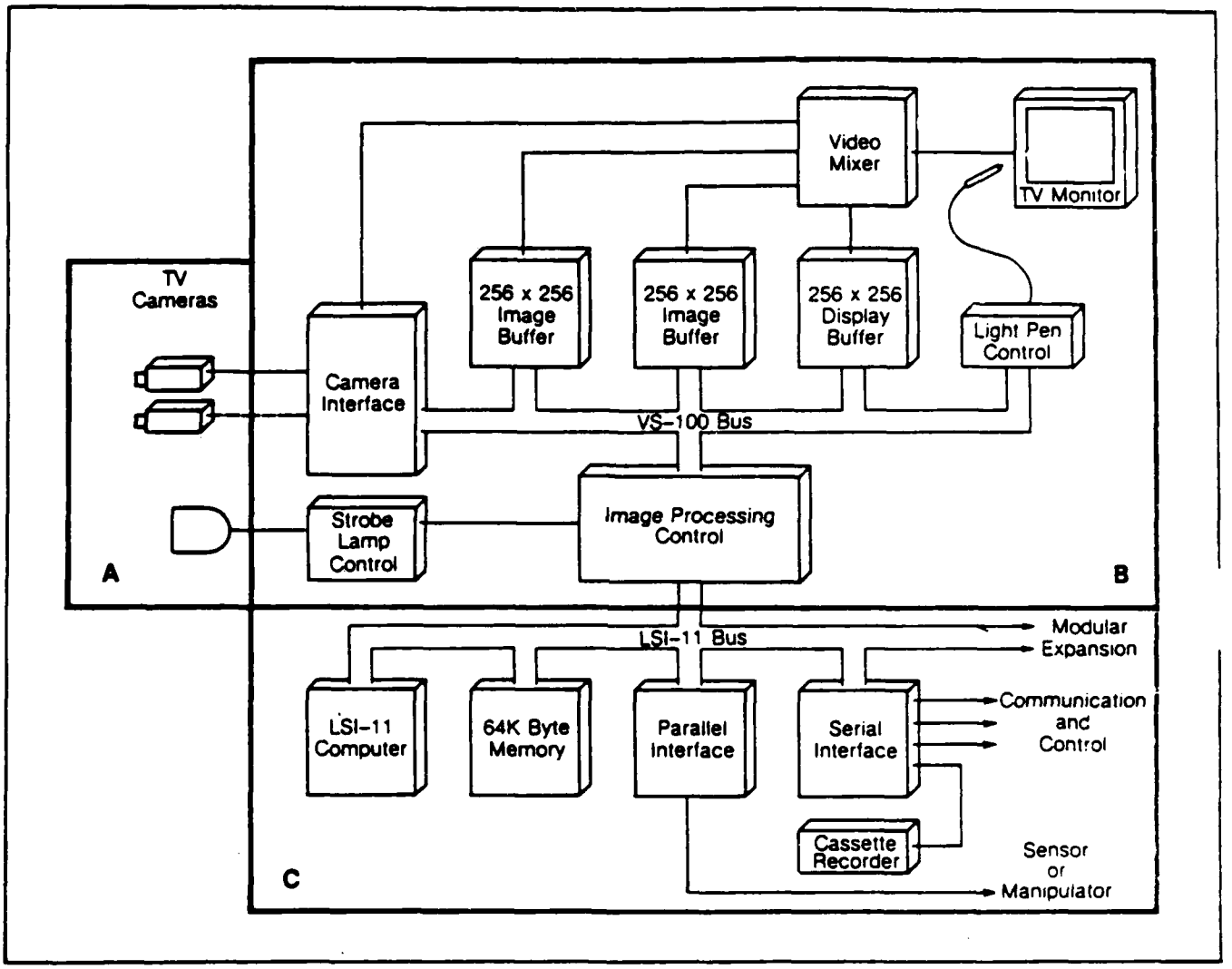


- For Inspection, Material Handling, and Assembly
- Simplified Light Pen Control
- Easy Programming by Showing
- Versatile Camera Inputs
- Programmable Output Control
- Powerful Computing Capability

Machine Intelligence Corporation
1120 San Antonio Road
Palo Alto, California 94303
(415) 968-4008

Model VS-

Functional Block Diagram



Theory of Operation

The VS–100 consists of one or more cameras (any of four types), an image processing unit, and a Digital Equipment LSI–11 microcomputer. It accepts images from up to two separate cameras (A), which may be operated with a strobe lamp triggered under VS–100 control. The image processing unit (B) transforms camera images into binary (black and white) images that are run-length encoded for data compression and subsequent processing. Efficient algorithms operating in the LSI–11 (C) perform a complete connectivity analysis of the encoded images, building data structures that represent essential features of each contiguous region. Up to 13 distinguishing features such as area, perimeter, center of gravity, number of holes, and minimum and maximum radii can be extracted for each region.

Object recognition is performed using a nearest neighbor classifier operating on a user-selectable subset of the features. Precise numerical measurements are computed to indicate the degree of confidence in the system's recognition of the object. If the degree of match of the selected features is above a user-adjustable threshold, the object is rejected as unknown or defective. Once an object is recognized, its position and orientation are determined.

Vision System

An Eye on the Future

The age of automated vision is here. In the factory, non-contact visual sensing can be used for inspection, measurement of critical dimensions, parts sorting, programmable part presentation, sensing for process control, and automated assembly. For a modest capital investment (that won't increase for multi-shift operation), machine vision provides:

- enhanced productivity,
- improved quality control, and
- greater reliability.

Machine Intelligence Corporation has utilized the results of over a decade of government-sponsored research in automated vision and engineered the VS-100 Vision System—a rugged, versatile, complete unit designed for industrial applications.

The VS-100 Vision System

The VS-100 is a highly advanced system, with both a broad range of image processing capabilities and ease of use. It recognizes and inspects images of complex objects against a contrasting background in real time. The objects can be anywhere in the field of view, in any orientation, even on a moving belt. Thus, a workpiece randomly positioned on a belt or a table can be located precisely, so that it can be inspected, or acquired and manipulated by an industrial robot or other positioning device.

Ease of Use

A most striking feature of the VS-100 is its human engineering. The VS-100 can be trained to analyze new objects simply by showing them to the system. Interactions with the system are menu-driven, using light-pen input; for most applications, users never need to touch a keyboard! Easy-to-use menus allow selection of the threshold level, relevant features, and other important parameters for specific applications. System control is by light pen, too. Calibration, training-by-showing, and storing and loading of prototype data can easily be done, with minimum training.

A Complete Development Tool

Because of its generality and simplicity of use, the VS-100 is ideal for developing and assessing applications of visual sensing. The VS-100 is a complete system including display, light pen, storage device for prototype information and a camera of your choice, as well as the visual processing unit. Yet its modular, expandable design permits easy system integration.

The VS-100 accepts on-off signals from external devices, and can in turn send on-off signals to other devices—up to sixteen in all. Three serial ports are available for communication with other computers—large hosts for system development or microcomputers for controlling gates, X-Y tables, industrial robots, or other manipulators. Special interfaces to other PDP-11 computers and several robots are available. The LSI-11 bus allows efficient interaction with a wide range of compatible devices (such as A/D and D/A converters and other LSI-11 modules) which you can add for your own purposes.

A Cost-Effective Application Device

The many sophisticated features of the VS-100 have been engineered for fast, consistent operation in a production environment. Where the system can be used without a large investment in ancillary equipment, the cost of the unit can be justified for a one-shift operation. In most instances, a complete system including the VS-100 and some computer-controlled manipulation provides adequate return on investment for a two-shift operation. The VS-100 provides high reliability: it operates at full accuracy around the clock, every day of the week. Battery power backup is an available option. Application-specific functions can be programmed by our staff.

What the VS-100 Can Do for You

Examples of applications in which the VS-100 can improve product quality and reduce costs include:

Inspection-
- Integrity and completeness
- Shape and size defects
- Flash
- Number, size, and positioning of holes
- Cosmetic stains
- Cracks and burrs
- Measurement of critical dimensions

Sensor-controlled acquisition and manipulation-
- Workpiece sorting
- Workpieces randomly positioned on conveyors
- Manufacturing processes requiring visual feedback
- Fastening operations
- In-process inspection

SPECIFICATIONS
Model VS-100 Vision System

General Features

- Human engineered to provide interactive menu-driven selection of threshold level, relevant features and other important parameters for specific applications, using light-pen input.
- Training-by-showing for simplified programming.
- Binary threshold settable by either light pen (interactively) or by program control.
- Settable black or white background.
- Fast run-length data compression hardware.
- Single pixel noise rejection–optionally settable.
- Software windowing of image area by light pen control.
- Menu-driven connectivity analysis software.
- Optionally reprocessing of selected image data.
- Menu-driven feature selection and calibration.
- DEC LSI-11/2 computer with EIS/FIS and 64K Byte RAM.
- Cassette recorder for program loading and prototype storage.
- 2 strobe lamp triggers for "freezing" moving objects.
- 2 quad-slots or 4-dual-slots available on LSI-11 backplane for customized applications, e.g., A/D, D/A, TTL interfaces.
- Processing time for many applications within a fraction of a second.

Input/Output

- Can accommodate up to 2 cameras, singly or in combination from the following types:
  a. GE TN2200, 128 x 128 solid state array (standard)
  b. GE TN2500, 240 x 240 solid state array (optional)
  c. Reticon LC600C256-1, 256 x 1 solid state array (optional)
  d. Standard (RS-170) vidicon with external sync (optional)
- Light pen input
- 4-port serial RS-232C, RS-422 or RS-423 interface (DLV11-J)
- 16-bit parallel TTL interface (DRV11)
- 2 frame buffers for up to 256 x 256 pixel arrays (binary)
- 1 display buffer (256 x 256) for graphics and text overlays
- 1 12" TV Display Monitor for displaying binary image data, processed image data, and analog images

Communications Interface

- Standard RS-232C, RS-422 or RS-423 serial communication ports for host and/or control computers.
- Data rate jumper-selectable from 150 to 38,400 baud.

Manipulator Interface

- Bi-directional 16-bit parallel interface may be used for manipulator control applications,—communication protocols are available for several major industrial robots.

Mechanical

- Cabinet enclosure, Can also be installed in a standard 19" rack.
- Rack-mount dimensions: 5¼" high; 17" wide; 24" deep
- Weight approximately 50 lbs.

Electrical

- Power: 105–125 VAC, 60 HZ, 300 watts maximum
- Optional: Battery stand-by supply, will maintain program and data storage for up to 4 hours.

Warranty

- The VS-100 Machine Vision System is guaranteed to be free of manufacturing parts and labor defects for a period of 90 days after delivery.

For further information or applications assistance, please write or call:

Machine Intelligence Corporation
1120 San Antonio Road
Palo Alto, California 94303
(415) 968-4008

SECTION I

INTRODUCTION

Univision is a camera-based processing system which enables VAL, the UNIMATE control system, to locate and identify objects within a given work area. The Univision system is composed of three subsystems, namely:

(1) Area array camera sensors (maximum of two)

(2) Vision processor, including:

  (a) Machine Intelligence Corporation (MIC) VS-100 Processor

  (b) Light pen and monitor for user interaction with the vision system

(3) Univision Interface Kit enabling VAL, appended with vision commands, to coummunicate with the vision processor. The Kit includes:

  (a) Interprocessor Communications Interface Card

  (b) Interprocessor Communications Cables

  (c) Vision software and diagnostics (down loadable from VAL's floopy disk)

A block diagram of the hardware system is shown in Figure 1.1.

The Univision system locates and categorizes stationary, objects within its field of view on the basis of features extracted from each object's silhouette - silhouettes generated from a spatially quantized image obtained from a single, fixed, overhead view. The control system, VAL, transforms the object location/orientation information into robot coordinates so that it can identify an appropriate user-defined, object-dependent grasp point(s), and then pick up and move the object as desired.

The key element in Univision is the VS-100 processor - a sophisticated image processing and pattern recognition system. Most of its sophistication is hidden from the user by its easy-to-use, high-level, menu-driven architecture. However, effective application of the Univision system will require, in most cases, that the user have a basic understanding of the "behind-the-scenes" operation of this device. The next section will discuss the operation of the Univision/VAL control system and will focus, in particular, on the VS-100 processing.

1.1 FUNCTIONAL OPERATION OF THE UNIVISION/PUMA CONTROL SYSTEM

Figure 1.2 shows a functional block diagram of the Univision/PUMA control system. The diagram illustrates the processing functions

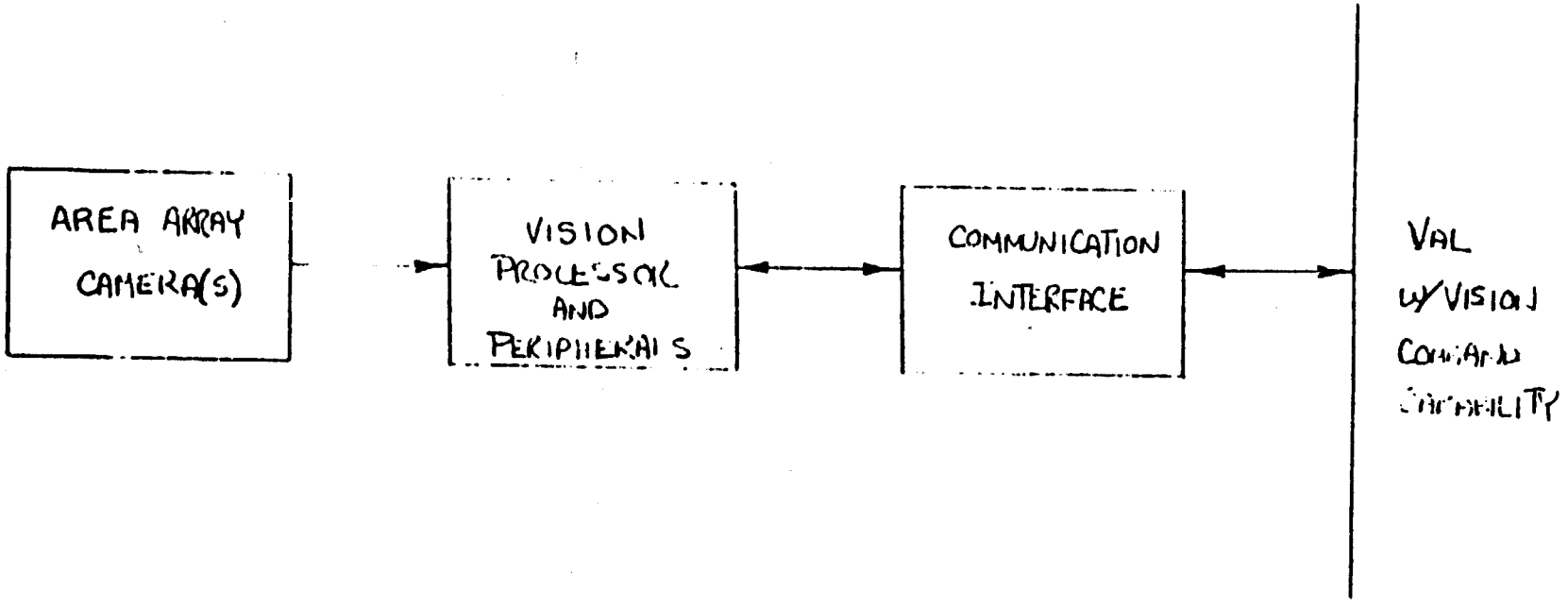


FIG. 1.1 BLOCK DIAGRAM OF THE VISION SYSTEM

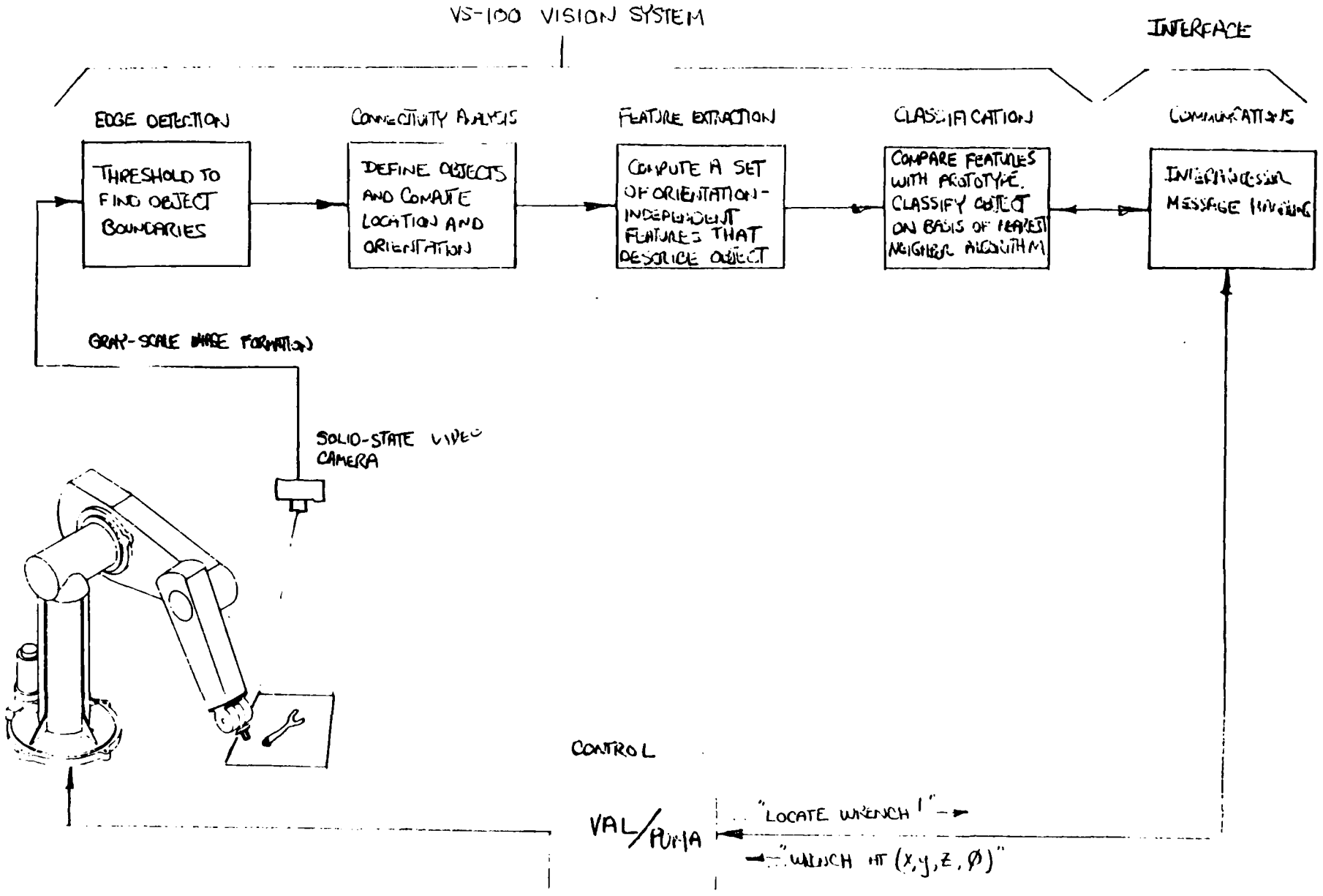


FIG 12 FUNCTIONAL BLOCK DIAGRAM OF THE UNIVISION/PUMA CONTROL SYSTEM

necessary to locate, recognize, and pick up parts. The basic functions are:

(1) Image Formation

(2) Boundary Detection

(3) Location and Orientation Computation

(4) Extraction of Orientation-Independent Features

(5) Object Classification (Recognition)

(6) Inter-Processor Communication

(7) Transformation of Object Location/Orientation from the Vision Frame to the Robot Frame

(8) Control

We will now describe, in detail, the internal operation of Univision/PUMA during a typical vision-aided task. We'll consider the task of picking up a wrench, located and oriented arbitrarily within Univision's field of view, and placing it in a bin. The steps required to complete this task are as follows:

1. VAL initiates the task by sending a message to the vision processor asking it to freeze the current video frame (picture) for further processing. One frame of video, consisting of an array of N1 x N2 analog voltages (gray scale), each proportional to the image intensity on a particular photo element (pixel) in the solid state array, is sent to the vision processor. The N2 "column voltages" are read out continuously in a raster-type scan for each of the N1 rows of the sensor array grid.

2. The vision system thresholds the video, on-the-fly, thus creating a binary image or silhouette. The thresholding operation defines each object's outline or boundary. To further simplify processing, only the position of threshold crossings (positive and/or negative) and the number of pixels, or run length, in between each crossing are stored in image memory for each raster scan line. This coding process is called run length encoding and results in near minimal storage of the information in the binary picture.

3. The vision system performs connectivity analysis which combines all run lengths which are part of an object's silhouette. This analysis also detects the presence of holes. Any objects whose silhouettes merge are subsequently treated as a single object after this step.

4. In parallel with "3.", the vision processor sums the centroids of all of the run lengths for each silhouette to obtain its centroid location in camera coordinates. Orientation, using any of a variet

of orientation algorithms, is also computed.

5. Feature extraction, the process of computing a set of quantities that can be used in object (pattern) classification, is then performed by the vision processor. In Univision, only orientation/position-independent features are computed because a part must be recognized regardless of its placement in the field of view. Typical features computed are area, compaction (perimeter * * 2/area), second moments, number of holes, etc. The VS-100 allows the user to choose these features, among a set of 13 and aids the user in selecting those features which are best for a given application.

6. Object classification is then performed by statistically comparing the computed features of the silhouette with those of silhouettes generated by a set of prototype objects. Typically, the prototypes are the set of objects that one would expect to "see" at the workstation. The user trains Univision to recognize these parts. During the training process, the vision system establishes the mean and variability of the features of the prototype silhouette over a variety of positions and orientations within the field of view and over the possible stable states of the prototype object.

   Silhouettes are classified as being a "member" of the prototype group if the statistical "distance" between the features of the prototype and the silhouette is shortest. This technique is called "nearest-neighbor" classification. The following example will explain the process and point out the importance of prototype training.

   EXAMPLE: We will suppose that (1) our prototypes are limited to two objects - an adjustable wrench and a fixed head wrench; and, (2) the feature that most easily classifies them is the compaction index, c = perimeter * * 2/area. During training, we "show" the vision system the two objects in a variety of possible locations, and orientations. In the case of the adjustable wrench, we also train on the expected range of wrench openings. This training procedure defines the mean and variability of the features used during classification. The spread (probability) functions of c for the two objects after training are illustrated in Figure 1.3. As shown, the adjustable wrench exhibits higher feature variability because its perimeter can vary significantly, as a function of opening, without significant changes in area.

   The statistical distance from a measured feature, c, to the average feature of a prototype $\bar{c}$ is measured in terms of the number of standard deviations d between c and $\bar{c}$. If σ defines the prototype variability established during prototyping, then the distance, d, is

$$d = \frac{c - \bar{c}}{\sigma}$$

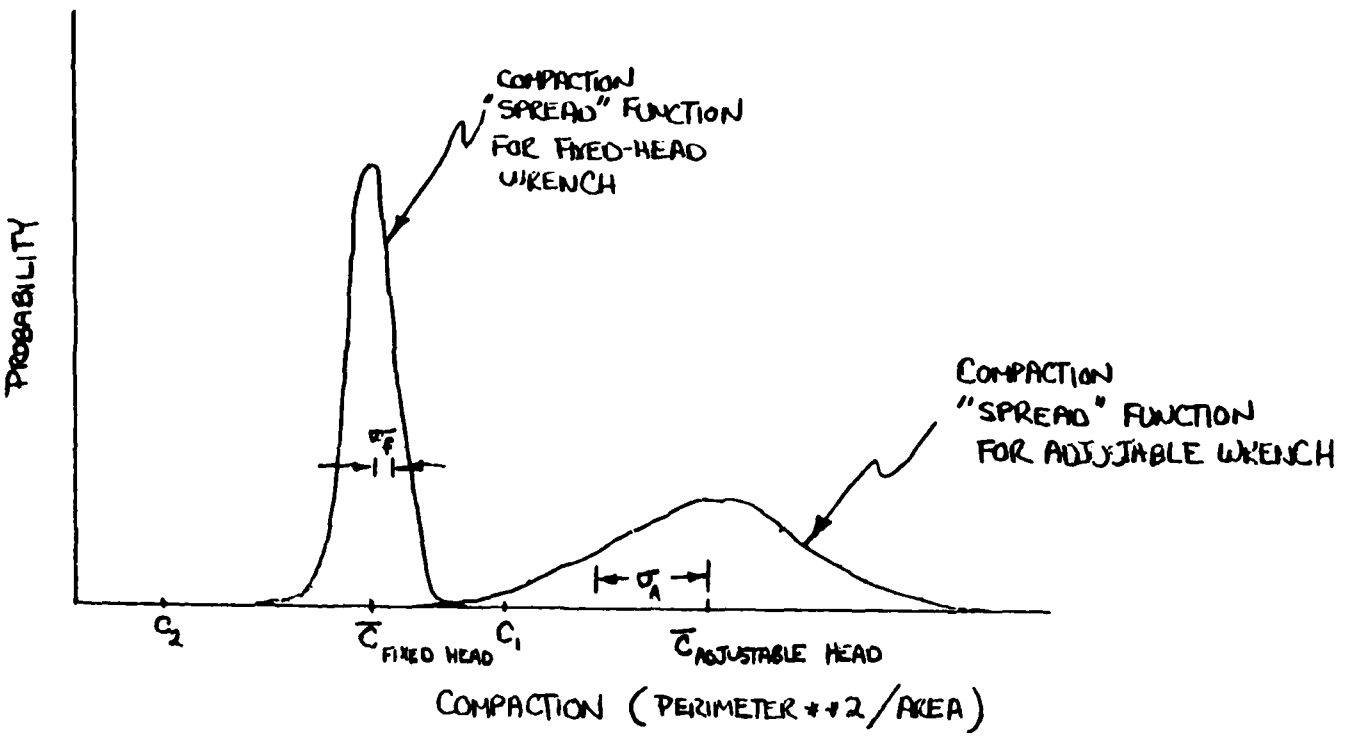


FIG. 1.3 ILLUSTRATION OF PROTOTYPE FEATURE PROBABILITY FUNCTIONS FOR FIXED HEAD AND ADJUSTABLE HEAD WRENCH

d is in a sense a probability measure - as d increases, the probability that the object matches the prototype decreases. For example, if $d > 3$, and the spread function of the prototype feature is normally (Gaussian) distributed, there is less than a 1% chance that the object is a member of the prototype classification

The nearest neighbor classification procedure computes the "distance" between the object feature(s) and each prototype's feature(s). The object is classified as a member of the prototype group if (1) its feature "distance", d^*, to the prototype is shortest; and, (2) $d^* < d_{max}$ where d_{max} establishes a statistical confidence limit beyond which the object cannot be classified - the probability that the object is a member of any prototype group is too small.

If we refer to Figure 1.3 we see then that an object with compaction index, c1, would be classified as an adjustable wrench. Note, however, that the numerical value of C1 is actually closer to $\bar{c}$ measured for a fixed head wrench. An object with feature c2 cannot

be classified since it clearly lies outside any reasonable confidence interval of being a member of either the wrench classifications.

As can be seen, prototype training is key to reliable object classification. Care must be taken by the user to ensure that the mean and variability of silhouette features measured during training adequately reflects that which will be encountered in practice.

The nearest-neighbor classification algorithm is done "behind-the-scenes" in the VS-100 and normally requires no user intervention except during prototype training. In this regard, the VS-100 system helps the user to train prototypes and allows him to evaluate, improve, and predict classification performance before he puts the system on-line.

7. Upon receiving a completion message on the original picture command, VAL asks the vision processor for the location of any objects that have been classified as a wrench.

8. The vision system reports the location and orientation of a wrench (if there is one) in its field of view by sending an appropriate message to VAL. Note that during the time the vision system is locating and identifying objects, VAL is free to complete other tasks. This overlapping mode of operation minimizes the effect of vision system processing on robot cycle times.

9. In the final step of the task, VAL:

   (a) Transforms the location and orientation of the wrench from

vision to robot coordinates using parameters defined during calibration.

(b) Determines the location and orientation of a user defined grip point. This feature allows a wide range of objects to be picked up in a reliable fashion.

(c) Checks with the VS-100 to see whether the robot gripper will have clearance to pick up the part at the grip location.

(d) Positions and orients the gripper.

(e) Picks up the wrench and places it in the bin.

Even a simple task like that described above requires significant vision processing capability. VAL has been programmed so that only a few, high-level, vision commands are needed to access information from the VS100 processor and to control its complex image processing and pattern recognition operations. This capability frees the VAL user to concentrate more on the important control aspects of a vision-aided task.

1.2 OVERVIEW OF THE MANUAL

The manual covers the theory, operation, maintenance, installation, and user application of Univision. Chapter 2 is a copy of the VS-100 processor reference manual which describes the setup and operation of the vision system and its peripherals. Chapter 3 is a supplement to "Users Guide to VAL" that describes the new VAL commands used for interrogation and control of the vision system. Chapter 4 is a user's guide to Univision which covers (1) practical considerations in setting up and using a Univision system; (2) setup of the camera system; (3) installation of the Univision Interface Kit and start-up procedures; (4) vision and vision-to-robot calibration; (5) prototype training; (6) computing object to robot transformations; and, (7) application programs. Chapter 5 contains an operating manual for the cameras (either TN-2500 or MIC 22) used in the application. Chapters 6, 7, 8 cover the spare parts list, mechanical drawings and electrical drawings respectively.

Three sections are included in the Appendix. Appendix A is the diagnostics manual for the VS-100 describing use of the diagnostic software and other debugging procedures needed to isolate failures in the VS-100 system. Appendix B describes in detail the hardware and software protocols used in the interprocessor communications interface. Appendix C is a copy of an applications note covering lens selection for the TN-2500 and TN-2200 (same sensor as the MIC-22) General Electric camera sensors.

1.3 FUNCTIONAL SPECIFICATION

Parts Recognition:

Maximum number of trained prototypes resident in the system at a given time 9

Maximum number of objects that can be recognized in the field of view 12
Reliable parts recognition requires adequate image contrast and requires that the part silhouettes do not merge.

Vision Sensor Resolution:

MIC 22	1 Part in 128 of Camera Field of View
G.E. TN-2500	1 Part in 244 of Width of Field
	1 Part in 248 of Height of Field

Location and Orientation Measurement Accuracy

(Ignoring effects of parallax, and robot vision calibration errors)

Location	Equal to or Better Than Vision Sensor Resolution
Orientation	Dependent on Object and on the Orientation Feature Selected

Environmental:

Camera w/o Enclosure	
Ambient Temp	0-50 C
Humidity	90% Max Non-Condensing
Vision Processor	
Ambient Temp	0-46 C
Humidity	90% Max Non-Condensing

Part 3: Interface Software Descriptions (19)

This document attempts to describe all of the improvements that Univision VAL-II has compared to Univisionl VAL-I.

Additions:

U.blink.blob - This routine will allow the user from the VAL keyboard to outline the selected blob with either a white or a black border. This routine can be very helpful in determining which blob the vision system is refering to.

U.Boot - Allows the VAL-II user to restart the vision system from the VAL-I keyboard. It is a destructive restart in that it will delete all prototypes. However, it can save the user from having to reload t vision system software from tape.

U.Delete.Blob - This routine will allow the user to delete blobs from the blob descriptor area. This will save vision system memory allowing more trainings per prototype.

U.Erase - This routine will erase the text overlay, this will allow the user to see a clear unobstructed digitized picture.

U.Recognize - This routine will compare all of the known prototypes to to the specified blob and return the prototype id number that best fits the blob. The routine will also return the difference from the best fit, and the difference to the second best fit.

U.Raster.Line - Allows the user to examine each line on the vision system to determine which pixels are on or off.

Overall Improvements-

1. By selecting the fast option VAL-II can process other information while the vision system is processing data. Upon completion VAL-I can readdress the vision system to get the processed data.

2. Overall the user has more options when analyzing the vision picture

Changes-

Findheap- Now is U.WHERE.ARE.YOU.PROTO. Information is contained in array element [0].

Train- Handled by U.TRAIN.BLOB. User can now use teach pendant to train prototypes.

VSTORE, VLOAD- Functions handled by U.SAVER, however, U.SAVER does not preserve switch settings, and orientation features. U.SAVER uploads the information into an array then the user must store the array to disk as a series of real values. U.SAVER will allow the user to delete prototype

Not Implemented-

Grip functions.

Routine: U.Boot

Description:

This routine will allow the user to boot the MIC vision system. CAUTION: during a reboot all prototype information will be erased. This routine has been provided for use when the vision system hangs.

Usage:

exec boot

Created: Sept 8, 1985
Revision: 1.0

Routine: U.Blink.Blob

General Description:

This diagnostic routine will aid in locating blobs by outlining them with a white or a black border.

Variables Description:

u.blobnum - This parameter must be set by the user before calling this routine.

The legal values range from one to the number of blobs that are currently on the display screen.

u.color - This parameter must be set by the user before calling this routine.

The variable has two legal values, zero and negative one.

If the variable is set equal to negative one, the outline is drawn in bright white.

If the variable is set equal to zero, the outline is drawn in black.

s.error - This variable indicates, in a logical fashion, the result of the execution of the routine. If it returns TRUE, it indicat that an error was detected by the vision system. The actual error code will be found in s.system.error. If it returns as FALSE, the routine completed with out an error.

s.message[] - Reserved for system use.
s.numbytes - Reserved for system use.
s.numwords - Reserved for system use.

s.system.error - This variable contains the error code that is returned from the vision system. It should be referenced when the variabl s.error indicates that a problem occured with the routine.

If s.error is FALSE, this variable will be equal to zero.

Subroutine Usage:

U.Blink.blob directly calls:

S.transfer.message
S.get.message

Usage Example:

```
.Program Test
1;*********************************************************
2;= This program will flash a blobs' border white and black until
3;= the user strikes the <REC> key on the teach pendant
```

```
4       PROMPT "PLEASE ENTER THE BLOB NUMBER TO BLINK --> ",S.BLOBNUM
5       TYPE "PRESS <REC> ON THE PENDANT TO STOP"
6       DO
7           U.COLOR=-1 ;SET COLOR TO WHITE
8           CALL U.BLINK.BLOB; TELL VISION TO MAKE BORDER WHITE
9           IF S.ERROR THEN
10           TYPE "ERROR: HALTING" ; STOP IF AN ERROR
11           HALT
12          END
13          FOR INDEX = 1 TO 500
14          END
15          U.COLOR=0 ;SET COLOR TO BLACK
16          CALL U.BLINK.BLOB ;TELL VISION TO MAKE BORDER BLACK
17          IF S.ERROR THEN
18           TYPE "ERROR: HALTING" ;STOP IF ERROR
19           HALT
20          END
21      UNTIL PENDANT(1) BAND 1 == 1
```

Routine: U.Change.camera

General Description:

This routine will allow the user to select the camera that will be used.

Variables Description:

u.camera - This variable must be set by the user before this routine is called. It must contain a number between one and four. (inclusive) The number that this variable contains will be the new camera number.

s.error - This variable indicates, in a logical fashion, the result of the execution of the routine. If it returns TRUE, it indicates that an error was detected by the vision system. The actual error code will be found in s.system.error. If it returns as FALSE, the routine completed with out an error.

s.message[] - Reserved for system use.
s.numbytes - Reserved for system use.
s.numwords - Reserved for system use.

s.system.error - This variable contains the error code that is returned from the vision system. It should be referenced when the variable s.error indicates that a problem occured with the routine.

If s.error is FALSE, this variable will be equal to zero.

Subroutine Usage:

U.Change.camera directly calls:

s.transfer.message
s.get.message

Usage Example:

```
.program usage
1     U.camera=1
2     CALL U.change.camera
3     CALL U.take.picture
4     FOR delay=1 TO 500
5     END
6     U.camera=2
7     CALL U.change.camera
8     CALL U.take.picture
9     FOR delay=1 TO 500
10    END
exec usage,-1
```

Routine: U.Delete.Blob

General Description:

This routine will remove a blobs' features from the MIC vision system.
(It does not delete a prototype.)

Variables Description:

u.delete.blob.num - This variable must be set by the user before calling this routine.

s.error - This variable indicates, in a logical fashion, the result of the execution of the routine. If it returns TRUE, it indicates that an error was detected by the vision system. The actual error code will be found in s.system.error. If it returns as FALSE, the routine completed with out an error.

s.message[] - Reserved for system use.
s.numbytes - Reserved for system use.
s.numwords - Reserved for system use.

s.system.error - This variable contains the error code that is returned from the vision system. It should be referenced when the variable s.error indicates that a problem occured with the routine.

If s.error is FALSE, this variable will be equal to zero.

Usage Example:

To remove blob descriptor 2 from the vision system
.DO U.DELETE.BLOB.NUM=2
.EXEC U.DELETE.BLOB

Routine: U.Erase

General Description

This routine will erase the any text or graphic overlays on the screen. The routine does not erase the current picture in the buffer.

Variables Description:

s.error - This variable indicates, in a logical fashion, the result of the execution of the routine. If it returns TRUE, it indicates that an error was detected by the vision system. The actual error code will be found in s.system.error. If it returns as FALSE, the routine completed with out an error.

s.message[] - Reserved for system use.
s.numbytes - Reserved for system use.
s.numwords - Reserved for system use.

s.system.error - This variable contains the error code that is returned from the vision system. It should be referenced when the variable s.error indicates that a problem occured with the routine.

If s.error is FALSE, this variable will be equal to zero.

Subroutine Usage:

Erase directly calls:

S.transfer.message
S.get.message

Usage Example:

```
.Program Demo
1       Call U.erase; Erase vision screen to make look good.
```

Routine: U.Initvision

General Description:

This routine will initialize the communication software. It will also set default values for other the other subroutines. This routine must be called before any other communication is attempted.

Variables Description:

Setup Section:

U.blobnum - Initialized value is one. (Set primary blob)
U.color - Initialized value is negative one. (Color= white.)
U.expected.shapes - Initialized value is nine.
U.fast - Initialized value is zero. (Slow Picture)
U.new.picture - Initialized value is zero. (No picture)
U.numblob.id - Initialized value is negative one. (Identify all)
U.picture.fast - Initialized value is zero. (Slow Picture)
U.camera - Initialized value is one.

Variables Section:

s.error - This variable indicates, in a logical fashion, the result of the execution of the routine. If it returns TRUE, it indicates that an error was detected by the vision system. The actual error code will be found in s.system.error. If it returns as FALSE, the routine completed with out an error.

s.exp - Reserved for system use.
s.fraction - Reserved for system use.
s.message[] - Reserved for system use.
s.numbytes - Reserved for system use.
s.numwords - Reserved for system use.

s.system.error - This variable contains the error code that is returned from the vision system. It should be referenced when the variable s.error indicates that a problem occured with the routine.

If s.error is FALSE, this variable will be equal to zero.

Subroutine Usage:

Initvision directly calls:

S.transfer.message
S.get.message

Usage Example:

```
.EXEC U.INITVISION
```

Routine: U.Recognize

General Description:

This routine will compare all known prototypes to the specified blob number and return the prototype identification number that corresponds to that blob. The routine will also return a two match numbers that indicate how close the blob is to first choice, and how close of a match it is to a second choice.

Variables Description:

s.diff1 - This variable contains the difference from the specified blob to the best matching prototype.

s.diff2 - This variable contains the difference from the specified blob to the next best prototype.

CAUTION: If the values contained in s.diff1 and s.diff2 are very close, this indicates that the vision system may not always correctly differentiate between the two prototypes. To correct this requires further training of both of the prototypes.

s.error - This variable indicates, in a logical fashion, the result of the execution of the routine. If it returns TRUE, it indicates that an error was detected by the vision system. The actual error code will be found in s.system.error. If it returns as FALSE, the routine completed with out an error.

s.message[] - Reserved for system use.

U.numblob - This variable must be set by the user before calling this routine. It contains the blob number that you want compared to all of the known prototypes.

s.numbytes - Reserved for system use.
s.numwords - Reserved for system use.

s.proto.id - This variable contains the ID number of the prototype that best matches the specified blob number.

s.system.error - This variable contains the error code that is returned from the vision system. It should be referenced when the variable s.error indicates that a problem occured with the routine.

If s.error is FALSE, this variable will be equal to zero.

Routine: U.Reprocess.picture

General Description:

This routine will instruct the vision system to reanalyze the current image.

Variables Description:

s.blob.count - Upon completion of the routine "Take.picture" this variable will contain a count of the number of blobs that are in the picture. The count is not always equal to the number of objects but rather is the number of connected regions in the image.

s.error - This variable indicates, in a logical fashion, the result of the execution of the routine. If it returns TRUE, it indicates that an error was detected by the vision system. The actual error code will be found in s.system.error. If it returns as FALSE, the routine completed with out an error.

s.message[] - Reserved for system use.
s.numbytes - Reserved for system use.
s.numwords - Reserved for system use.

u.fast - This parameter must be set by the user before calling the routine.

u.fast has two legal values; zero or one.

If u.fast is set equal to zero, then the vision system will take a slow picture. Slow means that the routine will wait for the vision system to completely process the command before proceeding.

If u.fast is set equal to one, then the vision system will take a fast picture. The routine will finish before vision completes all processing. Variable s.blob.count will be equal to zero regardless of the number of blobs that exsist in the image.

s.system.error - This variable contains the error code that is returned from the vision system. It should be referenced when the variable s.error indicates that a problem occured with the routine.

If s.error is FALSE, this variable will be equal to zero.

Subroutine Usage:

Reprocess.picture directly calls:

S.transfer.message
S.get.message

Routine: U.Take.picture

General Description:

This routine will instruct the vision system to take a picture of the current field of view and process the image.

Variables Description:

s.blob.count - Upon completion of the routine "Take.picture" this variable will contain a count of the number of blobs that are in the picture. The count is not always equal to the number of objects but rather is the number of connected regions in the image.

s.error - This variable indicates, in a logical fashion, the result of the execution of the routine. If it returns TRUE, it indicates that an error was detected by the vision system. The actual error code will be found in s.system.error. If it returns as FALSE, the routine completed with out an error.

s.message[] - Reserved for system use.
s.numbytes - Reserved for system use.
s.numwords - Reserved for system use.

u.picture.fast - This parameter must be set by the user before calling the routine.

u.picture.fast has two legal values; zero or one.

If u.picture.fast is set equal to zero, then the vision system will take a slow picture. Slow means that the routine will wait for the vision system to completely process the command before proceeding.

If u.picture.fast is set equal to one, then the vision system will take a fast picture. The routine will finish before vision completes all processing. Variable s.blob.count will be equal to zero regardless of the number of blobs that exsist in the image.

s.system.error - This variable contains the error code that is returned from the vision system. It should be referenced when the variable s.error indicates that a problem occured with the routine.

If s.error is FALSE, this variable will be equal to zero.

ubroutine Usage:

U.Take.Picture directly calls:

S.transfer.message
S.get.message

sage Example:

Program Test

```
        U.picture.fast = 0; Set slow mode.
        Call U.Take.picture; Ask the vision system to take a picture.
        If S.error then; Examine the variable to see if a problem occured.
          Type "Bad Routine  Error code is --> ",/d,s.system.error
    ;= The above line will display the error code and a short message.
        Else
          Type "Good Picture  Blob Count is --> ",/d,s.blob.count
    ;= The above will happen if everything worked.
        End
```

utine: U.Where.Are.You.Proto

neral Description:

is routine will analyze an image and return the coordinates of the centroid
a shape, along with an orientation measurement that can be used to create a
ansformation to locate an actual piece. The coordinates are returned in
its of pixels and it is the users responsibility to scale these pixels
to actual coordinates.

e orientation value is returned in radians and it must be converted to degrees
fore being used.

riables Description:

color - this variable has two legal values, negative one and zero. If the user sets this variable to negative one white shapes will be used. If the the user sets it to zero the vision system will use black shapes.

expected.shapes - this variable will tell the vision system how many matches to allow.

fast - this variable will tell the vision system whether or not the action is to occur in a fast mode of operation or not. If the user selects fast by setting this variable to one the user must recall this routine with this switch set to zero to collect the data.

new.picture - this variable instructs the vision system to -1 repicture, 0 for previous picture and 1 for picture.

numblob.id - this variable contains the number of prototypes to find, if you want the vision system to locate all of prototypes set this variable to -1.

blob.there[] - this variable indicates in a logical fashion if the prototype is in the field of view. If the prototype is in the current picture this variable will be set equal to TRUE. If the not in the field of view the variable will be

prototype is
set to FALSE. The element number is equated to the prototype number.

s.error - this variable indicates, in a logical fashion, the result of the execution of the routine. If it returns TRUE, it indicates that an error was detected by the vision system. The actual error code will be found in the variable s.system.error. If it returns a FALSE, the routine completed without an error.

s.message[] - reserved for system use
s.numbytes - reserved for system use
s.numwords - reserved for system use

s.orient.blob [] - Contains the angular measurement of the prototype whose number is the array offset.

s.system.error - This variable contains the actual error code returned from the vision system. It should be referenced when the variable s.error indicates that a problem occured during the execution of the routine.

If s.error is FALSE, this variable will be equal to zero.

s.xcenter.blob [] - Contains the value in pixels locating the prototype whose number is the array offset, in the X direction.

s.ycenter.blob [] - Contains the value in pixels locating the prototype whose number is the array offset, in the Y direction.

Example: For prototype one in the picture, its X location will be found in variable S.xcenter.blob[1], the Y location will be found in variable S.ycenter.blob[1], and finally the orientation will be found in S.orient.blob[1]. The number found between the brackets is the prototype number. If an unidentified shape is in the field of view, then it will be referenced by array offset zero.

POSSIBLE VALUES FOR S.SYSTEM.ERROR

Value Meaning

Corrective Action for the following errors:

Examine the variables that you are required to set. Reset the bad one to a legal value.

1 Bad Parameter has been passed to the routine. The Routine has not executed.

Corrective Action for the following errors:

None, a zero indicates a good run.

0 The routine ended with out any error detected by the vision system.

Corrective Action for the following errors:

Reattempt the communication with the vision system.

Attempt to reload both the vision system and the VAL-II system. if this error persists please note all actions that lead up to this and contact Unimation Inc.

-1 Not yet implemented. The command number passed to the vision system is not defined.

-3 The command number is out of range.

-5 The argument passed to the vision system is out of range.

-7 The command number is reserved for future use.

-17 The variable name passed to the vision system was not recieved properly.

-21 Bad Restart operation, this error should only occur during the execution of the boot routine.

-23 Vision System restarted during command.

-129 Error from User routine.

-131 User routine number out of range

-133 Buffer to small for return.

-135 No User.

Corrective Action for the following errors:

Rename the one of the prototypes.

-19 Duplicate prototype name.

Corrective Action for the following errors:

This collection of errors most probably result from specifying an illegal blob number.

-9 Blob is not active.

-11 Blob Specification is illegal.

-13 No Blob found.

Program: U.Saver

Instructions/Usage

U.Saver is a menu driven program that allows you to delete prototypes from the vision system, copy them from the vision system to val, and copy them from val to the vision system. It deals with only prototype information not orientation features, or other switches.

To upload prototypes select option 1, then enter the prototype number. Upon completion, store the real variable PROTOINFO[] to disk.

To download first load the real variables from disk and then execute the program. Select option 2 and the prototype will be restored.

To delete a prototype select option 3, then enter a 1 to confirm.

Program: U.train.blob

Instructions/Usage

U.Train.blob will allow you to add in a blobs description to a prototypes description. (this is called training) It will use the primary blob.

The user simply enters the prototype number and then the number of training times. By depressing <REC> on the pendant a picture will be taken, and the blob will be added into the prototype.

Note: This program calls the user routine u.take.picture and sets the flag u.picture.fast to zero (slow mode). This routine will not function reliably unless the vision system has been initialized.

Appendix B

VRS Software Program Listings

This Appendix contains the software program listings for the vision-robot system (VRS). The listings are grouped together by the tasks in which they were accomplished.

A miniature Table of Contents of the program listings follow:

Table of Contents

Task 1: Static Visual Servo Control Program Listings

```
- rl vision.menu
.PROGRAM vision.menu
   1 ;
   2 ; THIS PROGRAM PROVIDES A MENU FOR SELECTING VARIOUS OPTIONS
   3 ; WHEN USING THE VISION SYSTEM.  MOST OF PROGRAMS AVAILABLE,
   4 ; ARE SYSTEM PROGRAMS FROM THE UNIVISION1.PG FLOPPY DISK.
   5 ; THERE ISN'T MUCH DOCUMENTATION AVAILABLE FOR THESE PROGRAMS,
   6 ; HOWEVER THEY ARE FAIRLY SIMPLE AND USER FRIENDLY.
   7 ;
   8 ; THE VISION/ROBOT SYSTEM MUST BE INITIALIZED BEFORE EXECUTING
   9 ; ANY PROGRAMS.
  10 ; THE VISION/ROBOT INTERFACE IS INITIALIZED BY CALLING
  11 ; U.INITVISION.
  12 ;
  13 ; **********************WARNING*****************************
  14 ; *                                                        *
  15 ; * ONCE THE INTERFACE HAS BEEN INITIALIZED FOR THE        *
  16 ; * 1ST TIME, THE VISION/ROBOT SYSTEM MUST BE CALIBRATED.  *
  17 ; *                                                        *
  18 ; **********************************************************
  19 ;
  20 ; ==============  CAUTION  ==========================
  21 ;
  22 ; IF THE CAMERA IS EVER MOVED RELATIVE TO THE ROBOT, THE
  23 ; REFERENCE FRAME FOR OBJECTS VIEWED BY THE VISION SYSTEM
  24 ; WILL ALSO BE MOVED.  IN ORDER FOR THE ROBOT TO MOVE TO
  25 ; LOCATIONS DEFINED BY THE VISION SYSTEM, A RELATIVE
  26 ; TRANSFROMATION MUST BE DEFINED WHICH RELATES THE REFERENCE
  27 ; FRAME OF THE VISION SYSTEM TO THAT OF THE ROBOT;
  28 ; vision.robot.cal WILL ACCOMPLISH THIS TASK.
  29 ;
  30 ;=================================================
  31 ;
  32     10  TYPE /B, /C2; beeps the screen & skips 2 lines
  33         TYPE "************************************************"
  34         TYPE "*                                              *"
  35         TYPE "*     WELCOME TO AFIT'S VISION LAB             *"
  36         TYPE "*                                              *"
  37         TYPE "************************************************"
  38         num = 0; initialize parameter
  39         TYPE /C2, /B
  40         TYPE "  THE FOLLOWING OPTIONS ARE AVAILABLE   "
  41         TYPE /C1
  42         TYPE "    1.  INITIALIZE VISION/ROBOT COMMUNICATION ", /C1
  43         TYPE "    2.  VISION-TO-ROBOT CALIBRATION ", /C1
  44         TYPE "    3.  PROTOTYPE TRAINING          ", /C1
  45         TYPE "    4.  PROTOTYPE STORAGE AND DELETION  ", /C1
  46         TYPE "    5.  VISION DEMO   ", /C1
  47 ;
  48         TYPE "********************************************"
  49         TYPE "*                                          *"
  50         TYPE "*  -----------> WARNING <--------------    *"
  51         TYPE "*                                          *"
  52         TYPE "*    BEFORE EXECUTING OPTIONS 3,4, OR 5,   *"
  53         TYPE "*   OPTIONS 1 AND 2 MUST BE ACCOMPLISHED   *"
  54         TYPE "*                                          *"
  55         TYPE "********************************************", /C1
  56 ;
  57         TYPE "PLEASE ENTER THE NUMBER OF THE OPTION "
  58         PROMPT "  YOU DESIRE --->  ", num
  59  ;
  60  ;
  61  :
```

```
 62  ;
 63  ;
 64  ;
 65  ;
 66  ;
 67          CASE INT(num) OF
 68            VALUE 1:
 69              CALL u.initvision; INITIALIZE VISION/ROBOT SYSTEM
 70              TYPE /C2
 71              PROMPT "===> INTIALIZATION COMPLETE, PRESS <RETURN> TO CONTINU
 72            VALUE 2:
 73              CALL vision.robot.cal; PROGRAM TO CALIBRATE SYSTEM
 74            VALUE 3:
 75              CALL prototype.train; USE THIS TO TEACH THE VISION SYSTEM
 76                           ; DIFFERENT OBJECTS
 77            VALUE 4:
 78              CALL blob.store.or.delete; THIS WILL STORE OR DELETE A
 79                            ; PROTOTYPE INTO/FROM THE VISION SYSTEM'S
 80                            ; MEMORY.
 81            VALUE 5:
 82              CALL vision.demo.menu; THIS EXECUTES THE VISION DEMO
 83            ANY
 84              TYPE "THAT NUMBER IS NOT AN OPTION PLEASE TRY AGAIN"
 85              GOTO 10
 86          END
 87  ;
 88  ;
 89          GOTO 10
 90          RETURN
.END
```

```
.pl vision.robot.cal
.PROGRAM vision.robot.cal
   1 ;
   2 ; THIS PROGRAM EXECUTES THREE SYSTEM PROGRAMS.
   3 ; THE FIRST PROGRAM, u.boot, REBOOTS THE VISION SYSTEM.
   4 ; THIS HAS THE EFFECT OF DELETING ALL THE PROTOTYPES FROM
   5 ; MEMORY.  THEREFORE NEW PROTOTYPES MUST BE TRAINED.
   6 ;
   7 ; THE SECOND PROGRAM, u.k.teach, DETERMINES THE
   8 ; SCALE FACTOR (s.k.scale) USED IN CALIBRATING
   9 ; THE VISION-TO-ROBOT SYSTEM.  THIS PROGRAM
  10 ; RELATES THE CAMERA'S CHARACTERISTICS (ie. FOCAL
  11 ; LENGTH) TO CAMERA PIXEL SIZE.   IT DEFINES THE RATIO OF
  12 ; THE DISTANCE BETWEEN 2 LOCATIONS IN ROBOT COORDINATES
  13 ; TO THE DISTANCE IN PIXEL COORDINATES.
  14 ;
  15 ; A GOOD RANGE FOR s.k.scale IS APPROXIMATELY
  16 ;          1< s.k.scale < 2
  17 ;  where the units are in mm.
  18 ;
  19 ; THE THIRD SYSTEM PROGRAM CALLED IS u.frame.teach
  20 ; WHICH ESTABLISHES THE ROBOT/VISION WORK FRAME.
  21 ;
  22 ;
  23          CALL u.boot; REBOOT THE VISION SYSTEM
  24 ;
  25          TYPE /B, /C8
  26          TYPE "================================================="
  27          TYPE "!                                               !"
  28          TYPE "! THE  FOLLOWING PROGRAMS WILL ESTABLISH THE    !"
  29          TYPE "!            ROBOT-VISION WORK AREA             !"
  30          TYPE "!                                               !"
  31          TYPE "================================================="
  32          TYPE /C2
  33 ;
  34          TYPE " STEP 1 -->  SELECT THE o VIEW CURRENT CAMERA, OPTION", /C1
  35          TYPE " STEP 2 -->  FOCUS THE CAMERA USING THE CALIBRATION"
  36          TYPE "               DISK AS A GUIDE", /C1
  37 ;
  38          PROMPT "====== PRESS <RETURN> TO CONTINUE ======"
  39 ;
  40          CALL u.k.teach; SYSTEM PROGRAM USED TO DETERMINE s.k.scale
  41 ;
  42          TYPE /C1, /B
  43          TYPE "A GOOD VALUE FOR S.K.SCALE IS APPROXIMATELY", /C1
  44          TYPE "          1 < s.k.scale < 2   ", /C1
  45 ;
  46          PROMPT "====== PRESS <RETURN> TO CONTINUE ======"
  47 ;
  48          CALL u.frame.teach; SYSTEM PROGRAM TO WORK FRAME
  49 ;
  50          TYPE /B, /C1
  51          TYPE "============================================"
  52          TYPE "!                                          !"
  53          TYPE "! NOW A CHECK OF THE VISION-TO-ROBOT       !"
  54          TYPE "! TRANSFROMATION WILL BE VERIFIED AND      !"
  55          TYPE "! MODIFIED, IF NECESSARY.                  !"
  56          TYPE "!                                          !"
  57          TYPE "============================================"
  58          TYPE /B, /C1
  59 ;
  60 ;
  61 ;
```

```
;
;
;
;
;
;
        MOVE #s.safe.position; MAKES SURE ARM IS OUT OF CAMERA'S FOV
;
        TYPE "== PLACE THE CALIBRATION DISK IN THE APPOXIMATE =="
        TYPE "==       CENTER OF THE CAMERA'S FOV                ==", /C1
;
        PROMPT "====== PRESS <RETURN> TO CONTINUE ======"
        TYPE /C1
;
;
; THE FOLLOWING WILL INITIALIZE PARAMETERS REQUIRED IN
; THE REST OF THIS PROGRAM.
;
        x.off = 0
        y.off = 0
        z.off = 0
        xc = 0
        yc = 0
        zc = 0
;
        CALL u.take.picture; TAKE A PICTURE
        CALL u.recognize; IDENTIFY OBJECTS
        x = s.proto.id
        CALL u.where.are.you.proto; ANALIZES IMAGE AND PROVIDES
                          ; ITS x & y CENTROID POSITIONS
        xcent = s.k.scale*s.xcenter.blob[x]
        ycent = s.k.scale*s.ycenter.blob[x]
        SET center = camera:TRANS(xcent, ycent, 0, 0, -90, 0)
      ; camera IS THE THE RELATIVE TRANSFROMATION
      ; DETERMINED WHEN u.frame.teach WAS EXECUTED.
        SPEED 100
        MOVE center
;
        TYPE "======================================="
        TYPE "|                                     |"
        TYPE "| LOOK AT THE POSITION OF THE ROBOT   |"
        TYPE "| POINTER TO THAT OF THE CALIBRATION  |"
        TYPE "| DISK.  IF IT IS NOT IN THE CENTER   |"
        TYPE "| ANSWER THE FOLLOWING QUESTIONS      |"
        TYPE "|                                     |"
        TYPE "=======================================", /B, /C1
;
        TYPE "THE POINTER IS OFF CENTER BY + or - _______ mm's"
        PROMPT " IN THE 'X' DIRECTION -->", x.off
        TYPE /C1
        TYPE "THE POINTER IS OFF CENTER BY + or - _______ mm's"
        PROMPT " IN THE 'Y' DIRECTION -->", y.off
        TYPE /C1
        TYPE "THE POINTER IS TOO FAR ABOVE/BELOW THE TABLE BY "
        PROMPT " + or - _______ mm's -->", z.off
        TYPE /C1
;
        CALL camera.center.offset; THIS WILL CORRECT THE OFFSET
        SET spot = camera1:TRANS(xcent, ycent, 0, 0, -90, 0)
        SPEED 25
        MOVE spot
;
;
;
;
;
```

```
128 ;
129 ;
130         TYPE /B
131         TYPE "==> CHECK THE POINTER W.R.T. THE DISK AGAIN <=="
132         TYPE /C1
133         TYPE "THE POINTER IS OFF CENTER BY + or - ______ mm's"
134         PROMPT " IN THE 'X' DIRECTION -->", xc
135         TYPE /C1
136         TYPE "THE POINTER IS OFF CENTER BY + or - ______ mm's"
137         PROMPT " IN THE 'Y' DIRECTION -->", yc
138         TYPE /C1
139         TYPE "THE POINTER IS TOO FAR ABOVE/BELOW THE TABLE BY"
140         PROMPT " + or - ______ mm's -->", zc
141         TYPE /C1
142 ;
143         CALL camera.center.offset
144         SET spot = camera1:TRANS(xcent, ycent, 0, 0, -90, 0)
145         SPEED 25
146         MOVE spot
147 ;
148         TYPE "THIS SHOULD BE CLOSE ENOUGH, THE SYSTEM IS NOW"
149         TYPE " READY FOR YOUR CONVIENCE!!!!!!!!!!"
150         TYPE /C1
151 ;
152         PROMPT "=======> PRESS <RETURN> TO CONTINUE <======="
153 ;
154         READY
155 ;
156         CALL vision.menu; GO BACK TO MAIN MENU
.END
```

```
.pl prototype.train
.PROGRAM prototype.train
    1 ;
    2 ; THIS PROGRAM CALLS u.train.blob TO TEACH THE VISION
    3 ; SYSTEM THE VARIOUS PROTOTYPES THE USER IS INTERESTED
    4 ; IN RECOGNIZING.
    5 ;
    6 ; FOR THE VISION DEMO OPTION, PROTOTYPES 1 AND 2 SHOULD BE
    7 ; TRAINED AS A CIRCLE AND A RECTANGLE, RESPECTIVELY.  THE REASON
    8 ; FOR THIS CONSTRAINT LIES IN THE FACT THAT THE VISION
    9 ; DEMO HAS PROGRAMS WHICH ARE BASED ON THE PROTOTYPES BEING
   10 ; DEFINED WITH; 1 -- CIRCLE, 2 -- RECTANGLE.
   11 ;
   12          TYPE /C11
   13          TYPE "          ===> NOTE <===       "
   14          TYPE /B, /C1
   15          TYPE " IF THE USER IS TRAINING OBJECTS FOR THE VISION DEMO"
   16          TYPE /C1
   17          TYPE "==>  PROTOTYPE 1 SHOULD BE A CIRCLE  <=-", /C1
   18          TYPE "==>  PROTOTYPE 2 SHOULD BE A RECTANGLE  <==", /C3
   19 ;
   20          PROMPT "=======> PRESS <RETURN> TO CONTINUE <======="
   21          CALL u.train.blob
   22          CALL vision.menu; GO BACK TO MAIN MENU
 .END
```

```
.pl blob.store.or.delete
.PROGRAM blob.store.or.delete
   1 ;
   2 ; THIS PROGRAM IS USED WHEN STORING OR DELETING PROTOTYPE
   3 ; INFORMATION INTO/FROM THE VISION SYSTEM'S MEMORY.
   4 ;
   5 ; u.saver IS A MENU DRIVEN SYSTEM PROGRAM WHICH IS CALLED
   6 ; WHEN THE USER WISHES TO COMPLETELY DELETE A PROTOTYPE FROM
   7 ; THE VISION SYSTEM, OR TO COPY THEM FROM THE VISION SYSTEM
   8 ; TO VAL II AND VIS VERSA.  u.saver DEALS ONLY WITH PROTOTYPE
   9 ; INFORMATION, NOT ORIENTATION FEATURES, OR OTHER SWITCHES.
  10 ;
  11 ; u.delete.blob WOULD BE USED IF THE USER WISHED TO
  12 ; DELETE ONLY A PROTOTYPE'S FEATURES FROM MEMORY.
  13 ;
  14 ; u.boot REBOOTS THE VISION SYSTEM.  THIS HAS THE EFFECT OF
  15 ; DELETING ALL THE PROTOTYPES FROM MEMORY.
  16 ;
  17 ;
  18      15  TYPE /B, /C11
  19          TYPE "THE FOLLOWING OPTIONS ARE AVAILABLE", /C1
  20          TYPE "  1.  STORE, RECALL, OR DELETE A PROTOTYPE ", /C1
  21          TYPE "  2.  DELETE ONLY A PROTOTYPE'S FEATURES ", /C1
  22          TYPE "  3.  DELETE ALL PROTOTYPES FROM MEMORY ", /C1
  23          TYPE "  4.  GO BACK TO THE VISION LAB MAIN MENU", /C1
  24          TYPE "PLEASE ENTER THE NUMBER OF THE OPTION "
  25          PROMPT " YOU DESIRE -->  ", num
  26 ;
  27          CASE INT(num) OF
  28            VALUE 1:
  29              CALL u.saver; STORE, COPY, OR DELETE A PROTOTYPE
  30            VALUE 2:
  31              TYPE /B
  32              TYPE "ENTER THE NUMBER OF THE PROTOTYPE WHOSE FEATURES "
  33              PROMPT "  YOU WISH TO DELETE --> ", u.delete.blob.num
  34              TYPE /C1
  35              CALL u.delete.blob; DELETE A BLOB'S FEATURES
  36            VALUE 3:
  37              CALL u.boot; DELETE ALL PROTOTYPES FROM MEMORY
  38            VALUE 4:
  39              CALL vision.menu; GO BACK TO MAIN MENU
  40            ANY
  41              TYPE "THAT NUMBER IS NOT AN OPTION, PLEASE TRY AGAIN"
  42              GOTO 15
  43          END
  44 ;
  45          CALL vision.menu; GO BACK TO MAIN MENU
.END
```

```
.pl vision.demo.menu
.PROGRAM vision.demo.menu
    1 ;
    2 ; THIS PROGRAM PROVIDES A MENU FOR SELECTING VARIOUS CHOICES
    3 ; OF VISION DEMOS.  EACH TIME THE MENU PROGRAM IS EXECUTED,
    4 ; THE VISION/ROBOT INTERFACE IS INITIALIZED BY CALLING A
    5 ; SYSTEM PROGRAM, u.initvision, (LOCATED ON THE UNIVISION1.PG
    6 ; FLOPPY DISK).
    7 ;
    8         CALL u.initvision; INITIALIZE THE VISION/ROBOT INTERFACE
    9         u.new.picture = 1; SYSTEM LOOKS AT CURRENT PICTURE
   10         u.numblob.id = -1; VISION SYSTEM WILL IDENTIFY ALL OBJECTS IN FOV
   11         u.numblob = 1; BLOB 1 WILL BE COMPARED TO ALL
   12                       ; KNOWN PROTOTYPES
   13         u.blobnum = 2; set this number to the max number of
   14                       ; of blobs expected in the camera's FOV.
   15         u.picture.fast = 0; set slow mode
   16 ;
   17     10  TYPE /B, /C3; BEEPS THE SCREEN AND SKIPS 3 LINES
   18 ;
   19         TYPE "*********************************************"
   20         TYPE "*                                           *"
   21         TYPE "*     WELCOME TO AFIT'S VISION DEMO         *"
   22         TYPE "*                                           *"
   23         TYPE "*********************************************"
   24 ;
   25         SPEED 75; robot speed is set to 75% of its normal value
   26         READY
   27         TYPE /C1, /B
   28         TYPE "  THE FOLLOWING DEMOS ARE AVAILABLE   "
   29         TYPE /C1
   30         TYPE "    1.  OBJECT IDENTIFICATION   ", /C1
   31         TYPE "    2.  OBJECT IDENTIFICATION AND ROBOT POSITIONING ", /C1
   32         TYPE "    3.  DIRECT ROBOT POSITIONING  ", /C1
   33         TYPE "    4.  GO BACK TO THE VISION LAB MAIN MENU", /C1
   34 ;
   35         TYPE "PLEASE ENTER THE NUMBER OF THE DEMO "
   36         PROMPT "  YOU WISH TO VIEW --->  ", num
   37 ;
   38         CASE INT(num) OF
   39           VALUE 1:
   40             CALL ident.blobs; PROGRAM TO IDENTIFY BLOBS
   41           VALUE 2:
   42             CALL ident.blob.point.center; PROGRAM TO IDENTIFY BLOBS,
   43                  ; & DIRECTS THE ROBOT TO MOVE TO THE CENTER OF THE
   44                  ; LARGEST  BLOB IN THE CAMERA'S FIELD OF VIEW (FOV)
   45           VALUE 3:
   46             CALL point.center; DIRECTS THE ROBOT TO MOVE TO THE CENTER
   47                  ; TO THE LARGEST BLOB IN THE CAMERA'S FOV.
   48           VALUE 4:
   49             CALL vision.menu; GO TO MAIN MENU
   50           ANY
   51             TYPE "THAT NUMBER IS NOT AN OPTION PLEASE TRY AGAIN"
   52             GOTO 10
   53         END
   54 ;
   55 ;
   56         PROMPT "====== PRESS <RETURN> TO CONTINUE ======"
   57         GOTO 10
.END
```

```
.pl ident.blobs
.PROGRAM ident.blobs
    1 ;
    2 ; THIS PROGRAM CALL A SYSTEM PROGRAM, u.take.picture,
    3 ; WHICH ASKS THE VISION SYSTEM TO TAKE A PICTURE.
    4 ; THIS PROGRAM PROVIDES THE NUMBER OF BLOBS IN THE
    5 ; CAMERA'S FOV.
    6 ;
    7 ; AFTER COMPLETION OF THIS PROGRAM, u.recognize, IS
    8 ; CALLED TO IDENTIFY THE OBJECTS OBSERVED WHEN THE LAST
    9 ; PICTURE WAS TAKEN.
   10 ;
   11          CALL u.take.picture; ASK THE VISION SYSTEM TO TAKE A PICTURE
   12          nb = s.blob.count
   13          TYPE /B, /C1
   14          IF s.error THEN
   15              TYPE "BAD ROUTINE - ERROR CODE IS -->", /D, s.system.error
   16          ELSE
   17              TYPE "GOOD PICTURE -  THERE ARE ", /D, nb
   18              TYPE /B, "BLOB(S) IN THE CAMERA'S VIEW!"
   19          END
   20          CALL u.blink.blob; OUTLINE THE BLOB IN WHITE
   21          CALL u.recognize; IDENTIFY THE OBJECTS
   22          TYPE /B
   23          TYPE /C1, "****************************************"
   24          x = s.proto.id; SET NUMBER OF THE LARGEST PROTOTYPE
   25                        ; IDENTIFIED.
   26 ;
   27 ; IT SHOULD BE NOTED HERE THAT THE USER MUST PREVIOUSLY TRAIN
   28 ; THESE OBJECTS, WITH THE APPROPRIATE NUMBER, TO THE VISION
   29 ; SYSTEM.  FOR EXAMPLE, OBJECT "1" IS TRAINED TO BE A CIRCLE
   30 ; AND OBJECT 2 IS TRAINED TO BE A RECTANGLE.
   31 ; ALSO IF AN OBJECT IS NOT RECOGNIZED BY THE SYSTEM
   32 ;  ----   s.proto.id = 0  ----------
   33 ;
   34          CASE INT(x) OF
   35            VALUE 0:
   36              TYPE "* I DON'T RECOGNIZE ANY OBJECTS         *"
   37            VALUE 1:
   38              TYPE "* THE LARGEST OBJECT IS A CIRCLE        *"
   39            VALUE 2:
   40              TYPE "* THE LARGEST OBJECT IS A RECTANGLE     *"
   41          END
   42          TYPE "****************************************"
   43          TYPE /C1
   44          CALL locate; ASK THE VISION SYSTEM TO GIVE THE
   45                     ; CENTROID POSITION OF THE OBJECT.
   46          RETURN
 .END
```

```
.pl locate
.PROGRAM locate
    1 ;
    2 ; THIS PROGRAM CALLS u.where.are.you.proto TO DETERMINE
    3 ; AN OBJECTS CENTROID AND ORIENTATION.
    4 ;
    5 ; THE x & y COORDINATES ARE SCALED BY THE SCALE FACTOR, s.k.scale,
    6 ; WHICH DEFINES THE RATIO OF THE DISTANCE BETWEEN 2 LOCATIONS
    7 ; IN ROBOT COORDINATES TO THE DISTANCE IN PIXEL COORDINATES.
    8 ;
    9           CALL u.where.are.you.proto
   10           xcent = s.k.scale*s.xcenter.blob[x]
   11           ycent = s.k.scale*s.ycenter.blob[x]
   12           orien = 180*s.orient.blob[x]/PI
   13           TYPE "THE X COORDINATE OF THE CENTROID (in mms) IS", /D, xcent
   14           TYPE /C1
   15           TYPE "THE Y COORDINATE OF THE CENNTROID (in mms) IS", /D, ycent
   16           TYPE /C1
   17           TYPE "THE ORIENTATION IS", /D, orien
   18           TYPE /C2
   19           RETURN
.END
```

```
.pl ident.blob.point.center
.PROGRAM ident.blob.point.center
    1 ;
    2 ; THIS PROGRAMS CALLS VARIOUS PROGRAMS TO ENABLE THE
    3 ; VISION SYSTEM TO IDENTIFY AN OBJECT AND PROVIDE
    4 ; THE APPROPRIATE COMMANDS AND COORDINATES FOR THE ROBOT
    5 ; TO POINT TO THE CENTROID OF THE OBJECT.
    6 ;
    7         MOVE #s.safe.position; MOVE THE ARM TO A POSITION
    8                              ; OUTSIDE THE CAMERA'A FOV.
    9         TYPE /B, /C2
   10         CALL ident.blobs; VISION SYSTEM TAKES A PICTURE
   11                         ; IDENTIFIES BLOBS AND PROVIDES THE
   12                         ; OBJECTS X AND Y CENTROID POSITION.
   13         CALL camera.center.offset; THIS PROGRAM PROVIDES THE
   14                         ; CORRECTION FOR THE OFFSET
   15                         ; BETWEEN THE OBJECT'S CENTROID
   16                         ; AND THE POSITION OF THE POINTER
   17 ;
   18 ; THE NEXT EXPRESSION DEFINES THE LOCATION OF THE OBJECT
   19 ; IN THE ROBOT REFERENCE FRAME.
   20         SET spot = camera1:TRANS(xcent, ycent, 0, orien, -90, 0)
   21         SPEED 75
   22         MOVE spot; MOVE ARM TO THE DESIRED LOCATION
   23         RETURN
.END
```

```
.pl camera.center.offset
.PROGRAM camera.center.offset
   1 ;
   2 ; THIS PROGRAM SIMPLY PROVIDES THE CORRECTION FOR THE OFFSET
   3 ; BETWEEN THE OBJECT'S CENTROID AND THE POSITION OF THE ROBOT
   4 ; POINTER.
   5 ;
   6          SET camera1 = SHIFT(camera BY x.off+xc, y.off+yc, z.off+zc)
   7          RETURN
.END
```

```
.pl point.center
.PROGRAM point.center
    1 ;
    2 ; THIS PROGRAM IS BUILT FOR SPEED.  IT SIMPLY
    3 ; INSTRUCTS THE VISION SYSTEM TO TAKE A PICTURE, BY
    4 ; CALLING u.take.picture.  THE PROGRAM THEN IDENTIFIES
    5 ; THE OBJECTS USING u.recognize.  AND FINALLY, THE
    6 ; OBJECT'S CENTROID IS DETERMINED BY u.where.are.you.proto.
    7 ;
    8      10  MOVE #s.safe.position; MOVES ARM TO A POSITION
    9 ; OUTSIDE THE CAMERA'S FOV.
   10          CALL u.take.picture; VISION SYSTEM TAKES PICTURE
   11          IF s.blob.count > 0 THEN; CHECK FOR BLOBS IN FOV
   12              CALL u.recognize; IDENTIFY BLOBS
   13              x = s.proto.id
   14              CALL u.where.are.you.proto; PROVIDES OBJECT'S CENTROID
   15              xcent = s.K.scale*s.xcenter.blob[x]
   16              TYPE /C1
   17              TYPE "THE X COORDINATE OF THE CENTROID (in mms) IS", /D, xcent
   18              ycent = s.K.scale*s.ycenter.blob[x]
   19              TYPE "THE Y COORDINATE OF THE CENTROID (in mms) IS", /D, ycent
   20              orien = 180*s.orient.blob[x]/PI
   21              TYPE "ORIENTATION IN DEGREES IS", /D, orien
   22              TYPE /C2
   23              CALL camera.center.offset
   24              SET spot = camera1:TRANS(xcent, ycent, 0, orien, -90, 0)
   25 ;
   26              test = INRANGE(spot)
   27              IF test > 0 THEN
   28                  TYPE /2, /C1, "     ===> WARNING <===", /C1
   29                  TYPE "THE OBJECT IS OUT OF THE ROBOT'S REACH"
   30                  TYPE " PLEASE MOVE THE OBJECT AND TRY AGAIN ", /C1
   31                  PROMPT "=======> PRESS <RETURN> TO CONTINUE <======="
   32                  GOTO 10
   33              ELSE
   34                  SPEED 100
   35                  MOVE spot
   36              END
   37          ELSE
   38              TYPE /B, /C2
   39              TYPE /B, "THERE ARE NO BLOBS IN THE CAMERA'S VIEW"
   40              TYPE /C2
   41              TYPE "PLEASE PLACE AN OBJECT ON THE TABLE "
   42              TYPE "AND RERUN THE PROGRAM", /C2
   43          END
   44          RETURN
.END
```

```
.pl track.tars
.PROGRAM track.tars
   1 ;
   2 ; This program exhibits the charateristics of a Static look-and-
   3 ; move visual servo system during vision-robot system (VRS) motion.
   4 ;
   5 ; Before executing this program, ensure that there aren't any
   6 ; prototypes in the vision system's memory.  If there are, type
   7 ; EX U.BOOT, before executing this program.  This is necessary
   8 ; since this program will track any target placed in the camera's
   9 ; FOV.
  10 ;
  11 ; The program instructs the vision system to take a picture and
  12 ; provide the x any y offsets of the centroid of any target
  13 ; in the camera's FOV.
  14 ;
  15 ; Next a program entitled, center, is called.  Center
  16 ; provides the neccessary vision information to provide the x and y
  17 ; centroid positions for correction to the robot's position.
  18 ;
  19 ;
  20 ; The DO loop will repeat itself until the target is centered in the
  21 ; camera's FOV, and the arm has stopped.
  22 ;
  23 ;
  24       5  SPEED 100
  25          MOVE #start.position
  26          BREAK
  27          CALL u.initvision; initialize communication.
  28          u.numblob = 1; Blob 1 will be compared to all prototypes.
  29 ;
  30 ; Check to make sure the ball is in the camera's FOV.
  31 ;
  32      10  CALL u.take.picture
  33          IF s.blob.count < 1 THEN
  34              TYPE /B, /C1
  35              TYPE "*****************************************"
  36              TYPE "*                                       *"
  37              TYPE "*     THE BALL IN NOT IN THE CAMERA'S   *"
  38              TYPE "*        FIELD OF VIEW  (FOV)           *"
  39              TYPE "*                                       *"
  40              TYPE "*****************************************"
  41              TYPE /B, /C1
  42              TYPE "  ======> PLEASE POSITION THE BALL IN THE CAMERA'S FOV"
  43              TYPE /B, /C1
  44              PROMPT "  ======> PRESS <RETURN> TO CONTINUE <====== "
  45              GOTO 10
  46          ELSE
  47              HERE current; define current as present location.
  48      20      DO
  49                  TIMER 1 = 0
  50                  u.new.picture = 1
  51                  u.fast = 0
  52                  CALL u.where.are.you.proto; take picture and set
  53                                            ; x, y centroids
  54                  TYPE "The time to set the vision info was", TIMER(1), " se
  55                  CALL center; This program instructs the
  56                                            ; robot to center the camera
  57                                            ; over the target.
  58              UNTIL ball == 0
  59              GOTO 20
  60          END
.END
```

```
.pl center
.PROGRAM center
   1 ;
   2 ; THIS PROGRAM CHECKS TO SEE IF THE CENTROID OF THE
   3 ; BALL IS IN THE CENTER OF THE CAMERA'S FOV.
   4 ;
   5          ball = 1
   6          xcen = s.K.scale*s.xcenter.blob[]
   7          ycen = -s.K.scale*s.ycenter.blob[]
   8          TYPE "X POS IS ->", xcen
   9          TYPE "Y POS IS ->", ycen
  10          TYPE /C1
  11          IF ((ABS(xcen) < 10) AND (ABS(ycen) < 10)) THEN
  12              ball = 0
  13          ELSE
  14              FOR i = 1 TO 10
  15                  SET current = SHIFT(current BY xcen/10, ycen/15)
  16                  MOVE current
  17              END
  18          END
  19          RETURN
.END
```

```
.pl track.targ.scat
.PROGRAM track.targ.scat
   1 ;
   2 ; This program exhibits the charateristics of a Static look-and-
   3 ; move visual servo system during vision-robot system (VRS>) motion.
   4 ;
   5 ; The program instructs the vision system to take a picture and
   6 ; provide the x any y offsets of the centroid of any recognized balls
   7 ; in the camera's FOV.
   8 ;
   9 ; Once the ball has been identified by the system program, u.recognize,
  10 ; a program entitled, center.scat, is called.  Center.scat
  11 ; provides the neccessary vision information to provide the x and y
  12 ; centroid positions for correction to the robot's position.
  13 ;
  14 ; z position is determined by the size of the ball recognized.  The
  15 ; larger the ball, the closer the robot is to its desired objective.
  16 ;
  17 ; The DO loop will repeat itself until the ball is centered in the
  18 ; camera's FOV, and the arm has stopped at a predetermined height
  19 ; above the ball.
  20 ;
  21 ; See the program, search.track, for additional comments
  22 ; pertaining to the general use of this program.
  23 ;
  24       5  SPEED 100
  25          MOVE #start.position
  26          BREAK
  27          CALL u.initvision; initialize communication.
  28          u.numblob = 1; Blob 1 will be compared to all prototypes.
  29 ;
  30 ; Check to make sure the ball is in the camera's FOV.
  31 ;
  32      10  CALL u.take.picture
  33          IF s.blob.count < 1 THEN
  34              TYPE /B, /C1
  35              TYPE "*****************************************"
  36              TYPE "*                                       *"
  37              TYPE "*     THE BALL IN NOT IN THE CAMERA'S   *"
  38              TYPE "*        FIELD OF VIEW  (FOV)           *"
  39              TYPE "*                                       *"
  40              TYPE "*****************************************"
  41              TYPE /B, /C1
  42              TYPE "  =======> PLEASE POSITION THE BALL IN THE CAMERA'S FOV"
  43              TYPE /B, /C1
  44              PROMPT "  ======> PRESS <RETURN> TO CONTINUE <====== "
  45              GOTO 10
  46          ELSE
  47              HERE current; define current as present location.
  48              TIMER 2 = 0
  49              DO
  50                  TIMER 1 = 0
  51                  BREAK
  52                  u.new.picture = 1
  53                  u.fast = 0
  54                  CALL u.where.are.you.proto; take picture and set
  55                                      ; x, y centroids
  56                  CALL u.recognize; Identify the ball.
  57                  id = s.proto.id; which ball was identified.
  58 ;
  59 ;
  60 ;
```

```
61 ;
62 ;
63 ;
64 ;
65                TYPE /B, "**************************************************
66                TYPE "The ball recognized is prototype #", id
67                TYPE "The time to get the vision info wa", TIMER(1), " se
68                CALL center.scat; This program instructs the
69                                ; robot to move closer and
70                                ; centered on the ball.
71            UNTIL ball == 0
72            TYPE TIMER(2)
73        END
74        PROMPT "======> PRESS <RETURN> TO RUN THE PROGRAM AGAIN <====="
75        GOTO 5
76        RETURN
.END
```

```
.pl center.scat
.PROGRAM center.scat
   1 ;
   2 ; This program checks to see if the centroid of the ball is in
   3 ; the center of the camera's FOV.  Also, depending to which
   4 ; prototype was identified, the camera will move down in the
   5 ; '-z' direction an appropriate, predetermined, amount.
   6 ;
   7 ; The x and y locations (s.xcenter.blob[] and s.ycenter.blob[])
   8 ; provided by u.where.are.you.proto are scaled by the scale
   9 ; factor, s.K.scale.  s.K.scale is determined when the program u.
  10 ; u.K.teach is executed.  See the program, search.center, for a
  11 ; further explaination of s.K.scale.
  12 ;
  13 ; xcen and ycen are the resultant values when the x and y locations
  14 ; are scaled.  It should be noted that ycen is the negative value
  15 ; of s.ycenter.blob[].  This occurs since the robot's x,y frame is
  16 ; different than the camera's x,y frame.
  17 ; It should also be noted that, ycen, when scaled by s.K.scale
  18 ; does not provide accurate results.  Through experimentation it
  19 ; was found that if ycen is multiplied by 2/3's, good results occur.
  20 ;
  21 ; The ball will be considered 'centered' if the centroid is within
  22 ; '+ or -' 10 mm.  Once the object is identified as a prototype
  23 ; greater than #6. the robot will not move any closer to the ball.
  24 ;
  25 ; The FOR loop in this program breaks up the desired robot motion
  26 ; into only 10 steps.  This allows smooth movement with minimal
  27 ; overshoot of the target.  As can be seen in the loop, depending
  28 ; on which ball is identified, the arm will move down a designated
  29 ; amount.  Since the prototypes were trained at 25 mm increments
  30 ; the value of downward movements should be factors of 25.
  31 ;
  32          SPEED 50  ALWAYS
  33          ball = 1
  34          xcen = s.K.scale*s.xcenter.blob[id]
  35          ycen = -s.K.scale*s.ycenter.blob[id]
  36          TYPE "X POS IS ->", xcen
  37          TYPE "Y POS IS ->", ycen
  38          TYPE "**************************************************", /C1
  39 ;
  40 ; Check to see if the ball is centered and close enough to the ball.
  41 ;
  42          IF ((ABS(xcen) < 10) AND (ABS(ycen) < 10) AND ((id < 1) OR (id >=
  43              ball = 0
  44          ELSE
  45              FOR i = 1 TO 10
  46                  CASE INT(id) OF
  47                    VALUE 0, 9:
  48                      zoom = 0
  49                    VALUE 1, 2, 3:
  50                      zoom = -150
  51                    VALUE 4:
  52                      zoom = -75
  53                    VALUE 5, 7:
  54                      zoom = -50
  55                    VALUE 6, 8:
  56                      zoom = -25
  57                  END
  58 ;
  59 ;
  60 ;
```

```
 62 ;
 63 ;
 64 ;
 65 ; The next IF statement accounts for the initial offset between the
 66 ; position of the camera mount with respect to the PUMA end effector.
 67 ; See search.center for an explanation.
 68 ;
 69                 IF ((id <> 0) AND (id < 4)) THEN
 70                     SET current = SHIFT(current BY xcen/10, ycen/15+3.5, z
 71                 ELSE
 72                     SET current = SHIFT(current BY xcen/10, ycen/15, zoom/
 73                 END
 74             END
 75             MOVE current
 76         END
 77         RETURN
.END
```

```
.PROGRAM search
   1 ; This program will go through a "square conical" search
   2 ; (going from the largest possible area, to the smallest)
   3 ; in an attempt to acquire a target (ball) in the search area.
   4 ;
   5 ; Once the ball has been acquired by the vision system,
   6 ; a program entitled, search.track, is called.  This
   7 ; program will instruct the robot to center the ball in the
   8 ; camera's FOV and lower the camera closer to the ball.
   9 ;
  10 ; If the ball is not acquired during the first search, the
  11 ; search will be accomplished over and over, until a ball is
  12 ; placed in the search area, or the user terminates the program.
  13 ;
  14 ; Once the ball has been acquired, the system will go into a
  15 ; continuous "tracking" mode.  In this mode, the user can move
  16 ; the ball anywhere in the camera's FOV, and the robot/vision
  17 ; system will track the ball (the robot will not move closer to
  18 ; the ball, just track it).
  19 ;
  20 ; If the ball is removed completely from the camera's FOV, the
  21 ; search will start over from the beginning.  The search's
  22 ; starting point, #search.start, is defined as a VAL II precision
  23 ; (refer to a VAL II User's Guide for a definition of precision point).
  24 ;
  25          TIMER 1 = 0
  26     50   SPEED 100
  27          MOVE #search.start; This is the predifined starting point
   8 ;
  29          count = 0; Initialize counter used to instruct the
  30                   ; program to accomplish tracking only.
  31          m = 0; Initialize counter for square conical search.
  32          BREAK; Next command waits until robot motion stops.
  33          SPEED 50  ALWAYS
  34       5  m = m+1
  35          CALL u.initvision; Initialize vision/robot communication
  36          u.numblob = 1; Blob 1 will be compared to all prototypes.
  37 ;
  38          HERE start
  39          CALL u.take.picture; System program which checks for blobs.
  40 ;
  41 ; The following IF-THEN-ELSE is used to conduct the square search
  42 ; and acquisition of the ball.
  43 ;
  44 ; The search will continue until, u.take.picture, finds a blob
  45 ; with an area greater than the minimum blob pixels  (set by user)
  46 ; in the camera's FOV.
  47 ;
  48          IF s.blob.count < 1 THEN
  49              CASE INT(m) OF
  50                VALUE 1, 2, 3, 12, 13:
  51                  MOVE SHIFT(start BY 80)
  52                VALUE 4, 5, 6, 14:
  53                  MOVE SHIFT(start BY , 65)
   4                VALUE 7, 8, 9, 15:
  55                  MOVE SHIFT(start BY -80)
  56                VALUE 10, 11, 16:
  57                  MOVE SHIFT(start BY , -65)
  58 ;
  59 ;
  60 ;
```

```
 62 ;
 63 ;
 64 ;
 65                ANY
 66                  TYPE /B
 67                  TYPE "SEARCH HAS BEEN COMPLETED, THE BALL WAS"
 68                  TYPE "NOT IN THE SEARCH AREA.", /C1
 69                  TYPE "THE SEARCH WILL NOW REPEAT ITSELF.", /B
 70 ;
 71                  TYPE "PLEASE PLACE THE BALL IN THE SEARCH AREA,"
 72                  TYPE "OR TERMINATE THE PROGRAM BY TYPING,", /C1
 73                  TYPE "    'A'  FOLLOWING BY PRESSING THE <RETURN> KEY", /C1
 74 ;
 75                  DELAY 2; Delay program execution by 2 seconds.
 76                  GOTO 50
 77              END
 78              BREAK
 79          ELSE
 80 ;
 81 ; The following DO loop will continue forever, unless the user
 82 ; wishes to terminate.  Th terminate program execution: press
 83 ; "A" followed by <RETURN>.
 84 ;
 85 ; If the ball is removed from the camera's FOV. The search will
 86 ; start over.
 87 ;
 88              DO
 89                  CALL search.track
 90                  IF lost <> 0 THEN
 91                      TYPE /B, "THE BALL HAS BEEN ACQUIRED", /C1
 92                      TYPE /B, "TO STOP THIS PROGRAM:", /C1
 93                      TYPE "       TYPE 'A' FOLLOWED BY PRESSING THE "
 94                      TYPE "       <RETURN> KEY.", /C1
 95                  END
 96 ;
 97                  count = 15; Reset counter.  This is to prevent
 98                            ; the robot from moving closer to the
 99                            ; ball.  However tracking can still be
100                            ; accomplished.
101                  CALL u.take.picture; Check to see if ball is
102                            ; is still in the camera's FOV.
103              UNTIL s.blob.count == 0
104 ;
105              TYPE /C1, "*************************************************"
106              TYPE "*                                                *"
107              TYPE "*       THE BALL HAS BEEN REMOVED FROM           *"
108              TYPE "*     THE CAMERA'S FOV, THE SEARCH WILL          *"
109              TYPE "*    REPEAT UNTIL THE BALL IS REACQUIRED         *"
110              TYPE "*                                                *"
111              TYPE "*************************************************", /C2
112 ;
113              GOTO 50
114          END
115          GOTO 5
.END
```

```
.pl search.track
.PROGRAM search.track
   1 ;
   2 ; This program instructs the vision system to take a picture
   3 ; and give the x and y offsets of the centroid of any recognized
   4 ; balls in the camera's FOV.
   5 ;
   6 ; Once the ball has been identified by the system program,
   7 ; u.recognize, a program entitled, search.center, is called.
   8 ; Search.center provides the necessary vision information
   9 ; to provide the x and y centroid positions for correction to
  10 ; the robot's position.
  11 ;
  12 ; z position is determined by the size of the ball recognized.
  13 ; The larger the ball, the closer the robot is to its desired
  14 ; objective.
  15 ;
  16 ; The DO loop will repeat itself until the ball is centered in
  17 ; the camera's FOV, and the arm has stopped at a predetermined
  18 ; height above the ball.
  19 ;
  20 ; NOTE:  The reason the robot is not lowered all the way to the
  21 ; ball is that if the ball encompasses a majority of the camera's
  22 ; FOV, the centroid position can not be calculated accurately, and
  23 ; the robot/vision system will overshoot back and forth for an
  24 ; extended period of time.
  25 ;
  26 ; NOTE: The different ball sizes are trained to the vision system
  27 ; ahead of time.  Using a ball alleviates the problem of not having an
  28 ; autofocus camera lens.  Even though the ball gets blurry as the
  29 ; camera gets closer, it will still remain circular in the camera's
  30 ; 2-dimensional FOV.  u.train.blob was used to train the vision system.
  31 ; Training occurs as follows:
  32 ;  1) The user defines a starting position (the one used here is
  33 ;     designated as the following presion point, #start.position).
  34 ;  2) u.train.blob is called, the first object trained will be
  35 ;     designated as prototype '1'.
  36 ;  3) Once training has been accomplished, the user should verify
  37 ;     the training was succesful.  This is accomplished by using
  38 ;     the light pen to select the  o PROTOTYPE MENU, and then
  39 ;     checking the o DISPLAY DISCRIMATION MATRIX.  There should be
  40 ;     a large block next to the CURRENT SET row.  If not, the ball
  41 ;     should be retrained.
  42 ; To make life easier, the balls have been already trained at 25 mm
  43 ; intervals and stored on tape.  The tape is entitled 'VISION SERVO
  44 ;.CONTROL TARGETS'.  These can be loaded directly, without any additional
  45 ; training.
  46 ;
  47 ; The timer is used to delay program execution of the program
  48 ; and display the amount of time required for the vision system
  49 ; to process the image and send the required data back to the
  50 ; robot processor.
  51 ;
  52 ; Since this particular program is working as a Static look-and-
  53 ; move system, the delay is required to ensure the robot has
  54 ; stopped moving before the next picture is taken.  The reason for
  55 ; this is; if the vision system takes a picture in-between trained
  56 ; ball sizes, the picture taken will not be an identifiable object.
  57 ; (The area of the ball will not match any area trained earlier)
  58 ; Thus the vision system will not know how far down to move the arm.
  59 ;
  60 ;
```

```
 62 ;
 63 ;
 64 ;
 65          SPEED 25
 66          HERE current; Define current as present location.
 67 ;
 68 ; The following 2 parameters initialized are used incase the vision
  9 ; system gets lost, ie. it thinks it has the ball in its FOV, but
 70 ; in reality it doesn't.
 71 ;
 72          xcen = 0
 73          xcen1 = 1
 74 ;
 75          DO
 76              CALL u.initvision; Initialize communication.
 77              TIMER 1 = 0
 78              BREAK
 79              u.new.picture = 1
 80              u.fast = 0
 81              CALL u.where.are.you.proto; take picture and set
 82                                         ; x, y centroids
 83              CALL u.recognize; Identify the ball.
 84              id = s.proto.id; This comes form u.recognize.
 85                               ; It identifies which ball was identified.
 86 ;
 87              TYPE /B, "*************************************************"
 88              TYPE "The ball recognized is prototype #", id
 89              TYPE "The time to get the vision info was", TIMER(1), /C1
 90 ;
 91              CALL search.center; This program instructs the robot
 92                       ; to move closer and centered on the ball.
 93          UNTIL ball == 0
 94          TYPE "TIME IS ", TIMER(11)
 95          RETURN
.END
```

```
.pl search.center
.PROGRAM search.center
   1 ;
   2 ; This program checks to see if the centroid of the ball
   3 ; is in the center of the camera's FOV.  Also, depending on
   4 ; which prototype was identified, the camera will move down
   5 ; in the '-z' direction an appropriate, predetermined, amount.
   6 ;
   7 ; The x and y locations (s.xcenter.blob[] and s.ycenter.blob[])
   8 ; provided by u.where.are.you.proto are scaled by the scale
   9 ; factor, s.K.scale.  s.K.scale is determined when the program
  10 ; u.K.teach is executed.  It is important to get an accurate scale
  11 ; factor, with the camera mounted to the robot.  The following
  12 ; should be accomplished to ensure this:
  13 ;  1) Put the arm in the starting position (#start.position).
  14 ;  2) Execute u.K.teach, make sure the disK will be in the camera's
  15 ;     FOV when the robot is moved out of the way.
  16 ; The value of s.K.scale should be approximately .5.
  17 ;
  18 ; xcen and ycen are the resultant values when the x and y locations
  19 ; are scaled.  It should be noted that  ycen is the negative value
  20 ; of s.ycenter.blob[].  This occurs since the robot's x,y frame is
  21 ; different than the camera's x,y frame.
  22 ; It should also be noted that, ycen, when scaled by
  23 ; s.K.scale does not provide accurate results.  Throush experimentation
  24 ; it was found that if ycen is multiplied by 2/3's, sood results occur.
  25 ;
  26 ; The ball will be considered centered if the centroid is within '+ or -'
  27 ; 10 mm.  Once the object is identified as a prototype 9
  28 ;  the robot will not move any closer to the ball.
   9 ;
  30 ; The FOR loop is used to breaK up the robot motion into 10 steps.
  31 ; This allows smooth movement with minimal overshoot of the target.
  32 ; As can be seen in the loop, depending on which ball identified,
  33 ; the arm will move down a desisnated amount.  Since the prototypes
  34 ; were trained at 25 mm increments, the value of downward movements
  35 ; should be factors of 25.
  36 ;
  37         SPEED 50  ALWAYS
  38         lost = 1
  39         ball = 1
  40         xcen = s.K.scale*s.xcenter.blob[id]
  41         ycen = -s.K.scale*s.ycenter.blob[id]
  42         TYPE "The ball's centroid in the 'X' direction is", xcen
  43         TYPE "The ball's centroid in the 'Y' direction is", ycen
  44         TYPE "************************************************", /C1
  45 ;
  46 ; Check to see if the vision system is huns.    ,
  47 ;
  48         IF xcen == xcen1 THEN
  49             TYPE /C1, "****************************************"
  50             TYPE "*                                      *"
  51             TYPE "*       THE BALL HAS BEEN LOST         *"
  52             TYPE "*                                      *"
  53             TYPE "****************************************", /C2
  54             lost = 0
   5             ball = 0
  56         ELSE
  57 ;
  58 ; Check to see if the ball is centered and close enoush to the ball.
  59 ;
  60 ;
  61 ;
```

```
 63 ;
 64 ;
 65 ;
 66              IF ((ABS(xcen) < 10) AND (ABS(ycen) < 10) AND (id > 8)) THEN
 67                  ball = 0
 68 ;
 69              ELSE
 70                  count = count+1
 71                  FOR i = 1 TO 10
 72                      CASE INT(id) OF
 73                        VALUE 0:
 74 ;
 75 ; The following IF statement is to account for the situation when
 76 ; the ball's initial position in the search area is at a distance
 77 ; greater than that trained to the vision system.  The search area
 78 ; is such that at no point in search area will the ball be at a
 79 ; distance greater than 75 mm from an initial training position.
 80 ;
 81                            IF count <= 4 THEN
 82                                zoom = -25.
 83                            ELSE
 84                                zoom = 0
 85                            END
 86                        VALUE 1, 2, 3:
 87                          zoom = -150
 88                        VALUE 4:
 89                          zoom = -75
 90                        VALUE 5, 7:
 91                          zoom = -50
 92                        VALUE 6, 8:
 93                          zoom = -25
 94                        VALUE 9:
  5                         zoom = 0
 96                      END
 97 ;
 98 ; The following IF statement is used once the ball has been
 99 ; acquired and zoomed into.  The vision/robot system will just
100 ; become a tractor.  This is a safety measure, because in the
101 ; tracking mode, if only part of the ball is in the camera's
102 ; FOV when a picture is taken, the prototype identified might not
103 ; be what the actual prototype is.  For example, if the blob identified
104 ; was now prototype #1, the robot would think that it should lower
105 ; itself 150 mm, thus putting itself out of its possible range of
106 ; motion.
107 ;
108                      IF count > 6 THEN
109                          SET current = SHIFT(current BY xcen/10, ycen/25)
110                          IF ((ABS(xcen) < 10) AND (ABS(ycen) < 10)) THEN
111                              ball = 0
112                          END
113                          GOTO 25
114                      END
115 ;
116 ; The next IF statement accounts for the initial offset between
117 ; the position of the camera mount with respect to the PUMA
118 ; end effector.  This is a problem because x,y,z coordinates in
119 ; the VAL II controller are based on the location of joint 6,
120 ; not joint 3 (where the camera is presently mounted).
121 ;
122 ;
123 ;
124 ;
125 ;
126 ;
127 ;
```

```
129 ;
130                     IF ((id <> 0) AND (id < 3)) THEN
131                         IF m == 2 THEN
132                             SET current = SHIFT(current BY xcen/10+1, ycen)
133                         END
134                         IF ((m == 3) OR (m == 4)) THEN
135                             SET current = SHIFT(current BY xcen/10+2, ycen)
136                         ELSE
137                             SET current = SHIFT(current BY xcen/10, ycen/1)
138                         END
139 ;
140 ;
141                     ELSE
142                         IF id == 3 THEN
143                             SET current = SHIFT(current BY xcen/10, ycen/1)
144                         ELSE
145                             SET current = SHIFT(current BY xcen/10, ycen/1)
146                         END
147                     END
148     25              MOVE current
149                 END
150             END
151         END
152         xcen1 = xcen; This sets up the check for the situation where
153                  ; the vision system had the ball, then lost it.
154         RETURN
.END
```

Task 2: Dynamic Visual Servo Control Program Listings

```
.pl track.tars.dcat
.PROGRAM track.tars.dcat
    1 ;
    2 ; This program exhibits the charateristics of a Dynamic look-and-
    3 ; move visual servo system when the arm moves in a downward direction.
    4 ; The remainins movements are done in the more classical Static look-
    5 ; and-move servoins method.
    6 ;
    7 ; The program instructs the vision system to take a picture and
    8 ; provide the x any y offsets of the centroid of any recosnized balls
    9 ; in the camera's FOV.
   10 ;
   11 ; Once the ball has been identified by the system program, u.recosnize,
   12 ; a program entitled, center.dcat, is called.  Center.dcat
   13 ; provides the neccessary vision information to provide the x and y
   14 ; centroid positions for correction to the robot's position.
   15 ;
   16 ; z position is determined by the size of the ball recosnized.  The
   17 ; larser the ball, the closer the robot is to its desired objective.
   18 ;
   19 ; The DO loop will repeat itself until the ball is centered in the
   20 ; camera's FOV, and the arm has stopped at a predetermined height
   21 ; above the ball.
   22 ;
   23 ; See the program, search.ball.track, for additional comments
   24 ; pertainins to the seneral use of this program.
   25 ;
   26       5  SPEED 100
   27          MOVE #start.position
   28          BREAK
   29          CALL u.initvision; initialize communication.
   30          u.numblob = 1; Blob 1 will be compared to all prototypes.
   31 ;
   32 ; Check to make sure the ball is in the camera's FOV.
   33 ;
   34      10  CALL u.take.picture
   35          IF s.blob.count < 1 THEN
   36              TYPE /B, /C1
   37              TYPE "*******************************************"
   38                                                          *"
   39              TYPE "*
   40              TYPE "*     THE BALL IN NOT IN THE CAMERA'S     *"
   41              TYPE "*        FIELD OF VIEW  (FOV)             *"
   42              TYPE "*                                         *"
   43              TYPE "*******************************************"
   44              TYPE /B, /C1
   45              TYPE "  =======> PLEASE POSITION THE BALL IN THE CAMERA'S FOV"
   46              TYPE /B, /C1
   47              PROMPT "  ======> PRESS <RETURN> TO CONTINUE <====== "
   48              GOTO 10
   49          ELSE
   50              HERE current; define current as present location.
   51              TIMER 2 = 0
   52              DO
   53                  TIMER 1 = 0
   54                  u.new.picture = 1
   55                  u.fast = 0
   56                  CALL u.where.are.you.proto; take picture and set
   57                                             ; x, y centroids
   58                  CALL u.recosnize; Identify the ball.
   59                  id = s.proto.id; which ball was identified.
   60 ;
```

```
 62 ;
 63 ;
 64 ;
 65 ;
 66 ;
 67                TYPE /B, "**************************************************"
 68                TYPE "The ball recognized is prototype #", id
 69                TYPE "The time to set the vision info was", TIMER(1), " sec
 70                CALL center.dcat; This program instructs the
 71                                ; robot to move closer and
 72                                ; centered on the ball.
 73            UNTIL ball == 0
 74            TYPE TIMER(2)
 75        END
 76        PROMPT "======> PRESS <RETURN> TO RUN THE PROGRAM AGAIN <====="
 77        GOTO 5
 78        RETURN
.END
```

```
.Pl center.dcat
.PROGRAM center.dcat
   1 ;
   2 ; This program checks to see if the centroid of the ball is in
   3 ; the center of the camera's FOV.  Also, depending to which
   4 ; prototype was identified, the camera will move down in the
   5 ; '-z' direction an appropriate, predetermined, amount.
   6 ;
   7 ; The x and y locations (s.xcenter.blob[] and s.ycenter.blob[])
   8 ; provided by u.where.are.you.proto are scaled by the scale
   9 ; factor, s.K.scale.  s.K.scale is determined when the program u.
  10 ; u.K.teach is executed.  See the program, search.center, for a
  11 ; further explaination of s.K.scale.
  12 ;
  13 ; xcen and ycen are the resultant values when the x and y locations
  14 ; are scaled.  It should be noted that ycen is the negative value
  15 ; of s.ycenter.blob[].  This occurs since the robot's x,y frame is
  16 ; different than the camera's x,y frame.
  17 ; It should also be noted that, ycen, when scaled by s.K.scale
  18 ; does not provide accurate results.  Through experimentation it
  19 ; was found that if ycen is multiplied by 2/3's, good results occur.
  20 ;
  21 ; The ball will be considered 'centered' if the centroid is within
  22 ; '+ or -' 10 mm.  Once the object is identified as a prototype
  23 ; greater than #8. the robot will not move any closer to the ball.
  24 ;
  25 ; The FOR loop in this program breaks up the desired robot motion
  26 ; into only 2 steps.  This is what allows the dynamic look-and-move
  27 ; visual tracking to occur.  As the arm is moving down, a picture is
   8 ; taken as soon as possible in program execution.  The commands are
  29 ; serial, but a form of parallel operation is possible because once
  30 ; the MOVE command is executed the next commands are addressed.
  31 ; Therefore a picture is requested while the arm is moving form its
  32 ; last MOVE command.  See search.center for information pertaining to
  33 ; the values for downward movement.
  34 ;
  35         SPEED 25  ALWAYS
  36         ball = 1
  37         xcen = s.K.scale*s.xcenter.blob[id]
  38         ycen = -s.K.scale*s.ycenter.blob[id]
  39         TYPE "X POS IS ->", xcen
  40         TYPE "Y POS IS ->", ycen
  41         TYPE "**************************************************", /C1
  42 ;
  43 ; Check to see if the ball is centered and close enough to the ball.
  44 ;
  45         IF ((ABS(xcen) < 10) AND (ABS(ycen) < 10) AND ((id < 1) OR (id >= N
  46             ball = 0
  47         ELSE
  48             FOR i = 1 TO 2
  49                 CASE INT(id) OF
  50                   VALUE 0:
  51                     zoom = 0
  52                   VALUE 1, 2, 3:
  53                     zoom = -150
  54                   VALUE 4, 5, 6:
  55                     zoom = -50
  56                   VALUE 7, 8, 9:
  57                     zoom = 0
  58                 END
                                  155
  59 ;
  60 ;
```

```
 62 ;
 63 ;
 64 ;
 65 ;
 66 ; The next IF statement accounts for the initial offset between the
 67 ; position of the camera mount with respect to the PUMA end effector.
 68 ; See search.center for an explanation.
 69 ;
 70                  IF ((id <> 0) AND (id < 4)) THEN
 71                      SET current = SHIFT(current BY xcen/2, ycen/4+25, zoom)
 72                  ELSE
 73                      IF id == 4 THEN
 74                          SET current = SHIFT(current BY xcen/2, ycen/4-15, )
 75                      ELSE
 76                          SET current = SHIFT(current BY xcen/2, ycen/4, zoo)
 77                      END
 78                  END
 79                  MOVE current
 80              END
 81          END
 82          RETURN
.END
```

Appendix C

User's Manual

Appendix C contains the User's Manual for setting up this particular vision-robot system (VSR) for the various tasks accomplished in my thesis. The Manual provides step-by-step instructions to enable the user to execute the various programs.

Directions for Operating VRS

Step	Direction

===> NOTE <===

It is assumed that the user has read the following references --> (16; 17; 18), before using the VRS.

1. Mount the camera to the camera stand, with the camera pointing towards the black part of the table, when looking at the image through the camera.

2. Load the vision system's operating software by inserting the Univision S/W tape into the "TAPE 0" slot located on the right side of the vision system's video monitor. Depress the "RESTART/RELOAD" button (a blue light should be visible).

3. Turn the key to the "ON" position. The LED by the tape drive will flash while the system is loading.

===> NOTE <===

It takes approximately 5 minutes to load the system. Therefore, the user may accomplish Steps 16-26 to bring up the robot, while loading occurs.

A correct load will results when the Machine Intelligence Univision (3.09-B) logo is displayed.

4. Remove the tape from the drive.

5. Press the light pen to any white part of the screen.

===> NOTE <===

The following steps set up the vision system for Subtask 1. For setting up the vision system for the remaining Subtasks and Tasks accomplish Steps 15.

6. Select: o SYSTEM SETUP

o VIEW CURRENT CAMERA

7. Place the calibration disk in the camera's field of view (FOV).

8. Focus the camera and adjust the aperture to get as clear an image as possible.

9. Press the light pen to any part of the screen and then select the following:

   - o SET THRESHOLD

   Set the threshold value to 128 by pressing the light pen to an arrow until 128 shows up in the upper right corner of the monitor.

   128 provides the best contrast when using white objects against a black background.

10. Select:
   - o QUIT
   - o OPERATING OPTIONS

   Ensure "ONLY" the following options are "ON".

   - o CONNECTIVITY ANALYSIS
   - o FIRST MOMENTS
   - o KEEP ALL BLOBS
   - o NOISE SUPPRESSION
   - o OUTLINE BLOBS
   - o PERIMETERS
   - o RECOGNITION
   - o SECOND MOMENTS

11. Select:
   - o QUIT
   - o QUIT
   - o EXPERT OPERATION
   - o OPERATING OPTIONS

   In addition to the options listed in Step 10, ensure "ONLY" the following options are "ON".

   - o OBJECTS ONLY PROCESSED
   - o PROCESS ALL BLOBS

   ===> NOTE <===

   Other options are possible, refer to (17) for more information.

12. Select:
    - o QUIT
    - o QUIT
    - o UNIVISION
    - o VIEW CURRENT CAMERA

13. READY TO GO !!!!!!

14. Ensure the robot (operating under VAL II) is up and running. If not, accomplish Steps 16-26.

15. ONLY the following options from the various OPERATING OPTIONS should be selected when accomplishing Subtasks 2, 3, 4 and Task 2.

    - o CONNECTIVITY ANALYSIS
    - o FIRST MOMENTS
    - o NOISE SUPPRESSION
    - o OBJECTS ONLY PROCESSED

===> NOTE <===

The following steps will setup the PUMA 560 under the VAL II operating system.

16. Turn of the robot's dumb terminal.

17. Turn the power "ON" the Unimate Computer/Controller.

18. A message should appear asking the user to load VAL II from floppy disk.

19. Insert the 5 & 1/4" floppy disk entitled "VAL II 560.1.4.B" into the Unimate disk drive.

20. Type "Y <RETURN>" to the prompt asking the user to load VAL II.

21. Enter the number "798" for the robot serial number.

22. Approximately 2 minutes later, type "Y <RETURN>" to the prompt asking the user to INITIALIZE the system. A "." system prompt will appear.

===> NOTE <===

Initializing the system has the effect of erasing all programs stored in memory. Therefore, ensure a copy of all software is stored on disk.

23. Remove the Unimation disk and insert the Univision I system program disk 935H3 which contains the interface software.

24. Enter the following:

    ==> LOAD UNIVISION1.PG

    11 s.*.* files and 15 u.*.* files should be loaded and displayed.

25. Remove the Univision disk and insert the Vision Servo Control disk into the Unimate disk drive.

26. Enter the following:

    ==> LOAD VISION.SERVO.CONTROL

    The 19 programs listed in Appendix B should be loaded.

27. To execute Subtask 1, type the following:

        EX VISION.DEMO

    and accomplish the OPTIONS in order to initialize communications, calibrate the systems, train targets, and execute open loop, static look-and-move control.

28. For the remaining Subtasks and Task, the camera must be mounted on the third joint of the PUMA 560 (see Figure 10 in Chapter III) and the table removed from the robot workspace. Put the black part of the table on the floor for a work area. The vision system must also be re-calibrated by executing the following self-explanatory program:

        EX U.K.TEACH

    which determines the camera scale factor (s.k.scale should be approximately equal to 0.5).

29. To execute Subtask 2, delete all prototypes from the vision system's memory by typing the following:

    EX U.BOOT

    Next place a target under the camera's FOV. The system is ready for closed loop, static look-and-move visual servo control. Enter the following:

    EX TRACK.TARG

30. Before executing Subtask 3, the trained targets must be loaded into the vision system's memory. This is accomplished by inserting the vision cassette, labeled "VISION SERVO CONTROL TARGETS", into the vision system's "Tape 0" slot. Next select the following with the light pen:

    - o PROTOTYPE MENU
    - o LOAD ALL PROTOTYPES FROM MEMORY

    The system should now be ready. Place the white ball below the camera and enter the following:

    EX TRACK.TARG.SCAT

31. To execute Subtask 4, simply place the ball anywhere on the black background and enter the following:

    EX SEARCH

32. Finally, to execute Task 2, closed loop, dynamic look-and-move visual servo control, place the target near the center of the camera's FOV and enter the following:

    EX TRACK.TARG.DCAT

33. When finished for the day, follow the manuals for shutting the systems down.

Bibliography

1. Ahluwalia, Rashpal S. and Lynn M. Fogwell. "A Modular Approach to Visual Servoing," Proceedings of the IEEE 1986 International Conference on Robotics and Automation, 2: 943-950. IEEE Computer Society, Washington, 1986.

2. Bamba, Takao, et al. "A Visual Sensor for Arc-welding Robots," Robot Vision. 169-177. IFS(Publications) Ltd., U.K. and Springer-Verlag, Berlin, Heidelberg, New York, 1983.

3. Davis, Capt Dewayne. "Robots 11 Presentation on Robotic Aircraft Refueling." Copy of briefing given at Robots 11 Conference. Wright-Patterson AFB OH, 29 April 1987.

4. D'Azzo, John J. and Constantine H. Houpis. Linear Control Systems analysis and Design (Second Edition). New York: McGraw-Hill Book Company, 1981.

5. Harrell, R. C. et al. "Vision Guidance of a Robotic Tree Fruit Harvester," Intelligent Robots and Computer Vision, 579: 537-545. SPIE, 1985.

6. Hill, John and William T. Park. "Real Time Control of a Robot with a Mobile Camera," Proceedings of the Ninth ISIR. 233-246. Society of Manufacturing Engineers, Michigan, 1979.

7. Kim, Jin Kyoung et al. "Visual Servoing of a Robot Manipulator Using 3-D Information from Hand-Held Camera Motion," Proceedings of the 25th Conference on Decision and Control. 417-422 (December 1986).

8. Koren, Y. Robotics for Engineers, New York: McGraw-Hill Book company, 1985.

9. Leahy, Michael B. Jr. The RAL Hierarchical Control System User's Guide, Version 1.0. Rensselaer Polytechnic Institute, Troy NY, April 1986.

10. Palm, William J. and Ramiro Liscano. "Integrated Design of an End Effector for a Visual Servoing Algorithm," Journal of Robotic Systems, 3: 221-236 (Fall 1986).

11. Pugh, Alan. Robot Vision, U.K. and Springer-Verlag, Berlin, Heidelberg, New York: IFS(Publications) Ltd., 1983.

12. Sanderson, Arthur C. and Lee E. Weiss. "Adaptive Visual Servo Control of Robots," Robot Vision. 107-116. U.K. and Springer-Verlag, Berlin, Heidelberg, New York: IFS(Publications) Ltd., 1983.

13. Smith, John L., et al. "Robotic Concepts for Aircraft Turnaround," Report to USAF System Command, Aeronautical Systems Division/AFWAL/FIER. Wright-Patterson AFB, OH, 15 April 1986.

14. VanderBrug, G. J., et al. "A Vision System for Real Time Control of Robots," Proceedings of the Ninth ISIR. 213-231. Society of Manufacturing Engineers, Michigan, 1979.

15. Ward, M. R., et al. "CONSIGHT: A Practical Vision-Based Robot Guidance System," Proceedings of the Ninth ISIR. 195-211. Society of Manufacturing Engineers, Michigan, 1979.

16. Westinghouse Company, Unimation. Unimate PUMA Mark II Robot, 500 Series Equipment Manual for VAL II and VAL Plus Operating Systems 398U1. Unimation Incorporated, March 1985.

17. Westinghouse Company, Unimation. Preliminary Unimate Industrial Robot Programming Manual User's Guide to VAL II 398T1. Unimation Incorporated, May 1985.

18. Westinghouse Company, Unimation. Preliminary Univision I Robot Vision System Equipment Manual 398L1. Unimation Incorporated, June 1983.

19. Westinghouse Company, Unimation. Interface Description of Interface Between MIC Vision System and Unimations' VAL II Programming Language. Unimation Incorporated.

VITA

Mikel M. Miller was born on 8 May 1960 in Sioux City, Iowa, with most of the credit given to his parents Barbara M. and Dennis A. Miller. He graduated from high school in Williston, North Dakota in 1978 and attended North Dakota State University, Fargo, North Dakota, from which he received the degree of Bachelor of Science in Electrical and Electronic Engineering in November 1982. Upon graduation, he received a commission in the USAF through the Reserve Officer Training Corps program, in which he was a Distinguished Graduate. He married Colleen M. Conlin on 18 December 1982 at Saint Joseph's Church in Williston North Dakota. He entered the Air Force on active duty in April 1983. His first assignment as an Air Force Officer was to the 1000th Satellite Operations Group at Offutt AFB, Nebraska. His job titles included Satellite Systems Engineer and System Integration Engineer; responsible for the system test, integration, and evaluation of upgrades to the Defense Meteorological Satellite Program, Block 5D-2 satellite and associated ground system. While at Offutt, on 31 March 1984, his first child, a boy, Casey C., was born. He left the "Cornhusker" state when assigned to the Air Force Institute of Technology, School of Engineering, at Wright-Patterson AFB, Ohio in May of 1986. On 2 February 1987, his second child, a girl, Krista M., was born. Most of the credit for the birth of his children go to his wife, Colleen.

Permanent address: 814 1st Avenue West
Williston, North Dakota 58801

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION: UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION / AVAILABILITY OF REPORT: Approved for public release; distribution unlimited.
2b. DECLASSIFICATION / DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S): AFIT/GE/ENG/87D-45	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION: School of Engineering; 6b. OFFICE SYMBOL (If applicable): AFIT/ENG	7a. NAME OF MONITORING ORGANIZATION
6c. ADDRESS (City, State, and ZIP Code): Air Force Institute of Technology, Wright-Patterson AFB, OH 45433-6583	7b. ADDRESS (City, State, and ZIP Code)
8a. NAME OF FUNDING / SPONSORING ORGANIZATION: AFWAL/MLTZ; 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State, and ZIP Code): Wright-Patterson AFB, OH 45433	10. SOURCE OF FUNDING NUMBERS: PROGRAM ELEMENT NO; PROJECT NO.; TASK NO; WORK UNIT ACCESSION NO

11. TITLE (Include Security Classification): See Box 19

12. PERSONAL AUTHOR(S): Mikel M. Miller, Capt, USAF

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
MS Thesis	FROM _____ TO _____	1987 December	173

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	
12	09		Robot Vision, Visual Servoing, Vision Vision Servo Control

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

Title: IMPLEMENTATION OF A VISUAL SERVOING SYSTEM FOR EVALUATION OF ROBOTIC REFUELING APPLICATIONS

Thesis Chairman: Matthew Kabrisky, PhD
Professor Of Electrical Engineering

Approved for public release: IAW AFR 190-1
Lynn E. Wolaver 24 Feb 88
Dean for Research and Professional Development
Air Force Institute of Technology
Wright-Patterson AFB OH 45433

20. DISTRIBUTION / AVAILABILITY OF ABSTRACT	21. ABSTRACT SECURITY CLASSIFICATION
☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS	UNCLASSIFIED
22a. NAME OF RESPONSIBLE INDIVIDUAL: Matthew Kabrisky, PhD	22b. TELEPHONE (Include Area Code): (513)-255-5276; 22c. OFFICE SYMBOL: AFIT/ENG

DD Form 1473, JUN 86 — Previous editions are obsolete.

This research effort designed and integrated a visual servo control scheme for a PUMA 560 robot arm that derived its control information from a Machine Intelligence Corporation vision system. The vision system's TV camera was mounted to the PUMA's third joint. The integrated vision-robot system (VRS) used closed loop, static and dynamic, visual servo control techniques techniques to demonstrate the feasibility of ground refueling an aircraft.