OAK RIDGE
NATIONAL LABORATORY

MANAGED BY UT-BATTELLE
FOR THE DEPARTMENT OF ENERGY

ORNL/TM-2004/110

Proof-of-Concept Demonstration Results for Robotic Visual Servo Controllers

September 2004

Prepared by

P. V. Chawda
HERE Program

W. E. Dixon, Ph.D.

T. J. Flynn
HERE Program

E. B. Holcombe
SULI Program

L. J. Love, Ph.D.

J. C. Rowe

UT-BATTELLE

ORNL-27 (2-03)

ORNL/TM-2004/110

PROOF-OF-CONCEPT DEMONSTRATION RESULTS FOR ROBOTIC VISUAL SERVO CONTROLLERS

P. V. Chawda
W. E. Dixon, Ph.D.
T. J. Flynn
E. B. Holcombe
L. J. Love, Ph.D.
J. C. Rowe

Date Published: September 2004

Prepared by
OAK RIDGE NATIONAL LABORATORY
P.O. Box 2008
Oak Ridge, Tennessee 37831-6285
managed by
UT-Battelle, LLC
for the
U.S. DEPARTMENT OF ENERGY
under contract DE-AC05-00OR22725

CONTENTS

LIST OF FIGURES

ACKNOWLEDGEMENT

The authors express their gratitude for the support for this research project provided by the U.S. Department of Energy (DOE) Office of Science (SC) Office of Biological and Environmental Research (OBER) Environmental Management Sciences Program (EMSP) project ID No. 82797, the DOE SC Undergraduate Laboratory Internships (SULI) Program, and the Oak Ridge Institute for Science and Education (ORISE) Higher Education Research Experiences (HERE) Program.

ABSTRACT

There is significant motivation to provide robotic systems with improved autonomy as a means to significantly accelerate deactivation and decommissioning operations while also reducing the associated costs, removing human operators from hazardous environments, and reducing the required burden and skill of human operators. To achieve improved autonomy, fundamental research is focused on the challenges of developing visual servo controllers. The challenge in developing these controllers is that a camera provides 2-dimensional image information about the 3-dimensional Euclidean-space through a perspective (range dependent) projection that can be corrupted by uncertainty in the camera calibration matrix. Disturbances in this relationship (i.e., corruption in the sensor information) propagate erroneous information to the feedback controller of the robot, leading to potentially unpredictable task execution. This technical manual describes 3 proof-of-concept demonstrations of visual servo controllers developed from fundamental research aimed at these challenges. Specifically, one section describes the implementation of a cooperative visual servo control scheme with a camera-in-hand and a fixed camera to track a moving target despite uncertainty in the camera calibration and the unknown constant distance from the camera to a target where the camera is mounted on the end-effector of a 6 degrees-of-freedom hydraulic robot manipulator. The next section describes the implementation of 2 homography-based visual servo tracking and regulation controllers for a mobile robot with a calibrated camera despite an unknown time-varying distance from the camera to a target.

1. INTRODUCTION

Although a vision system can provide a robot with a unique sense of perception, several technical issues have impacted the design of robust visual servo controllers. One issue that was initially targeted in this project was the camera configuration trade-off between pixel resolution and field-of-view (FOV). Vision systems that utilize a camera mounted in a fixed configuration (i.e., the eye-to-hand configuration) are typically mounted far enough away from the robot workspace to ensure that the robot and desired target objects will remain in the camera's FOV. Unfortunately, by mounting the camera in this configuration the Euclidean-space area that corresponds to a pixel in the image-space can be quite large, resulting in low-resolution position measurements; hence, the precision and stability of the robot could be adversely affected. Moreover, many applications are ill-suited for the fixed camera configuration. For example, a robot may be required to position the camera for close-up tasks (i.e., the camera-in-hand configuration). For vision systems that utilize a camera mounted in the camera-in-hand configuration, the camera is naturally close to the workspace, providing for higher resolution measurements due to the fact that each pixel represents a smaller Euclidean-space area; however, the FOV of the camera is significantly reduced (i.e., an object may be located in the robot's workspace but be out of the camera's FOV due to the position of the end-effector).

The implementation of a cooperative visual servo control scheme is described in Section 2 for a camera-in-hand and a fixed camera configuration where the camera-in-hand is mounted on a Schilling 6-degrees-of-freedom (DOF) Titan II hydraulic robotic manipulator test bed that has played a vital role in several Department of Energy's teleoperation-based applications.[1] The objective of the cooperative visual servo control scheme is defined as the desire to track a moving target despite uncertainty in the camera calibration and an unknown constant distance from the camera to a target. To provide for greater robustness, the controller was developed despite parametric uncertainty in the camera calibration (e.g., focal length, image center, scaling factors, and camera position and orientation) and in the parameters of the dynamic model of the robot manipulator (e.g., mass, inertia, friction coefficients, and additive bounded disturbances). A nonlinear, Lyapunov-based design approach was utilized to construct a visual servoing controller for a robot manipulator that ensures uniformly ultimately bounded (UUB) end-effector tracking performance despite parametric uncertainty throughout the entire robot/camera system.[2,3] The UUB tracking result exploits information from both a fixed camera and a camera-in-hand, although both cameras contain parametric uncertainty in the calibration parameters. The advantages of the cooperative camera configuration are that: the fixed camera can be mounted so that a large robot workspace is visible; the camera-in-hand is mounted so that a high-resolution, close-up view of an object is achieved (facilitating the potential for more precise robotic motion); and the fixed camera provides a mechanism for determining the relative velocity of the robot end-effector with respect to the object for the camera-in-hand tracking problem. In comparison to previous work, this controller provides several enabling capabilities such as object tracking, robustness to disturbances in the projective image-space to Euclidean-space relationship, and robustness to parametric uncertainty in the robot dynamics (which includes unmodeled additive bounded disturbances). Despite the advancements of the cooperative visual servo control efforts, a significant limitation of the approach is that the distance from the camera to the target is required to remain constant or slowly time-varying.

An analytical method to compensate for a time-varying distance between the camera and the target can be obtained by using Lyapunov-based methods for control design and analysis in conjunction with photogrammetry techniques. Specifically, a homographic relationship between multiple images taken

by a single camera can be used to craft translation and rotation error systems that are suitable for controller development without requiring additional sensors (i.e., the system only requires a single camera). By utilizing homography-based methods, several visual servo controllers have been developed for robot manipulator applications.[4-10] However, the error system development, control design, and stability analysis for these efforts require fundamentally new control approaches for wheeled mobile robot (WMR) systems, since WMRs are underactuated systems subject to nonholonomic motion constraints (i.e., a typical feedback linearizing controller developed for holonomic systems can not be used to solve the problem).[11] The development of visual servo controllers for WMRs is especially motivated because the nonholonomic nature of WMR makes the Euclidean position difficult to accurately obtain. That is, the linear velocity of the WMR must first be numerically differentiated from the encoder readings (i.e., by the backwards difference algorithm) and then the nonlinear kinematic model must be numerically integrated to obtain the Euclidean position (i.e., dead reckoning). Since numerical differentiation/integration errors may accumulate over time, it is well known that navigation by dead reckoning is relatively inaccurate.

With the combined use of Lyapunov-based methods and the use of homography-based concepts, a visual servo controller was recently developed to ensure asymptotic regulation of the position/orientation of a WMR.[12] By decomposing the homography into separate translation and rotation components, measurable signals for the orientation and the scaled Euclidean position were obtained. Full Euclidean reconstruction is not possible due to the lack of depth information from the on-board camera to the target; hence, the resulting translation error system is unmeasurable. The contribution of this effort is that Lyapunov techniques are exploited to craft an adaptive controller that enables position and orientation regulation of the WMR despite the lack of depth information. This result is achieved with a monocular vision system, and the adaptive control design approach incorporates the full nonlinear kinematic equations of motion. Motivated by many practical applications that require a robotic system to move along a predefined or dynamically changing trajectory, a visual servo controller was also recently developed.[13] Specifically, a prerecorded image sequence (e.g., a video) of three target points is used to define a desired trajectory for the WMR. By comparing the target points from a stationary reference image with the corresponding target points in the live image and the prerecorded sequence of images, projective geometric relationships are exploited to construct Euclidean homographies. The information obtained by decomposing the Euclidean homography is used to develop kinematic controllers that are proven to yield either asymptotic tracking or regulation while actively compensating for the lack of depth information required for the translation error system. The implementation of these visual servo controllers is described in Section 3 for a Cybermotion K2A WMR test bed.

2. COOPERATIVE VISUAL SERVO CONTROL

2.1 OBJECTIVES

The aim of the efforts described in this section is to design control algorithms (realized by a software module and implemented on a typical robotic system employed for environmental managements tasks), which can force a robot manipulator to track an object moving with an unknown trajectory. To be reliably used in field operation, the tracking controller provides robustness to camera calibration errors and the parametric uncertainty associated with the robotic system. Moreover, camera information is incorporated from a fixed and an in-hand camera to provide both a large FOV and a high-resolution view of the robotic task to facilitate field implementation. To quantify this objective, a tracking error is defined as the difference between the actual Euclidean position of an object and the robot end-effector as follows:

$$e(t) = x(t) - x_0(t). \tag{1}$$

In (1), $e(t)$ denotes the planar position tracking error, and $x(t), x_0(t)$ denote the unknown Euclidean position of the camera-in-hand and the object, respectively. Based on (1), the error signal is clearly not measurable since the Euclidean position of the object is unknown. This fact significantly complicates the control design since the tracking error cannot be used as a feedback signal.

To achieve the control objective, one of the issues that have been addressed is the use of information from multiple uncalibrated cameras.[2,3] To address this problem, the following pinhole lens models were utilized for a global fixed camera and a local camera-in hand, respectively

$$y_0 = H_0 R_0 x_0 + p \tag{2}$$

$$y = HRR_K(x - x_0). \tag{3}$$

In the models given by (2) and (3), $y(t), y_0(t)$ denote the measurable image-space position of a target object (e.g., door frame, I-beam, bolt, laser point) determined by the in-hand and fixed camera, respectively. The matrices $H(z), H_0(z_0)$ given in (2) and (3) are diagonal, positive-definite matrices that are functions of the unknown constant focal length of each camera, the unknown constant camera scaling factors of each camera, and the distance, denoted by z, z_0, from the image-plane of each camera to the object. The matrices R, R_0 given in (2) and (3) denote constant rotation matrices for each camera that is a function of the unknown constant camera orientation, $R_K(q)$ denotes the rotation matrix for the eye-in-hand camera that is a function of the manipulator joints (since the orientation of this camera is not constant), and p is a vector of unknown constants including the projection of the camera's optical center on the image plane and the image center that is defined in the frame buffer coordinates. After taking the time derivative of (2) the following relationships for the fixed camera can be obtained

$$\dot{y}_0 = H_0 R_0 \dot{x}_0 \qquad \dot{x}_0 = R_0^{-1} H_0^{-1} \dot{y}_0. \tag{4}$$

After taking the time derivative of (3), the following expression can be obtained for the camera-in-hand

$$\dot{y} = HRJue + HRR_K(\dot{x} - \dot{x}_0) \tag{5}$$

where $J(q)$ denotes the manipulator Jacobian and $u(t)$ denotes the joint level control input. Based on the expressions given in (3)-(5), the following kinematic control input was designed[2]

$$u = J^{-1}(k + k_2(\rho_1(y_o) + \rho_2))R_K y \tag{6}$$

where k, k_2 denote known, positive constant control gains, and $\rho_1(\cdot), \rho_2$ denote positive bounding terms for the Euclidean trajectory. The use of the image-space data from the camera-in-hand provides a mechanism for incorporating the error feedback as indicated from (1) and (3). To exploit this characteristic, novel mathematical development was required due to the structure of (3)-(5).

2.2 RESULTS

By using Lyapunov-based stability analysis techniques, the control design given in (6), coupled with a robust torque control input to reject parametric uncertainties in the robot dynamic model, was analytically proven to force the end-effector of a robot manipulator to track an object moving in a plane with an unknown trajectory.[2] Specifically, for an object moving in a plane, the task-space tracking error defined in (1) was analytically proven to be exponentially driven to a small neighborhood about zero that could be made arbitrarily small by adjusting the control gains (i.e., UUB tracking).

To demonstrate the performance of the developed control strategy, an experimental test bed (see Fig. 1) was developed that includes: a 6-DOF Schilling Titan II hydraulic manipulator, two Dalsa (CA-D6-0256W) MotionVision area scan digital cameras that capture 955 frames per second with 8-bit gray scale at a 256x256 resolution, two Road Runner Model 24 video capture boards, two Pentium IV-based (1.9GHz) personal computers (PCs) operating under the real-time operating system QNX (a real-time micro-kernel based operating system), a pan-and-tilt unit (PTU), and a custom Arm Level Controller (ALC) based on commercial PC104 components. The ALC is part of a telerobotic manipulation system that has targeted applications in both waste tank remediation and facility deactivation and decommissioning operations as part of the U.S. Department of Energy's EM50 Program.[1] The ALC is controlled at the force/torque level by a linear, quaternion-based Euclidean strategy that operates at a control frequency of 200 Hz. Since the force/torque level controller is a closed (black box) system, the complete robust torque controller could not be implemented without replacing the existing ALC. Rather than replacing the existing controller, a plug-and-play retrofit was implemented in which the integral of the kinematic controller given in (6) was utilized to command desired joint positions. That is, the position commands that are typically provided through a human operating a master controller are replaced by desired position commands that are autonomously produced by the visual servo control scheme.

The uncalibrated fixed camera was mounted on a tripod and placed in the workspace and the camera-in-hand was mounted on the end-effector such that the yaw and pitch of the camera is coincident with the camera. For simplicity, the fixed camera was placed in the workspace so that the FOV was twice the size of the in-hand camera. A laser pointer is mounted on the PTU to project a laser point (to provide an easily detectable object feature) in the environment with some unknown (by the controller) trajectory. One of the PCs is connected to the fixed camera and will be utilized to capture the laser point image, extract the pixel coordinate of the laser point in the fixed camera's reference frame, and transmit the pixel coordinates to the shared memory of the second PC over a 100 Mb/sec dedicated Ethernet

connection. The second PC is connected to the camera-in-hand and is utilized to capture the laser point image, extract the pixel coordinate of the laser point in the in-hand camera's reference frame, acquire the pixel coordinates provided by the first PC from a shared memory location, acquire joint information from a shared memory location provided by the ALC, compute the kinematic control algorithm, and transmit the computed control to the ALC over a second 100 Mb/sec dedicated Ethernet connection. Although the cameras can acquire data at a rate of 955 frames per second, the communication interface between the existing ALC and the plug-and-play vision system was limited to 100 Hz. The ALC is utilized to perform the real-time I/O from the joint resolvers, transmit the joint positions to the camera-in-hand PC through shared memory, acquire the kinematic control input from the camera-in-hand PC, and calculate the joint-level control input. Joint resolvers were used to determine the joint rotation. The joint rotation signals were utilized to calculate the task-space position of the end-effector through the manipulator Jacobian. Communication between the PCs and the ALC is achieved through two RS232 serial ports (one port for reading data, one port for writing data).

Fig. 1. Experimental test bed including a Schilling Titan II hydraulic manipulator with a fixed camera and an in-hand camera.

The source code for the server program that determines the image-space coordinates of the tartet for the fixed camera and then writes the information to shared memory locations over a TCP/IP connection is provided in Section A.1 of Appendix A. The source code for the server program that determines the image-space coordinates of the target for the camera-in-hand and then writes the information directly into shared memory (the program executes on the resident PC) is provided in Section A.2 of

Appendix A. The source code for the client program that allocates shared memory locations for the fixed camera data, receives the fixed camera image-space coordinates of the target, and writes the information into shared memory is given in Section A.3 of Appendix A. The source code that reads data from the shared memory locations for the fixed camera and the camera-in-hand, reads robot joint information from the ALC, computes the visual servo controller, provides real-time plotting and controller gain adjustment capabilities through the real-time control environment QMotor,[14] and writes the desired trajectory to the ALC is given in Section A.4 of Appendix A.

To illustrate the application of the cooperative visual servoing strategy, several examples were examined. In the first example, the integral of the controller in (6) was implemented for cases when the laser point moved in circular, square, and "figure 8" trajectories. These trajectories were programmed into the PTU to generate the patterns, but knowledge of the pattern was not provided to the controller. The object trajectories that were recorded by the fixed camera are presented in Figs. 2–4, and the image-space errors determined by the in-hand camera are depicted in Figs. 5–7. For each case, a straight line path planner was utilized to position the in-hand camera so that the initial position of the target was in view and that the image plane of the in-hand camera is parallel to the robot's plane of motion. During this motion the robot is not under visual servo control, and hence, the image-space errors observed by the in-hand camera are set to zero (see the first few seconds of Figs. 5–7. Once the camera has been positioned, the rotation of the end-effector is fixed, along with the depth of the manipulator (denoted by the X-axis in the manipulator workspace), to ensure the robot moves in a planar motion (ensuring that the distance from the camera to the target remains constant). The desired position commands computed by the visual servo controller are depicted in Figs. 8–10 (i.e., the signals given in Figs. 8–10 represent the desired Euclidean position trajectory computed by the visual servo controller and supplied to the ALC).

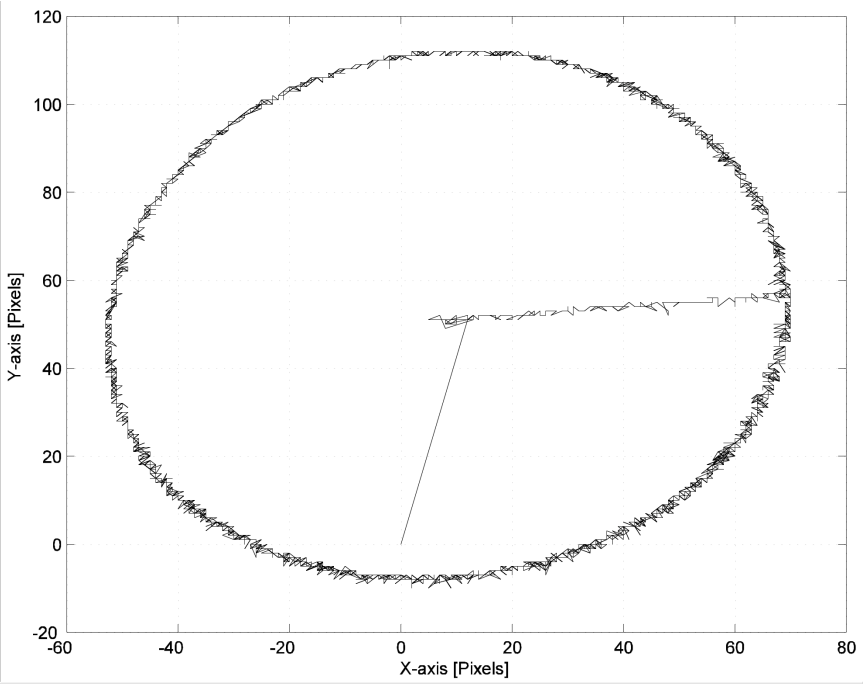

Fig. 2. Image-space object trajectory recorded by the fixed camera for a circular motion.

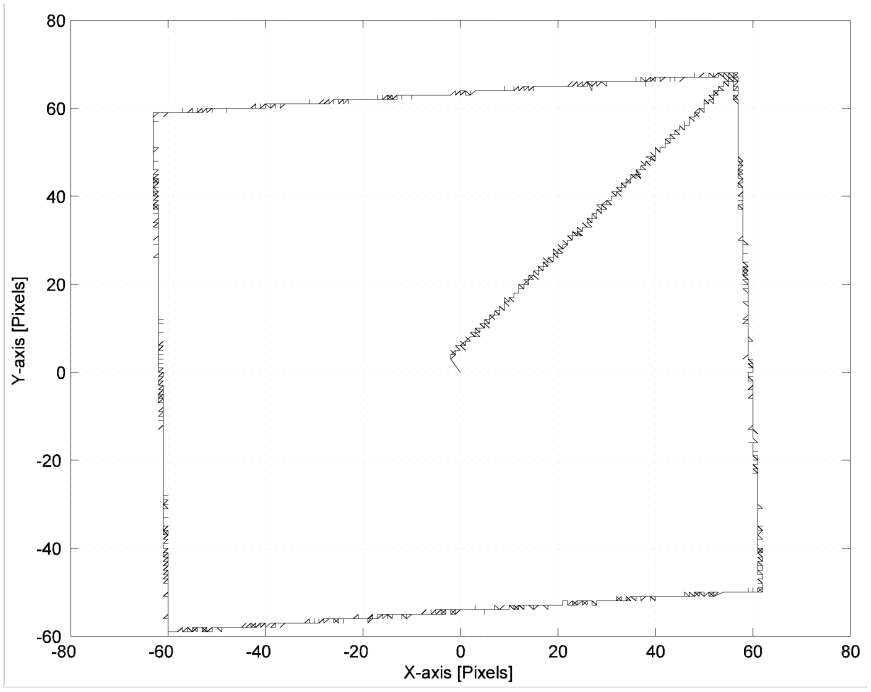

Fig. 3. Image-space object trajectory recorded by the fixed camera for a square motion.

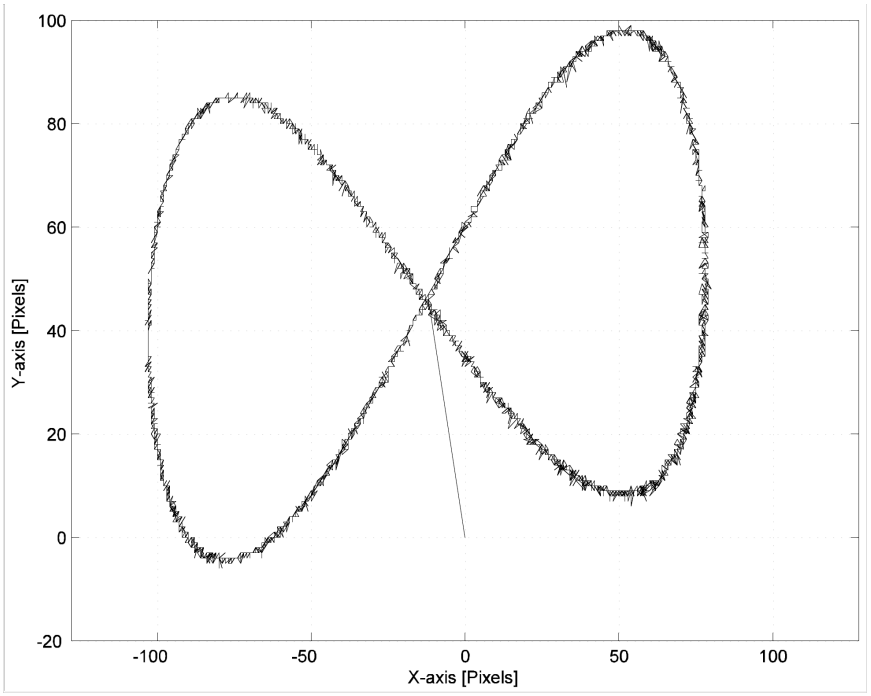

Fig. 4. Image-space object trajectory recorded by the fixed camera for a "figure 8" motion.

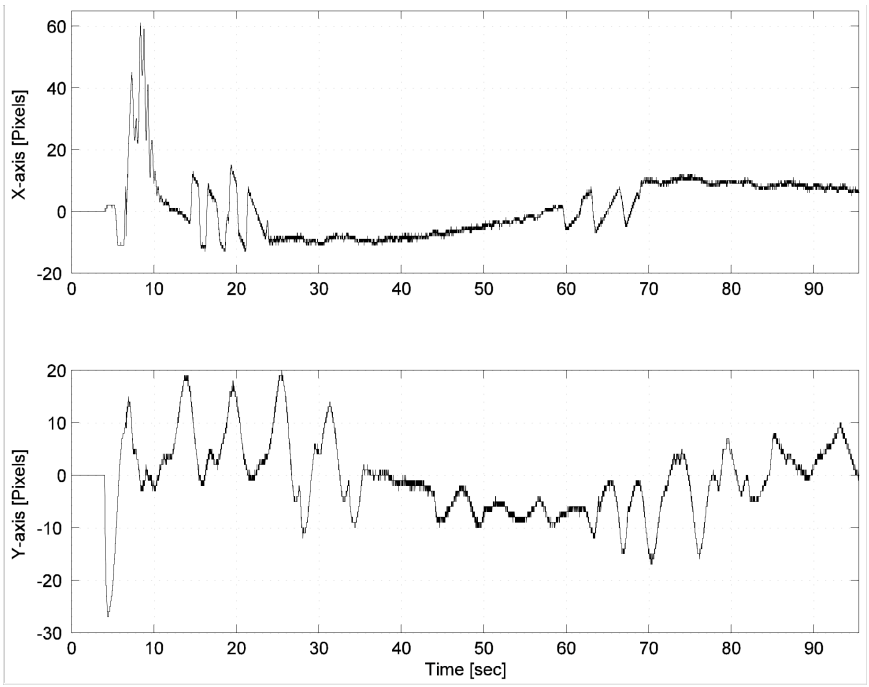

Fig. 5. Image-space tracking error recorded by the in-hand camera for a circle motion.

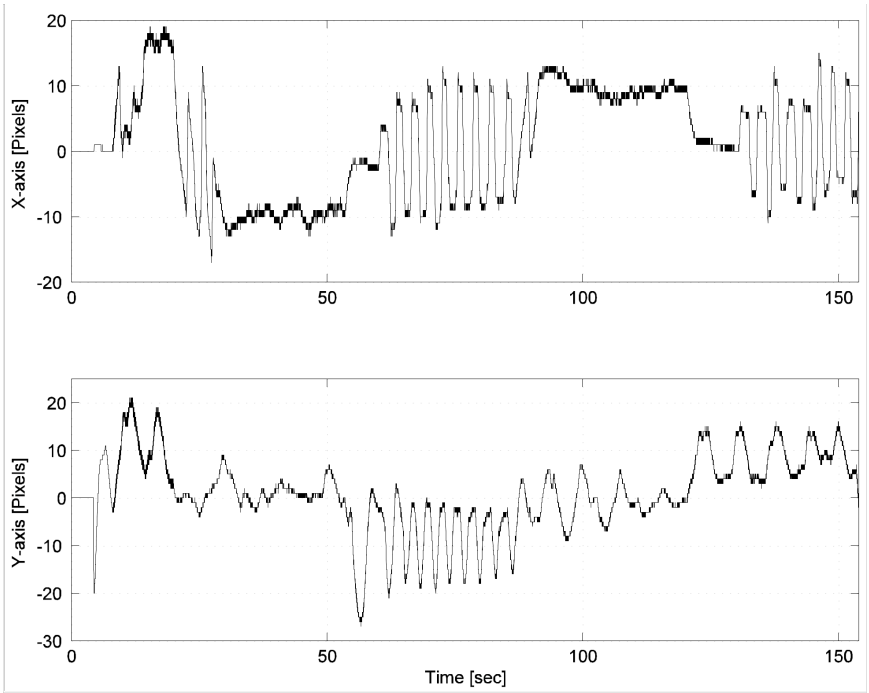

Fig. 6. Image-space tracking error recorded by the in-hand camera for a square motion.

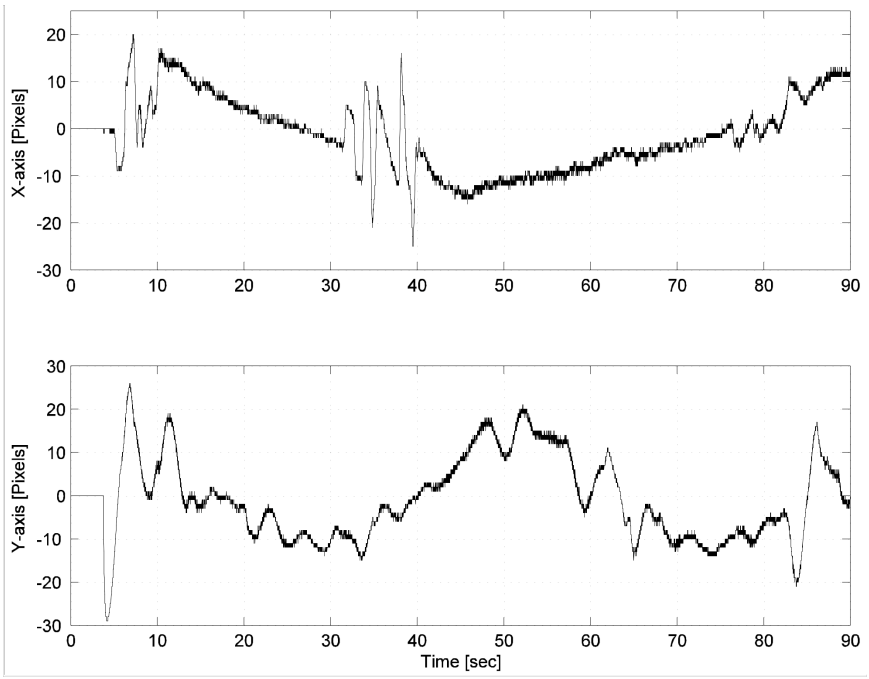

Fig. 7. Image-space tracking error recorded by the in-hand camera for a "figure 8" motion.

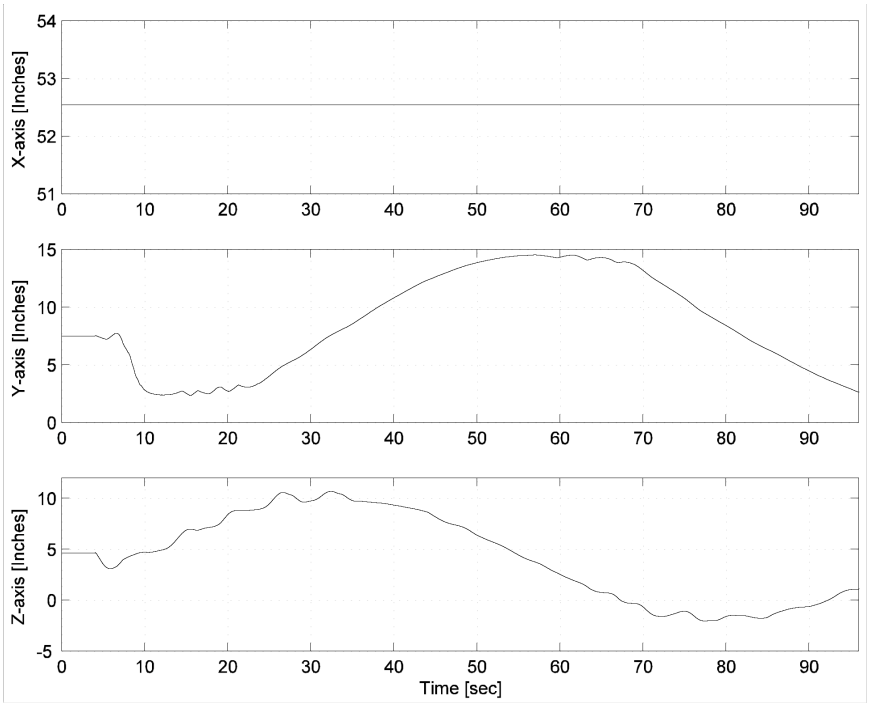

Fig. 8. Euclidean position commands computed by the visual servo controller and sent to the ALC for the circle trajectory.

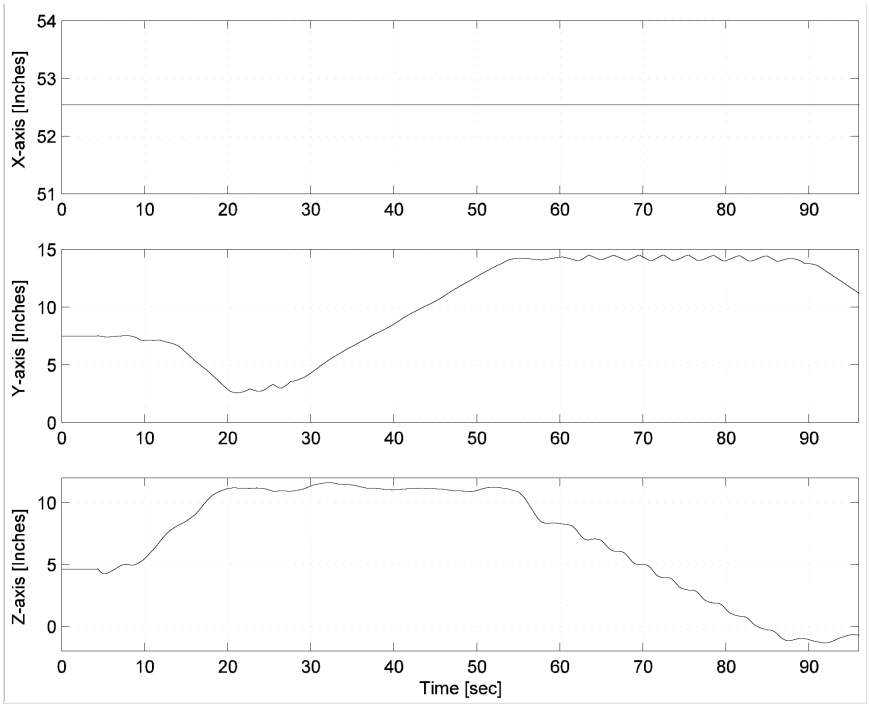

Fig. 9. Euclidean position commands computed by the visual servo controller and sent to the ALC for the square trajectory.

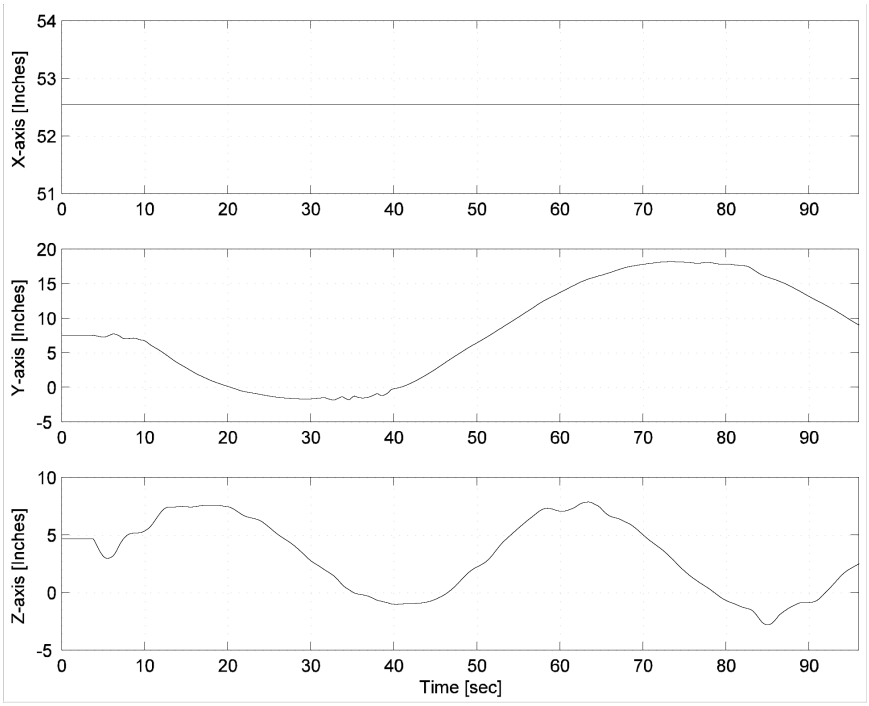

Fig. 10. Euclidean position commands computed by the visual servo controller and sent to the ALC for the "figure 8" trajectory.

2.3 DISCUSSION

The application of the cooperative uncalibrated visual control strategy is demonstrated through the results depicted in Figs. 2–10. To integrate the novel visual servoing research into an actual field system, a plug-and-play retrofit strategy was employed in which the kinematic controller was utilized to command desired joint positions to a linear joint level controller. Unfortunately, due to the restriction that the full robust controller could not be implemented, many of the robust control terms were omitted from the implementation, thus reducing the performance of the controller. Other factors that contributed to reduced tracking performance include: (1) the force/torque level controller is a linear controller (and hence, perfect tracking cannot be obtained), (2) the visual servo rate was limited to 100 Hz, and more significantly, (3) the robotic testbed is not a precision manipulator by construction (experimental tests have illustrated the repeatability of the end-effector position to be approximately 0.25 inches near its limits of motion). For the current experiment, an end-effector error of 0.25 inches corresponds to approximately 5 pixels by the in-hand camera.

Despite the lack of high-precision tracking, the impact of the research described in this section is the enabling technology that a manipulator can be used to automatically track an object moving with an unknown trajectory with multiple uncalibrated cameras. A novel aspect that is demonstrated is the fact that multiple uncalibrated cameras could be utilized (in a non-stereo vision approach) to enable both a close-up, higher resolution view and large FOV. In the current experiment, the in-hand camera provided a resolution of 19.32 pixels/inch and the resolution of the fixed camera was reduced to 9.75 pixels/inch. The developed cooperative visual servoing approach also solved the relative velocity problem that has stymied previous visual servo research for tracking object trajectories with an uncalibrated camera. The main limitation of the approach is that the unknown depth is required to be held constant (i.e., the manipulator is constrained to planar trajectories).

3. HOMOGRAPHY-BASED VISUAL SERVO CONTROL

3.1 OBJECTIVES

Photogrammetry methods such as the development of a homographic relationship between multiple images can be used to develop an adaptive visual servo controller that can compensate for an unknown time-varying distance from the camera to a target object.[5-10,12,13] An overview of the homography-based approach is provided in this section along with a description of the results obtained from a proof-of-concept demonstration for WMR tracking and regulation. Specifically, a homography-based visual servo control strategy was recently developed to force the Euclidean position/orientation of a camera mounted on a WMR (i.e., the camera-in-hand problem) to track a desired time-varying trajectory defined by a prerecorded sequence of images or to regulate the WMR to a desired position and orientation.[12,13] By comparing the feature points of an object from a reference image to feature points of an object in the current image and the prerecorded sequence of images, projective geometric relationships are exploited to enable the reconstruction of the Euclidean coordinates of the target points with respect to the WMR coordinate frame. The tracking control objective is naturally defined in terms of the Euclidean space; however, the translation error is unmeasurable. That is, the Euclidean reconstruction is scaled by an unknown distance from the camera to the target, and while the scaled position is measurable through the homography, the unscaled position error is unmeasurable. To overcome this obstacle, a Lyapunov-based control strategy can be employed that provides a framework for the construction of an adaptive update law to actively compensate for the unknown depth-related scaling constant.[12,13] In contrast to visual servo methods that linearize the system equations to facilitate Extended Kalman Filtering methods, the Lyapunov-based control design in this section is based on the full nonlinear kinematic model of the vision system and the mobile robot system.

3.2 GEOMETRIC MODEL

To illustrate the underlying homography-based methodology, and to motivate the research activities in this project, consider Figure 11 which depicts a reference plane, denoted by π, that is defined by four target points, denoted by $O_i \; \forall i = 1, 2, 3, 4$, that are considered to be coplanar and not collinear. If four coplanar target points are not available then the subsequent development can exploit the classic eight-points algorithm with no four of the eight target points being coplanar. The coordinate frame $\mathcal{F}^*$ depicted in Fig. 11 and Fig. 12 defines a reference position and orientation of a camera that corresponds to a reference image of a target. The coordinate frame $\mathcal{F}$ depicted in Fig. 11 and Fig. 12 defines the current position and orientation of a camera that corresponds to the current image of the target, and the coordinate frame $\mathcal{F}_d$ depicted in Fig. 11 and Fig. 12 defines the desired position and orientation of a camera that corresponds to the prerecorded desired image trajectory of the target (i.e., a video that describes the desired motion of the camera). The Euclidean coordinates of $O_i \; \forall i = 1, 2, 3, 4$ can be expressed in terms of $\mathcal{F}$, $\mathcal{F}_d$, and $\mathcal{F}^*$, respectively, as follows:

$$\bar{m}_i(t) = \begin{bmatrix} x_i(t) & y_i(t) & z_i(t) \end{bmatrix}^T \quad \bar{m}_{di}(t) = \begin{bmatrix} x_{di}(t) & y_{di}(t) & z_{di}(t) \end{bmatrix}^T \quad \bar{m}_i^* = \begin{bmatrix} x_i^* & y_i^* & z_i^* \end{bmatrix}^T \quad (7)$$

under the practical assumption that the distances from the origin of the respective coordinate frames to Π along the focal axis remain positive (i.e., $x_i(t), x_i^* > \Pi$ where $/_i$ denotes an arbitrarily small positive constant).

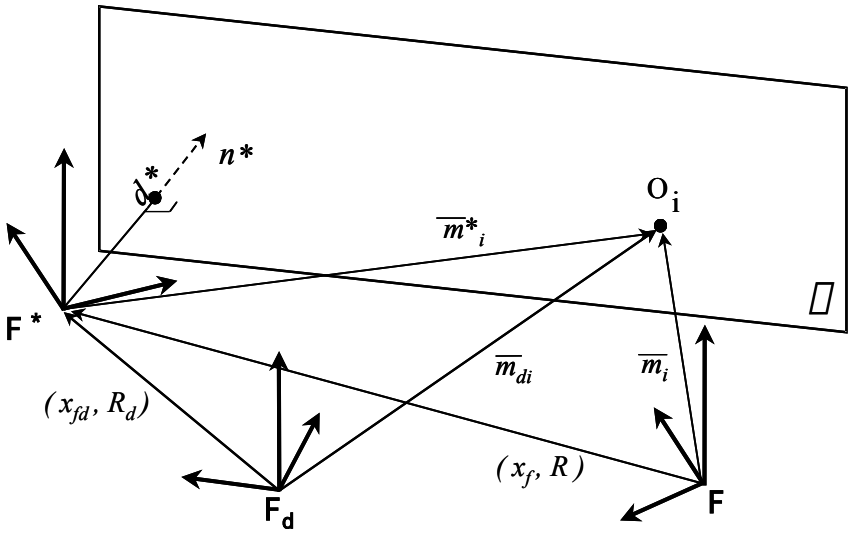

Fig. 11. Coordinate frame relationships.

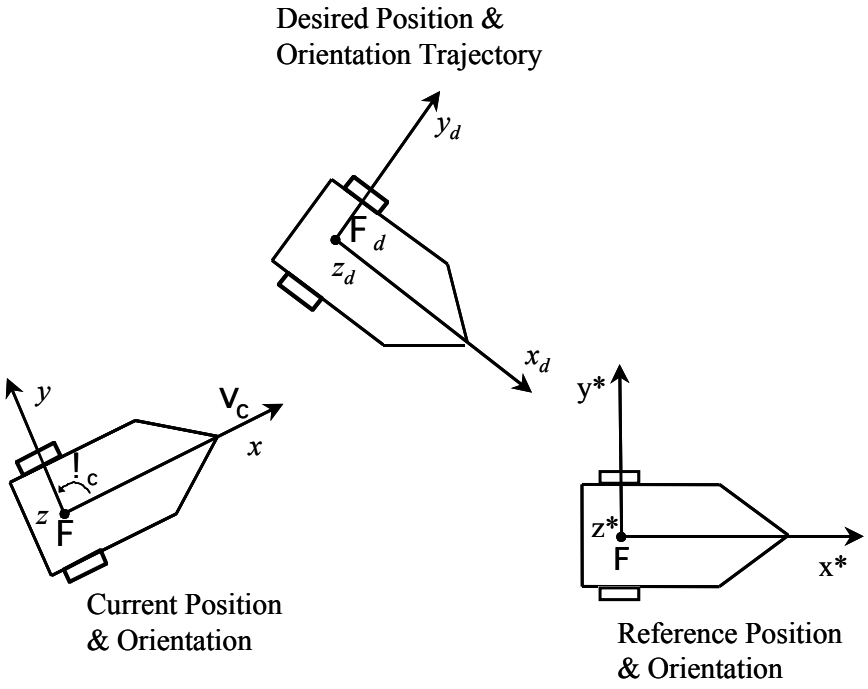

Fig. 12. WMR coordinate frames.

The time-varying translation and rotation from $\mathcal{F}$ to $\mathcal{F}^*$, are denoted by $x_f(t) \in \mathbb{R}^3$ and $R(t) \in SO(3)$, respectively, and the time-varying translation and rotation from $\mathcal{F}_d$ to $\mathcal{F}^*$, are denoted by $x_{fd}(t) \in \mathbb{R}^3$ and $R_d(t) \in SO(3)$, respectively. Specifically, $\bar{m}_i^*$ can be related to $\bar{m}_i(t)$ and $\bar{m}_{di}(t)$ as follows:

$$\bar{m}_i = x_f + R\bar{m}_i^* \qquad \bar{m}_{di} = x_{fd} + R_d\bar{m}_i^* \tag{8}$$

where

$$R \equiv \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad R_d \equiv \begin{bmatrix} \cos\theta_d & -\sin\theta_d & 0 \\ \sin\theta_d & \cos\theta_d & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{9}$$

for the WMR problem where $\theta(t)$ denotes the right-handed rotation angle that aligns the rotation of $\mathcal{F}$ with $\mathcal{F}^*$, and $\theta_d(t)$ denotes the right-handed rotation angle that aligns $\mathcal{F}_d$ with $\mathcal{F}^*$. Also from the geometry given in Fig. 1, the constant distance $d^* \in \mathbb{R}$ from the origin of $\mathcal{F}^*$ to π along the unit normal is given by

$$d^* = n^{*T}\bar{m}_i^* \tag{10}$$

where $n^* \in \mathbb{R}^3$ denotes the constant unit normal to π expressed in $\mathcal{F}^*$. Based on (10), the expressions in (8), can be written in terms of a normalized Euclidean coordinate vector as follows:

$$m_i = \alpha_i \underbrace{\left(R + x_h n^{*T}\right)}_{H} m_i^* \qquad m_{di} = \alpha_{di} \underbrace{\left(R_d + x_{hd} n^{*T}\right)}_{H_d} m_i^* . \tag{11}$$

In (11), $x_h(t) \in \mathbb{R}^3$ and $x_{hd}(t) \in \mathbb{R}^3$ denote the following scaled translation vectors

$$x_h = \frac{x_f}{d^*} \qquad x_{hd} = \frac{x_{fd}}{d^*} \tag{12}$$

$m_i(t)$, $m_{di}(t)$, $m_i^* \in \mathbb{R}^3$ denote the normalized Euclidean coordinates of O_i, expressed in $\mathcal{F}$, $\mathcal{F}_d$, and $\mathcal{F}^*$, respectively, that are defined as follows:

$$m_i = \frac{\bar{m}_i}{x_i} = \begin{bmatrix} 1 & \frac{y_i}{x_i} & \frac{z_i}{x_i} \end{bmatrix}^T \quad m_{di} = \frac{\bar{m}_{di}}{x_{di}} = \begin{bmatrix} 1 & \frac{y_{di}}{x_{di}} & \frac{z_{di}}{x_{di}} \end{bmatrix}^T \quad m_i^* = \frac{\bar{m}_i^*}{x_i^*} = \begin{bmatrix} 1 & \frac{y_i^*}{x_i^*} & \frac{z_i^*}{x_i^*} \end{bmatrix}^T \tag{13}$$

$\alpha_i(t), \alpha_{di}(t) \in \mathbb{R}$ $\forall i = 1, 2, 3, 4$ denote invertible depth ratios defined as follows:

$$\alpha_i(t) = \frac{x_i^*}{x_i} \qquad \alpha_{di}(t) = \frac{x_i^*}{x_{di}} \tag{14}$$

and $H(t), H_d(t) \in \mathbb{R}^{3\times3}$ denote the Euclidean homography matrices that are composed of the actual and desired rotation and translation components, respectively.

3.3. EUCLIDEAN RECONSTRUCTION

The relationship in (11) provides a means to quantify a translation and rotation error between $\mathcal{F}$ and $\mathcal{F}^*$ and between $\mathcal{F}_d$ and $\mathcal{F}^*$. However, since the Euclidean position of the coordinate frames cannot be directly measured, a Euclidean reconstruction can be developed by comparing the pixel coordinates of the target points in the current image to the pixel coordinates of the corresponding target points in the reference image. In addition to having a Euclidean coordinate, each target point O_i will also have a projected pixel coordinate that is defined as elements of $p_i(t) \in \mathbb{R}^3$ (the actual time-varying image points), $p_{di}(t) \in \mathbb{R}^3$ (the desired image point trajectory), and $p_i^* \in \mathbb{R}^3$ (the constant reference image points). The normalized Euclidean coordinates of the target points are related to the image data through the following pinhole lens models

$$p_i = Am_i \qquad p_{di} = Am_{di} \qquad p_i^* = Am_i^* \tag{15}$$

where $A \in \mathbb{R}^{3x3}$ is a known, constant, and invertible intrinsic camera calibration matrix. To calculate the Euclidean homography given in (11) from pixel information, the pinhole camera model given in (15) can be utilized to determine the following relationship

$$p_i = \alpha_i \underbrace{\left(AHA^{-1}\right)}_{G} p_i^* \qquad p_{di} = \alpha_{di} \underbrace{\left(AH_dA^{-1}\right)}_{G_d} p_i^* \tag{16}$$

where $G(t) = \left[g_{ij}(t)\right]$, $G_d(t) = \left[g_{dij}(t)\right]$ $\forall i, j = 1,2,3$ $\in \mathbb{R}^{3\times3}$ denote projective homography matrices. Provided the camera calibration parameters in A are known, various techniques can be used to determine the projective homography matrices up to a scalar multiple (e.g., the product $\alpha_i(t)G(t)$ can be determined). From the definition of $G(t)$ and $G_d(t)$ given in (16), various techniques can then be used[15,16] to decompose the homography to obtain $\alpha_i(t), \alpha_{di}(t), G(t), G_d(t), H(t), H_d(t)$, and the rotation and translation signals $R(t)$, $R_d(t)$, $x_{hd}(t)n^*$, and $x_h(t)n^*$. To facilitate the development of a visual servo controller, $R(t)$ can be expressed in the axis-angle representation, denoted by $\dot{e}(t) \in \mathbb{R}^3$, as follows:[17]

$$\dot{e} = u(t)\dot{\theta}(t). \tag{17}$$

A similar relationship can be developed for $R_d(t)$. For the representation in (17), $u(t) \in \mathbb{R}^3$ represents a unit rotation axis, and $\theta(t) \in \mathbb{R}$ denotes the respective rotation angle about $u(t)$ that is assumed to be confined to the following region

$$0 \le \theta(t) < \pi. \tag{18}$$

The unit rotation axis and the rotation angle can be obtained from $R(t)$ as follows:

$$\theta = \cos^{-1}\left(\frac{1}{2}\left(tr(R) - 1\right)\right) \qquad [u]_\times = \frac{R - R^T}{2\sin(\theta)} \tag{19}$$

where $tr(R)$ denotes the trace of $R(t)$. Hence, $R(t)$, $R_d(t)$, $x_h(t)$, $x_{hd}(t)$, $\alpha(t)$, $\alpha_d(t)$, $u(t)$, $u_d(t)$ and the depth ratios $\alpha_i(t)$ and $\alpha_{di}(t)$ are all known signals that can be used for control synthesis.

3.4. TRACKING CONTROL DEVELOPMENT

The tracking control objective is to ensure that the coordinate frame $\mathcal{F}$ tracks the time-varying trajectory of $\mathcal{F}_d$ (i.e., $\bar{m}_i(t)$ tracks $\bar{m}_{di}(t)$). Based on the previous development, the translation and rotation tracking error is defined as follows: [13]

$$e_1 = x_{h1} - x_{hd1} \qquad e_2 = x_{h2} - x_{hd2} \qquad e_3 = \alpha - \alpha_d \tag{20}$$

where $x_{h1}(t), x_{h2}(t), x_{hd1}(t)$, and $x_{hd2}(t)$ are elements of the $x_h(t)$ and $x_{hd}(t)$ translation vectors, and $\alpha(t)$ and $\alpha_d(t)$ are introduced in (9). Based on the definition in (20), it can be shown that the control objective is achieved if the tracking error $e(t) \to 0$ making $\bar{m}_i(t) \to \bar{m}_{di}(t)$.

Based on (20), the linear and angular velocity kinematic control inputs for the WMR can be designed as follows: [13]

$$v_c = k_v e_1 - \bar{e}_2 \alpha_c + \hat{d}\theta(x_{h2}\alpha_c - \dot{x}_{hd1}) \tag{21}$$

$$\alpha_c = k_\alpha e_3 - \dot{\alpha}_d - \dot{x}_{hd1}\bar{e}_2 \tag{22}$$

where k_v, k_α denote positive, constant control gains, and $\bar{e}_2(t)$ is an auxiliary signal defined as follows:

$$\bar{e}_2 = e_2 - x_{hd1} e_3 . \tag{23}$$

In (21), the parameter update law $\hat{d}\theta(t)$ is generated by the following differential equation[13]

$$\hat{d}\theta = -\gamma_1 e_1 \left(x_{h2}\alpha_c - \dot{x}_{hd1}\right) \tag{24}$$

where γ_1 is a positive, constant adaptation gain.

3.5. REGULATION CONTROL DEVELOPMENT

Due to the implications of Brockett's Condition, the previous tracking controller can not be used to solve the regulation control problem; hence, a separate regulation controller has been developed.[12] The regulation control objective is to ensure that $\mathcal{F}$ is regulated to $\mathcal{F}^*$. This objective is naturally defined in terms of the Euclidean position and orientation of the WMR. Specifically, the translation error between $\mathcal{F}$ and $\mathcal{F}^*$, denoted by $e_t(t)$, can be written for any target point as follows:

$$e_t(t) = \begin{bmatrix} e_{tx}(t) \\ e_{ty}(t) \end{bmatrix} = \begin{bmatrix} x(t) - x^* \\ y(t) - y^* \end{bmatrix} \begin{bmatrix} \cos\alpha & \sin\alpha \\ -\sin\alpha & \cos\alpha \end{bmatrix} \tag{25}$$

and the orientation error between $\mathcal{F}$ and $\mathcal{F}^*$, denoted by $e_o(t)$, can be written as follows:

$$e_o(t) = \theta(t). \tag{26}$$

Based on (25) and (26), the regulation control objective is to regulate $e_t(t)$ and $e_o(t)$ to zero. Since $e_t(t)$ is not measurable, an auxiliary error system can be defined as follows:[12]

$$r(t) = \begin{bmatrix} r_1(t) & r_2(t) & r_3(t) \end{bmatrix} = \begin{bmatrix} e_o(t) & \dfrac{e_{tx}(t)}{x^*} & \dfrac{e_{ty}(t)}{x^*} \end{bmatrix}. \tag{27}$$

As written, the last two elements of the vector in (27) do not appear to be measurable; however, after exploiting several properties from the geometry of the problem, the last two elements can be written as[12]

$$\begin{bmatrix} r_2(t) \\ r_3(t) \end{bmatrix} = \begin{bmatrix} \dfrac{1}{\theta}\left(\cos\theta + m_y^* \sin\theta\right) \\ \dfrac{1}{\theta} m_y \left(\sin\theta + m_y^* \cos\theta\right) \end{bmatrix} \tag{28}$$

where each element of (28) can be computed through the homography decomposition. Based on the error system given by (27) and (28), the following linear and angular velocity inputs can be designed to achieve the regulation control objective[12]

$$v_c = -k_2 \theta + \hat{x}^* r_3 \cos t + \hat{x}^* \omega_c r_3 \tag{29}$$

$$\omega_c = -k_1 \left(r_1 + \phi\right) \tag{30}$$

where k_1, k_2 denote positive, constant control gains, and $\theta(t)$ is an auxiliary signal defined as follows:

$$\theta = r_2 - r_3 \sin(t) \tag{31}$$

and $\phi(t)$ is an auxiliary signal defined as follows:

$$\phi = \left(\theta + r_3 \sin(t)\right)\left(r_3 - \theta \sin(t)\right). \tag{31}$$

In (29), the parameter update law $\hat{x}^*(t)$ is generated by the following differential equation

$$\dot{\hat{x}}^* = \tilde{A}\left(-r_3 \theta \cos t - \theta \omega_c r_3\right) \tag{32}$$

where $\tilde{A}$ is a positive, constant adaptation gain.[12]

3.6. RESULTS

To demonstrate the controller experimentally, the proof-of-concept WMR test bed depicted in Fig. 3 was developed. The WMR test bed consists of the following components: a modified K2A mobile robot (with an inclusive Pentium 133 MHz PC) manufactured by Cybermotion, Inc., a Dalsa (CA-D6-0256W) MotionVision area scan digital camera that captures 955 frames per second with 8-bit gray scale at a 256x256 resolution, a Road Runner Model 24 video capture board, two Pentium IV-based PCs operating under the real-time operating system QNX. The camera and image processing PC (operating under QNX) were mounted on the top of the WMR as depicted in Fig. 3. The internal WMR computer (also operating under QNX) hosts the control algorithm that was written in "C++", and implemented using

Qmotor 3.0.[14] In addition to the image processing PC, a second PC (operating under the MS Windows 2000 operating system) was used to remotely login to the internal mobile robot PC via the QNX Phindows application. The remote PC was used to access the graphical user interface of Qmotor for execution of the control program, gain adjustment, and data management, plotting, and storage. Light-emitting diodes (LEDs) were rigidly attached to a rigid structure that was used as the target, where the intensity of the LEDs contrasted sharply with the background. Due to the contrast in intensity, a simple thresholding algorithm was used to determine the image coordinates of each LED. The mobile robot is controlled by a torque input applied to a drive and a steer motor. To facilitate a torque controller the actual linear and angular velocity of the mobile robot is required. To acquire these signals a backwards difference algorithm was applied to the drive and steering motor encoders. Encoder data acquisition and the control implementation were performed at a frequency of 1.0 kHz using the Quanser MultiQ I/O board. For simplicity the electrical and mechanical dynamics of the system were not incorporated in the control design (i.e., the emphasis of this experiment is to illustrate the visual servo controller). However, since the developed kinematic controller is differentiable, standard backstepping techniques could be used to incorporate the mechanical and electrical dynamics. Permanent magnet DC motors provide steering and drive actuation through a 106:1 and a 96:1 gear coupling, respectively. The modified K2A mobile robot has an approximate mass of 165 kg, an inertia of approximately 4.643 kg-m^2, and a wheel radius of 0.010 m.

Fig. 13. WMR test bed.

To implement the tracking control experiment, the WMR was driven along a desired path as the camera recorded the trajectory. The program given in Section B.1 of Appendix B was used to capture the image-space coordinates of the four target points and write the coordinates in a file. One set of four image-space coordinates was used as the reference image. The program given in Section B.2 of Appendix B reads the file of image-space coordinates, computes the homography, decomposes the homography, computes the desired trajectory signals required by the visual servo tracking controller (i.e., $x_{hd1}(t)$, $x_{hd2}(t)$, and $\theta_d(t)$), and writes the desired trajectory signals to an output file. The output file was modified offline to compute the derivative of $x_{hd1}(t)$, $x_{hd2}(t)$, and $\theta_d(t)$ and to filter all the desired trajectory signals to reduce numerical noise artifacts. The filtered desired trajectory signals $x_{hd1}(t)$, $\dot{x}_{hd1}(t)$, $x_{hd2}(t)$, $\theta_d(t)$, and $\dot{\theta}_d(t)$ were then stored in a data file that is read by the subsequent WMR tracking control program (Section B.5 of Appendix B). Figure 14 and Fig. 15 depict the desired

translation and rotation signals, respectively. The server program (executing on the image processing PC) given in Section B.3 of Appendix B determines the image-space coordinates of the target in the live image, computes the homography, decomposes the homography, computes the current translation and rotation signals (i.e., $x_{h1}(t)$, $x_{h2}(t)$, and $\square(t)$) and then writes the information to shared memory locations over a TCP/IP connection. The client program (executing on the internal WMR PC) given in Section B.4 of Appendix B allocates shared memory locations for the live camera data, receives the current translation and rotation signals of the target, and writes the information into shared memory. The control program (executing on the internal WMR PC) given in Section B.5 of Appendix B reads data from the shared memory locations for the live camera, reads the WMR encoder signals (to enable a high-gain feedback torque control loop), computes the visual servo controller, provides real-time plotting and controller gain adjustment capabilities through the real-time control environment QMotor[14] (viewed through the remote PC), and commands a motor voltage signal to the drive and steering motors.

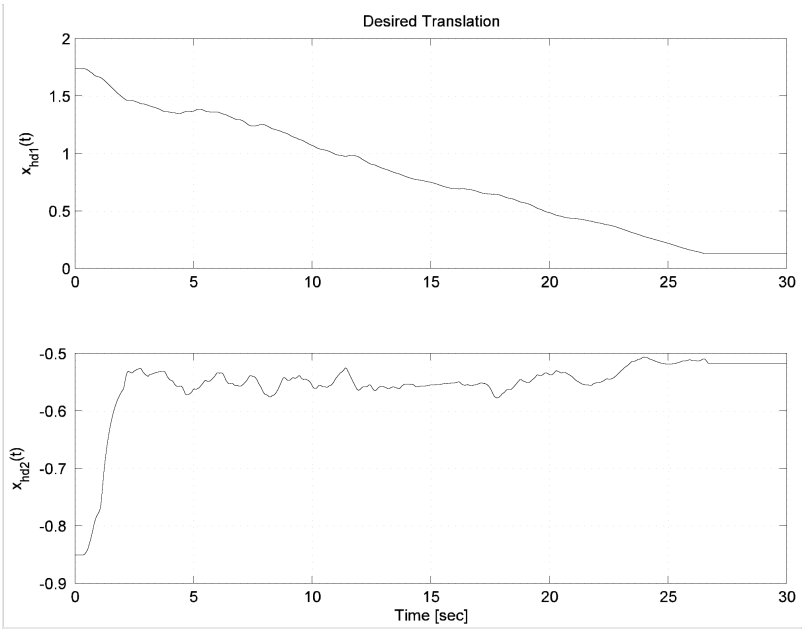

Fig. 14. Desired WMR translation trajectory.

The control gains were adjusted to reduce the position/orientationtracking error with the adaptation gains set to zero and the initial adaptive estimate set to zero. Once the position/orientation tracking error response could not be significantly improved by further adjustments of the feedback gains, the adaptation gains were adjusted to allow the parameter estimation to reduce the position/orientation tracking error. The unitless position/orientation tracking errors $e_1(t)$ and $e_2(t)$, are depicted in Fig. 16 and Fig. 17, respectively. Figure 18 illustrates that the adaptive estimate for the depth parameter $\hat{x}\square(t)$ approaches a constant. Figure 19 illustrates the linear and angular velocity of the WMR. The control torque inputs are presented in Fig. 20 and represent the torques applied after the gearing mechanism.

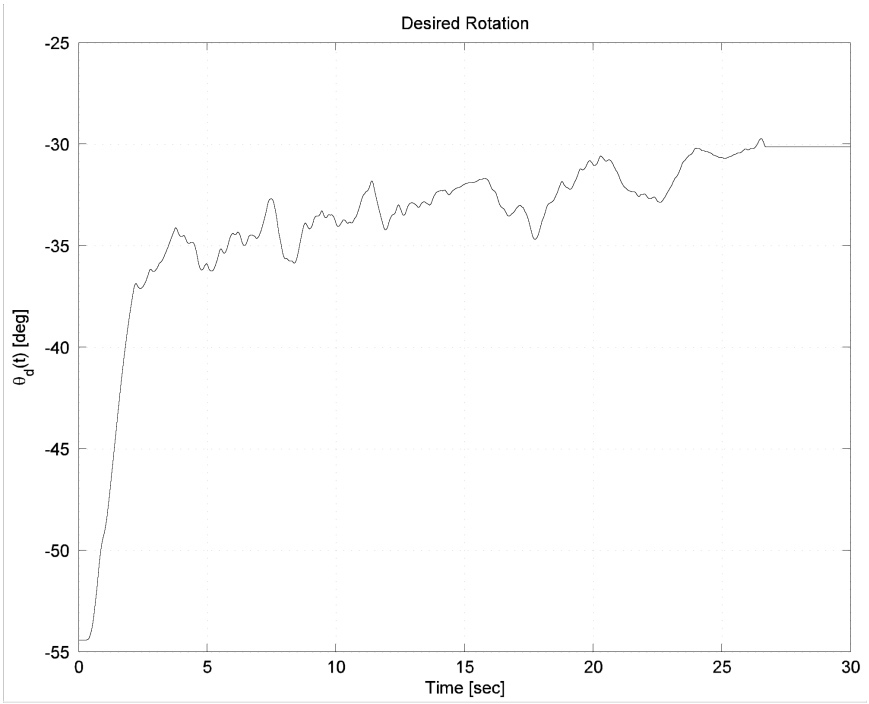

Fig. 15. Desired WMR rotation trajectory.

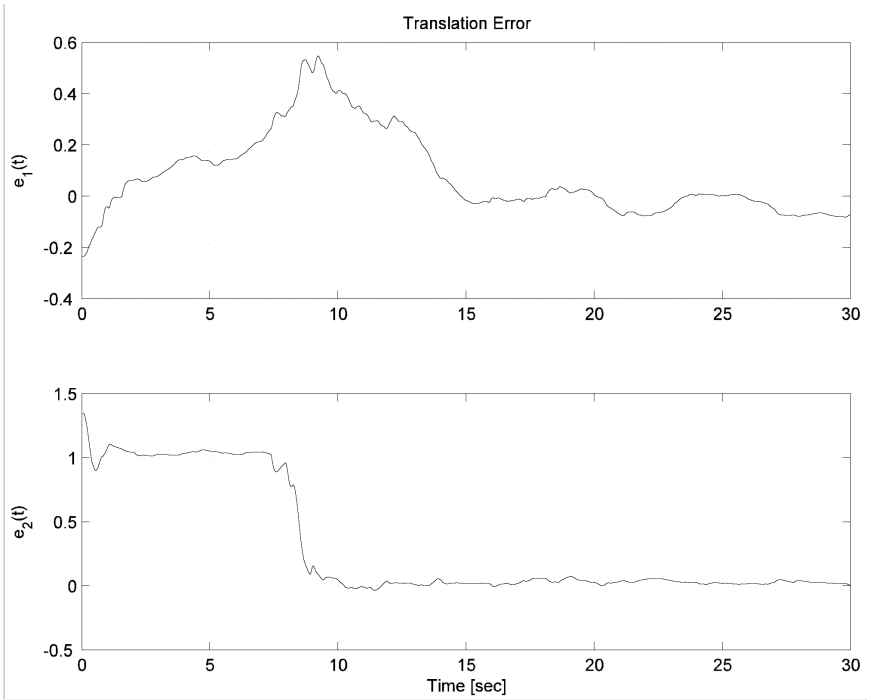

Fig. 16. WMR translation tracking error.

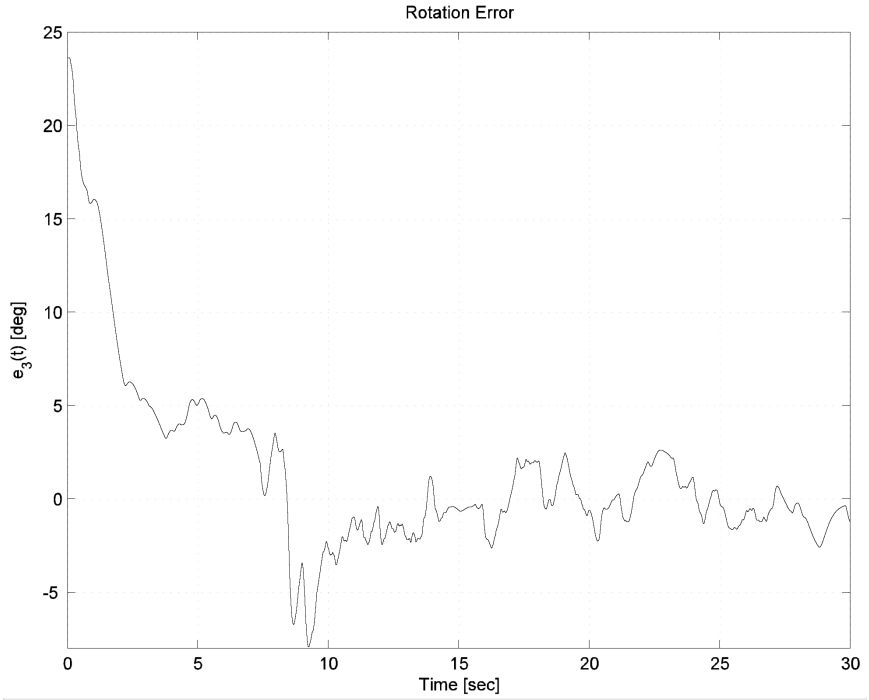

Fig. 17. WMR rotation tracking error.

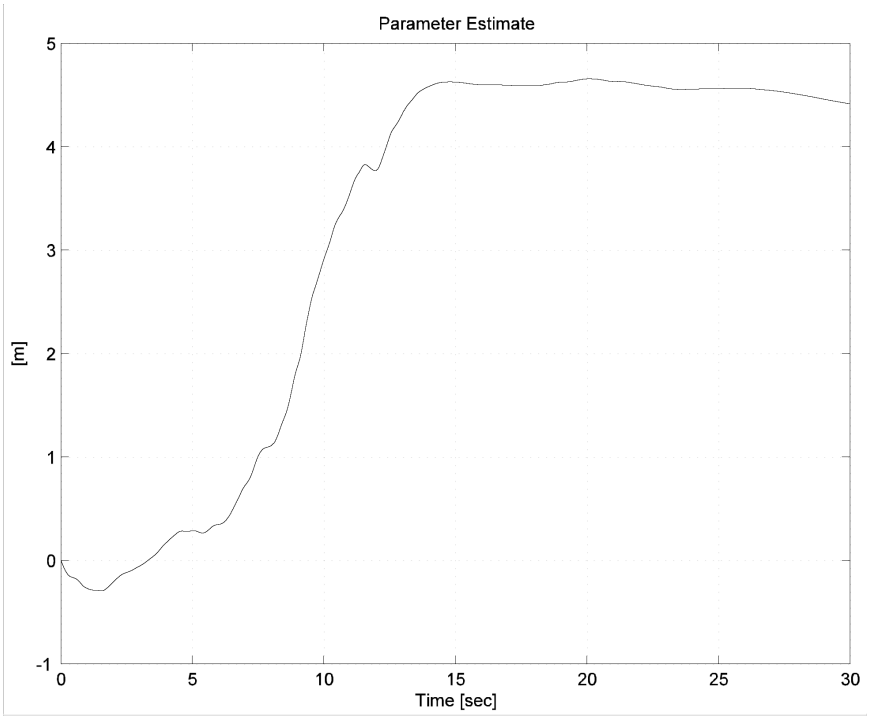

Fig. 18. Parameter estimate convergence.

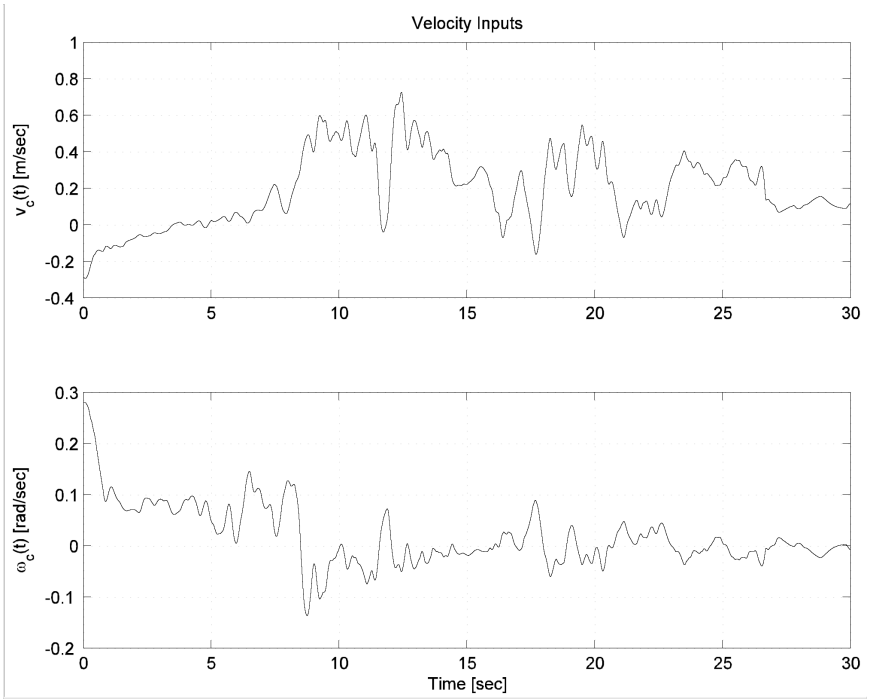

Fig. 19. Linear and angular velocity control inputs.

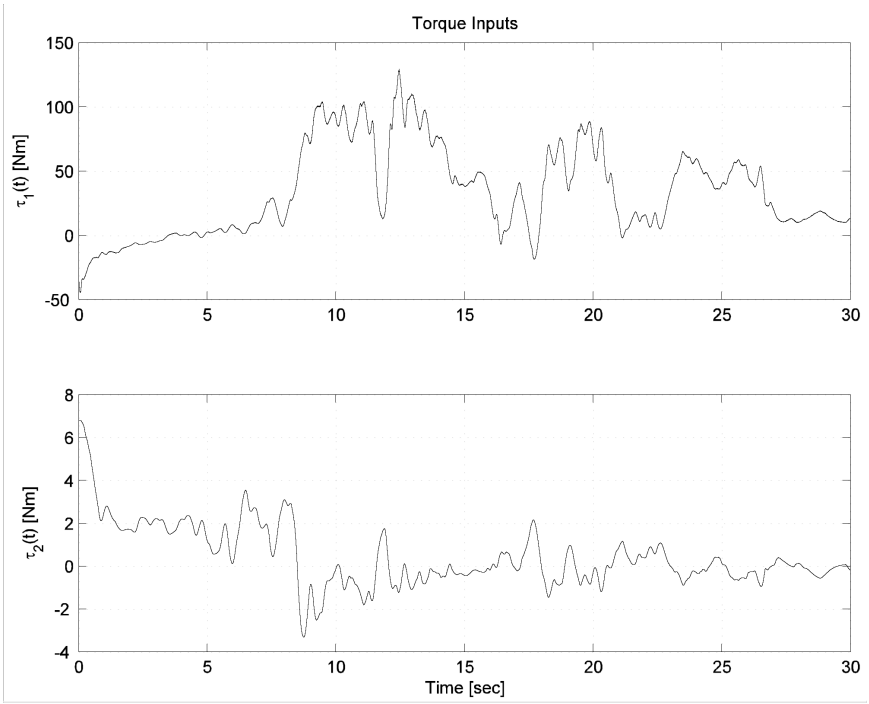

Fig. 20. Computed drive and steer motor torques.

To implement the regulation control experiment, the WMR was driven to a desired position and orientation and a snaphot was recorded along with the desired image-space coordinates of the feature points of a target. The WMR was then moved away from the desired location and the goal was to enable the WMR to return to the same position and orientation under visual servo control. A server program (executing on the image processing PC) similar to the tracking control problem determines the image-space coordinates of the target in the live image, computes the homography, decomposes the homography, computes the control signals $\square(t)$, m_y^*, and $\square(t)$, and then writes the signals to shared memory locations over a TCP/IP connection. Another client program (executing on the internal WMR PC) allocates shared memory locations for the live camera data, receives $\square(t)$, $m_y(t)$, and $\square(t)$, and writes the information into shared memory. The control program (executing on the internal WMR PC) given in Section B.6 of Appendix B reads data from the shared memory locations for the live camera, reads the WMR encoder signals (to enable a high-gain feedback torque control loop), computes the visual servo controller, provides real-time plotting and controller gain adjustment capabilities through the real-time control environment QMotor[14] (viewed through the remote PC), and commands a motor voltage signal to the drive and steering motors. Figure 21 and Fig. 22 illustrate the translation and rotation regulation errors obtained after adjusting the control gains as described for the tracking controller. Figure 23 illustrates the depth ratio error (i.e., the desired distance from the camera to the target divided by the actual distance from the camera to the target), where perfect regulation of the range to the target would be represented by a ratio of 1. Figure 24 illustrates the convergence of the depth parameter estimate, and Fig. 25 illustrates the computed drive and steering motor torques.

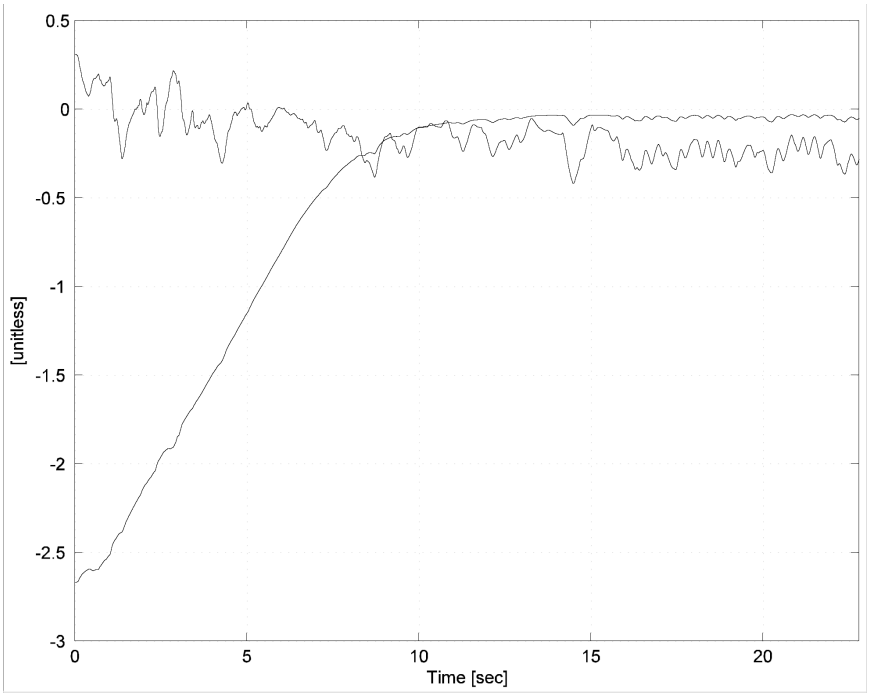

Fig. 21. WMR translation regulation errors.

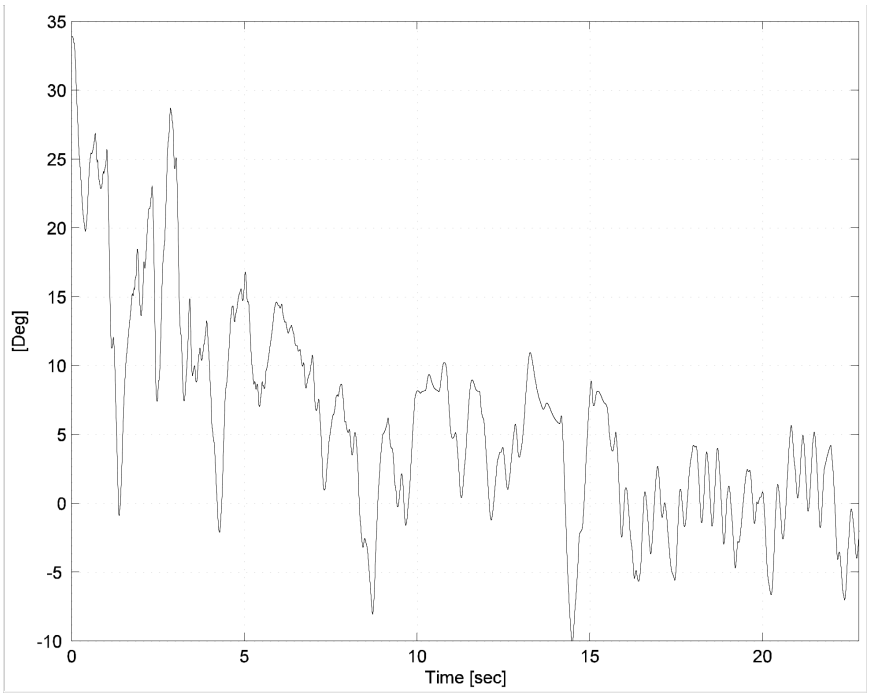

Fig. 22. WMR rotation regulation error.

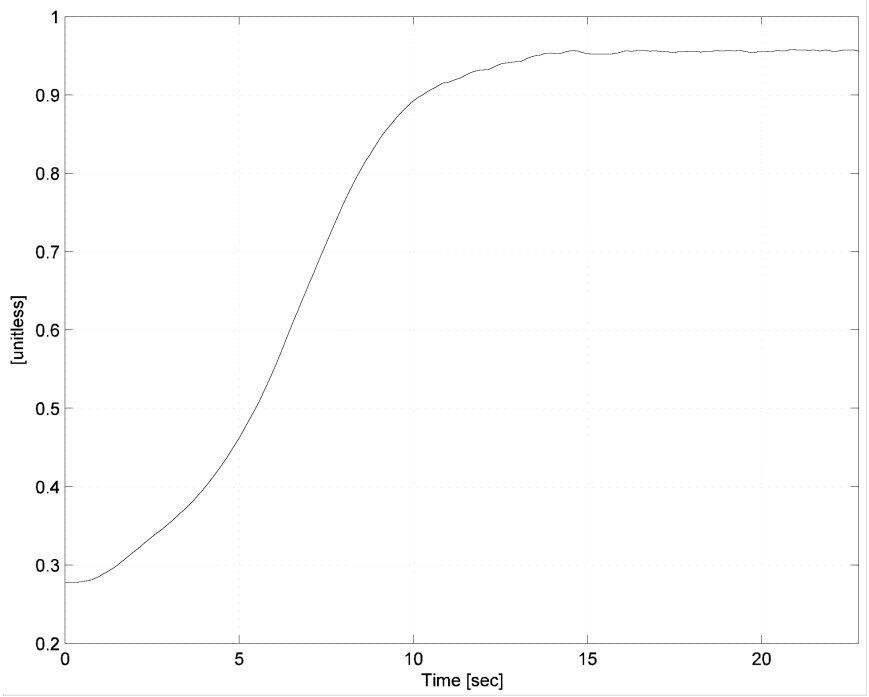

Fig. 23. Depth ratio error.

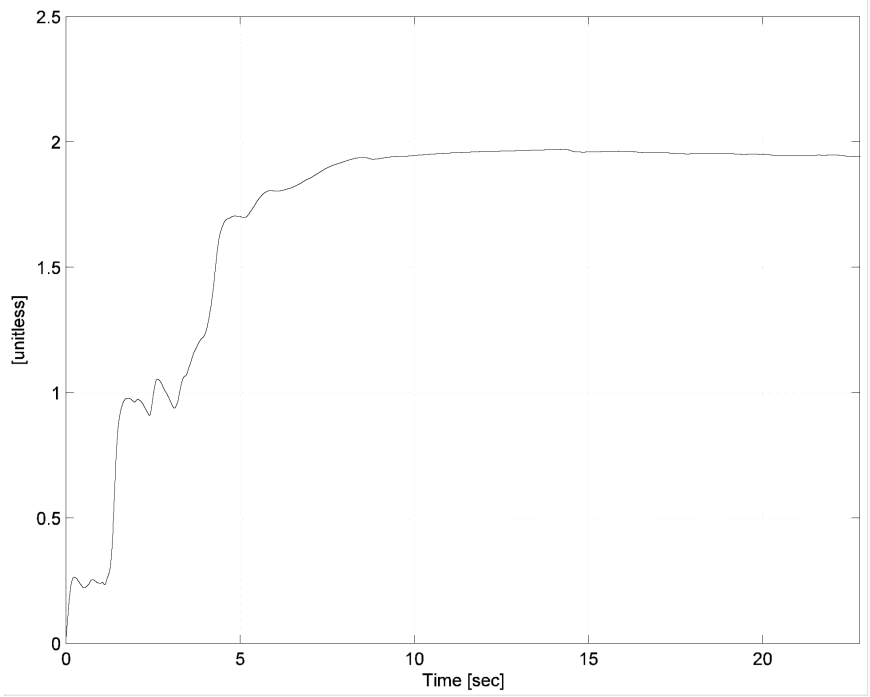

Fig. 24. Parameter estimate convergence.

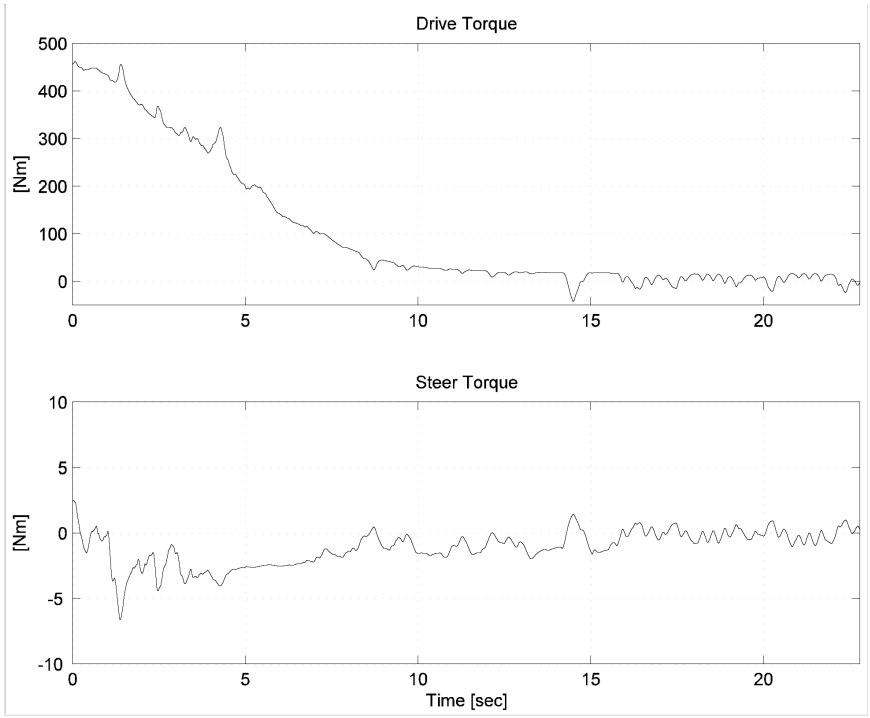

Fig. 25. Computed drive and steer motor torques.

4. CONCLUSIONS

Autonomous capabilities of a robot manipulator system and a WMR system were demonstrated through several visual servo controllers. Specifically, the results from three proof-of-concept visual servo controllers that are based on recent fundamental research results are described. The implementation of a cooperative visual servo control scheme was described in which a camera-in-hand and a fixed camera were used to track a moving target despite uncertainty in the camera calibration and the unknown constant distance from the camera to a target where the camera is mounted on the end-effector of a 6-DOF hydraulic robot manipulator. The implementation of homography-based visual servo tracking and regulation controllers for a WMR were also described. The results from the three demonstrations validate recent new advancements in visual servo control theory. With additional applied research and integration with advanced image-processing methods, the developed control design methods could impact numerous government, military, and private industries that require autonomous robotic operation.

REFERENCES

[1] M. W. Noakes, L. J. Love, P. D. Lloyd, "Telerobotic Planning and Control for DOE D&D Operations," *Proceedings of the IEEE Int. Conf. on Robotics and Automation*, Washington, DC, May 11-15, 2002, pp. 3485-3492.

[2] W. E. Dixon, E. Zergeroglu, Y. Fang, and D. M. Dawson, "Object Tracking by a Robot Manipulator: A Robust Cooperative Visual Servoing Approach," *Proceedings of the IEEE International Conference on Robotics and Automation*, Washington, DC, May 2002, pp. 211-216.

[3] W. E. Dixon and L. J. Love, "Lyapunov-Based Visual Servo Control for Robotic Deactivation and Decommissioning," *Proceedings of the Biennial ANS International Spectrum Conference*, Reno, Nevada, August 2002.

[4] V. Chitrakaran, D. M. Dawson, W. E. Dixon, and J. Chen, "Identification of a Moving Object's Velocity with a Fixed Camera," *Automatica*, to appear: see also *Proceedings of the IEEE Conference on Decision and Control*, Maui, Hawaii USA, December 2003, pp. 5402-5407.

[5] W. E. Dixon, Y. Fang, D. M. Dawson, and J. Chen, "Adaptive Range Identification for Exponential Visual Servo Control," *Proceedings of the 2003 IEEE International Symposium on Intelligent Control*, Houston, Texas, October 2003, pp. 46-51.

[6] Y. Fang, A. Behal, W. E. Dixon, and D. M. Dawson, "Adaptive 2.5D Visual Servoing of Kinematically Redundant Robot Manipulators," *Proceedings of the IEEE Conference on Decision and Control*, Las Vegas, Nevada, December 2002, pp. 2860-2865.

[7] Y. Fang, W. E. Dixon, and D. M. Dawson, "Adaptive 2.5D Visual Servoing of Cartesian Robots," *Proceedings of the International Conference on Control, Automation, Robotics and Vision (ICARCV)*, Kunming, China, December 6-9, 2004, submitted.

[8] Y. Fang, W. E. Dixon, D. M. Dawson, and J. Chen, "An Exponential Class of Model-Free Visual Servoing Controllers in the Presence of Uncertain Camera Calibration," *International Journal of Robotics and Automation*, submitted: see also, *Proceedings of the IEEE Conference on Decision and Control*, Maui, Hawaii, December 2003, pp. 5390-5395.

[9] J. Chen, A. Behal, D. M. Dawson, and W. E. Dixon, "Adaptive Visual Servoing in the Presence of Intrinsic Calibration Uncertainty," *Proceedings of the IEEE Conference on Decision and Control*, Maui, Hawaii, December 2003, pp. 5396-5401.

[10] W. E. Dixon, "Teach by Zooming: A Camera Independent Alternativeto Teach By Showing Visual Servo Control," *Proceedings of the IEEE International Conference on Intelligent Robots and Systems*, Las Vegas, Nevada, October 2003, pp. 749-754.

[11] W. E. Dixon, D. M. Dawson, E. Zergeroglu, and A. Behal, *Nonlinear Control of Wheeled Mobile Robots*, Springer-Verlag London Ltd, 2000.

[12] Y. Fang, W. E. Dixon, D. M. Dawson, and P. Chawda, "Homography-Based Visual Servoing of Wheeled Mobile Robots," *IEEE Transactions on Systems, Man, and Cybernetics -Part B: Cybernetics*, submitted: see also, *Proceedings of the IEEE Conference on Decision and Control*, Las Vegas, Nevada, December 2002, pp. 2866-2871.

[13] J. Chen, W. E. Dixon, D. M. Dawson, and M. McIntyre, "Homography-Based Visual Servo Tracking Control of a Wheeled Mobile Robot," *IEEE Transactions on Robotics and Automation*, submitted: see also, *Proceedings of the IEEE International Conference on Intelligent Robots and Systems*, Las Vegas, Nevada, October 2003, pp. 1814-1819.

[14] M. S. Loffler, N. P. Costescu, and D. M. Dawson, "QMotor 3.0 and the QMotor Robotic Toolkit: A PC-Based Control Platform," *IEEE Trans. Control Systems Magazine* **22**(3), 12-26, (2002).

[15] O. Faugeras, *Three-Dimensional Computer Vision*, The MIT Press, Cambridge, Massachusetts, 2001.

[16] Z. Zhang and A. R. Hanson, "Scaled Euclidean 3D Reconstruction Based on Externally Uncalibrated Cameras," *Proccedings of the IEEE Symp. on Comp. Vision*, 1995, pp. 37-42.

[17] M. W. Spong and M. Vidyasagar, *Robot Dynamic and Control*, John Wiley and Sons, Inc: New York, New York, 1989.

APPENDIX A. DEVELOPED SOFTWARE FOR COOPERATIVE VISUAL SERVO CONTROL

A.1 SERVER FOR FIXED CAMERA

```
///========================================================================
/// File:        fixedserver.cpp
/// Description: Writes the bright spot pixel information from the fixed cam
///              into shared memory locations xPosfix, yPosfix created by the
///              netclient.
///              Error Codes:
///                 -1  FIFO Overflow
///                 -2  Missed frame (not enough computing power)
///                 -3  No pixels brighter than the threshhold
///                 -4  Hardware exception interrupt
///                 -5  Server exited
///========================================================================

#include "RoadRunner.hpp"
#include "DmaBuffer.hpp"
#include "SharedMemory.hpp"
#include "CmdLineArgs.hpp"
#include <sys/types.h>
#include <sys/sched.h>
#include <unistd.h>
#include <sys/stat.h>
#include <iostream.h>
#include <fcntl.h>
#include <sys/mman.h>
#include <stdlib.h>
#include <signal.h>
#include <math.h>
#include <sys/types.h>
#include <sys/qnx_glob.h>

// Global variables needed by cleanup functions
RoadRunner *rr = 0;
DmaBuffer *frameBuffer0;
DmaBuffer *frameBuffer1;
int *xPosfix;
int *yPosfix;

// Function prototypes
void signalHandler(int);
void cleanUpAndExit(void);

///========================================================================
/// Name:        main
/// Description: The main program function
///========================================================================
int main(int argc, char *argv[])
{
       // Command line parameter stuff
       CmdLineArgs args(argc, argv);
       const char *cameraFile;
       int board;

       // Camera/image parameters
       int xSize = 0;
       int ySize = 0;
       int bytesPerPixel = 0;
       int imageSize;
       // DMA buffer and bright spot stuff
       char *buffer[2];
```

```
char *currentPixel;
int x;
int y;
int brightX;
int brightY;
int tap;
int pixel;
int brightValue;
int bank;

// Shared memory stuff
SharedMemory xPosfixShm;
SharedMemory yPosfixShm;

// Get command line parameters
cameraFile = args.getStringOption("camera", "DaCad256E4Raw.cam");
board = args.getIntegerOption("board", 0);
if(args.d_status.isStatusError())
{
       cerr << "server: Error parsing command line" << endl;
       cerr << args.d_status.getMessageText();
       exit(0);
}

// Create the shared memories
xPosfix = (int *)xPosfixShm.create("xPosfix", sizeof(int));
if(xPosfixShm.isStatusError())
{
       cerr << "Error creating xPosfix shared memory" << endl;
       exit(0);
}
yPosfix = (int *)yPosfixShm.create("yPosfix", sizeof(int));
if(yPosfixShm.isStatusError())
{
       cerr << "Error creating yPosfix shared memory" << endl;
       exit(0);
}
*xPosfix = -3;
*yPosfix = -3;

// Find the board
rr = new RoadRunner(board);
if(rr->d_status.isStatusError())
{
       cerr << "server: Error locating RoadRunner" << endl;
       cerr << rr->d_status.getMessageText();
       exit(-1);
}

// attach the signal handler to shut down the board when we exit by:
signal(SIGTERM, signalHandler);    // the slay utility
signal(SIGINT, signalHandler);     // or by pressing CNTL-C


// Print out info about the board
rr->printBoardInfo(255);

// Set up the board
rr->initialize(cameraFile);
if(rr->d_status.isStatusError())
{
       cerr << "server: Error initializing board" << endl;
       cerr << rr->d_status.getMessageText();
       cleanUpAndExit();
}
```

```
// Get info
xSize = rr->getXSize();
ySize = rr->getYSize();
bytesPerPixel = (int)ceil( (double)rr->getBitsPerPixel() / (double)8);
imageSize = xSize * ySize * bytesPerPixel;

// Allocate the DMA buffers
frameBuffer0 = new DmaBuffer(imageSize);
if(frameBuffer0->d_status.isStatusError())
{
       cerr << "server: Error allocating frame buffer 0" << endl;
       cerr << frameBuffer0->d_status.getMessageText();
       cleanUpAndExit();
}
buffer[0] = (char *)frameBuffer0->getVirtualAddress();

frameBuffer1 = new DmaBuffer(imageSize);
if(frameBuffer1->d_status.isStatusError())
{
       cerr << "server: Error allocating frame buffer 1" << endl;
       cerr << frameBuffer0->d_status.getMessageText();
       cleanUpAndExit();
}
buffer[1] = (char *)frameBuffer1->getVirtualAddress();

// Set up the DMA descriptors
if(rr->loadQtab(frameBuffer0->getPhysicalAddress(), 0, -1))
{
       cerr << "server: Error loading QTAB bank 0" << endl;
       cerr << rr->d_status.getMessageText();
       cleanUpAndExit();
}
if(rr->loadQtab(frameBuffer1->getPhysicalAddress(), 1, -1))
{
       cerr << "server: Error loading QTAB bank 1" << endl;
       cerr << rr->d_status.getMessageText();
       cleanUpAndExit();
}

// Enable interrupts
rr->enableDmaInterrupt();
rr->enableFifoInterrupt();
rr->enableHwInterrupt();

// Start out by aquiring into buffer 0
rr->setNextBank(0);
bank = 0;


if(rr->startDma())
{
       cerr << "server: Error starting DMA transfer" << endl;
       cerr << rr->d_status.getMessageText();
       cleanUpAndExit();
}

cout << "Processing frames...." << endl;

// Run at high priority
qnx_spawn_options.priority = 28;

// Start acquiring
rr->grab();
```

```
// Just keep them frames comin'
for(;;)
{
	// If we are filling buffer 0, we'll next fill buffer 1 (but we'll
	// continue processing buffer 0)
	if(bank)
		rr->setNextBank(0);
	else
		rr->setNextBank(1);

	// Wait for an interrupt
	switch(rr->waitForAnyInterrupt(0))
	{
		case RoadRunner::e_dmaInt:
			// The buffer has been filled.
			// Check to see if we've missed frames
			if(rr->checkForInterrupt(RoadRunner::e_dmaInt) != -1)
			{
				*xPosfix = -2;
				break;
			}

			// Find the bright spot.
			brightX = -3;
			brightY = -3;
			//brightValue = 48;
			brightValue = 0;
			currentPixel = buffer[bank];
			for(y = 0; y < ySize; y++)
			{
				for(x = 0, tap = 0; tap < 4; tap++)
				{
					for(pixel = 0; pixel < 65; pixel++, x++)
					{
						if(*currentPixel > brightValue)
						{
							// Remember the brightest spot
							brightX = x;
							brightY = y;
							brightValue = *currentPixel;
						}
						currentPixel = currentPixel + 4;
					}
					currentPixel = currentPixel - 259;
				}
				currentPixel = currentPixel + 268;
			}
			*xPosfix = brightX;
			*yPosfix = brightY;

			break;
		case RoadRunner::e_fifoInt:
			// FIFO overflow.  Indicate error
			*xPosfix = -1;
			*yPosfix = -1;
			cout << "o";

			// Recover
			rr->reset();
			if(rr->d_status.isStatusError())
			{
				cerr << "server: Error resetting after overflow"
		<< endl;
```

```
                                cerr << rr->d_status.getMessageText();
                                rr->shutdown();
                                cleanUpAndExit();
                        }
                        // Aquire into bank 0 first
                        rr->setNextBank(0);
                        bank = 0;
                        // Start acquiring again
                        rr->grab();
                        // Set up to aquire into bank 1 next.
                        break;
                case RoadRunner::e_hwInt:
                        // HW exception!
                        *xPosfix = -4;
                        *yPosfix = -4;
                        cout << "h";

                        // Recover
                        if(rr->reset())
                        {
                                cerr << "server: Error resetting after hardware
                                exception!" << endl;
                                cerr << rr->d_status.getMessageText();
                                rr->shutdown();
                                cleanUpAndExit();
                        }
                        // Aquire into bank 0 first
                        rr->setNextBank(0);
                        bank = 0;
                        // Start acquiring again
                        rr->grab();
                        // Set up to aquire into bank 1 next.
                        break;
                default:
                        cerr << "server: Unknown interrupt or error!" << endl;
                        cleanUpAndExit();
        }
        // OK, that frame is done.  Switch to the other bank.
        bank ^= 1;
    }
}

///========================================================================
/// Name:        signalHandler
/// Description: Catches signals in order to shut down to board before
///              exiting.
///========================================================================
void signalHandler(int)
{
    cleanUpAndExit();
}

///========================================================================
/// Name:        cleanUpAndExit
/// Description: Cleans up and exits
///========================================================================
void cleanUpAndExit(void)
{
    if(xPosfix)
        *xPosfix = -5;
        if(yPosfix)
        *yPosfix = -5;

    if(rr)
    {
```

```
                // Check for error so if we didn't map the registers we dodn't get
                // SIGSEG
                if(rr->d_status.isStatusOk())
                        rr->shutdown();
                delete rr;
        }

        if(frameBuffer0)
                delete frameBuffer0;

        if(frameBuffer1)
                delete frameBuffer1;

        exit(0);
}
```

A.2 SERVER FOR THE CAMERA-IN-HAND

```
///========================================================================
/// File:        handserver.cpp
/// Description: Writes the bright spot pixel information from the in-hand
///              into shared memory locations xPoshand, yPoshand.
///              Error Codes:
///                 -1  FIFO Overflow
///                 -2  Missed frame (not enough computing power)
///                 -3  No pixels brighter than the threshhold
///                 -4  Hardware exception interrupt
///                 -5  Server exited
/// History:
/// 04/17/02 WED Created
/// 06/12/02 WED & TJF got acceptable in-hand data from this server!
///========================================================================

#include "RoadRunner.hpp"
#include "DmaBuffer.hpp"
#include "SharedMemory.hpp"
#include "CmdLineArgs.hpp"
#include <sys/types.h>
#include <sys/sched.h>
#include <unistd.h>
#include <sys/stat.h>
#include <iostream.h>
#include <fcntl.h>
#include <sys/mman.h>
#include <stdlib.h>
#include <signal.h>
#include <math.h>

// Global variables needed by cleanup functions
RoadRunner *rr = 0;
DmaBuffer *frameBuffer0;
DmaBuffer *frameBuffer1;
int *xPosfix;
int *yPosfix;

// Function prototypes
void signalHandler(int);
void cleanUpAndExit(void);
///========================================================================
/// Name:        main
/// Description: The main program function
///========================================================================
int main(int argc, char *argv[])
{
```

```
// Command line parameter stuff
CmdLineArgs args(argc, argv);
const char *cameraFile;
int board;

// Camera/image parameters
int xSize = 0;
int ySize = 0;
int bytesPerPixel = 0;
int imageSize;

// DMA buffer and bright spot stuff
char *buffer[2];
char *currentPixel;
int x;
int y;
int brightX;
int brightY;
int tap;
int pixel;
int brightValue;
int bank;

// Shared memory stuff
SharedMemory xPosfixShm;
SharedMemory yPosfixShm;

// Get command line parameters
cameraFile = args.getStringOption("camera", "DaCad256E4Raw.cam");
board = args.getIntegerOption("board", 0);
if(args.d_status.isStatusError())
{
       cerr << "server: Error parsing command line" << endl;
       cerr << args.d_status.getMessageText();
       exit(0);
}

// Create the shared memories
xPosfix = (int *)xPosfixShm.create("xPosfix", sizeof(int));
if(xPosfixShm.isStatusError())
{
       cerr << "Error creating xPosfix shared memory" << endl;
       exit(0);
}
yPosfix = (int *)yPosfixShm.create("yPosfix", sizeof(int));
if(yPosfixShm.isStatusError())
{
       cerr << "Error creating yPosfix shared memory" << endl;
       exit(0);
}
*xPosfix = -3;
*yPosfix = -3;

// Find the board
rr = new RoadRunner(board);
if(rr->d_status.isStatusError())
{
       cerr << "server: Error locating RoadRunner" << endl;
       cerr << rr->d_status.getMessageText();
       exit(-1);
}

// attach the signal handler to shut down the board when we exit by:
signal(SIGTERM, signalHandler);    // the slay utility
signal(SIGINT, signalHandler);     // or by pressing CNTL-C
```

```
// Print out info about the board
rr->printBoardInfo(255);

// Set up the board
rr->initialize(cameraFile);
if(rr->d_status.isStatusError())
{
       cerr << "server: Error initializing board" << endl;
       cerr << rr->d_status.getMessageText();
       cleanUpAndExit();
}

// Get info
xSize = rr->getXSize();
ySize = rr->getYSize();
bytesPerPixel = (int)ceil( (double)rr->getBitsPerPixel() / (double)8);
imageSize = xSize * ySize * bytesPerPixel;

// Allocate the DMA buffers
frameBuffer0 = new DmaBuffer(imageSize);
if(frameBuffer0->d_status.isStatusError())
{
       cerr << "server: Error allocating frame buffer 0" << endl;
       cerr << frameBuffer0->d_status.getMessageText();
       cleanUpAndExit();
}
buffer[0] = (char *)frameBuffer0->getVirtualAddress();

frameBuffer1 = new DmaBuffer(imageSize);
if(frameBuffer1->d_status.isStatusError())
{
       cerr << "server: Error allocating frame buffer 1" << endl;
       cerr << frameBuffer0->d_status.getMessageText();
       cleanUpAndExit();
}
buffer[1] = (char *)frameBuffer1->getVirtualAddress();

// Set up the DMA descriptors
if(rr->loadQtab(frameBuffer0->getPhysicalAddress(), 0, -1))
{
       cerr << "server: Error loading QTAB bank 0" << endl;
       cerr << rr->d_status.getMessageText();
       cleanUpAndExit();
}
if(rr->loadQtab(frameBuffer1->getPhysicalAddress(), 1, -1))
{
       cerr << "server: Error loading QTAB bank 1" << endl;
       cerr << rr->d_status.getMessageText();
       cleanUpAndExit();
}

// Enable interrupts
rr->enableDmaInterrupt();
rr->enableFifoInterrupt();
rr->enableHwInterrupt();

// Start out by aquiring into buffer 0
rr->setNextBank(0);
bank = 0;

if(rr->startDma())
{
       cerr << "server: Error starting DMA transfer" << endl;
       cerr << rr->d_status.getMessageText();
```

```
        cleanUpAndExit();
}

cout << "Processing frames...." << endl;

// Run at high priority
qnx_scheduler(0, 0, SCHED_FIFO, 28, 0);

// Start acquiring
rr->grab();

// Just keep them frames comin'
for(;;)
{
        // If we are filling buffer 0, we'll next fill buffer 1 (but we'll
        // continue processing buffer 0)
        if(bank)
                rr->setNextBank(0);
        else
                rr->setNextBank(1);

        // Wait for an interrupt
        switch(rr->waitForAnyInterrupt(0))
        {
                case RoadRunner::e_dmaInt:
                        // The buffer has been filled.
                        // Check to see if we've missed frames
                        if(rr->checkForInterrupt(RoadRunner::e_dmaInt) != -1)
                        {
                                *xPosfix = -2;
                                break;
                        }

                        // Find the bright spot.
                        brightX = -3;
                        brightY = -3;
                        //brightValue = 48;
                        brightValue = 0;
                        currentPixel = buffer[bank];
                        for(y = 0; y < ySize; y++)
                        {
                                for(x = 0, tap = 0; tap < 4; tap++)
                                {
                                        for(pixel = 0; pixel < 65; pixel++, x++)
                                        {
                                                if(*currentPixel > brightValue)
                                                {
                                                        // Remember the brightest spot
                                                        brightX = x;
                                                        brightY = y;
                                                        brightValue = *currentPixel;
                                                }
                                                currentPixel = currentPixel + 4;
                                        }
                                        currentPixel = currentPixel - 259;
                                }
                                currentPixel = currentPixel + 268;
                        }
                        *xPosfix = brightX;
                        *yPosfix = brightY;

                        break;
                case RoadRunner::e_fifoInt:
                        // FIFO overflow.  Indicate error
                        *xPosfix = -1;
```

```
                        *yPosfix = -1;
                        cout << "o";

                        // Recover
                        rr->reset();
                        if(rr->d_status.isStatusError())
                        {
                                cerr << "server: Error resetting after overflow" << endl;
                                cerr << rr->d_status.getMessageText();
                                rr->shutdown();
                                cleanUpAndExit();
                        }
                        // Aquire into bank 0 first
                        rr->setNextBank(0);
                        bank = 0;
                        // Start acquiring again
                        rr->grab();
                        // Set up to aquire into bank 1 next.
                        break;
                case RoadRunner::e_hwInt:
                        // HW exception!
                        *xPosfix = -4;
                        *yPosfix = -4;
                        cout << "h";

                        // Aquire into bank 0 first
                        rr->setNextBank(0);
                        bank = 0;
                        // Start acquiring again
                        rr->grab();
                        // Set up to aquire into bank 1 next.
                        break;
                default:
                        cerr << "server: Unknown interrupt or error!" << endl;
                        cleanUpAndExit();
        }
        // OK, that frame is done.  Switch to the other bank.
        bank ^= 1;
    }
}

///============================================================================
/// Name:        signalHandler
/// Description: Catches signals in order to shut down to board before
///              exiting.
///============================================================================
void signalHandler(int)
{
    cleanUpAndExit();
}

///============================================================================
/// Name:        cleanUpAndExit
/// Description: Cleans up and exits
///============================================================================
void cleanUpAndExit(void)
{
    if(xPosfix)
        *xPosfix = -5;

    if(yPosfix)
        *yPosfix = -5;

    if(rr)
    {
```

```
                // Check for error so if we didn't map the registers we dodn't get
                // SIGSEG
                if(rr->d_status.isStatusOk())
                        rr->shutdown();
                delete rr;
        }

        if(frameBuffer0)
                delete frameBuffer0;

        if(frameBuffer1)
                delete frameBuffer1;

        exit(0);
}
```

A.3 SHARED MEMORY CLIENT

```
///=============================================================================
/// File:        netClient.cpp
/// Description: Sets up the server side of a TCP/IP socket and reads bright
///              spot values from the netServer into shared memory.
///=============================================================================
#include "CSocket.hpp"
#include "SharedMemory.hpp"
#include "CmdLineArgs.hpp"
#include <iostream.h>
#include <sys/sched.h>
#include <stdlib.h>

///=============================================================================
/// Name:        main
/// Description: The main program function
///=============================================================================
void main(int argc, char *argv[])
{
        CmdLineArgs args(argc, argv);
        Socket serverSocket;
        int socketPortNumber;
        SharedMemory xPoshandShm;
        SharedMemory yPoshandShm;
        int *xPoshand;
        int *yPoshand;
        int position[2];

        // Get command line parameters
        socketPortNumber = args.getIntegerOption("port", 10000);
        if(args.d_status.isStatusError())
        {
                cerr << "netClient: Error parsing command line" << endl;
                cerr << args.d_status.getMessageText();
                exit(-1);
        }
        // Create shared memory locations
        xPoshand = (int *) xPoshandShm.create ("xPoshand", sizeof(int));
        yPoshand = (int *) yPoshandShm.create ("yPoshand", sizeof(int));

        // Set up the socket
        cout << "Waiting for data on port " << socketPortNumber << endl;
        serverSocket.initServerSocket(socketPortNumber);

        // Go to high priority for a more deterministic response
        qnx_scheduler (0, 0, SCHED_FIFO, 27, 0);
        for(;;)
```

```
	{
		serverSocket.receive(position, 2*sizeof(int));
		*xPoshand = position[0];
		*yPoshand = position[1];
		if(*xPoshand == -5)
		{
			cout << "Video Server exited.  Quitting." << endl;
			break;
		}
	}
	serverSocket.close();
}
```

A.4 CONTROL PROGRAM

```
// ========================================================================
//  Control Program : CompVis.cpp
//  Description     : In-hand and fixed, robust hydraulic arm control
//                  : Qmotor program is for kinematic level control
//  Author          : Warren Dixon, John Rowe, Lonnie Love, T.J. Flynn
//  Start Date      : Mon Feb 18 3:30 2002
// ========================================================================

// ----- QRTS libraries -----
#include "ControlProgram.hpp"
#include "IOBoardClient.hpp"
#include "SharedMemory.hpp"
#include "CSocket.hpp"
#include "CmdLineArgs.hpp"
#include "TitanT2.h" // change to a .hpp file and put into /usr/qrts/include
#include "Vector.hpp"
#include "Matrix.h" // put Matrix.hpp in /usr/qrts/include
#include "StraightLinePathPlanner.hpp"
#include "TitanIIStraightLinePathPlanner.hpp"
#include "ButterworthFilter.hpp"

// ----- C standard libraries -----
#include <unistd.h>
#include <math.h>
#include <sys/sched.h>
#include <iostream.h>
#include <conio.h>
#include <errno.h>
#include <stdlib.h>
#include <stdio.h>
#include <netdb.h>
#include <arpa/inet.h>
#include <sys/socket.h>
#include <netinet/in.h>

#include "alc_defs.h"

#define ALC_IP "192.168.20.2"

//=========================================================================
// Class definition of the CompVis class
//=========================================================================

class CompVis : public ControlProgram
{
	protected:

		// ----- Log Variables -----
		double q0;     //Quaternion variables
```

```
        double q1;
        double q2;
        double q3;
        double j1;     //individual joint positions
        double j2;
        double j3;
        double j4;
        double j5;
        double j6;
        double dx_d; //desired task-space (x) velocity
        double dy_d; //desired task-space (y) velocity
        double x_t;    //desired task-space (x) position
        double y_t;    //desired task-space (y) position
        double z_t;    //desired task-space (z) position
        double x_hand; //pixel-space x-coordinate from in-hand camera
        double y_hand; //pixel-space y-coordinate from in-hand camera
        double x_handPrev; // previous control cycle x-hand pixel data
        double y_handPrev; // previous control cycle y-hand pixel data
        double x_fixed;       //pixel-space x-coordinate from fixed camera
        double y_fixed;       //pixel-space y-coordinate from fixed camera
        double R[3][3]; //quaternion matrix which forms roation matrix
        double RT[3][3]; //stores transpose of quaternion matrix R
        double y[3][1]; // multiplies RT to form rotation matrix
        double rotMatrix[3][1]; //rotation matrix formed from RT * y
        double xDot[3][1]; // cartesian positions differentiated
        double xDotxOld; // stores x-component of xDot
        double xDotyOld; // stores y-component of xDot
        double xAreaSum; // running sum of area under curve wrt x
        double yAreaSum; // running sum of area under curve wrt y
        int homeFlag; // indicates whether arm is at "home" position
int pathNeededFlag; // flag to determine if SLPP is needed
double x_h, y_h, z_h; // cartesian/translation "home" variables
        double q0_h, q1_h, q2_h, q3_h; // "home" orientation/rotation
        double temp; // temporary storage during matrix multiplication
        int row; // used for looping during matrix multiplication
        int col; // used for looping during matrix multiplication
        TitanT2 theTitanT2; // the TitanT2 object used to calculate
                // forward/inverse kinematics
        ColVector currentTipPosition; // current x, y, and z cartesian
                              // coordinates of the end-effector
        ColVector currentTipQuaternion; // 4 quaternions representing the
                               // current orientation of the tip
        ColVector homeyTipPosition; // x, y, and z cartesian "home"
                         // coordinates of the tip
        ColVector homeyTipQuaternion; // "home" orientation
        ColVector desiredTipPosition; // stores desired cartesian x, y,
                              // z position of tip

        // straight line path planner object for the T2
        TitanIIStraightLinePathPlanner theTitanTIISLPP;

        ColVector desiredJointPosition; // holds 6 desired joint positions
        ColVector currentJointPosition; // holds 6 current joint positions
        ColVector homeJoint; // 6 "home" joint positions
        float maxTranslationalVelocity;
        float maxRotationalVelocity;
        float translationalAccel;
        float rotationalAccel;
        int numIterations; // number of points iterated
        double x_d, y_d, z_d;
        double q0_d, q1_d, q2_d, q3_d;
        double jd_1, jd_2, jd_3, jd_4, jd_5, jd_6;
        double errorflag;
        double initflag;
```

```
        double x_offset, y_offset;
        double R11, R12, R21, R22;
        double rotMatrix10, rotMatrix20;
        double controlPeriod;
        double xFixedDot, yFixedDot;
        double yFixedPrev, xFixedPrev;

        // ----- Control Parameters -----
        double kp1, kp2; // Control Gain
        double kn;
        double rho_sq; // rho^2
        double rhoControlGain; //control gain for rho_sq
        double pixelThreshold; // allowable amount of x/y pixel
                               // displacement between cycles

        // ----- Ethernet Comm Variables
        outPKT alcPkt;
        outPKT *pAlcPkt;
        int sock;
        struct sockaddr_in server;
        char buf[1024];
        int status;
        hlcPACKET *pInPkt;
        int first, cntr;

        // ----- Other Variables -----
        SharedMemory xPoshandShm;    //Shared Memory Setup
        SharedMemory yPoshandShm;
        SharedMemory xPosfixShm;
        SharedMemory yPosfixShm;
        int *xPoshand;               //interger pointer to shared memory
        int *yPoshand;
        int *xPosfix;
        int *yPosfix;
        ButterworthFilter<double> filter;

public:

        // Constructor.
        CompVis(int argc, char *argv[]) : ControlProgram (argc, argv),
        currentTipPosition(3),
        currentTipQuaternion(4),
        homeyTipPosition(3),
        homeyTipQuaternion(4),
        desiredTipPosition(3),
        currentJointPosition(6),
        desiredJointPosition(6),
        theTitanTIISLPP(),
        homeJoint(6)
        {};

        // Destructor. Usually no need to make changes here
        ~CompVis () {};

        virtual int enterControl();
        virtual int startControl();
        virtual int control();
        virtual int stopControl();
        virtual int exitControl();
};


//============================================================================
// CompVis::enterControl
// ----------------------------------------------------------------------------
```

```
// This function is called when the control program is loaded. In standalone
// mode, this happens immediately. When using the GUI, it happens when the
// user loads the control program.
//========================================================================

int CompVis::enterControl()
{
       // ----- Log Variables -----

       registerLogVariable(&x_hand, "x_hand", "X_hand");
       registerLogVariable(&y_hand, "y_hand", "Y_hand");

       registerLogVariable(&j1, "joint_1", "Joint 1");
       registerLogVariable(&j2, "joint_2", "Joint 2");
       registerLogVariable(&j3, "joint_3", "Joint 3");
       registerLogVariable(&j4, "joint_4", "Joint 4");
       registerLogVariable(&j5, "joint_5", "Joint 5");
       registerLogVariable(&j6, "joint_6", "Joint 6");

       registerLogVariable(&x_fixed, "x_fixed", "X_fixed");
       registerLogVariable(&y_fixed, "y_fixed", "Y_fixed");

       registerLogVariable(&x_d, "x_d", "x_d");
       registerLogVariable(&y_d, "y_d", "y_d");
       registerLogVariable(&z_d, "z_d", "z_d");

       registerLogVariable(&q0_d, "q0_d", "q0_d");
       registerLogVariable(&q1_d, "q1_d", "q1_d");
       registerLogVariable(&q2_d, "q2_d", "q2_d");
       registerLogVariable(&q3_d, "q3_d", "q3_d");

       registerLogVariable(&jd_1, "jd_1", "Joint 1 des");
       registerLogVariable(&jd_2, "jd_2", "Joint 2 des");
       registerLogVariable(&jd_3, "jd_3", "Joint 3 des");
       registerLogVariable(&jd_4, "jd_4", "Joint 4 des");
       registerLogVariable(&jd_5, "jd_5", "Joint 5 des");
       registerLogVariable(&jd_6, "jd_6", "Joint 6  des");
       registerLogVariable(&homeFlag, "homeflag", "homeflag");
      registerLogVariable(&errorflag, "errorflag", "errorflag");
       registerLogVariable(&initflag, "initflag", "initflag");

       registerLogVariable(&R11, "R11", "R11");
       registerLogVariable(&R12, "R12", "R12");
       registerLogVariable(&R21, "R21", "R21");
       registerLogVariable(&R22, "R22", "R22");

       registerLogVariable(&rotMatrix10, "rotMatrix10", "rotMatrix10");
       registerLogVariable(&rotMatrix20, "rotMatrix20", "rotMatrix20");
       registerLogVariable(&xFixedDot, "xfd", "xfd");
       registerLogVariable(&yFixedDot, "yfd", "yfd");

       registerLogVariable(&controlPeriod, "controlPeriod", "control period");

       // ----- Control Parameters -----

       registerControlParameter(&kp1, "kp1", "Proportional Gain");
      registerControlParameter(&kp2, "kp2", "Proportional Gain");
       registerControlParameter(&kn, "kn", "Damping Gain");
       registerControlParameter(&rhoControlGain, "rhoControlGain", "Rho Squared       Gain");
       registerControlParameter(&pixelThreshold, "pixelThreshold", "Max. pixel
       displacement");

       // Set all control parameters initially to zero
       clearAllControlParameters();

       qnx_scheduler(0, 0, SCHED_FIFO, 27, 0);
```

```
        // initialize ethernet comm
        pAlcPkt = &alcPkt;
        pInPkt  = (hlcPACKET *)&buf[0];

        // initialize packet to ALC
        for( int i=0; i<7; i++ )
                alcPkt.rightArmPos[i] = (float)0;

        alcPkt.mode = GUI_NONE;
        alcPkt.armSwitch = 0;
        alcPkt.panelSwitch = 0;

        // Create socket
        sock = socket( AF_INET, SOCK_STREAM, 0 );

        if( sock < 0 )
        {
                perror("Opening stream socket" );
                return(-1);
        }

        // Connect socket
        server.sin_family = AF_INET;
        server.sin_addr.s_addr = inet_addr( ALC_IP );
        server.sin_port =     htons( 9000 );

        if( connect( sock, (struct sockaddr*)&server, sizeof(server) ) < 0 )
        {
                perror( "connecting stream socket" );
                return(-1);
        }

        return 0;
}

//=============================================================================
// CompVis::startControl
// -----------------------------------------------------------------------------
// Called each time a control run is started. If running from the GUI, this
// will be called each time the START button is pushed.
//=============================================================================

int CompVis::startControl()
{
printf("Start Control\n");
                clearAllLogVariables();
                // ----- Initialize your clients here -----
        filter.setCutOffFrequency(10);
                filter.setSamplingTime(d_controlPeriod);
                filter.setDampingRatio(1.0);
                filter.setAutoInit();

                homeFlag = 0;
                pathNeededFlag = 1;
                temp = 0;
                numIterations = 0;
                initflag = 0;

                // initial values to indicate no in-hand pixel data has been read
                x_handPrev = 9999;
                y_handPrev = 9999;
                xFixedPrev = 9999;
                yFixedPrev = 9999;
```

```
// "home" joint column vector
homeJoint[0] = 0.14;
homeJoint[1] = 0.46;
homeJoint[2] = -1.69;
homeJoint[3] = 1.23;
homeJoint[4] = 0.01;
homeJoint[5] = -0.15;

// use "home" joint info to get homey tip and homey quaternion
theTitanT2.CalcForwardKinematics(homeJoint);

homeyTipQuaternion = theTitanT2.GetTipQuarternion();

homeyTipPosition = theTitanT2.GetTipPosition();

// "home" path parameters
maxTranslationalVelocity = 5;
maxRotationalVelocity = 45;
translationalAccel = 4.83;
rotationalAccel = 180;

pAlcPkt->mode = GUI_NONE;

// Read the 6 joint positions from ALC
cntr  = 0;
do
{
       if( send( sock, pAlcPkt, sizeof(alcPkt), 0 ) < 0 )
       {
              perror( "writing to stream socket" );
              return(-1);
       }

       status = recv( sock, &buf[0], sizeof(buf), 0 );
       if( status < 0 )
       {
              perror( "recv called" );
              return(-1);
       }

       if( pInPkt->mode == AM_ERROR )
              pAlcPkt->mode = GUI_CLEAR_ERROR;
       else
              pAlcPkt->mode = GUI_ENABLE;

       cntr++;
       if( cntr > 5 )
       {
              printf("Start Control count exceeded\n");
              return(-1);
       }
       sleep(1);
       } while( (pInPkt->mode != AM_SERVO) || (status != sizeof(hlcPACKET)) );

j1 = (double)pInPkt->armPos[0];
j2 = (double)pInPkt->armPos[1];
j3 = (double)pInPkt->armPos[2];
j4 = (double)pInPkt->armPos[3];
j5 = (double)pInPkt->armPos[4];
j6 = (double)pInPkt->armPos[5];

first = 1; // variable to alter first loop in control

sleep(1);

if( send( sock, pAlcPkt, sizeof(alcPkt), 0 ) < 0 )
```

```
		{
			perror( "writing to stream socket" );
			return(-1);
		}

		return 0;
}


//=============================================================================
// CompVis::control
//-----------------------------------------------------------------------------
// Called each control cycle. Do your input, control computations, and output
// here. If you return 0, the control will continue to execute. If you return
// nonzero, the control will abort. You may want to abort if some error
// condition occurs (excessive velocity,  etc.)
//=============================================================================

int CompVis::control()
{
	// ----- Take care of moving arm to home position first -----

	pAlcPkt->mode = GUI_TR;

	if(homeFlag != 1)
	{
		// read in 6 current joint positions
		status = recv( sock, &buf[0], sizeof(buf), 0 );

		if( status < 0 )
		{
			perror( "recv called" );
			return(-1);
		}

		j1 = (double)pInPkt->armPos[0];
		j2 = (double)pInPkt->armPos[1];
		j3 = (double)pInPkt->armPos[2];
		j4 = (double)pInPkt->armPos[3];
		j5 = (double)pInPkt->armPos[4];
		j6 = (double)pInPkt->armPos[5];

		// store newly read joint positions in column vector
		currentJointPosition[0] = j1;
		currentJointPosition[1] = j2;
		currentJointPosition[2] = j3;
		currentJointPosition[3] = j4;
		currentJointPosition[4] = j5;
		currentJointPosition[5] = j6;

		// use current joint positions to get corresponding current
		// cartesian points and current quaternion
		theTitanT2.CalcForwardKinematics(currentJointPosition);

		// store newly calculated tip x, y, and z positions
		currentTipPosition = theTitanT2.GetTipPosition();

		// store newly calculated tip orientation
		currentTipQuaternion = theTitanT2.GetTipQuarternion();

		// create a straight line path plan if not already made
		if(pathNeededFlag == 1)
		{
		theTitanTIISLPP.ResetPathPlanner(currentTipPosition,
		homeyTipPosition,
```

```
        currentTipQuaternion,
        homeyTipQuaternion,
        maxTranslationalVelocity,
        translationalAccel,
        maxRotationalVelocity,
        rotationalAccel,
        d_controlPeriod);

        // ensure that only 1 SLPP is created while moving arm to "home"
               pathNeededFlag = 0;
        }

        // iterate x, y, and z tip positions and quaternion to next point
        theTitanTIISLPP.iteratePathPlanner();
        numIterations = numIterations + 1;

        // get 6 new joint positions based on iterated Cartesian points
        // and quaternion

        theTitanT2.CalcInverseKinematics(theTitanTIISLPP.GetPositionVector(),
        theTitanTIISLPP.GetQuaternion());

        // send new desired joint position data to ALC to move arm
desiredJointPosition = theTitanT2.GetJointPosition();

        pAlcPkt->rightArmPos[0] = desiredJointPosition[0];
        pAlcPkt->rightArmPos[1] = desiredJointPosition[1];
        pAlcPkt->rightArmPos[2] = desiredJointPosition[2];
        pAlcPkt->rightArmPos[3] = desiredJointPosition[3];
        pAlcPkt->rightArmPos[4] = desiredJointPosition[4];
        pAlcPkt->rightArmPos[5] = desiredJointPosition[5];

        if( send( sock, pAlcPkt, sizeof(alcPkt), 0 ) < 0 )
        {
               perror( "writing to stream socket" );
               return(-1);
        }

         x_d = homeyTipPosition[0];
         y_d = homeyTipPosition[1];
         z_d = homeyTipPosition[2];

         q0_d = homeyTipQuaternion[0];
         q1_d = homeyTipQuaternion[1];
         q2_d = homeyTipQuaternion[2];
         q3_d = homeyTipQuaternion[3];

         jd_1 = desiredJointPosition[0];
         jd_2 = desiredJointPosition[1];
         jd_3 = desiredJointPosition[2];
         jd_4 = desiredJointPosition[3];
         jd_5 = desiredJointPosition[4];
         jd_6 = desiredJointPosition[5];

        // Must initialize the trapazoidal rule for homey position

        xDotxOld = y_d;
        xDotyOld = z_d;
        xAreaSum = xDotxOld;
        yAreaSum = xDotyOld;

        if(numIterations >= theTitanTIISLPP.GetNumberOfPathPoints())
        {
               homeFlag = 1; // now at (or close enough to) the "home"
               return 0; }
```

```
        else
                return 0; // otherwise just keep iterating to "home"
}


// ----- End home positioning of the arm -----


// ----- Begin actual kinematic control of the arm for tracking -----

// Read in joint positions at beginning of every control cycle
status = recv( sock, &buf[0], sizeof(buf), 0 );
if( status < 0 )
{
        perror( "recv called" );
        return(-1);
}

j1 = (double)pInPkt->armPos[0];
j2 = (double)pInPkt->armPos[1];
j3 = (double)pInPkt->armPos[2];
j4 = (double)pInPkt->armPos[3];
j5 = (double)pInPkt->armPos[4];
j6 = (double)pInPkt->armPos[5];

currentJointPosition[0] = j1;
currentJointPosition[1] = j2;
currentJointPosition[2] = j3;
currentJointPosition[3] = j4;
currentJointPosition[4] = j5;
currentJointPosition[5] = j6;

theTitanT2.CalcForwardKinematics(currentJointPosition);

currentTipQuaternion = theTitanT2.GetTipQuarternion();

//currentTipPosition = theTitanT2.GetTipPosition();

// read x & y IN-HAND data from shared memory

xPoshand = (int *)xPoshandShm.attach("xPoshand", sizeof(int));
if(xPoshandShm.isStatusError())
{
        cerr << "Error opening shared memory \"xPoshand\": ";
        cerr << strerror(errno) << endl;
        return 0;
}

yPoshand = (int *)yPoshandShm.attach("yPoshand", sizeof(int));
if(yPoshandShm.isStatusError())
{
        cerr << "Error opening shared memory \"yPoshand\": ";
        cerr << strerror(errno) << endl;
        return 0;
}
if (initflag != 1)
        {
        x_offset = *xPoshand - (double)128;
        y_offset = (double)128 - *yPoshand;
        initflag = 1;
        }

x_hand = *xPoshand - (double)128 - x_offset;
y_hand = (double)128 - *yPoshand - y_offset;

// check amount of x/y pixel displacement between successive cycles
```

```
 // and abort if too large

 // check if first cycle of pixel data
 if(x_handPrev == 9999 && y_handPrev == 9999)
 {
        x_handPrev = x_hand;
        y_handPrev = y_hand;
 }

 if(fabs(x_hand - x_handPrev) > pixelThreshold)
 {
        errorflag = 5;
        x_hand = x_handPrev;
 }

 if(fabs(y_hand - y_handPrev) > pixelThreshold)
 {
         errorflag = 5;
         y_hand = y_handPrev;
 }

 x_handPrev = x_hand;
 y_handPrev = y_hand;

 // read x & y FIXED data from shared memory

 xPosfix = (int *)xPosfixShm.attach("xPosfix", sizeof(int));
 if(xPosfixShm.isStatusError())
 {
        cerr << "Error opening shared memory \"xPosfix\" : ";
        cerr << strerror(errno) << endl;
        return 0;
 }

 yPosfix = (int *)yPosfixShm.attach("yPosfix", sizeof(int));
 if(yPosfixShm.isStatusError())
 {
        cerr << "Error opening shared memory \"yPosfix\": ";
        cerr << strerror(errno) << endl;
        return 0;
 }

 x_fixed = *xPosfix - (double)128;
 y_fixed = (double)128 - *yPosfix;

// check if first cycle of pixel data
 if(xFixedPrev == 9999 && yFixedPrev == 9999)
  {
          xFixedPrev = x_fixed;
          yFixedPrev = y_fixed;
  }

 xFixedDot = filter.filter((x_fixed - xFixedPrev) / d_controlPeriod);
yFixedDot = filter.filter((y_fixed - yFixedPrev) / d_controlPeriod);

 xFixedPrev = x_fixed;
 yFixedPrev = y_fixed;

 //do rho-squared calculation

 rho_sq = rhoControlGain * (pow(x_fixed, 2) + pow(y_fixed, 2));

 //do rotation matrix calculation
 //first initialize the matrix R

 // q0 - q3 replaced by currentTipQuaternion[0] - [3]
```

```
R[0][0] = 1;
R[0][1] = R[0][2] = R[1][0] = R[2][0] = 0;
R[1][1] = R11 = 1 - (2 * (pow(currentTipQuaternion[1], 2) +
pow(currentTipQuaternion[3], 2)));
R[1][2] = R12 = 2 * ((currentTipQuaternion[2] * currentTipQuaternion[3])
                     - (currentTipQuaternion[0] *currentTipQuaternion[1]));
R[2][1] = R21 = 2 * ((currentTipQuaternion[2] * currentTipQuaternion[3])
                     + (currentTipQuaternion[0] *currentTipQuaternion[1]));
R[2][2] = R22 = 1 - (2 * (pow(currentTipQuaternion[1], 2)
                     + pow(currentTipQuaternion[2], 2)));

// take transpose of matrix R to form RT

RT[0][0] = 1;
RT[0][1] = RT[0][2] = RT[1][0] = RT[2][0] = 0;
RT[1][1] = 1 - (2 * (pow(currentTipQuaternion[1], 2) + pow(currentTipQuaternion[3],
2)));
RT[1][2] = 2 * ((currentTipQuaternion[2] * currentTipQuaternion[3])
                     + (currentTipQuaternion[0] *currentTipQuaternion[1]));
RT[2][1] = 2 * ((currentTipQuaternion[2] * currentTipQuaternion[3])
                     - (currentTipQuaternion[0] *currentTipQuaternion[1]));
RT[2][2] = 1 - (2 * (pow(currentTipQuaternion[1], 2) + pow(currentTipQuaternion[2],
2)));

// finally form rotation matrix rotMatrix = RT * y
// after initializing y

y[0][0] = 0;
y[1][0] = x_hand;
y[2][0] = y_hand;

for(row = 0; row < 3; row++)
{
       for(col = 0; col < 3; col++)
       temp = temp + (RT[row][col] * y[col][0]);
       rotMatrix[row][0] = temp;
       temp = 0;
}

rotMatrix10 = rotMatrix[1][0];
rotMatrix20 = rotMatrix[2][0];

controlPeriod = d_controlPeriod;

// now calculate xDot (also known as U)
xDot[1][0] = - (kp1 + (kn * (xFixedDot * xFixedDot))) * rotMatrix[1][0];
xDot[2][0] =  (kp2 + (kn * (yFixedDot * yFixedDot))) * rotMatrix[2][0];


// numerically integrate xDot to get x-desired and y-desired by
// Trapezoidal Rule

xAreaSum += d_controlPeriod * ((xDotxOld + xDot[1][0]) / 2);
yAreaSum += d_controlPeriod * ((xDotyOld + xDot[2][0]) / 2);

// save old integration endpoints for next iteration
xDotxOld = xDot[1][0];
xDotyOld = xDot[2][0];

x_t = xAreaSum;
y_t = yAreaSum;

// save DESIRED x, y, and z cartesian positions
desiredTipPosition[0] = homeyTipPosition[0];
desiredTipPosition[1] = x_t;
desiredTipPosition[2] = y_t;
```

```
        // calculate desired joint positions from corresponding desired
        // cartesian positions and desired quaternion
        theTitanT2.CalcInverseKinematics(desiredTipPosition, homeyTipQuaternion);

        // get and send desired joint positioning data to ALC
        desiredJointPosition = theTitanT2.GetJointPosition();

        pAlcPkt->rightArmPos[0] = desiredJointPosition[0];
        pAlcPkt->rightArmPos[1] = desiredJointPosition[1];
        pAlcPkt->rightArmPos[2] = desiredJointPosition[2];
        pAlcPkt->rightArmPos[3] = desiredJointPosition[3];
        pAlcPkt->rightArmPos[4] = desiredJointPosition[4];
        pAlcPkt->rightArmPos[5] = desiredJointPosition[5];

        if( send( sock, pAlcPkt, sizeof(alcPkt), 0 ) < 0 )
        {
               perror( "writing to stream socket" );
               return(-1);
        }


        y_d = x_t;
        z_d = y_t;
        x_d = homeyTipPosition[0];

        q0_d = homeyTipQuaternion[0];
        q1_d = homeyTipQuaternion[1];
        q2_d = homeyTipQuaternion[2];
        q3_d = homeyTipQuaternion[3];

         jd_1 = desiredJointPosition[0];
         jd_2 = desiredJointPosition[1];
         jd_3 = desiredJointPosition[2];
         jd_4 = desiredJointPosition[3];
         jd_5 = desiredJointPosition[4];
         jd_6 = desiredJointPosition[5];

        return 0;
}


//============================================================================
// CompVis::stopControl()
//----------------------------------------------------------------------------
// Called each time a control run ends. If running from the GUI, this
// will be called each time the STOP button is pushed, or when a timed run
// ends, or when the control aborts itself.
//============================================================================

int CompVis::stopControl()
{
printf("Stop Control\n");
        pAlcPkt->mode = GUI_ENABLE;

        if( send( sock, pAlcPkt, sizeof(alcPkt), 0 ) < 0 )
        {
                perror( "writing to stream socket" );
                return(-1);
        }

        status = recv( sock, &buf[0], sizeof(buf), 0 );
        if( (status < 0) || (status != sizeof(hlcPACKET)) )
        {
               perror( "recv called" );
               return(-1);
```

```
        }
        return 0;
}

//============================================================================
// CompVis::exitControl
// ----------------------------------------------------------------------------
// This function is called when the control is unloaded. In standalone
// mode, this happens after one control run has completed. When using
// the GUI, it happens when the user loads a new control program or
// exits the GUI.
//============================================================================

int CompVis::exitControl()
{
        return 0;
}


//============================================================================
// main()
//----------------------------------------------------------------------------
// The main function instantiates the object and goes into the mainloop
//============================================================================

main (int argc, char *argv[])
{
        CompVis cp (argc, argv);
        cp.run();
}
```

APPENDIX B. DEVELOPED SOFTWARE FOR HOMOGRAPHY-BASED VISUAL SERVO TRACKING AND REGULATION CONTROL DEMONSTRATIONS

B.1 DESIRED HOMOGRAHPY COMPUTATION

```
///========================================================================
/// File:        pixServer4print.cpp
/// Description: Writes the bright spot pixel information from the in-hand
///              into shared memory locations xPoshand, yPoshand. Also prints
///              pixel information to the screen and a data file.
///              (xyPoints.dat).
///              Error Codes:
///                 -1  FIFO Overflow
///                 -2  Missed frame (not enough computing power)
///                 -3  No pixels brighter than the threshhold
///                 -4  Hardware exception interrupt
///                 -5  Server exited
///========================================================================

#include "RoadRunner.hpp"
#include "DmaBuffer.hpp"
#include "SharedMemory.hpp"
#include "CmdLineArgs.hpp"
#include <sys/types.h>
#include <sys/sched.h>
#include <unistd.h>
#include <sys/stat.h>
#include <iostream.h>
#include <stdio.h>
#include <conio.h>
#include <errno.h>
#include <fcntl.h>
#include <sys/mman.h>
#include <stdlib.h>
#include <signal.h>
#include <math.h>
#include <sys/types.h>
#include <sys/qnx_glob.h>

// Global variables needed by cleanup functions
RoadRunner *rr = 0;
DmaBuffer *frameBuffer0;
DmaBuffer *frameBuffer1;

// Function prototypes
void signalHandler(int);
void cleanUpAndExit(void);

///========================================================================
/// Name:        main
/// Description: The main program function
///========================================================================

int main(int argc, char *argv[])
{
       CmdLineArgs args(argc, argv);
       const char *cameraFile;
       int board;

       // Camera/image parameters
       int xSize = 0;
       int ySize = 0;
       int bytesPerPixel = 0;
       int imageSize;
```

```
// DMA buffer and bright spot stuff
char *buffer[2];
char *currentPixel;
int x;
int y;
int brightX[4];
int brightY[4];
int tap;
int pixel;
int brightValue[4];
int brightThresh;
int bank;
int dot;
int toCompare;

FILE * dataFile;
dataFile = fopen ("xyPoints.dat","w+");

// Get command line parameters
cameraFile = args.getStringOption("camera", "DaCad256E4Raw.cam");
board = args.getIntegerOption("board", 0);
if(args.d_status.isStatusError())
{
       cerr << "server: Error parsing command line" << endl;
       cerr << args.d_status.getMessageText();
       exit(0);
}

// Find the board
rr = new RoadRunner(board);
if(rr->d_status.isStatusError())
{
       cerr << "server: Error locating RoadRunner" << endl;
       cerr << rr->d_status.getMessageText();
       exit(-1);
}

// attach the signal handler to shut down the board when we exit by:
signal(SIGTERM, signalHandler);    // the slay utility
signal(SIGINT, signalHandler);     // or by pressing CNTL-C

// Print out info about the board
rr->printBoardInfo(255);

// Set up the board
rr->initialize(cameraFile);
if(rr->d_status.isStatusError())
{
       cerr << "server: Error initializing board" << endl;
       cerr << rr->d_status.getMessageText();
       cleanUpAndExit();
}

// Get info
xSize = rr->getXSize();
ySize = rr->getYSize();
bytesPerPixel = (int)ceil( (double)rr->getBitsPerPixel() / (double)8);
imageSize = xSize * ySize * bytesPerPixel;

// Allocate the DMA buffers
frameBuffer0 = new DmaBuffer(imageSize);
if(frameBuffer0->d_status.isStatusError())
{
       cerr << "server: Error allocating frame buffer 0" << endl;
```

```
        cerr << frameBuffer0->d_status.getMessageText();
        cleanUpAndExit();
}
buffer[0] = (char *)frameBuffer0->getVirtualAddress();

frameBuffer1 = new DmaBuffer(imageSize);
if(frameBuffer1->d_status.isStatusError())
{
        cerr << "server: Error allocating frame buffer 1" << endl;
        cerr << frameBuffer0->d_status.getMessageText();
        cleanUpAndExit();
}
buffer[1] = (char *)frameBuffer1->getVirtualAddress();

// Set up the DMA descriptors
if(rr->loadQtab(frameBuffer0->getPhysicalAddress(), 0, -1))
{
        cerr << "server: Error loading QTAB bank 0" << endl;
        cerr << rr->d_status.getMessageText();
        cleanUpAndExit();
}
if(rr->loadQtab(frameBuffer1->getPhysicalAddress(), 1, -1))
{
        cerr << "server: Error loading QTAB bank 1" << endl;
        cerr << rr->d_status.getMessageText();
        cleanUpAndExit();
}

// Enable interrupts
rr->enableDmaInterrupt();
rr->enableFifoInterrupt();
rr->enableHwInterrupt();

// Start out by aquiring into buffer 0
rr->setNextBank(0);
bank = 0;

if(rr->startDma())
{
        cerr << "server: Error starting DMA transfer" << endl;
        cerr << rr->d_status.getMessageText();
        cleanUpAndExit();
}

cout << "Processing frames...." << endl;

// Run at high priority
qnx_spawn_options.priority = 28;


// Start acquiring
rr->grab();

// Just keep them frames comin'
for(;;)
{
        // If we are filling buffer 0, we'll next fill buffer 1 (but we'll
        // continue processing buffer 0)
        if(bank)
                rr->setNextBank(0);
        else
                rr->setNextBank(1);

        // Wait for an interrupt
        switch(rr->waitForAnyInterrupt(0))
```

```
{
        case RoadRunner::e_dmaInt:
                // The buffer has been filled.
                // Check to see if we've missed frames
                if(rr->checkForInterrupt(RoadRunner::e_dmaInt) != -1)
                {
                        break;
                }

                // Find the brightest 3 spots that are in a square
                dot = 0;
                toCompare = 0;
                brightX[0] = -9;
                brightY[0] = -9;
                brightX[1] = -9;
                brightY[1] = -9;
                brightX[2] = -9;
                brightY[2] = -9;
                brightX[3] = -9;
                brightY[3] = -9;
                brightValue[0] = -1;
                brightValue[1] = -1;
                brightValue[2] = -1;
                brightValue[3] = -1;
                brightThresh = 23;
                currentPixel = buffer[bank];
                for(y = 0; y < ySize; y++)
               {
        for(x = 0, tap = 0; tap < 4; tap++)
               {
          for(pixel = 0; pixel < 65; pixel++, x++)
          {
               if(*currentPixel >= brightThresh)
               {
               //cout << "Pixel found over threshhold" << endl;

//store the brightest pixel data then compare it with those within a 8x8 area
//find where the current pixel lies in relation to your other found pixels

                toCompare = 0;
                for(int i = 0; i <= 3; i++){
                if(((brightX[i]+8) >= x) && ((brightX[i]-8) <= x) &&
                ((brightY[i]+8) >= y) && ((brightY[i]-8) <= y)){
                dot = i;
                toCompare = 1;
                break;  }}

                if(toCompare){
                if(brightValue[dot] < *currentPixel){
                brightValue[dot] = *currentPixel;
                brightX[dot] = x;
                brightY[dot] = y;

                }
                }

                else{
                if(brightX[0] == -9){
                dot = 0;
                brightValue[dot] = *currentPixel;
                brightX[dot] = x;
                brightY[dot] = y;
                }

                else if(brightX[1] == -9){
```

```
        dot = 1;
        brightValue[dot] = *currentPixel;
        brightX[dot] = x;
        brightY[dot] = y;
        }

        else if(brightX[2] == -9){
        dot = 2;
        brightValue[dot] = *currentPixel;
        brightX[dot] = x;
        brightY[dot] = y;
        }

        else if(brightX[3] == -9){
        dot = 3;
        brightValue[dot] = *currentPixel;
        brightX[dot] = x;
        brightY[dot] = y;
        }
        }
        }
    currentPixel = currentPixel + 4;

  } //x/4
  currentPixel = currentPixel - 259;
} //tap

        currentPixel = currentPixel + 268;
        } //ySize


        fprintf(dataFile, "%12d\t%12d\t%12d\t%12d\t\n", brightX[0],
        brightX[1], brightX[2], brightX[3]);
        fprintf(dataFile, "%12d\t%12d\t%12d\t%12d\t\n", brightY[0],
        brightY[1], brightY[2], brightY[3]);


        break;
        case RoadRunner::e_fifoInt:
        // FIFO overflow.  Indicate error
        cout << "o";

        // Recover
        rr->reset();
        if(rr->d_status.isStatusError())
        {
               cerr << "server: Error resetting after overflow" << endl;
               cerr << rr->d_status.getMessageText();
               rr->shutdown();
               cleanUpAndExit();
        }
        // Aquire into bank 0 first
        rr->setNextBank(0);
        bank = 0;
        // Start acquiring again
        rr->grab();
        // Set up to aquire into bank 1 next.
        break;
        case RoadRunner::e_hwInt:
        // HW exception!
        cout << "h";

        // Recover
        if(rr->reset())
        {
```

```
                                        cerr << "server: Error resetting after hardware
                                        exception!" << endl;
                                        cerr << rr->d_status.getMessageText();
                                        rr->shutdown();
                                        cleanUpAndExit();
                                }
                                // Aquire into bank 0 first
                                rr->setNextBank(0);
                                bank = 0;
                                // Start acquiring again
                                rr->grab();
                                // Set up to aquire into bank 1 next.
                                break;
                                default:
                                cerr << "server: Unknown interrupt or error!" << endl;
                                cleanUpAndExit();
                }
                // OK, that frame is done.  Switch to the other bank.
                bank ^= 1;
        }
}

///========================================================================
/// Name:        signalHandler
/// Description: Catches signals in order to shut down to board before
///              exiting.
///========================================================================


void signalHandler(int)
{
        cleanUpAndExit();
}


///========================================================================
/// Name:        cleanUpAndExit
/// Description: Cleans up and exits
///========================================================================
void cleanUpAndExit(void)
{

        if(rr)
        {
                // Check for error so if we didn't map the registers we didn't get
                // SIGSEG
                if(rr->d_status.isStatusOk())
                        rr->shutdown();
                delete rr;
        }

        if(frameBuffer0)
                delete frameBuffer0;

        if(frameBuffer1)
                delete frameBuffer1;

        exit(0);
}
```

B.2 DESIRED TRAJECTORY GENERATOR

```
////////////////////////////////////////////////////////////////////////////// Compute the
homography by solving linear equations,using svd and faugeras // decomposition algorithm
// reference - numerical recipies in C
// written by Prakash Chawda
//////////////////////////////////////////////////////////////////////////////

#include <stdio.h>
#include <stdlib.h>
#include<math.h>
#define NRANSI
#include "nr.h"
#include "nrutil.h"

#define NP 20
#define MP 20
#define MAXSTR 80

int main(void)
{
       int i,j,k,l,m,n,*indx,number;
       float fu,fv,uo,vo;
       float p,*x,*w,*nn,*n1,*t1,*t,*ttt,*x_h,*n_star,*n_star_h,*cas,*px1,
              *px_s1,*px2,*px_s2,*px3,*px_s3,*px4,*px_s4;
       float *m1,*m2,*m3,*m_s1,*m_s2,*m_s3;
       float **c,**a,**b,**a1,**b1,**b2,**mp,**mp_s,**Gn,**alpha_g33,**H,**u,
              **v,**vt,**ut,**n_s,**alpha,**R1,**R,**tmp,**tmp1,**A,**Ai;
       float alpha1_g33,alpha2_g33,alpha3_g33,alpha4_g33,d_sign;
       float d1,d2,d3,s,det1,det2; double num1,num2; float x1,x3;
       float ww,qq,zz,dd,st,ct,d,theta; double csth,snth;
       char dummy[MAXSTR];

       int pp1,pp2,pp3,pp4,pp5,pp6,pp7,pp8;
       float theta_old = -8;

    FILE * dataFile;
    FILE * outputFile;
    FILE * outputFile0;
    FILE * outputFile11;
    FILE * outputFile12;
    FILE * outputFile13;
    FILE * outputFile14;
    FILE * outputFile15;
    dataFile = fopen("4Points.dat", "r");
    outputFile = fopen("desired4new.dat", "w");
    outputFile0 = fopen("Hmatrix.dat", "w");
    outputFile12 = fopen("n_smatrix.dat", "w");
    outputFile11 = fopen("n_starmatrix.dat", "w");
    outputFile14 = fopen("x_matrix.dat", "w");
    outputFile13 = fopen("Gn_matrix.dat", "w");
    outputFile15 = fopen("alpha_matrix.dat", "w");

    while(!feof(dataFile)){

      indx=ivector(1,NP);
      x=vector(1,NP);
      px1=vector(1,NP);
       px2=vector(1,NP);
       px3=vector(1,NP);
       px4=vector(1,NP);
       px_s1=vector(1,NP);
       px_s2=vector(1,NP);
       px_s3=vector(1,NP);
```

```
px_s4=vector(1,NP);
m1=vector(1,NP);
m2=vector(1,NP);
m3=vector(1,NP);
m_s1=vector(1,NP);
m_s2=vector(1,NP);
m_s3=vector(1,NP);
w=vector(1,NP);
nn=vector(1,NP);
n1=vector(1,NP);
t1=vector(1,NP);
t=vector(1,NP);
ttt=vector(1,NP);
x_h=vector(1,NP);
n_star=vector(1,NP);
n_star_h=vector(1,NP);
cas=vector(1,NP);
mp=matrix(1,NP,1,NP);
mp_s=matrix(1,NP,1,NP);
Ai=matrix(1,NP,1,NP);
A=matrix(1,NP,1,NP);
a=matrix(1,NP,1,NP);
a1=matrix(1,NP,1,NP);
b1=matrix(1,NP,1,NP);
b2=matrix(1,NP,1,NP);
b=matrix(1,NP,1,NP);
c=matrix(1,NP,1,NP);
Gn=matrix(1,NP,1,NP);
H=matrix(1,NP,1,NP);
alpha_g33=matrix(1,NP,1,NP);
alpha=matrix(1,NP,1,NP);
R1=matrix(1,NP,1,NP);
R=matrix(1,NP,1,NP);
n_s=matrix(1,MP,1,NP);
u=matrix(1,MP,1,NP);
v=matrix(1,NP,1,NP);
vt=matrix(1,NP,1,NP);
ut=matrix(1,NP,1,NP);
tmp=matrix(1,NP,1,NP);
tmp1=matrix(1,NP,1,NP);


{
       fu=1235.29456134; fv=1229.7192456; uo=130; vo=130;

       A[1][1]=1;  A[1][2]=0;   A[1][3]=0;
       A[2][1]=vo; A[2][2]=fv;  A[2][3]=0;
       A[3][1]=uo; A[3][2]=0;   A[3][3]=fu;

       /*reference points hard-coded here*/
       px_s1[1]=1;px_s1[2]= 260-69 ;px_s1[3]= 260-84;
       px_s2[1]=1;px_s2[2]= 260-158 ;px_s2[3]= 260-86;
       px_s3[1]=1;px_s3[2]= 260-68 ;px_s3[3]= 260-173;
       px_s4[1]=1;px_s4[2]= 260-157 ;px_s4[3]= 260-174;

       n=3;m=1;

 px1[1] = 1;
 px2[1] = 1;
 px3[1] = 1;
       px4[1] = 1;

 //get pixel data
       fscanf(dataFile,"%f",&px1[2]);
```

```
fscanf(dataFile,"%f",&px2[2]);
fscanf(dataFile,"%f",&px3[2]);
fscanf(dataFile,"%f",&px4[2]);
fscanf(dataFile,"%f",&px1[3]);
fscanf(dataFile,"%f",&px2[3]);
fscanf(dataFile,"%f",&px3[3]);
fscanf(dataFile,"%f",&px4[3]);

pp1=px1[3];pp2=px1[2];pp3=px2[3];pp4=px2[2];pp5=px3[3];pp6=px3[2];
pp7=px4[3];pp8=px4[2];

px1[3] = 260 - pp1;
px1[2] = 260 - pp2;
px2[3] = 260 - pp3;
px2[2] = 260 - pp4;
px3[3] = 260 - pp5;
px3[2] = 260 - pp6;
px4[3] = 260 - pp7;
px4[2] = 260 - pp8;

/// Inverse of the camera matrix ////
Ai[1][1]=1;   Ai[1][2]=0;   Ai[1][3]=0;
Ai[2][1]=-vo/fv; Ai[2][2]=1/fv;  Ai[2][3]=0;
Ai[3][1]=-uo/fu; Ai[3][2]=0;   Ai[3][3]=1/fu;


       for (l=1;l<=n;l++) {
              m_s1[l]=0.0;
              for (j=1;j<=n;j++)
                     m_s1[l] += (Ai[l][j]*px_s1[j]);
       }


a[1][1]= px1[2];a[1][2]= px1[2]*px_s1[2]; a[1][3]= px1[2]*px_s1[3];  a[1][4]= -1;
a[1][5]= -px_s1[2];       a[1][6]= -px_s1[3];       a[1][7]=0;       a[1][8]=0;
a[2][1]= 0;     a[2][2]= 0;               a[2][3]= 0;               a[2][4]= px1[3];
a[2][5]= px1[3]*px_s1[2]; a[2][6]= px1[3]*px_s1[3]; a[2][7]=-px1[2]; a[2][8]= -
px1[2]*px_s1[2];
a[3][1]= px2[2];a[3][2]= px2[2]*px_s2[2]; a[3][3]= px2[2]*px_s2[3]; a[3][4]= -1;
a[3][5]= -px_s2[2];       a[3][6]= -px_s2[3];       a[3][7]=0;       a[3][8]=0;
a[4][1]= 0;     a[4][2]= 0;               a[4][3]= 0;               a[4][4]= px2[3];
a[4][5]= px2[3]*px_s2[2]; a[4][6]= px2[3]*px_s2[3]; a[4][7]=-px2[2]; a[4][8]= -
px2[2]*px_s2[2];
a[5][1]= px3[2];a[5][2]= px3[2]*px_s3[2]; a[5][3]= px3[2]*px_s3[3]; a[5][4]= -1;
a[5][5]= -px_s3[2];       a[5][6]= -px_s3[3];       a[5][7]=0;       a[5][8]=0;
a[6][1]= 0;     a[6][2]= 0;               a[6][3]= 0;               a[6][4]= px3[3];
a[6][5]= px3[3]*px_s3[2]; a[6][6]= px3[3]*px_s3[3]; a[6][7]=-px3[2]; a[6][8]= -
px3[2]*px_s3[2];
a[7][1]= px4[2];a[7][2]= px4[2]*px_s4[2]; a[7][3]= px4[2]*px_s4[3]; a[7][4]= -1;
a[7][5]= -px_s4[2];       a[7][6]= -px_s4[3];       a[7][7]=0;       a[7][8]=0;
a[8][1]= 0;     a[8][2]= 0;               a[8][3]= 0;               a[8][4]= px4[3];
a[8][5]= px4[3]*px_s4[2]; a[8][6]= px4[3]*px_s4[3]; a[8][7]=-px4[2]; a[8][8]= -
px4[2]*px_s4[2];


b[1][1]= 0; b[2][1]= px1[2]*px_s1[3]; b[3][1]= 0; b[4][1]= px2[2]*px_s2[3]; b[5][1]=
0; b[6][1]= px3[2]*px_s3[3]; b[7][1]= 0; b[8][1]= px4[2]*px_s4[3];

       n=8;m=1;
       // Save matrix a for later testing
       for (l=1;l<=n;l++)
              for (k=1;k<=n;k++) c[k][l]=a[k][l];
       // Do LU decomposition
       ludcmp(c,n,indx,&p);
```

```
// Solve equations for the right-hand vector
for (k=1;k<=m;k++) {
       for (l=1;l<=n;l++) x[l]=b[l][k];
       lubksb(c,n,indx,x);
       fprintf(outputFile14, " %12.6f %12.6f %12.6f %12.6f %12.6f %12.6f %12.6f
       %12.6f\n", x[1],x[2],x[3],x[4],x[5],x[6],x[7],x[8]);

       Gn[1][1] = x[1]; Gn[1][2] = x[2]; Gn[1][3] = x[3];
       Gn[2][1] = x[4]; Gn[2][2] = x[5]; Gn[2][3] = x[6];
       Gn[3][1] = x[7]; Gn[3][2] = x[8]; Gn[3][3] = 1;
       fprintf(outputFile13, " %12.6f %12.6f %12.6f\n  %12.6f %12.6f %12.6f\n
       %12.6f %12.6f %12.6f\n\n", Gn[1][1], Gn[1][2], Gn[1][3],Gn[2][1],
       Gn[2][2], Gn[2][3],Gn[3][1], Gn[3][2], Gn[3][3]);


       for (l=1;l<=n;l++) {
              b[l][k]=0.0;
              for (j=1;j<=n;j++)
                     b[l][k] += (a[l][j]*x[j]);
       }

}
alpha1_g33= 1/(Gn[1][1]+Gn[1][2]*px_s1[2]+Gn[1][3]*px_s1[3]);
alpha2_g33= 1/(Gn[1][1]+Gn[1][2]*px_s2[2]+Gn[1][3]*px_s2[3]);
alpha3_g33= 1/(Gn[1][1]+Gn[1][2]*px_s3[2]+Gn[1][3]*px_s3[3]);
alpha4_g33= 1/(Gn[1][1]+Gn[1][2]*px_s4[2]+Gn[1][3]*px_s4[3]);

alpha_g33[1][1]=alpha1_g33;alpha_g33[2][1]=alpha2_g33;
alpha_g33[3][1]=alpha3_g33;alpha_g33[4][1]=alpha4_g33;

if(alpha1_g33<0)
       d_sign=-1;
if(alpha1_g33==0)
       d_sign=0;
if(alpha1_g33>0)
       d_sign=1;

if(d_sign==1)
       if(alpha2_g33<0 || alpha3_g33<0 || alpha4_g33<0)
              printf("first=Stopped at GnConstruction\n");

if(d_sign==-1)
       if(alpha2_g33>0 || alpha3_g33>0 || alpha4_g33>0)
              printf("second=Stopped at GnConstruction\n");

for(i=1;i<=3;i++){
              for(j=1;j<=3;j++){
              tmp1[i][j]=0.0;
                     for(k=1;k<=3;k++){
              tmp1[i][j]= tmp1[i][j]+Ai[i][k]*Gn[k][j];} } }

for(i=1;i<=3;i++){
              for(j=1;j<=3;j++){
                H[i][j]=0.0;
                     for(k=1;k<=3;k++){
              H[i][j]= H[i][j]+tmp1[i][k]*A[k][j];} } }
fprintf(outputFile0, " %12.6f %12.6f %12.6f\n %12.6f %12.6f %12.6f\n %12.6f
%12.6f %12.6f\n\n",
H[1][1],H[1][2],H[1][3],H[2][1],H[2][2],H[2][3],H[3][1],H[3][2],H[3][3] );

/*Calculate svd of H matrix */

m=3;n=3;
```

```
/* copy original matrix into u */
for (k=1;k<=m;k++)
       for (l=1;l<=n;l++) {
                     u[k][l]=H[k][l];
       }
/* perform decomposition */

svdcmp(u,m,n,w,v);

/* sort elements of w-matrixin decreasing order and positive*/
for(i=1;i<=3;i++)
       {
              if(w[i]<0)
                     {
                            w[i] = (-1)*w[i];
                            for(j=1;j<=3;j++)
                                   {
                                          u[j][i] = (-1)*u[j][i];
                                          v[j][i] = (-1)*v[j][i];
                                   }
                     }
       }


if(w[1]<w[2] && w[2]<w[3])
       {
              ttt[1] = ttt[2] = ttt[3] = 0.0;
              ww=w[1]; w[1] = w[3]; w[3] =ww;
              for(i=1;i<=3;i++)
                     {
                            ttt[i]=u[i][1];
                            u[i][1]=u[i][3];
                            u[i][3]=ttt[i];
                     } ttt[1] = ttt[2] = ttt[3] = 0.0;
              for(i=1;i<=3;i++)
                     {
                            ttt[i]=v[i][1];
                            v[i][1]=v[i][3];
                            v[i][3]=ttt[i];
                     }
       }
if(w[1]<w[2] && w[2]>w[3])
       {
              ttt[1] = ttt[2] = ttt[3] = 0.0;
              ww=w[1]; w[1] = w[2]; w[2] =ww;
              for(i=1;i<=3;i++)
                     {
                            ttt[i]=u[i][1];
                            u[i][1]=u[i][2];
                            u[i][2]=ttt[i];
                     } ttt[1] = ttt[2] = ttt[3] = 0.0;
              for(i=1;i<=3;i++)
                     {
                            ttt[i]=v[i][1];
                            v[i][1]=v[i][2];
                            v[i][2]=ttt[i];
                     }
       }
if(w[1]>w[2] && w[2]<w[3])
       {
              ttt[1] = ttt[2] = ttt[3] = 0.0;
              ww=w[2]; w[2] = w[3]; w[3] =ww;
              for(i=1;i<=3;i++)
                     {
```

```
                        ttt[i]=u[i][2];
                        u[i][2]=u[i][3];
                        u[i][3]=ttt[i];
                } ttt[1] = ttt[2] = ttt[3] = 0.0;
        for(i=1;i<=3;i++)
                {
                        ttt[i]=v[i][2];
                        v[i][2]=v[i][3];
                        v[i][3]=ttt[i];
                }
        }

det1=u[1][1]*(u[2][2]*u[3][3]-u[3][2]*u[2][3]) - u[1][2]*(u[2][1]*u[3][3]-
u[3][1]*u[2][3]) + u[1][3]*(u[2][1]*u[3][2]-u[3][1]*u[2][2]);
det2=v[1][1]*(v[2][2]*v[3][3]-v[3][2]*v[2][3]) - v[1][2]*(v[2][1]*v[3][3]-
v[3][1]*v[2][3]) + v[1][3]*(v[2][1]*v[3][2]-v[3][1]*v[2][2]);

d1=w[1];d2=w[2];d3=w[3];


/*Transpose of U obtained above ,to get ut */
for (k=1;k<=n;k++) {
        for (l=1;l<=n;l++)
                ut[l][k]=u[k][l];}

/*Transpose of V obtained above ,to get vt */
for (k=1;k<=n;k++) {
        for (l=1;l<=n;l++)
                vt[l][k]=v[k][l];}

s=det1*det2;

 num1=((d1*d1)-(d2*d2))/((d1*d1)-(d3*d3));
 num2=((d2*d2)-(d3*d3))/((d1*d1)-(d3*d3));
 x1=  sqrt(num1);
 x3=  sqrt(num2);


/* Faugeras Decomposition starts here*/

m=3;n=3;
number=0;       /*number of solutions for decomposition*/
n_star_h[1]=1;n_star_h[2]=0;n_star_h[3]=0;
/* case 1 */
n1[1]=x1; n1[2]=0; n1[3]=x3;
for(j=1;j<=n;j++){
        n_s[j][1]=0.0;
        for(l=1;l<=m;l++)
                n_s[j][1]+=v[j][l]*n1[l];}

if(((n_s[1][1]*1)+(n_s[2][1]*m_s1[2])+(n_s[3][1]*m_s1[3]))>0){
        number=number+1;
cas[number]=1;}

/*  case 2 */
n1[1]=x1; n1[2]=0; n1[3]=(-1)*x3;
for(j=1;j<=n;j++){
        n_s[j][2]=0.0;
        for(l=1;l<=m;l++)
                n_s[j][2]+=v[j][l]*n1[l];}

if(((n_s[1][2]*1)+(n_s[2][2]*m_s1[2])+(n_s[3][2]*m_s1[3]))>0){
        number=number+1;
cas[number]=2;}
```

```
/*  case 3 */
n1[1]=(-1)*x1; n1[2]=0; n1[3]=x3;
for(j=1;j<=n;j++){
       n_s[j][3]=0.0;
       for(l=1;l<=m;l++)
              n_s[j][3]+=v[j][l]*n1[l];}

if(((n_s[1][3]*1)+(n_s[2][3]*m_s1[2])+(n_s[3][3]*m_s1[3]))>0){
       number=number+1;
cas[number]=3;}


/*  case 4 */
n1[1]= (-1)*x1; n1[2]=0; n1[3]=(-1)*x3;
for(j=1;j<=n;j++){
       n_s[j][4]=0.0;
       for(l=1;l<=m;l++)
              n_s[j][4]+=v[j][l]*n1[l];}

if(((n_s[1][4]*1)+(n_s[2][4]*m_s1[2])+(n_s[3][4]*m_s1[3]))>0){
       number=number+1;
cas[number]=4;}


k=cas[1]; l=cas[2];

qq=((n_star_h[1]*n_s[1][k])+(n_star_h[2]*n_s[2][k])+(n_star_h[3]*n_s[3][k]));

zz=((n_star_h[1]*n_s[1][l])+(n_star_h[2]*n_s[2][l])+(n_star_h[3]*n_s[3][l]));

if(qq>zz)
       i=1;
else
       i=2;
m=3;n=3;
if((s*d_sign)>0)
       {
       dd=d2;

       if(cas[i]==1){
              n1[1]=x1; n1[2]=0; n1[3]=x3;
              for(j=1;j<=n;j++){
                     n_star[j]=0.0;
                     for(l=1;l<=m;l++)
                            n_star[j]+=v[j][l]*n1[l];}

              st=(d1-d3)*x1*x3/d2;
              t1[1]=(d1-d3)*x1; t1[2]=(d1-d3)*0; t1[3]=(d1-d3)
              *(-1)*(x3);
       }

       else if(cas[i]==2){
              n1[1]=x1; n1[2]=0; n1[3]=(-1)*x3;
              for(j=1;j<=n;j++){
                     n_star[j]=0.0;
                     for(l=1;l<=m;l++)
                            n_star[j]+=v[j][l]*n1[l];}

              st=(-1)*(d1-d3)*x1*x3/d2;
              t1[1]=(d1-d3)*x1;t1[2]=(d1-d3)*0;t1[3]=(d1-d3)*x3;
       }

       else if(cas[i]==3){
```

```
                n1[1]=(-1)*x1; n1[2]=0; n1[3]=x3;
                for(j=1;j<=n;j++){
                        n_star[j]=0.0;
                        for(l=1;l<=m;l++)
                                n_star[j]+=v[j][l]*n1[l];}

                st=(-1)*(d1-d3)*x1*x3/d2;
                t1[1]=(d1-d3)*(-1)*(x1); t1[2]=(d1-d3)*0;
                t1[3]=(d1-d3)*(-1)*(x3);
        }

        else if(cas[i]==4){
                n1[1]=(-1)*x1; n1[2]=0; n1[3]=(-1)*x3;
                for(j=1;j<=n;j++){
                        n_star[j]=0.0;
                        for(l=1;l<=m;l++)
                                n_star[j]+=v[j][l]*n1[l];}
                st=(d1-d3)*x1*x3/d2;
                t1[1]=(d1-d3)*(-1)*(x1); t1[2]=(d1-d3)*0;
                t1[3]=(d1-d3)*x3;
        }

        ct=((d2*d2)+(d1*d3))/((d1+d3)*(d2));
        R1[1][1]=ct; R1[1][2]=0; R1[1][3]=(-1)*st; R1[2][1]=0; R1[2][2]=1;
        R1[2][3]=0; R1[3][1]=st; R1[3][2]=0; R1[3][3]=ct;

        for(i=1;i<=3;i++){
                for(j=1;j<=3;j++){
                tmp[i][j]=0.0;
                        for(k=1;k<=3;k++){
                tmp[i][j]=     tmp[i][j]+R1[i][k]*vt[k][j];} } }

        for(i=1;i<=3;i++){
                for(j=1;j<=3;j++){
                tmp1[i][j]=0.0;
                        for(k=1;k<=3;k++){
                tmp1[i][j]=    tmp1[i][j]+u[i][k]*tmp[k][j];} } }

        for(i=1;i<=3;i++){
                for(j=1;j<=3;j++){
                R[i][j]=0.0;
                        for(k=1;k<=3;k++){
                R[k][j]= s*tmp1[k][j];} } }


        for(j=1;j<=n;j++){
                        nn[j]=0.0;
                        for(l=1;l<=m;l++)
                                nn[j]+=R[j][l]*n_star[l];}
}

else if(s*d_sign<0)
        {
          dd=-d2;
          m=3;n=3;
          if(cas[i]==1){
                n1[1]=x1; n1[2]=0; n1[3]=x3;
                for(j=1;j<=n;j++){
                        n_star[j]=0.0;
                        for(l=1;l<=m;l++)
                                n_star[j]+=v[j][l]*n1[l];}

                st=(d1+d3)*x1*x3/d2;
                t1[1]=(d1+d3)*x1; t1[2]=(d1+d3)*0; t1[3]=(d1+d3)*x3;
```

```
}
else if(cas[i]==2){
       n1[1]=x1; n1[2]=0; n1[3]=(-1)*x3;
       for(j=1;j<=n;j++){
              n_star[j]=0.0;
              for(l=1;l<=m;l++)
                     n_star[j]+=v[j][l]*n1[l];}

       st=(-1)*(d1+d3)*x1*x3/d2;
       t1[1]=(d1+d3)*x1;t1[2]=(d1+d3)*0;
       t1[3]=(d1+d3)*(-1)*(x3);
}
else if(cas[i]==3){
       n1[1]=(1)*-x1; n1[2]=0; n1[3]=x3;
       for(j=1;j<=n;j++){
              n_star[j]=0.0;
              for(l=1;l<=m;l++)
                     n_star[j]+=v[j][l]*n1[l];}

       st=(-1)*(d1+d3)*x1*x3/d2;
       t1[1]=(d1+d3)*(-1)*(x1); t1[2]=(d1+d3)*0; t1[3]=(d1+d3)*(x3);
}

else if(cas[i]==4){
       n1[1]=(-1)*x1; n1[2]=0; n1[3]=(-1)*x3;
       for(j=1;j<=n;j++){
              n_star[j]=0.0;
              for(l=1;l<=m;l++)
                     n_star[j]+=v[j][l]*n1[l];}

       st=(d1+d3)*x1*x3/d2;
       t1[1]=(d1+d3)*(-1)*(x1); t1[2]=(d1+d3)*0; t1[3]=(d1+d3)*(-
       1)*(x3);
}

ct=((d1*d3)-(d2*d2))/((d1-d3)*(d2));
R1[1][1]=ct; R1[1][2]=0; R1[1][3]=st;
R1[2][1]=0; R1[2][2]=-1; R1[2][3]=0; R1[3][1]=st; R1[3][2]=0;
R1[3][3]=(-1)*ct;

for(i=1;i<=3;i++){
       for(j=1;j<=3;j++){
       tmp[i][j]=0.0;
              for(k=1;k<=3;k++){
       tmp[i][j]=    tmp[i][j]+R1[i][k]*vt[k][j];} } }

for(i=1;i<=3;i++){
       for(j=1;j<=3;j++){
       tmp1[i][j]=0.0;
              for(k=1;k<=3;k++){
       tmp1[i][j]= tmp1[i][j]+u[i][k]*tmp[k][j];} } }


for(i=1;i<=3;i++){
       for(j=1;j<=3;j++){
       R[i][j]=0.0;
              for(k=1;k<=3;k++){
       R[k][j]= s*tmp1[k][j];} } }


for(j=1;j<=n;j++){
              nn[j]=0.0;
              for(l=1;l<=m;l++)
                     nn[j]+=R[j][l]*n_star[l];}
```

```
        }

                for(j=1;j<=n;j++){
                                t[j]=0.0;
                                for(l=1;l<=m;l++)
                                        t[j]+=u[j][l]*t1[l];}

                d=s*dd;

                x_h[1]=t[1]/d;  x_h[2]=t[2]/d;  x_h[3]=t[3]/d;

                alpha[1][1] = alpha_g33[1][1]*d; alpha[2][1] = alpha_g33[2][1]*d;
                alpha[3][1] = alpha_g33[3][1]*d;

                fprintf(outputFile15, " %12.6f %12.6f %12.6f\n",
                alpha[1][1],alpha[2][1],alpha[3][1]);

                csth = R[1][1];
                snth = R[2][1];

                theta = acos(((R[1][1]+R[2][2]+R[3][3])-1)/2);
                //theta = acos(csth);
                if(snth < 0)
                        theta = (-1)*theta;

                if(theta_old == -8)
                   theta_old = theta;

     if((theta_old < 1 && theta > 0) || (theta_old > 1 && theta < 0)){
        theta = (-1)*theta;
     }

     theta_old = theta;

         fprintf(outputFile11, " %12.6f %12.6f %12.6f\n", n_star[1], n_star[2],
         n_star[3]);
         fprintf(outputFile12, " %12.6f %12.6f %12.6f %12.6f\n  %12.6f %12.6f %12.6f
         %12.6f\n %12.6f %12.6f %12.6f %12.6f\n %12.6f %12.6f %12.6f %12.6f\n\n\n",
         n_s[1][1], n_s[1][2], n_s[1][3], n_s[1][4],n_s[2][1], n_s[2][2], n_s[2][3],
         n_s[2][4],n_s[3][1], n_s[3][2], n_s[3][3], n_s[3][4],n_s[4][1], n_s[4][2],
         n_s[4][3], n_s[4][4]);

 fprintf(outputFile, " %12.6f %12.6f %12.6f\n", x_h[1], x_h[2], theta);
}
free_matrix(R,1,NP,1,NP);
free_matrix(R1,1,NP,1,NP);
free_matrix(tmp,1,NP,1,NP);
free_matrix(tmp1,1,NP,1,NP);
free_matrix(ut,1,NP,1,NP);
free_matrix(vt,1,NP,1,NP);
free_matrix(v,1,NP,1,NP);
free_matrix(u,1,NP,1,NP);
free_matrix(n_s,1,NP,1,NP);
free_matrix(alpha,1,NP,1,NP);
free_matrix(alpha_g33,1,NP,1,NP);
free_matrix(Gn,1,NP,1,NP);
free_matrix(H,1,NP,1,NP);
free_matrix(a1,1,NP,1,NP);
free_matrix(b1,1,NP,1,NP);
free_matrix(b2,1,NP,1,NP);
free_matrix(mp,1,NP,1,NP);
free_matrix(mp_s,1,NP,1,NP);
free_matrix(c,1,NP,1,NP);
free_matrix(b,1,NP,1,NP);
```

```
        free_matrix(a,1,NP,1,NP);
        free_matrix(Ai,1,NP,1,NP);
        free_matrix(A,1,NP,1,NP);
        free_vector(px1,1,NP);
        free_vector(px_s1,1,NP);
        free_vector(px2,1,NP);
        free_vector(px_s2,1,NP);
        free_vector(px3,1,NP);
        free_vector(px_s3,1,NP);
        free_vector(px4,1,NP);
        free_vector(px_s4,1,NP);
        free_vector(m1,1,NP);
        free_vector(m_s1,1,NP);
        free_vector(m2,1,NP);
        free_vector(m_s2,1,NP);
        free_vector(m3,1,NP);
        free_vector(m_s3,1,NP);
        free_vector(x,1,NP);
        free_vector(w,1,NP);
        free_vector(nn,1,NP);
        free_vector(n1,1,NP);
        free_vector(t1,1,NP);
        free_vector(t,1,NP);
        free_vector(ttt,1,NP);
        free_vector(x_h,1,NP);
        free_vector(n_star,1,NP);
        free_vector(n_star_h,1,NP);
        free_vector(cas,1,NP);
        free_ivector(indx,1,NP);

    }

        fclose(dataFile);
        fclose(outputFile);
        fclose(outputFile0);
        fclose(outputFile11);
        fclose(outputFile12);
        fclose(outputFile13);
        fclose(outputFile14);
        fclose(outputFile15);
        return 0;
}
#undef NRANSI
```

B.3 WMR TRACKING SERVER PROGRAM

```
///====================================================================
/// File       : final.cpp
/// Description: identifies the bright spot pixel information from the
///              camera and computes the homography by solving
///              linear equations, using svd and the Faugeras
///                   decomposition algorithm, and writes some parameters
///                   into shared memory locations.
///              Error Codes:
///                 -1  FIFO Overflow
///                 -2  Missed frame (not enough computing power)
///                 -3  No pixels brighter than the threshold
///                 -4  Hardware exception interrupt
///                 -5  Server exited
///====================================================================

#include "RoadRunner.hpp"
#include "DmaBuffer.hpp"
```

```
#include "SharedMemory.hpp"
#include "CmdLineArgs.hpp"
#include "CSocket.hpp"

extern "C" {
#include <sys/types.h>
#include <sys/sched.h>
#include <unistd.h>
#include <sys/stat.h>
#include <stdio.h>
#include <conio.h>
#include <errno.h>
#include <fcntl.h>
#include <sys/mman.h>
#include <stdlib.h>
#include <signal.h>
#include <math.h>
#include <sys/types.h>
#include <sys/qnx_glob.h>
#include <nr.h>
#include <nrutil.h>
}
#define NRANSI
#define NP 20
#define MP 20
#define MAXSTR 80
#define PI 3.141592653589

// Global variables needed by cleanup functions
RoadRunner *rr = 0;
DmaBuffer *frameBuffer0;
DmaBuffer *frameBuffer1;
float * dataPacket;
Socket clientSocket;

// Function prototypes
void signalHandler(int);
void cleanUpAndExit(void);

///=========================================================================
/// Name:        main
/// Description: The main program function
///=========================================================================

int main(int argc, char *argv[])
{

        CmdLineArgs args(argc, argv);
        const char *cameraFile;
        int board;
        const char *socketServerName;
        int socketPortNumber;

        // Camera/image parameters
        int xSize = 0;
        int ySize = 0;
        int bytesPerPixel = 0;
        int imageSize;

        // DMA buffer and bright spot stuff
        char *buffer[2];
        char *currentPixel;
        int xx;
        int y;
        int brightX[3];
```

```
int brightY[3];
int brightXOld[3];
int brightYOld[3];
int tap;
int pixel;
int brightValue[3];
int brightThresh;
int bank;
int dot;
int toCompare;

brightXOld[0] = 101;
brightYOld[0] = 97;
brightXOld[1] = 140;
brightYOld[1] = 98;
brightXOld[2] = 141;
brightYOld[2] = 137;

//data packet allocation
int variablesToSend = 3;
dataPacket = (float*) calloc(variablesToSend,sizeof(float));

// homography stuff
int i,j,k,l,m,n,*indx,number;
float fu,fv,uo,vo;
float p,*x,*w,*nn,*n1,*t1,*t,*ttt,*x_h,*n_star,*n_star_h,*cas;
float  *px1,*px_s1,*px2,*px_s2,*px3,*px_s3;
float *m1,*m2,*m3,*m_s1,*m_s2,*m_s3;
float **c,**a,**b,**a1,**b1,**b2,**mp,**mp_s,**H1,**H2,**H;
float **u,**v,**vt,**ut,**n_s,**alpha,**R1,**R,**tmp,**tmp1,**Ai;
float alpha1,alpha2,alpha3,d_sign;
float d1,d2,d3,s,det1,det2; double num1,num2; float x1,x3;
float ww,qq,zz,dd,st,ct,d,theta; double csth,snth;


// Get command line parameters
cameraFile = args.getStringOption("camera", "DaCad256E4Raw.cam");
board = args.getIntegerOption("board", 0);
socketServerName = args.getStringOption("server","160.91.84.81");
socketServerName =args.getStringOption("server","160.91.84.111");
socketPortNumber = args.getIntegerOption("port", 10000);
if(args.d_status.isStatusError())
{
       cerr << "server: Error parsing command line" << endl;
       cerr << args.d_status.getMessageText();
       exit(0);
}

// Find the board
rr = new RoadRunner(board);
if(rr->d_status.isStatusError())
{
       cerr << "server: Error locating RoadRunner" << endl;
       cerr << rr->d_status.getMessageText();
       exit(-1);
}


// attach the signal handler to shut down the board when we exit
signal(SIGTERM, signalHandler);    // the slay utility
signal(SIGINT, signalHandler);     // or by pressing CNTL-C

// Print out info about the board
rr->printBoardInfo(255);
```

```
// Set up the board
rr->initialize(cameraFile);
if(rr->d_status.isStatusError())
{
       cerr << "server: Error initializing board" << endl;
       cerr << rr->d_status.getMessageText();
       cleanUpAndExit();
}

// Get info
xSize = rr->getXSize();
ySize = rr->getYSize();
bytesPerPixel=(int)ceil((double)rr->getBitsPerPixel()/(double)8);
imageSize = xSize * ySize * bytesPerPixel;

// Allocate the DMA buffers
frameBuffer0 = new DmaBuffer(imageSize);
if(frameBuffer0->d_status.isStatusError())
{
       cerr << "server: Error allocating frame buffer 0" << endl;
       cerr << frameBuffer0->d_status.getMessageText();
       cleanUpAndExit();
}

buffer[0] = (char *)frameBuffer0->getVirtualAddress();
frameBuffer1 = new DmaBuffer(imageSize);
if(frameBuffer1->d_status.isStatusError())
{
       cerr << "server: Error allocating frame buffer 1" << endl;
       cerr << frameBuffer0->d_status.getMessageText();
       cleanUpAndExit();
}
buffer[1] = (char *)frameBuffer1->getVirtualAddress();

// Set up the DMA descriptors
if(rr->loadQtab(frameBuffer0->getPhysicalAddress(), 0, -1))
{
       cerr << "server: Error loading QTAB bank 0" << endl;
       cerr << rr->d_status.getMessageText();
       cleanUpAndExit();
}
if(rr->loadQtab(frameBuffer1->getPhysicalAddress(), 1, -1))
{
       cerr << "server: Error loading QTAB bank 1" << endl;
       cerr << rr->d_status.getMessageText();
       cleanUpAndExit();
}

// Set up the client socket
cout << "Trying to connect to " << socketServerName << ", port ";
cout << socketPortNumber << "...";
clientSocket.initClientSocket();
if(clientSocket.connect((char *)socketServerName, socketPortNumber) == -1)
{
       cout << "Connection failed" << endl;
       cleanUpAndExit();
}
else
{
       cout << "Connected" << endl;
}

// Use this for a more deterministic response
clientSocket.disableNagle();
```

```
        // Enable interrupts
        rr->enableDmaInterrupt();
        rr->enableFifoInterrupt();
        rr->enableHwInterrupt();

        // Start out by aquiring into buffer 0
        rr->setNextBank(0);
        bank = 0;

        if(rr->startDma())

        {
                cerr << "server: Error starting DMA transfer" << endl;
                cerr << rr->d_status.getMessageText();
                cleanUpAndExit();
        }

        cout << "Processing frames and transmitting data...." << endl;

        // Run at high priority
        qnx_spawn_options.priority = 28;

        // Start acquiring
        rr->grab();

        // Just keep the frames comin'
        for(;;)
        {
        // If we are filling buffer 0, we'll next fill buffer 1 (but //we'll continue
        processing buffer 0)
                if(bank)
                        rr->setNextBank(0);
                else
                        rr->setNextBank(1);

                // Wait for an interrupt
                switch(rr->waitForAnyInterrupt(0))
                {
                case RoadRunner::e_dmaInt:
                         //The buffer has been filled.
                         //Check to see if we've missed frames
                        if(rr->checkForInterrupt(RoadRunner::e_dmaInt) != -1)
                        {
                                dataPacket[0] = -2;
                                dataPacket[1] = -2;
                                dataPacket[2] = -2;
                                clientSocket.send(dataPacket, 3 * sizeof(float));
                                break;
                        }
                // Find the brightest 3 spots that are in a square grouping.
                                dot = 0;
                                toCompare = 0;
                                brightX[0] = -9;
                                brightY[0] = -9;
                                brightX[1] = -9;
                                brightY[1] = -9;
                                brightX[2] = -9;
                                brightY[2] = -9;
                                brightValue[0] = -1;
                                brightValue[1] = -1;
                                brightValue[2] = -1;
                                brightThresh = 32;
                                currentPixel = buffer[bank];

for(y = 0; y < ySize; y++)
```

```
{
        for(xx = 0, tap = 0; tap < 4; tap++)
        {
                for(pixel = 0; pixel < 65; pixel++, xx++)
                {
                if(*currentPixel >= brightThresh)
                        {
//store the brightest pixel data then compare it with those within a
//8x8 area find where the current pixel lies in relation to your other
//found pixels
                toCompare = 0;
for(int i = 0; i <= 2; i++){
if(((brightX[i]+8) >= xx) && ((brightX[i]-8) <= xx) && ((brightY[i]+8) >= y) && ((brightY[i]-
8) <= y)){
                dot = i;
                toCompare = 1;
                break;
        }
}
if(toCompare){
        if(brightValue[dot] < *currentPixel){
                brightValue[dot] = *currentPixel;
                brightX[dot] = xx;
                brightY[dot] = y;
        }
}

else{
        if(brightX[0] == -9){
                dot = 0;
                brightValue[dot] = *currentPixel;
                brightX[dot] = xx;
                brightY[dot] = y;
        }

else if(brightX[1] == -9){
                dot = 1;
                brightValue[dot] = *currentPixel;
                brightX[dot] = xx;
                brightY[dot] = y;
        }

else if(brightX[2] == -9){
                dot = 2;
                brightValue[dot] = *currentPixel;
                brightX[dot] = xx;
                brightY[dot] = y;
                }
           }
        }
            currentPixel = currentPixel + 4;
    }
            currentPixel = currentPixel - 259;
   }
            currentPixel = currentPixel + 268;
  }
// Declaring variables used
     indx=ivector(1,NP);
       x=vector(1,NP);
       px1=vector(1,NP);
       px2=vector(1,NP);
       px3=vector(1,NP);
       px_s1=vector(1,NP);
       px_s2=vector(1,NP);
       px_s3=vector(1,NP);
```

```
        m1=vector(1,NP);
        m2=vector(1,NP);
        m3=vector(1,NP);
        m_s1=vector(1,NP);
        m_s2=vector(1,NP);
        m_s3=vector(1,NP);
        w=vector(1,NP);
        nn=vector(1,NP);
        n1=vector(1,NP);
        t1=vector(1,NP);
        t=vector(1,NP);
        ttt=vector(1,NP);
        x_h=vector(1,NP);
        n_star=vector(1,NP);
        n_star_h=vector(1,NP);
        cas=vector(1,NP);
        mp=matrix(1,NP,1,NP);
        mp_s=matrix(1,NP,1,NP);
        Ai=matrix(1,NP,1,NP);
        a=matrix(1,NP,1,NP);
        a1=matrix(1,NP,1,NP);
        b1=matrix(1,NP,1,NP);
        b2=matrix(1,NP,1,NP);
        b=matrix(1,NP,1,NP);
        c=matrix(1,NP,1,NP);
        H1=matrix(1,NP,1,NP);
        H2=matrix(1,NP,1,NP);
        H=matrix(1,NP,1,NP);
        alpha=matrix(1,NP,1,NP);
        R1=matrix(1,NP,1,NP);
        R=matrix(1,NP,1,NP);
        n_s=matrix(1,MP,1,NP);
        u=matrix(1,MP,1,NP);
        v=matrix(1,NP,1,NP);
        vt=matrix(1,NP,1,NP);
        ut=matrix(1,NP,1,NP);
        tmp=matrix(1,NP,1,NP);
        tmp1=matrix(1,NP,1,NP);

//check to see if any points were missed
if(brightX[0] == -9 || brightX[1] == -9 || brightX[2] == -9 || brightY[0] == -9 || brightY[1]
== -9 || brightY[2] == -9){

        px1[2] = 260 - brightXOld[0];
        px1[3] = 260 - brightYOld[0];
        px2[2] = 260 - brightXOld[1];
        px2[3] = 260 - brightYOld[1];
        px3[2] = 260 - brightXOld[2];
        px3[3] = 260 - brightYOld[2];

}

else{

        px1[2] = 260 - brightX[0];
        px1[3] = 260 - brightY[0];
        px2[2] = 260 - brightX[1];
        px2[3] = 260 - brightY[1];
        px3[2] = 260 - brightX[2];
        px3[3] = 260 - brightY[2];

        brightXOld[0] = brightX[0];
        brightXOld[1] = brightX[1];
        brightXOld[2] = brightX[2];
        brightYOld[0] = brightY[0];
```

```
        brightYOld[1] = brightY[1];
        brightYOld[2] = brightY[2];

}
//Homography stuff starts here

fu=1235.29456134; fv=1229.7192456; uo=130; vo=130;
Ai[1][1]=1;   Ai[1][2]=0;   Ai[1][3]=0;
Ai[2][1]=vo; Ai[2][2]=fv;  Ai[2][3]=0;
Ai[3][1]=uo; Ai[3][2]=0;   Ai[3][3]=fu;

/*reference points hard-coded here*/
px_s1[1]=1;px_s1[2]= 260 - 96   ;px_s1[3]=260 - 79;
px_s2[1]=1;px_s2[2]= 260 - 187 ;px_s2[3]= 260 - 81;
px_s3[1]=1;px_s3[2]= 260 - 186 ;px_s3[3]= 260 - 173;

px1[1] = 1;
px2[1] = 1;
px3[1] = 1;

n=3;m=1;
for (l=1;l<=n;l++)
        for (k=1;k<=n;k++) a[k][l]=Ai[k][l];

for (l=1;l<=n;l++)
        for (k=1;k<=m;k++) b[l][k]=px1[l];

n=3;m=1;
/* Save matrix a for later testing */
for (l=1;l<=n;l++)
        for (k=1;k<=n;k++) c[k][l]=a[k][l];
                /* Do LU decomposition */
                ludcmp(c,n,indx,&p);

/* Solve equations for the right-hand vector */
for (k=1;k<=m;k++) {
        for (l=1;l<=n;l++) x[l]=b[l][k];
                lubksb(c,n,indx,x);
                for(j=1;j<=n;j++)
                m1[j]=-x[j];
        for (l=1;l<=n;l++) {
                b[l][k]=0.0;
                for (j=1;j<=n;j++)
                        b[l][k] += (a[l][j]*x[j]);
                        }
                }

n=3;m=1;
for (l=1;l<=n;l++)
        for (k=1;k<=n;k++) a[k][l]=Ai[k][l];

for (l=1;l<=n;l++)
        for (k=1;k<=m;k++) b[l][k]=px2[l];

n=3;m=1;
/* Save matrix a for later testing */
for (l=1;l<=n;l++)
        for (k=1;k<=n;k++) c[k][l]=a[k][l];
                /* Do LU decomposition */
                ludcmp(c,n,indx,&p);

/* Solve equations for the right-hand vector */
for (k=1;k<=m;k++) {
        for (l=1;l<=n;l++) x[l]=b[l][k];
                lubksb(c,n,indx,x);
```

```
                for(j=1;j<=n;j++)
                m2[j]=-x[j];
        for (l=1;l<=n;l++) {
                b[l][k]=0.0;
                for (j=1;j<=n;j++)
                        b[l][k] += (a[l][j]*x[j]);
                        }
                }
n=3;m=1;
for (l=1;l<=n;l++)
        for (k=1;k<=n;k++) a[k][l]=Ai[k][l];

for (l=1;l<=n;l++)
        for (k=1;k<=m;k++) b[l][k]=px3[l];

n=3;m=1;
/* Save matrix a for later testing */
for (l=1;l<=n;l++)
        for (k=1;k<=n;k++) c[k][l]=a[k][l];

/* Do LU decomposition */
                ludcmp(c,n,indx,&p);

/* Solve equations for the right-hand vector */
for (k=1;k<=m;k++) {
        for (l=1;l<=n;l++) x[l]=b[l][k];
                lubksb(c,n,indx,x);
                for(j=1;j<=n;j++)
                m3[j]=-x[j];
        for (l=1;l<=n;l++) {
                b[l][k]=0.0;
                for (j=1;j<=n;j++)
                        b[l][k] += (a[l][j]*x[j]);
                        }
                }
n=3;m=1;
for (l=1;l<=n;l++)
        for (k=1;k<=n;k++) a[k][l]=Ai[k][l];

for (l=1;l<=n;l++)
        for (k=1;k<=m;k++) b[l][k]=px_s1[l];

n=3;m=1;
/* Save matrix a for later testing */
for (l=1;l<=n;l++)
        for (k=1;k<=n;k++) c[k][l]=a[k][l];
                /* Do LU decomposition */
                ludcmp(c,n,indx,&p);

/* Solve equations for the right-hand vector */
for (k=1;k<=m;k++) {
        for (l=1;l<=n;l++) x[l]=b[l][k];
                lubksb(c,n,indx,x);
                for(j=1;j<=n;j++)
                m_s1[j]=-x[j];
        for (l=1;l<=n;l++) {
                b[l][k]=0.0;
                for (j=1;j<=n;j++)
                        b[l][k] += (a[l][j]*x[j]);
                        }
                }
n=3;m=1;
for (l=1;l<=n;l++)
        for (k=1;k<=n;k++) a[k][l]=Ai[k][l];
```

```
for (l=1;l<=n;l++)
       for (k=1;k<=m;k++) b[l][k]=px_s2[l];

n=3;m=1;
/* Save matrix a for later testing */
for (l=1;l<=n;l++)
       for (k=1;k<=n;k++) c[k][l]=a[k][l];
              /* Do LU decomposition */
              ludcmp(c,n,indx,&p);

/* Solve equations for the right-hand vector */
for (k=1;k<=m;k++) {
       for (l=1;l<=n;l++) x[l]=b[l][k];
              lubksb(c,n,indx,x);
              for(j=1;j<=n;j++)
              m_s2[j]=-x[j];
              for (l=1;l<=n;l++) {
                     b[l][k]=0.0;
              for (j=1;j<=n;j++)
                     b[l][k] += (a[l][j]*x[j]);
                     }
              }
n=3;m=1;
for (l=1;l<=n;l++)
       for (k=1;k<=n;k++) a[k][l]=Ai[k][l];

for (l=1;l<=n;l++)
       for (k=1;k<=m;k++) b[l][k]=px_s3[l];
n=3;m=1;
/* Save matrix a for later testing */
for (l=1;l<=n;l++)
       for (k=1;k<=n;k++) c[k][l]=a[k][l];
              /* Do LU decomposition */
              ludcmp(c,n,indx,&p);

/* Solve equations for the right-hand vector */
for (k=1;k<=m;k++) {
       for (l=1;l<=n;l++) x[l]=b[l][k];
              lubksb(c,n,indx,x);
              for(j=1;j<=n;j++)
              m_s3[j]=-x[j];
              for (l=1;l<=n;l++) {
                     b[l][k]=0.0;
              for (j=1;j<=n;j++)
                     b[l][k] += (a[l][j]*x[j]);
                     }
              }

mp[1][1]=m1[2];       mp[2][1]=m1[3];       mp[3][1]=m2[2];
mp_s[1][1]=m_s1[2];   mp_s[2][1]=m_s1[3];   mp_s[3][1]=m_s2[2];
mp[4][1]=m2[3];       mp[5][1]=m3[2];       mp[6][1]=m3[3];
mp_s[4][1]=m_s2[3];   mp_s[5][1]=m_s3[2];   mp_s[6][1]=m_s3[3];

b1[1][1]=mp_s[2][1]; b1[2][1]=mp_s[4][1]; b1[3][1]=mp_s[6][1];

a[1][1]=mp[2][1];  a[1][2]=mp[2][1]*mp_s[1][1];  a[1][3]=mp[2][1]*mp_s[2][1];
a[2][1]=mp[4][1];  a[2][2]=mp[4][1]*mp_s[3][1];  a[2][3]=mp[4][1]*mp_s[4][1];
a[3][1]=mp[6][1];  a[3][2]=mp[6][1]*mp_s[5][1];  a[3][3]=mp[6][1]*mp_s[6][1];

n=3;m=1;
for (l=1;l<=n;l++)
       for (k=1;k<=m;k++) b[l][k]=b1[l][k];
/* Save matrix a for later testing */
for (l=1;l<=n;l++)
       for (k=1;k<=n;k++) c[k][l]=a[k][l];
```

```
                /* Do LU decomposition */
                ludcmp(c,n,indx,&p);

/* Solve equations for the right-hand vector */
for (k=1;k<=m;k++) {
        for (l=1;l<=n;l++) x[l]=b[l][k];
                lubksb(c,n,indx,x);
                for(j=1;j<=n;j++)
                H1[j][1]=x[j];
                for (l=1;l<=n;l++) {
                        b[l][k]=0.0;
                for (j=1;j<=n;j++)
                        b[l][k] += (a[l][j]*x[j]);
                        }
                }

a[1][1]=mp[2][1];  a[1][2]=mp[2][1]*mp_s[1][1];  a[1][3]=mp[2][1]*mp_s[2][1];
a[2][1]=mp[4][1];  a[2][2]=mp[4][1]*mp_s[3][1];  a[2][3]=mp[4][1]*mp_s[4][1];
a[3][1]=mp[6][1];  a[3][2]=mp[6][1]*mp_s[5][1];  a[3][3]=mp[6][1]*mp_s[6][1];

b2[1][1]=mp_s[2][1]*mp[1][1];
b2[2][1]=mp_s[4][1]*mp[3][1];
b2[3][1]=mp_s[6][1]*mp[5][1];

for (l=1;l<=n;l++)
        for (k=1;k<=m;k++) b[l][k]=b2[l][k];

n=3;m=1;
/* Save matrix a for later testing */
for (l=1;l<=n;l++)
        for (k=1;k<=n;k++) c[k][l]=a[k][l];
                /* Do LU decomposition */
                ludcmp(c,n,indx,&p);

/* Solve equations for the right-hand vector */
for (k=1;k<=m;k++) {
        for (l=1;l<=n;l++) x[l]=b[l][k];
                lubksb(c,n,indx,x);
                for(j=1;j<=n;j++)
                H2[j][1]=x[j];
                for (l=1;l<=n;l++) {
                        b[l][k]=0.0;
                for (j=1;j<=n;j++)
                        b[l][k] += (a[l][j]*x[j]);
                        }
                }

H[1][1]=H1[1][1];  H[1][2]=H1[2][1];  H[1][3]=H1[3][1];
H[2][1]=H2[1][1];  H[2][2]=H2[2][1];  H[2][3]=H2[3][1];
H[3][1]=0;         H[3][2]=0;         H[3][3]=1;

alpha1=mp[2][1]/mp_s[2][1];
alpha2=mp[4][1]/mp_s[4][1];
alpha3=mp[6][1]/mp_s[6][1];
alpha[1][1]=alpha1;alpha[2][1]=alpha2;alpha[3][1]=alpha3;

if(alpha1<0)
        d_sign=-1;
if(alpha1==0)
        d_sign=0;
if(alpha1>0)
        d_sign=1;

/*Calculate svd of H matrix */
m=3;n=3;
```

```
/* copy original matrix into u */
for (k=1;k<=m;k++)
	for (l=1;l<=n;l++) {
		u[k][l]=H[k][l];
	}
/* perform decomposition */
svdcmp(u,m,n,w,v);

/* sort elements of w-matrixin decreasing order and positive*/
for(i=1;i<=3;i++)
{
	if(w[i]<0)
		{
		w[i] = (-1)*w[i];
		for(j=1;j<=3;j++)
			{
				u[j][i] = (-1)*u[j][i];
				v[j][i] = (-1)*v[j][i];
			}
		}
}
if(w[1]<w[2] && w[2]<w[3])
	{
		ttt[1] = ttt[2] = ttt[3] = 0.0;
		ww=w[1]; w[1] = w[3]; w[3] =ww;
		for(i=1;i<=3;i++)
			{
				ttt[i]=u[i][1];
				u[i][1]=u[i][3];
				u[i][3]=ttt[i];
			} ttt[1] = ttt[2] = ttt[3] = 0.0;
		for(i=1;i<=3;i++)
			{
				ttt[i]=v[i][1];
				v[i][1]=v[i][3];
				v[i][3]=ttt[i];
			}
	}
if(w[1]<w[2] && w[2]>w[3])
	{
		ttt[1] = ttt[2] = ttt[3] = 0.0;
		ww=w[1]; w[1] = w[2]; w[2] =ww;
		for(i=1;i<=3;i++)
			{
				ttt[i]=u[i][1];
				u[i][1]=u[i][2];
				u[i][2]=ttt[i];
			} ttt[1] = ttt[2] = ttt[3] = 0.0;
		for(i=1;i<=3;i++)
			{
				ttt[i]=v[i][1];
				v[i][1]=v[i][2];
				v[i][2]=ttt[i];
			}
	}
if(w[1]>w[2] && w[2]<w[3])
	{
		ttt[1] = ttt[2] = ttt[3] = 0.0;
		ww=w[2]; w[2] = w[3]; w[3] =ww;
		for(i=1;i<=3;i++)
			{
				ttt[i]=u[i][2];
				u[i][2]=u[i][3];
				u[i][3]=ttt[i];
```

```
                        } ttt[1] = ttt[2] = ttt[3] = 0.0;
                for(i=1;i<=3;i++)
                        {
                                ttt[i]=v[i][2];
                                v[i][2]=v[i][3];
                                v[i][3]=ttt[i];
                        }
        }
det1=   u[1][1]*(u[2][2]*u[3][3]-u[3][2]*u[2][3]) - u[1][2]*(u[2][1]*u[3][3]-u[3][1]*u[2][3])
        + u[1][3]*(u[2][1]*u[3][2]-u[3][1]*u[2][2]);
det2=   v[1][1]*(v[2][2]*v[3][3]-v[3][2]*v[2][3]) - v[1][2]*(v[2][1]*v[3][3]-v[3][1]*v[2][3])
        + v[1][3]*(v[2][1]*v[3][2]-v[3][1]*v[2][2]);

d1=w[1];d2=w[2];d3=w[3];

/*Transpose of U obtained above ,to get ut */
for (k=1;k<=n;k++) {
        for (l=1;l<=n;l++)
                ut[l][k]=u[k][l];}

/*Transpose of V obtained above ,to get vt */
for (k=1;k<=n;k++) {
        for (l=1;l<=n;l++)
                vt[l][k]=v[k][l];}

s=det1*det2;
num1=((d1*d1)-(d2*d2))/((d1*d1)-(d3*d3));
num2=((d2*d2)-(d3*d3))/((d1*d1)-(d3*d3));
x1=  sqrt(num1);
x3=  sqrt(num2);

/* Faugeras Decomposition starts here*/
m=3;n=3;
number=0;
/* case 1 */
n1[1]=x1; n1[2]=0; n1[3]=x3;
for(j=1;j<=n;j++){
        n_s[j][1]=0.0;
        for(l=1;l<=m;l++)
                n_s[j][1]+=v[j][l]*n1[l];}

if(((n_s[1][1]*1)+(n_s[2][1]*m_s1[2])+(n_s[3][1]*m_s1[3]))>0){
        number=number+1;
        cas[number]=1;}

/*  case 2 */
n1[1]=x1; n1[2]=0; n1[3]=(-1)*x3;
for(j=1;j<=n;j++){
        n_s[j][2]=0.0;
        for(l=1;l<=m;l++)
                n_s[j][2]+=v[j][l]*n1[l];}

if(((n_s[1][2]*1)+(n_s[2][2]*m_s1[2])+(n_s[3][2]*m_s1[3]))>0){
        number=number+1;
        cas[number]=2;}

/*  case 3 */
n1[1]=(-1)*x1; n1[2]=0; n1[3]=x3;
for(j=1;j<=n;j++){
        n_s[j][3]=0.0;
        for(l=1;l<=m;l++)
                n_s[j][3]+=v[j][l]*n1[l];}

if(((n_s[1][3]*1)+(n_s[2][3]*m_s1[2])+(n_s[3][3]*m_s1[3]))>0){
```

```
        number=number+1;
        cas[number]=3;}

/*  case 4 */
n1[1]= (-1)*x1; n1[2]=0; n1[3]=(-1)*x3;
for(j=1;j<=n;j++){
        n_s[j][4]=0.0;
        for(l=1;l<=m;l++)
                n_s[j][4]+=v[j][l]*n1[l];}

if(((n_s[1][4]*1)+(n_s[2][4]*m_s1[2])+(n_s[3][4]*m_s1[3]))>0){
        number=number+1;
        cas[number]=4;}

n_star_h[1]=1;n_star_h[2]=0;n_star_h[3]=0;
k=cas[1]; l=cas[2];
qq=((n_star_h[1]*n_s[1][k])+(n_star_h[2]*n_s[2][k])+
        (n_star_h[3]*n_s[3][k]));
zz=((n_star_h[1]*n_s[1][l])+(n_star_h[2]*n_s[2][l])+
        (n_star_h[3]*n_s[3][l]));

if(qq>zz)
        i=1;
else
        i=2;
m=3;n=3;
if((s*d_sign)>0)
        {
        dd=d2;
        if(cas[i]==1){
                n1[1]=x1; n1[2]=0; n1[3]=x3;
                for(j=1;j<=n;j++){
                        n_star[j]=0.0;
                        for(l=1;l<=m;l++)
                                n_star[j]+=v[j][l]*n1[l];}

                st=(d1-d3)*x1*x3/d2;
                t1[1]=(d1-d3)*x1; t1[2]=(d1-d3)*0; t1[3]=(d1-d3)*(-1)*(x3);
                        }
        else if(cas[i]==2){
                n1[1]=x1; n1[2]=0; n1[3]=(-1)*x3;
                for(j=1;j<=n;j++){
                        n_star[j]=0.0;
                        for(l=1;l<=m;l++)
                                n_star[j]+=v[j][l]*n1[l];}

                st=(-1)*(d1-d3)*x1*x3/d2;
                t1[1]=(d1-d3)*x1;t1[2]=(d1-d3)*0;t1[3]=(d1-d3)*x3;
                        }
        else if(cas[i]==3){
                n1[1]=(-1)*x1; n1[2]=0; n1[3]=x3;
                for(j=1;j<=n;j++){
                        n_star[j]=0.0;
                        for(l=1;l<=m;l++)
                                n_star[j]+=v[j][l]*n1[l];}

st=(-1)*(d1-d3)*x1*x3/d2;
t1[1]=(d1-d3)*(-1)*(x1); t1[2]=(d1-d3)*0; t1[3]=(d1-d3)*(-1)*(x3);
                        }

        else if(cas[i]==4){
                n1[1]=(-1)*x1; n1[2]=0; n1[3]=(-1)*x3;
                        for(j=1;j<=n;j++){
                        n_star[j]=0.0;
```

```
                        for(l=1;l<=m;l++)
                                n_star[j]+=v[j][l]*n1[l];}
                                st=(d1-d3)*x1*x3/d2;
                                t1[1]=(d1-d3)*(-1)*(x1); t1[2]=(d1-d3)*0;
                                t1[3]=(d1-d3)*x3;
                        }

                        ct=((d2*d2)+(d1*d3))/((d1+d3)*(d2));
                        R1[1][1]=ct; R1[1][2]=0; R1[1][3]=(-1)*st;
                        R1[2][1]=0; R1[2][2]=1; R1[2][3]=0;
                        R1[3][1]=st; R1[3][2]=0; R1[3][3]=ct;

                        for(i=1;i<=3;i++){
                                for(j=1;j<=3;j++){
                                tmp[i][j]=0.0;
                                        for(k=1;k<=3;k++){
                                tmp[i][j]=tmp[i][j]+R1[i][k]*vt[k][j];} } }

                        for(i=1;i<=3;i++){
                                for(j=1;j<=3;j++){
                                tmp1[i][j]=0.0;
                                        for(k=1;k<=3;k++){
                                tmp1[i][j]=tmp1[i][j]+u[i][k]*tmp[k][j];} } }

                        for(i=1;i<=3;i++){
                                for(j=1;j<=3;j++){
                                R[i][j]=0.0;
                                        for(k=1;k<=3;k++){
                                R[k][j]= s*tmp1[k][j];} } }

                        for(j=1;j<=n;j++){
                                nn[j]=0.0;
                                for(l=1;l<=m;l++)
                                        nn[j]+=R[j][l]*n_star[l];}}

else if(s*d_sign<0)
        {
        dd=-d2;
        m=3;n=3;
        if(cas[i]==1){
                n1[1]=x1; n1[2]=0; n1[3]=x3;
                for(j=1;j<=n;j++){
                        n_star[j]=0.0;
                        for(l=1;l<=m;l++)
                                n_star[j]+=v[j][l]*n1[l];}

                st=(d1+d3)*x1*x3/d2;
                t1[1]=(d1+d3)*x1; t1[2]=(d1+d3)*0; t1[3]=(d1+d3)*x3;
                        }
        else if(cas[i]==2){
                n1[1]=x1; n1[2]=0; n1[3]=(-1)*x3;
                for(j=1;j<=n;j++){
                        n_star[j]=0.0;
                        for(l=1;l<=m;l++)
                                n_star[j]+=v[j][l]*n1[l];}

                st=(-1)*(d1+d3)*x1*x3/d2;
                t1[1]=(d1+d3)*x1;t1[2]=(d1+d3)*0;t1[3]=(d1+d3)*(-1)*(x3);
                        }
        else if(cas[i]==3){
                n1[1]=(1)*-x1; n1[2]=0; n1[3]=x3;
                for(j=1;j<=n;j++){
                        n_star[j]=0.0;
                        for(l=1;l<=m;l++)
                                n_star[j]+=v[j][l]*n1[l];}
```

```
                        st=(-1)*(d1+d3)*x1*x3/d2;
                        t1[1]=(d1+d3)*(-1)*(x1); t1[2]=(d1+d3)*0;
                        t1[3]=(d1+d3)*(-1)*(x3);
                }

else if(cas[i]==4){
        n1[1]=(-1)*x1; n1[2]=0; n1[3]=(-1)*x3;
        for(j=1;j<=n;j++){
                n_star[j]=0.0;
                for(l=1;l<=m;l++)
                        n_star[j]+=v[j][l]*n1[l];}

                        st=(d1+d3)*x1*x3/d2;
                        t1[1]=(d1+d3)*(-1)*(x1); t1[2]=(d1+d3)*0;
                        t1[3]=(d1+d3)*(-1)*(x3);
                }

                ct=((d1*d3)-(d2*d2))/((d1-d3)*(d2));
                R1[1][1]=ct; R1[1][2]=0; R1[1][3]=st;
                R1[2][1]=0; R1[2][2]=-1; R1[2][3]=0;
                R1[3][1]=st; R1[3][2]=0; R1[3][3]=(-1)*ct;

                for(i=1;i<=3;i++){
                        for(j=1;j<=3;j++){
                        tmp[i][j]=0.0;
                                for(k=1;k<=3;k++){
                        tmp[i][j]=tmp[i][j]+R1[i][k]*vt[k][j];} } }

                for(i=1;i<=3;i++){
                        for(j=1;j<=3;j++){
                                tmp1[i][j]=0.0;
                                for(k=1;k<=3;k++){
                        tmp1[i][j]= tmp1[i][j]+u[i][k]*tmp[k][j];} } }


                for(i=1;i<=3;i++){
                        for(j=1;j<=3;j++){
                                R[i][j]=0.0;
                                for(k=1;k<=3;k++){
                                        R[k][j]= s*tmp1[k][j];} } }


                for(j=1;j<=n;j++){
                                nn[j]=0.0;
                                for(l=1;l<=m;l++)
                                        nn[j]+=R[j][l]*n_star[l];}}

                for(j=1;j<=n;j++){
                                t[j]=0.0;
                                for(l=1;l<=m;l++)
                                        t[j]+=u[j][l]*t1[l];}

                d=s*dd;
                x_h[1]=t[1]/d;  x_h[2]=t[2]/d;  x_h[3]=t[3]/d;

                csth = R[1][1];       snth = R[2][1];

                theta = acos(csth);
                if(snth < 0)
                        theta = (-1)*theta;

dataPacket[0] = x_h[1];
dataPacket[1] = x_h[2];
dataPacket[2] = theta;
clientSocket.send(dataPacket, 3 * sizeof(float));
```

```
        free_matrix(R,1,NP,1,NP);
        free_matrix(R1,1,NP,1,NP);
        free_matrix(tmp,1,NP,1,NP);
        free_matrix(tmp1,1,NP,1,NP);
        free_matrix(ut,1,NP,1,NP);
        free_matrix(vt,1,NP,1,NP);
        free_matrix(v,1,NP,1,NP);
        free_matrix(u,1,NP,1,NP);
        free_matrix(n_s,1,NP,1,NP);
        free_matrix(alpha,1,NP,1,NP);
        free_matrix(H1,1,NP,1,NP);
        free_matrix(H2,1,NP,1,NP);
        free_matrix(H,1,NP,1,NP);
        free_matrix(a1,1,NP,1,NP);
        free_matrix(b1,1,NP,1,NP);
        free_matrix(b2,1,NP,1,NP);
        free_matrix(mp,1,NP,1,NP);
        free_matrix(mp_s,1,NP,1,NP);
        free_matrix(c,1,NP,1,NP);
        free_matrix(b,1,NP,1,NP);
        free_matrix(a,1,NP,1,NP);
        free_matrix(Ai,1,NP,1,NP);
        free_vector(px1,1,NP);
        free_vector(px_s1,1,NP);
        free_vector(px2,1,NP);
        free_vector(px_s2,1,NP);
        free_vector(px3,1,NP);
        free_vector(px_s3,1,NP);
        free_vector(m1,1,NP);
        free_vector(m_s1,1,NP);
        free_vector(m2,1,NP);
        free_vector(m_s2,1,NP);
        free_vector(m3,1,NP);
        free_vector(m_s3,1,NP);
        free_vector(x,1,NP);
        free_vector(w,1,NP);
        free_vector(nn,1,NP);
        free_vector(n1,1,NP);
        free_vector(t1,1,NP);
        free_vector(t,1,NP);
        free_vector(ttt,1,NP);
        free_vector(x_h,1,NP);
        free_vector(n_star,1,NP);
        free_vector(n_star_h,1,NP);
        free_vector(cas,1,NP);
        free_ivector(indx,1,NP);
        break;

case RoadRunner::e_fifoInt:
        // FIFO overflow.  Indicate error
        dataPacket[0] = -1;
        dataPacket[1] = -1;
        dataPacket[2] = -1;
        clientSocket.send(dataPacket, 3 * sizeof(float));
        cout << "o";

        // Recover
        rr->reset();
        if(rr->d_status.isStatusError())
                {
                        cerr << "server: Error resetting after overflow" << endl;
                        cerr << rr->d_status.getMessageText();
                        rr->shutdown();
                        cleanUpAndExit();
                }
```

```
        // Aquire into bank 0 first
        rr->setNextBank(0);
        bank = 0;
        // Start acquiring again
        rr->grab();
        // Set up to aquire into bank 1 next.
        break;
        case RoadRunner::e_hwInt:
        // HW exception!
        dataPacket[0] = -4;
        dataPacket[1] = -4;
        dataPacket[2] = -4;
        clientSocket.send(dataPacket, 3 * sizeof(float));
        cout << "h";
        // Recover
        if(rr->reset())
                {
                        cerr << "server: Error resetting after hardware exception!" << endl;
                        cerr << rr->d_status.getMessageText();
                        rr->shutdown();
                        cleanUpAndExit();
                }
        // Aquire into bank 0 first
        rr->setNextBank(0);
        bank = 0;
        // Start acquiring again
        rr->grab();
        // Set up to aquire into bank 1 next.
        break;
        default:
        cerr << "server: Unknown interrupt or error!" << endl;
        cleanUpAndExit();
        }
        bank ^= 1;
        }
}

#undef NRANSI

///====================================================================
/// Name:        signalHandler
/// Description: Catches signals in order to shut down before exiting
///====================================================================

void signalHandler(int)
{
        dataPacket[0] = -5;
        dataPacket[1] = -5;
        dataPacket[2] = -5;
        clientSocket.send(dataPacket, 3 * sizeof(float));
        cleanUpAndExit();
}

///====================================================================
/// Name:        cleanUpAndExit
/// Description: Cleans up and exits
///====================================================================
void cleanUpAndExit(void)
{
        if(dataPacket[0])
                dataPacket[0] = -5;
        if(dataPacket[1])
                dataPacket[1] = -5;
        if(dataPacket[2])
                dataPacket[2] = -5;
```

```
	clientSocket.send(dataPacket, 3 * sizeof(float));

	if(rr)
	{
// Check for error so if we didn't map the registers we didn't get a //SIGSEG
		if(rr->d_status.isStatusOk())
			rr->shutdown();
		delete rr;
	}
	if(frameBuffer0)
		delete frameBuffer0;
	if(frameBuffer1)
		delete frameBuffer1;

	free(dataPacket);

	exit(0);
}
```

B.4 WMR TRACKING SHARED MEMORY CLIENT

```
///============================================================================
/// File:        netClient.cpp
/// Description: Sets up the server side of a TCP/IP socket and reads bright
///              spot values from the netServer into shared memory.
/// Note:        To be run in conjunction with KineVis to set up SharedMemory.
///============================================================================

#include "CSocket.hpp"
#include "SharedMemory.hpp"
#include "CmdLineArgs.hpp"
#include <iostream.h>
#include <sys/sched.h>
#include <stdlib.h>

///============================================================================
/// Name:        main
/// Description: The main program function
///============================================================================
void main(int argc, char *argv[])
{
	CmdLineArgs args(argc, argv);
	Socket serverSocket;
	int socketPortNumber;
	SharedMemory x_h1Shm;
	SharedMemory x_h2Shm;
	SharedMemory thetaShm;
	float *x_h1_p;
	float *x_h2_p;
	float *theta_p;
	float dataPacket[3];

	// Get command line parameters
	socketPortNumber = args.getIntegerOption("port", 10000);

	if(args.d_status.isStatusError())
	{
		cerr << "netClient: Error parsing command line" << endl;
		cerr << args.d_status.getMessageText();
		exit(-1);
	}

	// Create shared memory locations
```

```
	x_h1_p = (float *) x_h1Shm.create ("x_h1", sizeof(float));
	if(x_h1Shm.isStatusError())
	{
		cerr << "Error creating x_h1 shared memory" << endl;
		exit(0);
	}
	x_h2_p = (float *) x_h2Shm.create ("x_h2", sizeof(float));
	if(x_h2Shm.isStatusError())
	{
		cerr << "Error creating x_h2 shared memory" << endl;
		exit(0);
	}
	theta_p = (float *) thetaShm.create ("theta", sizeof(float));
	if(thetaShm.isStatusError())
	{
		cerr << "Error creating theta shared memory" << endl;
		exit(0);
	}

	// Set up the socket
	cout << "Waiting for data on port " << socketPortNumber << endl;
	serverSocket.initServerSocket(socketPortNumber);

	// Go to high priority for a more deterministic response
	qnx_scheduler (0, 0, SCHED_FIFO, 27, 0);

	for(;;)
	{
		serverSocket.receive(dataPacket, 3*sizeof(float));
		if(dataPacket[0]!= -1 ||dataPacket[0]!= -4 ||dataPacket[0]!= -2)
		{

			*x_h1_p = dataPacket[0];
			*x_h2_p = dataPacket[1];
			*theta_p = dataPacket[2];
		}

		if(*x_h1_p == -5)
		{
			cout << "Video Server exited.  Quitting." << endl;
			break;
		}
	}
	serverSocket.close();
}
```

B.5 WMR TRACKING CONTROL PROGRAM

```
// ====================================================================
//Control Program : KineVis.cpp
//Description     : In-hand image; normal kinematic controlling of WMR
//                  Qmotor program is for kinematic level control
//Date Written    : June 27,2003
// ====================================================================


#include "ControlProgram.hpp"
#include "IOBoardClient.hpp"
#include "filter.hpp"
#include "SharedMemory.hpp"
#include "CmdLineArgs.hpp"
#include <iostream.h>
#include <conio.h>
#include <errno.h>
```

```
#include <stdlib.h>
#include <unistd.h>
#include <math.h>
#include <stdio.h>
#include <netdb.h>
#include <arpa/inet.h>
#include <sys/socket.h>
#include <sys/sched.h>
#include <netinet/in.h>

class KineVis : public ControlProgram
{
       protected: // Protected data

/*----------------filter declarations------------------*/

double  Unfilt_Velocity_SM,Unfilt_Velocity_DM,Unfilt_Velocity_Smi;
double  Unfilt_Velocity_Dmi,unfilt_x_h1, unfilt_x_h2, unfilt_theta;

              Filter Filt_Velocity_SM;
              Filter Filt_Velocity_DM;
              Filter Filt_Velocity_SMi;
              Filter Filt_Velocity_DMi;

              Filter Filt_XH1;
              Filter Filt_XH2;
              Filter Filt_TH;

              double x_h1f, x_h2f, thetaf;

// Log Variables
double encoder[2];
double analogInputs[4];
//ADC readings from both joystick and current sensors
// current sensors 0,1  spot for joystick 2,3

/*---------------output VD Variables----------------*/

double VD_Safety,VD_Pos_SM,VD_Pos_DM,VD_ang_vel,VD_lin_vel;
double VD_SMvolts,VD_SM_Tor,VD_DM_volts,VD_DM_Tor;
double VD_Voltage0,VD_Voltage1,VD_current0,VD_current1;

/*--------------input VF variables------------------*/

double VF_joy,VF_Velfilter_SM,VF_Velfilter_DM,Velfilter_SM_I;
double Velfilter_DM_I, VF_Filt_XH1, VF_Filt_XH2, VF_Filt_TH;

 /*---The state variables and reference variables---*/

double Vlin,Vlin_old,Vang,Vang_old;
double outputVoltage[2],PI;

/*----------------Shared Memory Setup---------------*/

SharedMemory x_h1Shm;
SharedMemory x_h2Shm;
SharedMemory thetaShm;

//float pointers to shared memory

float *x_h1_p;
float *x_h2_p;
float *theta_p;

/*--------Trajectory/Kinematic Variables-------*/
```

```
double kv;
double kw;
double gamma;
double x_h1;
double x_h2;

double d_sh;
double dd_sh;
double dd_sh_old;
double theta;
double e1;
double e2;
double e3;
double VD_e3deg;
double e2_b;

double w_c;
double v_c;

double VD_x_hd1, VD_x_hd2, VD_theta_d, VD_dtheta_d, VD_dx_hd1;

/*--------Array Memory Allocation Variables--------*/

int arraySize;
//must set to number of lines in desired trajectory file
float * x_hd1;
float * x_hd2;
float * dx_hd1;
float * theta_d;
float * dtheta_d;
/*----Global counter for desired trajectory information-----*/

int desiredSet;

/*------intermediate Variables and control variables-------*/

double sj_tor_com,dj_tor_com,Pos_SM_old,Pos_DM_old,Pos_SM,Pos_DM;
double joy1,joy2,ke1,ke2,tau1,tau2,eta1,eta2;

/*---------------------Friction terms--------------------*/

double Fs_ang,Fs_lin,Fd_ang,Fd_lin;
double lin_sgn,ang_sgn, Fric_ang, Fric_lin;

/*-------------------Hardware Servers--------------------*/

IOBoardClient *d_iobc;

///================================================
/// Public methods:
///================================================

public:

       KineVis
       (
              int argc,                     // # of cmd line arguments
              char *argv[],                 // Array of cmd line arguments
              char *timerServerName="ts0", //timer server providing timing
              char *ioboardServerName="iobs0",
               //  the IOBoard server providing I/O
              int priority = 27
       );
// Constructor, initialize hardware servers, etc. here. Called only //once, when the control
program is loaded.
```

```
        ~KineVis ();
               // Destructor, do cleanup here. Called only once, when the //control program is
        unloaded.

        int startControl();
        // Called each time the control is started by the Supervisor (eg. //the START button
        is pushed. Optional.

        int control ();
// Main control function. Called each time through the control loop.

        int handleMessage (pid_t pid, char *message);
// Called when a message that is not from the Timer server arrives.

        int stopControl();
// Called at the end of each control run. Optional.

        int exitControl();
// Called when exiting Qmotor or loading a different control program.
};

///================================================================
/// name   KineVis::KineVis (constructor replaces enterControl)
///----------------------------------------------------------------
/// input  : argc - # of command line args passed to this program
///          argv - Ptr to array of command line args
///          timerServerName - Name of the timer server which will
///                  -provide the timing for this control program.
/// Constructor for the user's control program. This is called when
/// the program first starts. Connections to needed servers should be
/// made here, log and control variables are initialized and registered
/// here. Any setup that must occur when the program is first started
/// should be placed here.
///================================================================

KineVis::KineVis(int argc, char *argv[], char *timerServerName,
                     char *ioboardServerName, int priority)
        : ControlProgram (argc,argv,timerServerName,priority)
{

        // Initialize your servers here. IOBoard server is used here.

        d_iobc = new IOBoardClient (ioboardServerName);

        if (d_iobc->isStatusError ())
 // If couldn't locate the IOBoard server,
        {
 // set the status and return.
        cerr << "Could not locate IOBoard server" << endl;
        d_status.setStatusError ();
        return;
        }

// Set control program parameters. If running under a Supervisor (like
//the GUI), these values will be overriden by the Supervisor.

        PI = 3.141592653;

        d_runForever = 0;
        d_controlFrequency = 1000;   // Control freq. in Hz
        d_controlDuration = 100.0;
// Control duration in seconds (-1 = run forever)


        registerControlParameter (&VF_joy, "joy");
```

```
        registerControlParameter (&VF_Velfilter_SM, "Velfilter_SM");
        registerControlParameter (&VF_Velfilter_DM, "Velfilter_DM");
        registerControlParameter (&Velfilter_SM_I, "Velfilter_SMi");
        registerControlParameter (&Velfilter_DM_I, "Velfilter_DMi");
        registerControlParameter (&VF_Filt_XH1, "VF_Filt_XH1");
        registerControlParameter (&VF_Filt_XH2, "VF_Filt_XH2");
        registerControlParameter (&VF_Filt_TH, "VF_Filt_TH");
        registerControlParameter (&kv, "kv");
        registerControlParameter (&kw, "kw");
        registerControlParameter (&ke1, "ke1");
        registerControlParameter (&ke2, "ke2");
        registerControlParameter (&gamma, "gamma");


         registerLogVariable (&e1, "e1");
         registerLogVariable (&e2, "e2");
         registerLogVariable (&e3, "e3");
         registerLogVariable (&v_c, "v_c");
         registerLogVariable (&w_c, "w_c");
         registerLogVariable (&d_sh, "d_sh");
         registerLogVariable (&tau1, "tau1");
         registerLogVariable (&tau2, "tau2");
         registerLogVariable (&x_h1, "x_h1");
         registerLogVariable (&x_h2, "x_h2");
         registerLogVariable (&theta, "theta");
         registerLogVariable (&eta1, "eta1");
         registerLogVariable (&eta2, "eta2");
         registerLogVariable (&e2_b, "e2_b");
         registerLogVariable (&VD_x_hd1, "x_hd1");
         registerLogVariable (&VD_x_hd2, "x_hd2");
         registerLogVariable (&VD_theta_d, "theta_d");
         registerLogVariable (&VD_dtheta_d, "dtheta_d");
         registerLogVariable (&VD_dx_hd1, "VD_dx_hd1");
         registerLogVariable (&VD_e3deg, "VD_e3deg");
         registerLogVariable (encoder, "encoder", "encoder position", 2);

        arraySize = 24348; //# lines in datafile for desired trajectory
        x_hd1 = (float*) calloc (arraySize,sizeof(float));
        x_hd2 = (float*) calloc (arraySize,sizeof(float));
        dx_hd1 = (float*) calloc (arraySize,sizeof(float));
        theta_d = (float*) calloc (arraySize,sizeof(float));
        dtheta_d = (float*) calloc (arraySize,sizeof(float));

        //get desired variables from file
        FILE * dataFile;
        dataFile = fopen("desiredFullFilt.dat", "r");

        int i = 0;
        for(i=0;i<arraySize;i++){
                fscanf(dataFile,"%f",&x_hd1[i]);
                fscanf(dataFile,"%f",&dx_hd1[i]);
                fscanf(dataFile,"%f",&x_hd2[i]);
                fscanf(dataFile,"%f",&theta_d[i]);
                fscanf(dataFile,"%f",&dtheta_d[i]);
                //i++;
        }

        fclose(dataFile);

        desiredSet = 0;
        d_status.setStatusOk (); // Constructor succeeded
}
```

```
///======================================================================
/// name   KineVis:~KineVis
///----------------------------------------------------------------------
/// Destructor for the user's control program. Called only once, as
/// the control program exits. Put any cleanup stuff that must happen
/// when your conrol program quits here.
///======================================================================

KineVis::~KineVis()
{
       delete d_iobc; // Disconnect from the IOBoard server
}

///======================================================================
/// name  KineVis::startControl
///----------------------------------------------------------------------
/// Called each time a control run is started. If running from the
/// GUI, this will be called each time the START button is pushed. Do
/// setup that must occur before each control run here
/// (e.g., initializing some counters, etc.)
///======================================================================

int KineVis::startControl() {

       clearAllLogVariables();

/*------------------Initializing Log Variables--------------------*/

       v_c = 0;
       w_c = 0;
       e1 = 0;
       e2 = 0;
       e3 = 0;

/*----------------zero out the dacs and encoders------------------*/

       d_iobc->setDacValue (0, 0.0);
       d_iobc->setDacValue (1, 0.0);

       d_iobc->setEncoderValue (0, 0.0);
       d_iobc->setEncoderValue (1, 0.0);

       VD_Safety = 1.0;
       Pos_SM_old = Pos_SM = 0.0;
       Pos_DM_old = Pos_DM = 0.0;

/*-------------------Initializing the filters--------------------*/

       initfilter(VF_Velfilter_SM, &Filt_Velocity_SM, d_controlPeriod);
       initfilter(VF_Velfilter_DM, &Filt_Velocity_DM, d_controlPeriod);
       initfilter(Velfilter_SM_I, &Filt_Velocity_SMi, d_controlPeriod);
       initfilter(Velfilter_DM_I, &Filt_Velocity_DMi, d_controlPeriod);
       initfilter(VF_Filt_XH1, &Filt_XH1, d_controlPeriod);
       initfilter(VF_Filt_XH2, &Filt_XH2, d_controlPeriod);
       initfilter(VF_Filt_TH, &Filt_TH, d_controlPeriod);

/*-----------------Initialize Other Variables------------------*/

       dd_sh_old = 0;
       desiredSet = 0;

       x_h1_p = (float *) x_h1Shm.attach("x_h1", sizeof(float));
       if(x_h1Shm.isStatusError())
       {
```

```
		cerr << "Error opening shared memory \"x_h1\": ";
		cerr << strerror(errno) << endl;
		return 0;
	}

	x_h2_p = (float *)x_h2Shm.attach("x_h2", sizeof(float));
	if(x_h2Shm.isStatusError())
	{
		cerr << "Error opening shared memory \"x_h2\": ";
		cerr << strerror(errno) << endl;
		return 0;
	}

	theta_p = (float *)thetaShm.attach("theta", sizeof(float));
	if(thetaShm.isStatusError())
	{
		cerr << "Error opening shared memory \"theta\": ";
		cerr << strerror(errno) << endl;
		return 0;
	}

	return (0);
}

///=====================================================================
/// name  KineVis::control
///---------------------------------------------------------------------
/// return : 0 - Control run may continue, nonzero - Control should
///stop Called each control cycle. Do your input, control
///computations, and output here. If you return 0, the control will
///continue to execute. If you return nonzero, the control will abort.
///You may want to abort if some error condition occurs (excessive
///velocity,  etc.)
///=====================================================================

int KineVis::control ()
{
// Input Routine

/*--Reading the encoder positions,joystick position and current sensors---*/

	encoder[0] = d_iobc->getEncoderValue (0);
	analogInputs[0] = d_iobc->getAdcValue (0);
	analogInputs[2] = d_iobc->getAdcValue (2) - 1.73;
	encoder[1] = d_iobc->getEncoderValue (1);
	analogInputs[1] = d_iobc->getAdcValue (1);
	analogInputs[3] = d_iobc->getAdcValue (3) - 1.73;

	Unfilt_Velocity_DMi = (analogInputs[0] -0.0425) * 2.0;
	Unfilt_Velocity_SMi = (analogInputs[1] -0.0171) * 2.0;

	VD_current0 = filter(Unfilt_Velocity_DMi, &Filt_Velocity_DMi);
	VD_current1 = filter(Unfilt_Velocity_SMi, &Filt_Velocity_SMi);
/*---------Getting drive and steer wheel positions-----------*/

	Pos_SM = encoder[1]*2.0*PI /1024.0;
	VD_Pos_SM  = 180.0*Pos_SM/PI;

	Pos_SM_old = Pos_SM;

	Pos_DM = encoder[0]*0.10617*2.0*PI/(98304.0);
	//position in meters  (98304 = 1024*96)
	VD_Pos_DM = Pos_DM;
```

```
/*-------------Filtering & Calculating Velocity------------*/

	Unfilt_Velocity_SM = (Pos_SM - Pos_SM_old)/ d_controlPeriod;
	Unfilt_Velocity_DM = (Pos_DM - Pos_DM_old)/ d_controlPeriod;

	Vang = filter(Unfilt_Velocity_SM, &Filt_Velocity_SM);
	Vlin = filter(Unfilt_Velocity_DM, &Filt_Velocity_DM);

	VD_ang_vel = Vang;
	VD_lin_vel = Vlin;

	Pos_SM_old = Pos_SM;
	Pos_DM_old = Pos_DM;

	joy1= analogInputs[2];
	joy2= analogInputs[3];

/*-============-Retrieve Variables from Shared Memory-=============-*/

	if (VF_joy != 1 )
	{
		unfilt_x_h1 = (double) *x_h1_p;
		unfilt_x_h2 = (double) *x_h2_p;
		unfilt_theta = (double) *theta_p;

		x_h1 = filter(unfilt_x_h1, &Filt_XH1);
		x_h2 = filter(unfilt_x_h2, &Filt_XH2);
		theta = filter(unfilt_theta, &Filt_TH);

		VD_x_hd1 = x_hd1[desiredSet];
		VD_dx_hd1 = dx_hd1[desiredSet];
		VD_x_hd2 = x_hd2[desiredSet];
		VD_theta_d = theta_d[desiredSet];
		VD_dtheta_d = dtheta_d[desiredSet];

/*-===============-Control calculations-==================-*/

	///error calculations
		e1 = (double) x_h1 - x_hd1[desiredSet];
		e2 = (double) x_h2 - x_hd2[desiredSet];
		e3 = (double) theta - theta_d[desiredSet];

		VD_e3deg = (e3 * 180) / PI;

		///controller
		e2_b = e2 - x_hd1[desiredSet] * e3;

		///angular velocity controller
		w_c = (double) (kw * e3 - dtheta_d[desiredSet] - dx_hd1[desiredSet] * e2_b);

		//parametric update law for linear velocity
		dd_sh = gamma * e1 * (x_h2 * w_c - dx_hd1[desiredSet]);

	//trapazoidal rule for integration of dd_sh to get d_sh

		d_sh += (d_controlPeriod * (dd_sh_old + dd_sh)) * 0.5;
		dd_sh_old = dd_sh;

	///linear velocity controller
		v_c = (double) (kv * e1 - e2_b * w_c + d_sh * (x_h2 * w_c -
		dx_hd1[desiredSet]));

	///increment desiredSet for next iteration
	desiredSet++;
```

```
            if(desiredSet == (arraySize-3)){
            desiredSet = (arraySize-4);
            }

    } //end if not on joy

/************* Torque output *****************/

        eta1 = (double) v_c - Vlin;
        eta2 = (double) w_c - Vang;
        tau1 = ke1 * eta1;
        tau2 = ke2 * eta2;
        dj_tor_com = tau1;
        sj_tor_com = tau2;

        if (VF_joy == 1.0 )
        {
                joy1 *= 0.5 ;
                joy2 *= 1.0 ;

                outputVoltage[0] = joy1;
                outputVoltage[1] = joy2;

        }

        else {
        /* -- convert the torque computed for the steering joint
           -- to the torque applied to the motor  */

                VD_SM_Tor = sj_tor_com/106.0;
                VD_DM_Tor = dj_tor_com/96.0;
                VD_SM_volts = 5.0 * (VD_SM_Tor/1.260);
                VD_DM_volts = 5.0 * (VD_DM_Tor/2.040);

   //-----------As a safety measure we set the following --------

        if (VD_SM_volts > 4.25)
                VD_SM_volts = 4.25;
        else if (VD_SM_volts < -4.25)
                VD_SM_volts  = -4.25;

        if (VD_DM_volts > 4.25 )
                VD_DM_volts = 4.25;
        else if (VD_DM_volts < -4.25)
                VD_DM_volts  = -4.25;

        if(VD_SM_volts > 4.5 || VD_SM_volts < -4.5 || VD_DM_volts > 4.5 || VD_DM_volts < -4.5)
                VD_Safety = 0.0;

                VD_Voltage0 = (VD_Safety * VD_SM_volts);
                VD_Voltage1 = (VD_Safety * VD_DM_volts);
        outputVoltage[0] = VD_Voltage0; //-(VD_Safety * VD_SM_volts);
        outputVoltage[1] = VD_Voltage1; //-(VD_Safety * VD_DM_volts);
        }

        d_iobc->setDacValue (0, outputVoltage[0]);
        d_iobc->setDacValue (1, outputVoltage[1]);

        return 0;
}
```

```
///====================================================================
/// name  KineVis::handleMessage
///--------------------------------------------------------------------
/// return : 0 - Control run may continue, nonzero - Control should
/// stop This optional function allows your control program to receive
/// QNX messages from processes other than the Timer server. If a msg
/// arrives and it is not from the Timer server, this function will be
/// called to handle it. Based on the message you may choose to stop
/// the control (return nonzero) or continue the control (return 0).
///====================================================================

int KineVis::handleMessage (pid_t pid, char *message)
{
       int msg;

       msg = *((int *)message); // Convert the msg to an integer
       Reply (pid, 0, 0);// Reply to the msg so the sender is unblocked

       if (msg == 1)  // If the msg was "1" abort the control.
       return (1);

       return (0);           // The control may continue.
}


///====================================================================
/// name  KineVis::stopControl
/// -------------------------------------------------------------------
/// Called each time a control run ends. If running from the GUI, this
/// will be called each time the STOP button is pushed, or when a
///timed run ends, or when the control aborts itself.
///====================================================================

int KineVis::stopControl() {
       int channel;

       for (channel = 0; channel < 8; channel++)
              d_iobc->setDacValue (channel, 0.0);
 // Zero out the DAC we were using.

       return (0);
}

//====================================================================
// name  KineVis::exitControl
// -------------------------------------------------------------------
// This function is called when the control is unloaded. In standalone
// mode, this happens after one control run has completed. When using
// the GUI, it happens when the user loads a new control program or
// exits the GUI.
//====================================================================

int KineVis::exitControl()
{
       delete d_iobc; // Disconnect from the IOBoard server
       free(x_hd1);
       free(x_hd2);
       free(dx_hd1);
       free(theta_d);
       free(dtheta_d);
       return 0;
}

//====================================================================
// main()
```

```
//-------------------------------------------------------------------
// The main function instantiates the object and goes into the main //loop
//===================================================================

main (int argc, char *argv[])
{

// Instantiate the control program.
// The first 3 arguments are required (argc, argv, and the name of the
//timer server. The fourth argument and beyond are determined by the
//user.Here it's just the name of the IOBoard server used for I/O. If
//you use additional servers you may pass their names as arguments
//here.
       KineVis cp (argc,argv,"ts0","iobs0",27);

// If couldn't start the control (maybe couldn't connect to one of the
// hardware servers, etc.), abort.

       if (!cp.d_status.isStatusOk())
       {
              cerr << "Can't start control" << endl;
              abort ();
       }

       // This mainLoop() is defined by the superclass ControlProgram.

       cp.mainLoop();
}
```

B.6 WMR REGULATION CONTROLLER

```
//=============================================================================
//  Control Program : KineVis.cpp
//  Description     : Fixed Camera; normal kinematic controlling of WMR
//                  : Qmotor program is for kinematic level control
//  Author          : Warren Dixon, Eric Holcombe, Prakash Chawda
//=============================================================================

#include "ControlProgram.hpp"
#include "IOBoardClient.hpp"
#include "filter.hpp"
#include "SharedMemory.hpp"
#include "CmdLineArgs.hpp"
#include "ButterworthFilter.hpp"
#include <iostream.h>
#include <conio.h>
#include <errno.h>
#include <stdlib.h>
#include <unistd.h>
#include <math.h>
#include <stdio.h>
#include <netdb.h>
#include <arpa/inet.h>
#include <sys/socket.h>
#include <sys/sched.h>
#include <netinet/in.h>

class KineVis : public ControlProgram
{
       protected: // Protected data

/*----------------filter declarations------------------*/

              double Unfilt_Velocity_SM,Unfilt_Velocity_DM,Unfilt_Velocity_SMi,
```

```
                Unfilt_Velocity_DMi, unfilt_gamma1, unfilt_m3_2, unfilt_theta;

                Filter        Filt_Velocity_SM;
                Filter        Filt_Velocity_DM;
                Filter        Filt_Velocity_SMi;
                Filter        Filt_Velocity_DMi;

                ButterworthFilter<float> Filt_GAMMA1;
                ButterworthFilter<float> Filt_TH1;
                ButterworthFilter<float> Filt_M3_2;

                double gamma1f, m3_2f, thetaf;

                // Log Variables
                double encoder[2];
                double analogInputs[4];    //ADC readings from joystick

/*---------------output VD Variables-----------------*/

                double VD_Safety,VD_Pos_SM,VD_Pos_DM,VD_ang_vel,VD_lin_vel,
                VD_SM_volts,VD_SM_Tor,VD_DM_volts,VD_DM_Tor,VD_Voltage0,
                VD_Voltage1,VD_current0,VD_current1;


/*--------------input VF variables------------------*/

                double VF_joy,VF_Velfilter_SM,VF_Velfilter_DM,Velfilter_SM_I, Velfilter_DM_I,
                VF_Filt_GAMMA1, VF_Filt_M3_2, VF_Filt_TH;


/*--------------The state variables and reference variables----------------*/

                double Vlin,Vlin_old,Vang,Vang_old;
                double outputVoltage[2],PI;

/*---------------------------Shared Memory Setup-----------------------------*/

                SharedMemory gamma1Shm;
                SharedMemory thetaShm;
                SharedMemory m3_2Shm;

                //float pointers to shared memory

                float *gamma1_p;
                float *m3_2_p;
                float *theta_p;

/*----------------------Trajectory/Kinematic Variables-----------------------*/

                double kv;
                double kw;
                double the_TAU;
                double gamma1;
                double m3_2;
                double m_s1_2;
                double alpha_h;
                double dalpha_h;
                double dalpha_h_old;
                double theta;
                double r1;
                double r2;
                double r3;
                double the_X;
                double ETA;
                double w_c;
```

```
              double v_c;

              double sj_tor_com,dj_tor_com,Pos_SM_old,Pos_DM_old,Pos_SM,
                      Pos_DM, joy1,joy2,ke1,ke2,tau1,tau2,eta1,eta2;

/*------------------------Friction terms---------------------------*/
              double  Fs_ang,Fs_lin,Fd_ang,Fd_lin,lin_sgn,ang_sgn, Fric_ang, Fric_lin;

/*------------------------Hardware Servers---------------------------*/
              IOBoardClient *d_iobc;
///================================================
/// Public methods:
///================================================
       public:

              KineVis
              (
                     int argc,                    // # of cmd line arguments
                     char *argv[],                // Array of cmd line arguments
                     char *timerServerName="ts0",  // Name of the timer server
                     char *ioboardServerName="iobs0",// Name of the IOBoard
                     int priority = 27
              );


              ~KineVis ();

              int startControl();
                     // Called each time the control is started by the Supervisor
                     // START button is pushed. Optional.

              int control ();
                     // Main control function. Called each time

              int handleMessage (pid_t pid, char *message);
                     // Called when a message that is not from the Timer server
                     // This allows you to receive QNX message from other

              int stopControl();
                     // Called at the end of each control run. Optional.

              int exitControl();
                     // Called when exiting Qmotor or loading a different control
};


///=============================================================================
/// name   KineVis::KineVis (constructor replaces enterControl)
///-----------------------------------------------------------------------------
/// input  : argc - # of command line args passed to this program
///          argv - Ptr to array of command line args
///          timerServerName - Name of the timer server which will provide the
///                            timing for this control program.
/// Constructor for the user's control program. This is called when the
/// program first starts. Connections to needed servers should be made here,
/// log and control variables are initialized and registered here. Any setup
/// that must occur when the program is first started should be placed here.
///=============================================================================

KineVis::KineVis(int argc, char *argv[], char *timerServerName,
       char *ioboardServerName, int priority)
       : ControlProgram (argc,argv,timerServerName,priority)
{
```

```
        // Initialize your servers here. IOBoard server is used here.

        d_iobc = new IOBoardClient (ioboardServerName);

        if (d_iobc->isStatusError ()) // If couldn't locate the IOBoard server,
        {                             // set the status and return.
                cerr << "Could not locate IOBoard server" << endl;
                d_status.setStatusError ();
                return;
        }

        // Set control program parameters. If running under a Supervisor (like
        // GUI), these values will be overriden by the Supervisor.

        PI = 3.141592653;

        d_runForever = 0;
        d_controlFrequency = 1000;   // Control freq. in Hz
        d_controlDuration = 100.0;    // Control duration in seconds

        // Register your control variables. Only registered control variables
        // appear in the Supervisor (ie. the GUI). You do NOT have to register
        // your variables, only the ones you want to change from the GUI. The
        // 2nd parameter is the description that will be seen in the GUI.

        registerControlParameter (&VF_joy, "joy");
        registerControlParameter (&VF_Velfilter_SM, "Velfilter_SM");
        registerControlParameter (&VF_Velfilter_DM, "Velfilter_DM");
        registerControlParameter (&Velfilter_SM_I, "Velfilter_SMi");
        registerControlParameter (&Velfilter_DM_I, "Velfilter_DMi");
        registerControlParameter (&VF_Filt_GAMMA1, "VF_Filt_GAMMA1");
        registerControlParameter (&VF_Filt_M3_2, "VF_Filt_M3_2");
        registerControlParameter (&VF_Filt_TH, "VF_Filt_TH");
        registerControlParameter (&kv, "kv");
        registerControlParameter (&kw, "kw");
        registerControlParameter (&ke1, "ke1");
        registerControlParameter (&ke2, "ke2");
        registerControlParameter (&the_TAU, "the_TAU");

        // Initialize your log variables here.

        // Register your log variables. Only registered log variables can
        // be logged/plotted by the Supervisor. The 2nd parameter is the
        // description that will be seen in the GUI.

        //trajectory vars
        registerLogVariable (&r1, "r1");
        registerLogVariable (&r2, "r2");
        registerLogVariable (&r3, "r3");
        registerLogVariable (&v_c, "v_c");
        registerLogVariable (&w_c, "w_c");
        registerLogVariable (&tau1, "tau1");
        registerLogVariable (&tau2, "tau2");
        registerLogVariable (&gamma1, "gamma1");
        registerLogVariable (&m3_2, "m3_2");
        registerLogVariable (&theta, "theta");
        registerLogVariable (&eta1, "eta1");
        registerLogVariable (&eta2, "eta2");
        registerLogVariable (encoder, "encoder", "encoder position", 2);
        registerLogVariable (&alpha_h, "alpha");
}

///============================================================================
/// name   KineVis:~KineVis
```

```
///------------------------------------------------------------------------
/// Destructor for the user's control program. Called only once, as the
/// control program exits. Put any cleanup stuff that must happen when your
/// conrol program quits here.
///========================================================================

KineVis::~KineVis()
{
       delete d_iobc; // Disconnect from the IOBoard server
}


///========================================================================
/// name  KineVis::startControl
///------------------------------------------------------------------------
/// Called each time a control run is started. If running from the GUI, this
/// will be called each time the START button is pushed. Do setup that must
/// occur before each control run here (eg. initializing some counters, etc.)
///========================================================================

int KineVis::startControl() {

       clearAllLogVariables();

/*-------------------Initializing Log Variables--------------------*/

       v_c = 0;
       w_c = 0;
       r1 = 0;
       r2 = 0;
       r3 = 0;

/*----------------zero out the dacs and encoders------------------*/

       d_iobc->setDacValue (0, 0.0);
       d_iobc->setDacValue (1, 0.0);

       d_iobc->setEncoderValue (0, 0.0);
       d_iobc->setEncoderValue (1, 0.0);

       VD_Safety = 1.0;

       Pos_SM_old = Pos_SM = 0.0;
       Pos_DM_old = Pos_DM = 0.0;

/*--------------------Initializing the filters--------------------*/

       initfilter(VF_Velfilter_SM, &Filt_Velocity_SM, d_controlPeriod);
       initfilter(VF_Velfilter_DM, &Filt_Velocity_DM, d_controlPeriod);
       initfilter(Velfilter_SM_I, &Filt_Velocity_SMi, d_controlPeriod);
       initfilter(Velfilter_DM_I, &Filt_Velocity_DMi, d_controlPeriod);

       Filt_GAMMA1.setCutOffFrequency(VF_Filt_GAMMA1);
       Filt_GAMMA1.setSamplingTime(d_controlPeriod);
       Filt_GAMMA1.setDampingRatio(1.0);
       Filt_GAMMA1.setAutoInit();

       Filt_TH1.setCutOffFrequency(VF_Filt_TH);
       Filt_TH1.setSamplingTime(d_controlPeriod);
       Filt_TH1.setDampingRatio(1.0);
       Filt_TH1.setAutoInit();

       Filt_M3_2.setCutOffFrequency(VF_Filt_M3_2);
       Filt_M3_2.setSamplingTime(d_controlPeriod);
```

```
	Filt_M3_2.setDampingRatio(1.0);
	Filt_M3_2.setAutoInit();

/*-----------------Initialize Other Variables-------------------*/

	dalpha_h_old = 0;


	gamma1_p = (float *) gamma1Shm.attach("gamma1", sizeof(float));
	if(gamma1Shm.isStatusError())
	{
		cerr << "Error opening shared memory \"gamma1\": ";
		cerr << strerror(errno) << endl;
		return 0;
	}

	theta_p = (float *)thetaShm.attach("theta", sizeof(float));
	if(thetaShm.isStatusError())
	{
		cerr << "Error opening shared memory \"theta\": ";
		cerr << strerror(errno) << endl;
		return 0;
	}

	m3_2_p = (float *)m3_2Shm.attach("m3_2", sizeof(float));
	if(m3_2Shm.isStatusError())
	{
		cerr << "Error opening shared memory \"m3_2\": ";
		cerr << strerror(errno) << endl;
		return 0;
	}
	return (0);
}

///=============================================================================
/// name  KineVis::control
///-----------------------------------------------------------------------------
/// return : 0 - Control run may continue, nonzero - Control should stop
/// Called each control cycle. Do your input, control computations, and output
/// here. If you return 0, the control will continue to execute. If you return
/// nonzero, the control will abort. You may want to abort if some error
/// condition occurs (excessive velocity,  etc.)
///=============================================================================
int KineVis::control ()
{
// Input Routine

/*-------Reading the encoder positions,joystick position and current sensors----------*/

	encoder[0] = d_iobc->getEncoderValue (0);
	analogInputs[0] = d_iobc->getAdcValue (0);
	analogInputs[2] = d_iobc->getAdcValue (2) - 1.73;
	encoder[1] = d_iobc->getEncoderValue (1);
	analogInputs[1] = d_iobc->getAdcValue (1);
	analogInputs[3] = d_iobc->getAdcValue (3) - 1.73;

	Unfilt_Velocity_DMi = (analogInputs[0] -0.0425) * 2.0;
	Unfilt_Velocity_SMi = (analogInputs[1] -0.0171) * 2.0;

	VD_current0 = filter(Unfilt_Velocity_DMi, &Filt_Velocity_DMi);
	VD_current1 = filter(Unfilt_Velocity_SMi, &Filt_Velocity_SMi);

/*--------------Getting drive and steer wheel positions--------------------*/

	Pos_SM = encoder[1]*2.0*PI /1024.0;
```

```
        VD_Pos_SM  = 180.0*Pos_SM/PI;

        Pos_SM_old = Pos_SM;

        Pos_DM = encoder[0]*0.10617*2.0*PI/(98304.0);   //position in meters  //(98304 =
                                                                    1024*96)
        VD_Pos_DM = Pos_DM;

/*----------------Filtering & Calculating Velocity-------------------*/

        Unfilt_Velocity_SM = (Pos_SM - Pos_SM_old)/ d_controlPeriod;
        Unfilt_Velocity_DM = (Pos_DM - Pos_DM_old)/ d_controlPeriod;

        Vang = filter(Unfilt_Velocity_SM, &Filt_Velocity_SM);
        Vlin = filter(Unfilt_Velocity_DM, &Filt_Velocity_DM);

        VD_ang_vel = Vang;
        VD_lin_vel = Vlin;

        Pos_SM_old = Pos_SM;
        Pos_DM_old = Pos_DM;

        joy1= analogInputs[2];
        joy2= analogInputs[3];

/*-============-Retrieve Variables from Shared Memory-=============-*/
//***remember these are just pointers***

        if (VF_joy != 1 )
        {

                //assign pointer stuff to local variables

                unfilt_gamma1 = (double) *gamma1_p;
                unfilt_theta = (double) *theta_p;
                unfilt_m3_2 = (double) *m3_2_p;

                gamma1 = Filt_GAMMA1.filter(unfilt_gamma1);
                theta = Filt_TH1.filter(unfilt_theta);
                m3_2 = Filt_M3_2.filter(unfilt_m3_2);
                m_s1_2 = 0.04;


/*-===============-Control calculations-==================-*/

        /*--------------------
        ----The Kinematics----
        --------------------*/

                ///error calculations

                r1 = (double) -1 * theta;
                r2 = (double) -1 * (1/gamma1 - cos(theta) + m_s1_2 * sin(theta));
                r3 = (double) (1/gamma1 * m3_2 - sin(theta) - m_s1_2 * cos(theta));

                ///controller
                ETA = r2 - r3 * sin(d_controlPeriod);

                the_X = (ETA + r3 * sin(d_controlPeriod)) * (r3 - ETA * sin(d_controlPeriod));

                ///angular velocity controller
                w_c = (double) (kw * (r1 + the_X));

                //parametric update law for linear velocity
                dalpha_h = the_TAU * (-1 * r3 * ETA * cos(d_controlPeriod) - w_c * ETA * r3);
```

```
            //trapazoidal rule for integration of dd_sh to get d_sh
            alpha_h += (d_controlPeriod * (dalpha_h_old + dalpha_h)) * 0.5;
            dalpha_h_old = dalpha_h;

            ///linear velocity controller
            v_c = (double) (kv * ETA + alpha_h * r3 * cos(d_controlPeriod) + alpha_h * w_c
            * r3);

   } //end if not on joy

      /************* Torque output *****************/

      eta1 = (double) v_c - Vlin;
      eta2 = (double) w_c - Vang;
      tau1 = - ke1 * eta1;
      tau2 = ke2 * eta2;
      dj_tor_com = tau1;
      sj_tor_com = tau2;

      if (VF_joy == 1.0 )
      {

            joy1 *= 0.5 ;
            joy2 *= 1.0 ;

            outputVoltage[0] = joy1;
            outputVoltage[1] = joy2;

      }

      else {
      /* -- convert the torque computed for the steering joint
         -- to the torque applied to the motor
      */

            VD_SM_Tor = sj_tor_com/106.0;
            VD_DM_Tor = dj_tor_com/96.0;


/* -- convert Nm computed by the control to volts ----

 -- NOTE THAT THE LINEAR AMP FOR THE STEER MOTOR SHOULD BE 12
 -- AND THE DRIVE MOTOR AMP SHOULD BE ADJUSTED TO 8

 -- 5 takes into account the fact that the multiQ can only supply +-5 volts --
 -- 5/1.260  is the (max volts)/(max NM) conversion factor for the -steer motor--
 -- 5/2.040 is the volts/NM conversion factor for the  -drive motor--*/

            VD_SM_volts = 5.0 * (VD_SM_Tor/1.260);
            VD_DM_volts = 5.0 * (VD_DM_Tor/2.040);

 /*-----------As a safety measure we set the following ---------------
 ----------1/2 power safety feature to protect the MultiQ-----------*/

            if (VD_SM_volts > 4.25)
                  VD_SM_volts = 4.25;
            else if (VD_SM_volts < -4.25)
                  VD_SM_volts  = -4.25;

            if (VD_DM_volts > 4.25 )
                  VD_DM_volts = 4.25;
            else if (VD_DM_volts < -4.25)
                  VD_DM_volts  = -4.25;
```

```
            if(VD_SM_volts > 4.5 || VD_SM_volts < -4.5 || VD_DM_volts > 4.5 || VD_DM_volts
            < -4.5)
            VD_Safety = 0.0;


            VD_Voltage0 = (VD_Safety * VD_SM_volts);
            VD_Voltage1 = (VD_Safety * VD_DM_volts);

            outputVoltage[0] = VD_Voltage0; //-(VD_Safety * VD_SM_volts);
            outputVoltage[1] = VD_Voltage1; //-(VD_Safety * VD_DM_volts);
      }

      d_iobc->setDacValue (0, outputVoltage[0]);
      d_iobc->setDacValue (1, outputVoltage[1]);

      //cout << "finished control loop" << endl;
      return 0;
}


///=============================================================================
/// name  KineVis::handleMessage
///-----------------------------------------------------------------------------
/// return : 0 - Control run may continue, nonzero - Control should stop
/// This optional function allows your control program to receive QNX messages
/// from processes other than the Timer server. If a msg arrives and it is not
/// from the Timer server, this function will be called to handle it. Based on
/// the message you may choose to stop the control (return nonzero) or
/// continue the control (return 0).
///=============================================================================

int KineVis::handleMessage (pid_t pid, char *message)
{
      int msg;

      msg = *((int *)message); // Convert the msg to an integer
      Reply (pid, 0, 0);       // Reply to the msg so the sender is unblocked


      if (msg == 1)  // If the msg was "1" abort the control.
            return (1);

      return (0);           // The control may continue.
}

///=============================================================================
/// name  KineVis::stopControl
///-----------------------------------------------------------------------------
/// Called each time a control run ends. If running from the GUI, this
/// will be called each time the STOP button is pushed, or when a timed run
/// ends, or when the control aborts itself.
///=============================================================================

int KineVis::stopControl() {

      int channel;

      for (channel = 0; channel < 8; channel++)
            d_iobc->setDacValue (channel, 0.0); // Zero out the DAC we were using.

      return (0);
}
```

```
//============================================================================
// name  KineVis::exitControl
// ----------------------------------------------------------------------------
// This function is called when the control is unloaded. In standalone
// mode, this happens after one control run has completed. When using
// the GUI, it happens when the user loads a new control program or
// exits the GUI.
//============================================================================

int KineVis::exitControl()
{
	delete d_iobc; // Disconnect from the IOBoard server
	/*free(x_hd1);
	free(x_hd2);
	free(dx_hd1);
	free(theta_d);
	free(dtheta_d); */
	return 0;
}


//============================================================================
// main()
//----------------------------------------------------------------------------
// The main function instantiates the object and goes into the mainloop
//============================================================================

main (int argc, char *argv[])
{
	KineVis cp (argc,argv,"ts0","iobs0",27);

	// If couldn't start the control (maybe couldn't connect to one of the
	// hardware servers, etc.), abort.
	if (!cp.d_status.isStatusOk())
	{
		cerr << "Can't start control" << endl;
		abort ();
	}
	cp.mainLoop();
}
```

INTERNAL DISTRIBUTION

1. E. C. Fox
2. D. J. Hill
3. J. F. Jansen
4. C. M. Kendrick
5. R. F. Lind
6. P. D. Lloyd
7. L. J. Love
8. M. W. Noakes
9. F. G. Pin
10. B. S. Richardson
11. J. C. Rowe
12. R. B. Shelton
13. Central Research Library
14. ORNL Laboratory Records–RC
15. ORNL Laboratory Records–OSTI

EXTERNAL DISTRIBUTION

16. P. V. Chawda, 341 EIB Fluor Daniel, Dept. of Electrical Engineering, Clemson University, Clemson, SC 29634
17. W. E. Warren Dixon, Assistant Professor, Department of Mechanical and Aerospace Engineering, University of Florida, P.O. Box 116250, Room 312 MAE-A, Gainesville, FL 32611
18. T. J. Flynn, 5180 Parkstone Dr., Suite 100, Chantilly, VA 20151
19. Teresa Fryberger, SC-75/Germantown Building, U.S. Department of Energy, 1000 Independence Ave., S.W., Washington, DC 20585-1290
20. Roland Hirsch, SC-73/Germantown Building, U.S. Department of Energy, 1000 Independence Ave., S.W., Washington, DC 20585-1290
21. E. B. Holcombe, N15E, PNC Courts, P.O. Box 6459, Clemson University, Clemson, SC 29634