Inverse Kinematics and Gaze Stabilization for the Rochester Robot Head

John Soong and Christopher Brown

The University of Rochester
Computer Science Department
Rochester, New York 14627

Technical Report 394

August 1991

Abstract

The Rochester robot head is a three degree of freedom camera platform providing independent pan axes and a shared tilt axis for two cameras. The head can be moved by a six degree of freedom robot arm. In this context, the gaze stabilizing problem is to cancel the effects of arbitrary head motion (in six d.o.f.) by compensating pan and tilt motions. We explore the kinematic formulation of the gaze stabilization problem, and experimentally compare two approaches (positional differences and Jacobian calculation). Various errata in an earlier TR [Brown and Rimey, 1988] are corrected and some of its formulae are verified experimentally. We describe a method of sending robot positions back to the host computer, and use this facility to implement the gaze control schemes. This report includes descriptions of relevant local software.

This material is based upon work supported by the National Science Foundation under Grants numbered IRI-8920771 and IRI-8903582.

1 Gaze Stabilization

The Rochester robot head is a three degree of freedom binocular camera mount (two individual camera pan motors mounted on a common tilt platform), itself mounted on the end of a six degree of freedom industrial (Puma) robot arm [Brown, 1988]. Gaze stabilization for the Rochester robot compensates for head motion (effected by motion of the head-carrying robot arm) with camera motion (caused by the pan and tilt motors on the head). The goal is to produce a stable image despite head movement. The benefits of gaze stabilization are well known and will not be enlarged upon here.

The form of the gaze-stabilization problem we have chosen is the following: compensate with camera pan and tilt motions for head motions in translation and rotation, so as to keep a given point (fixed in space) at the center of the field of view of the camera. This monocular form of the problem is all we can solve in general since our tilt platform is shared. However in many practical cases the resulting tilt commands are identical and both cameras can be centered on the target.

Visual clues can be used to stabilize gaze by active tracking of patterns in the field of view [Brown, 1988] using correlation techniques. Functionally this is like the human "smooth pursuit" system. Vergence is another way to stabilize gaze [Olson and Coombs, 1990; Coombs and Brown, 1990]. At Rochester, vergence and tracking have been implemented together to track a moving object visually from a moving platform [Coombs, 1991].

However, our goal in this work is to do gaze stabilization without visual input. Functionally this capability is like the human vestibular-occular reflex (VOR). The VOR relies on input from accelerometers measuring head rotation, and there is a similar reflex using the otolith organs to sense head translational accelerations. For the robo, one might use sensors such as accelerometers, tachometers, or position encoders to ascertain the robot's motion. We do not have such sensors, however. Gaze stabilization might use some form of "efferent copy", information, such as whatever movement commands were given to the robot. In our lab there are technical problems with this approach (Sec. 5). Intercepting the low-level motor command signals (to the robot arm's joints) might be possible but would require additional electronics and also a kinematic simulation to know what their cumulative effect was.

Given our technical situation the practical approach is to query the robot to get its head location and send that information back to the control program. Robot head location is reported as (x, y, z, O, A, T), the 3-D location of the origin of head coordinates and the Euler angles that determine its orientation [Brown and Rimey, 1988]. This information, along with the known (x, y, z) position of the target, can be converted into pan and tilt motions that will bring the object to the center of the field of the camera. Differencing these pans and tilts gives us velocities, which we use to control the camera angles velocity-streaming mode. Alternatively, differencing the sequence of output location sextuples yields a sequence of velocity signals in robot location parameters. Inverse jacobean calculations translate these velocities into error signals in pan and tilt angle velocities. In either case, the resulting pan and tilt angle velocity errors are input to a PID (proportional, integral, derivative of error) controller for the camera motors.

Here we describe the relevant mathematics, the software, and the experiments. This report is fairly self contained, but could well be read in conjuntion with [Brown and Rimey, 1988], which it sometimes corrects and certainly expands upon.

2 Transform Basics

If $\vec{x}$ is a vector denoting a point in LAB coordinates, and A and B are transforms, then B $\vec{x}$ gives the coordinates of ("is") the point, in LAB coordinates, that results from rotating and translating $\vec{x}$ by B. AB $\vec{x}$ is the point resulting from applying B to x, then A to the result, where rotating and translating by A is done with respect to the original LAB coordinate system. Alternatively, AB $\vec{x}$ means applying to $\vec{x}$ the transformation A, followed by the transform B *expressed in the frame A*. If 1 is the link to LAB, then the transformation A_n in the matrix chain $A_1\ A_2 \cdots A_{n-1}\ A_n$ is conceptualized as taking place in the coordinate system induced by all previous movements, $A_1\ A_2 \cdots A_{n-1}$. The resulting coordinate system is $A_1\ A_2 \cdots A_n$. More detail on transform basics can be found in [Paul, 1981].

2.1 Matrices, Coordinate Systems, Transforms, and Definitions

These definitions are not quite the same as those of [Brown and Rimey, 1988].

Rot_x(a)	Rotation around the x-axis by angle a.
Rot_y(a)	Rotation around the y-axis by angle a.
Rot_z(a)	Rotation around the z-axis by angle a.
Precision Points	Six angles defining the rotations of the robot joints.
(O, A, T)	"Orientation, Altitude, and Twist" angles describing the orientation of TOOL axes in terms of LAB. Like Euler angles but not: see Section 2.2 in [Brown and Rimey, 1988].
Loc	(X, Y, Z, O, A, T), same as compound transformation in this document. A generic location ((X, Y, Z) position and (O,A,T) orientation) used by VAL. May be relative.
Head	(ϕ, θ_L, θ_R), Camera rotations defining the head configuration. ϕ is shared tilt, θs are individual pan angles.
NULLTOOL	VAL's default value for TOOL. It corresponds to a relative location of (X, Y, Z, O, A, T) = (0, 0, 0, 90, -90, 0).
TOOL	A user-definable coordinate system rigidly attached to joint 6 of the Puma.
FLANGE_TOOL	Compound transformation = (0, 0, 0, -180, 0, -90).
JOINT_6_FLANGE	CS with TOOL of VAL set to FLANGE_TOOL (see JOINT_6_FLANGE).
FLANGE	CS differing from JOINT_6_FLANGE by a translation (TOOL_X_OFFSET, TOOL_Y_OFFSET, 0) (see FLANGE and Section 2.2).

2.2 JOINT_6_FLANGE and FLANGE

JOINT_6_FLANGE

The head is rigidly attached to the sixth robot link. When the eyes are facing "forward" (Head = $\vec{0}$), JOINT_6_FLANGE is a coordinate system whose axes are oriented to be consistent with

the camera imaging model. That is, in JOINT_6_FLANGE, Z is out along the direction the head is facing (parallel to the optic axis of cameras if Head = $\vec{0}$). Y is "down", increasing with the row number addresses of pixels in an image, and X is "to the right", increasing with the column numbers of pixel addresses in the image. The origin of JOINT_6_FLANGE is the same as the origin of the compound transformation describing the robot's configuration. JOINT_6_FLANGE in this technical report is the same as FLANGE in [Brown and Rimey, 1988].

FLANGE

FLANGE differs from JOINT_6_FLANGE by a translation. Thus the FLANGE of this report differs from that of [Brown and Rimey, 1988]. To transform a point p_{j6f} from JOINT_6_FLANGE to a point p_f in FLANGE, we make use of the following matrix equation,

$$p_f = \text{TRANS(X,Y,Z)}\ p_{j6f},$$

where X = -68.95mm (= - TOOL_X_OFFSET), Y = 149.2mm (= - TOOL_Y_OFFSET), Z = 0mm (= - TOOL_Z_OFFSET).

3 Inverse Kinematics

3.1 Inverse Kinematics: O,A,T from TOOL

The mathematics in [Brown and Rimey, 1988] Section 9 was checked using the FRAME command in VAL, which allows us to compare VAL's idea of the current O,A,T with the value we expect from the work in Section 9. Happily this mathematics checks out correctly.

3.2 Inverse Kinematics: ((ϕ, θ) from (x, y, z))

Section 10 of [Brown and Rimey, 1988] derives ϕ, θ) from (x, y, z), but there is a bug in the formulation. The problem is that the original equations describe a transformation into a coordinate system that is neither JOINT_6_FLANGE nor FLANGE – the proper offsets are missing. In this section we compute the answer for FLANGE. In Section 3.4, a new EYE coordinate system is defined for which the original equations are correct.

The equations to transform (x, y, z) to ϕ and θ are:

$$\phi = atan2(-y, z) - \arcsin(\frac{d}{h}) \quad (1)$$

$$\theta_L = atan2(x - LEFT\_OFFSET, z\cos(\phi) - y\sin(\phi)) \quad (2)$$

$$\theta_R = atan2(x - RIGHT\_OFFSET, z\cos(\phi) - y\sin(\phi)) \quad (3)$$

where ϕ, θ_L, and θ_R are the pitch angle, the left yaw angle, and the right yaw angle respectively (refer to Section 2 for definitions of LEFT_OFFSET and RIGHT_OFFSET).

3.3 Transforming a point in LAB to FLANGE

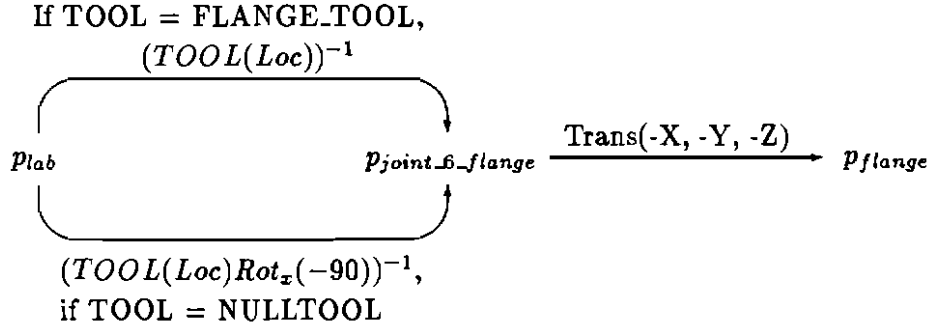
Transforming a point p_{lab} in LAB to a point p_{flange} in FLANGE requires the following transformation equations.

$$p_{joint\_6\_flange} = (TOOL(Loc))^{-1} p_{lab} \quad (4)$$

$$p_{flange} = Trans(-X, -Y, -Z) p_{joint\_6\_flange} \quad (5)$$

where $p_{joint\_6\_flange}$ is the point defined in JOINT_6_FLANGE, X = TOOL_X_OFFSET, Y = TOOL_Y_OFFSET, Z = ZERO, and TOOL on the VAL is set to FLANGE_TOOL.

These equations can be visualized using the following transformation diagram.



In the diagram, following an arrow means *premultiplying* the point at the tail of the arrow by a homogeneous matrix to derive the point at the head of the arrow.

Eq. 4 brings p_{lab} from LAB to TOOL (= JOINT_6_FLANGE) and assigns the resulting point to $p_{joint\_6\_flange}$. Eq. 5 brings $p_{joint\_6\_flange}$ from JOINT_6_FLANGE to FLANGE and assigns the resulting point to p_{flange}. Then we can use Eqs. 1 – 3 to calculate what the camera angles should be in order to move the camera motors to look at the point expressed in FLANGE with compound transformation Loc.

3.4 The Full Jacobian Calculation

We acknowledge Jeff Schneider at this point for his initial foray into the Jacobian solution, from which we learned much about the problem and its solution via Mathematica. So far the static relationships between robot head positions and eye positions have been described. This section derives the equations that relate robot head velocities to eye velocities. That is, here we calculate the Jacobian of eye pitch (ϕ) and yaw (θ) versus x, y, z, o, a, t of the robot head. We can use the Jacobian to keep the eyes fixed on a specific point while the robot makes arbitrary moves. This is like using a robotic "proprioceptive" sense that reports robot position (not velocity or acceleration). The variables in the Jacobian are the fixation point and the current robot head position. With

these variables set, the eye motor velocities necessary to hold the eyes fixed on the target are the matrix product of the Jacobian and the head velocity:

$$\begin{bmatrix} \phi' \\ \theta' \end{bmatrix} = \begin{bmatrix} \frac{\partial \phi}{\partial x} & \frac{\partial \phi}{\partial y} & \frac{\partial \phi}{\partial z} & \frac{\partial \phi}{\partial o} & \frac{\partial \phi}{\partial a} & \frac{\partial \phi}{\partial t} \\ \frac{\partial \theta}{\partial x} & \frac{\partial \theta}{\partial y} & \frac{\partial \theta}{\partial z} & \frac{\partial \theta}{\partial o} & \frac{\partial \theta}{\partial a} & \frac{\partial \theta}{\partial t} \end{bmatrix} \begin{bmatrix} x' \\ y' \\ z' \\ o' \\ a' \\ t' \end{bmatrix}$$

where the primes indicate velocities.

Mathematica [Wolfram, 1988] was used to do most of the math in this section, which is based on the notation of Craig [Craig, 1986]. First, it is necessary to define the notation that is used here.

ϕ - tilt, pitch, or up/down motion of the robot eyes

θ - yaw, or left/right motion of the robot eyes

$\vec{loc} = \begin{bmatrix} xyzoat \end{bmatrix}$ - position and orientation of the robot head in LAB coordinates

The position of the robot head refers to the value returned to the user by VAL when its position is requested. It represents the transformation matrices A_1 through A_7 [Brown and Rimey, 1988]. For the Jacobian calculated here, TOOL = NULLTOOL, hence according to section 5 of [Brown and Rimey, 1988], $A_8 = Rot_x(-90)$

$\vec{l} = \begin{bmatrix} x_{lab} y_{lab} z_{lab} \end{bmatrix}$ - position of visual target in LAB CS.

$\vec{e} = \begin{bmatrix} x_e y_e z_e \end{bmatrix}$ - position of visual target in EYE CS.

The definition of the EYE coordinate system is *not* FLANGE or JOINT_6_FLANGE. This section uses Fig. 2 and 3 from [Brown and Rimey, 1988]. Modified versions are reproduced here (Figs. 1, 2 and 3)

As noted earlier, there are some mistakes in [Brown and Rimey, 1988]. In particular Section 10 of [Brown and Rimey, 1988] states that the given target point is in FLANGE coordinates. It is actually in EYE coordinates as defined in Fig. 2 and Fig. 3. For the Jacobian calculations we have chosen to use EYE coordinates rather than the JOINT_6_FLANGE coordinates of Sec. 3.2.

The position of the robot head and the target are given in LAB coordinates. The first step is to transform the target from LAB coordinates to EYE coordinates. In order to do this, the entire transform chain from A_1 to A_{10} must be inverted. Note that the first eight transform a point into JOINT_6_FLANGE coordinates and the last two make the change to EYE coordinates. The equation for the first seven is given in Section 7 of [Brown and Rimey, 1988], which gives TOOL coordinates. Again, there is some confusion. "alias" and "alibi" should be exchanged in the sentences on either side of the equation. Here we name the five rotations in the original equation r_1 to r_5 and the translation tr_1. Therefore, for TOOL = NULLTOOL:

$r_1 = Rot_z(-90)$

$r_2 = Rot_y(90)$

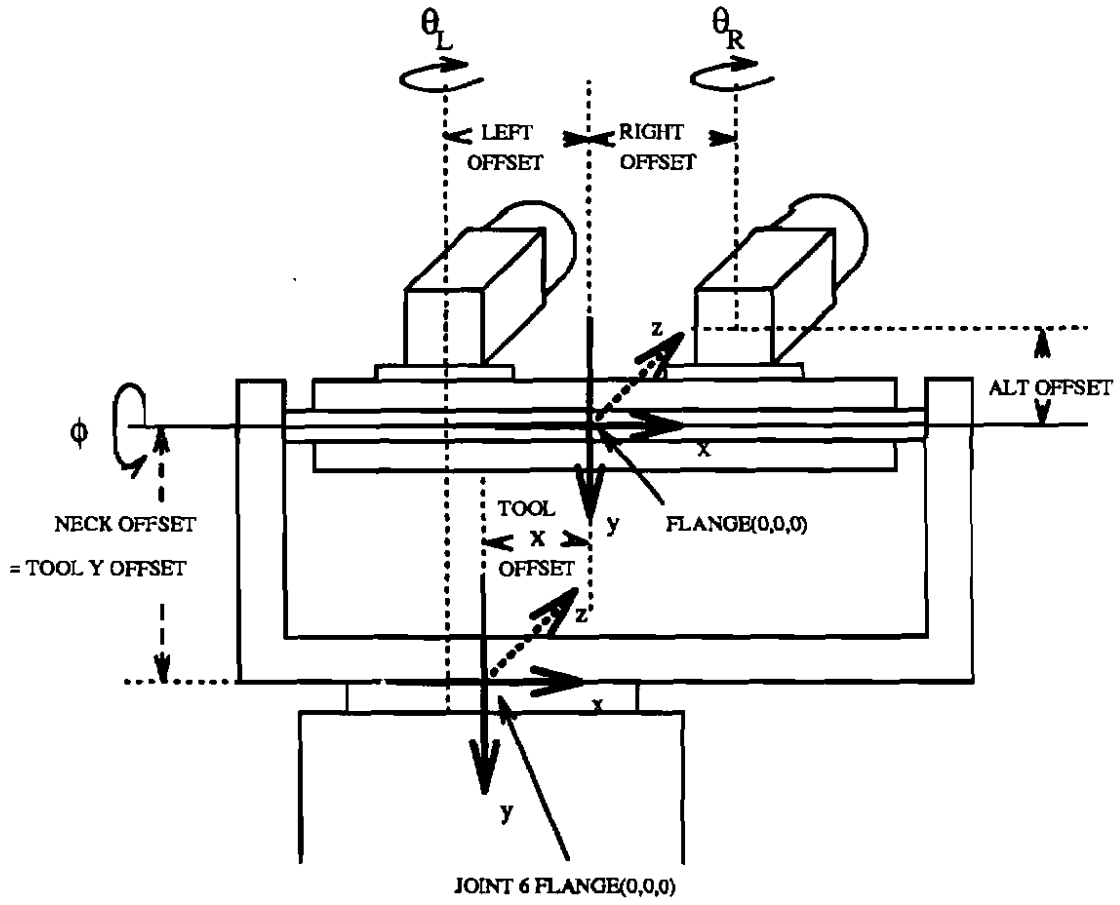


Figure 1: Axes and link offsets in the robot head

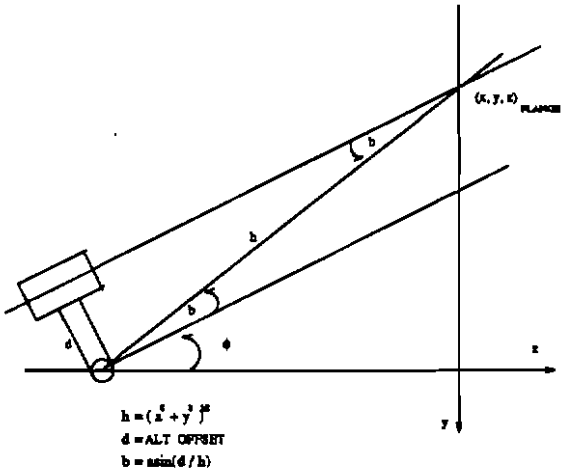


Figure 2: Distances and angles for computing the ϕ to aim at a point $\vec{x}$

$r_3 = Rot_x(-o)$

$r_4 = Rot_y(a)$

$r_5 = Rot_z(t)$

$tr_1 = Trans(x, y, z)$

$r_6 = A_8 = Rot_x(-90)$

$tr_2 = A_9 A_{10} = Trans(LEFT\_OFFSET + TOOL\_X\_OFFSET, TOOL\_Y\_OFFSET, 0.0)$
if EYE CS = LEFT EYE CS.

$tr_2 = A_9 A_{10} = Trans(RIGHT\_OFFSET + TOOL\_X\_OFFSET, TOOL\_Y\_OFFSET, 0.0)$
if EYE CS = RIGHT EYE CS.

$$A_1 A_2 A_3 A_4 A_5 A_6 A_7 A_8 A_9 A_{10} = r_1 r_2 r_3 r_4 r_5 tr_1 r_6 tr_2 = T_{1,10}$$

To move the target from LAB to EYE coordinates this chain of transform matrices must be inverted.

$$\vec{e} = T_{10,1} \vec{l}$$
$$T_{10,1} = tr_2^{-1} r_6^{-1} tr_1^{-1} r_5^{-1} r_4^{-1} r_3^{-1} r_2^{-1} r_1^{-1}$$

Now define R to be the 3x3 rotation matrix taken from $T_{10,1}$ and refer to the jth element of the ith row as R_{ij}. Filling in the proper values and solving for $T_{10,1}$ yields:

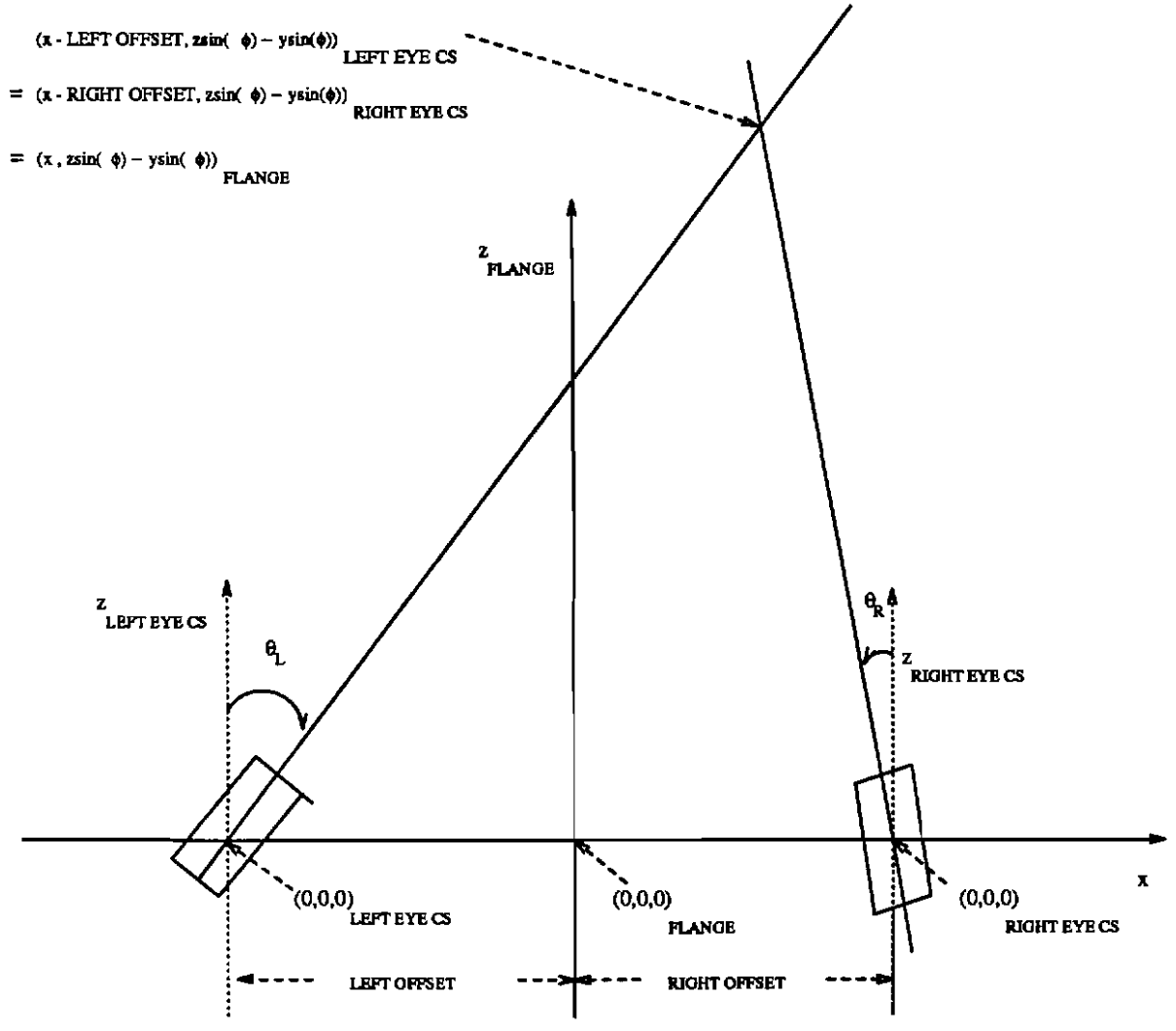


Figure 3: Distances to compute θ_L and θ_R to aim camera at $\vec{x}$.

$$R = \begin{bmatrix} -\cos(t)\sin(a)\sin(o) + \cos(o)\sin(t) & \cos(o)\cos(t)\sin(a) + \sin(o)\sin(t) & -\cos(a)\cos(t) \\ -\cos(a)\sin(o) & \cos(a)\cos(o) & \sin(a) \\ \cos(o)\cos(t) + \sin(a)\sin(o)\sin(t) & \cos(t)\sin(o) - \cos(o)\sin(a)\sin(t) & \cos(a)\sin(t) \end{bmatrix}$$

and the $T_{10,1}$ for left eye cs:

$$T_{10,1} = \begin{bmatrix} R_{11} & R_{12} & R_{13} & (-xR_{11} - yR_{12} - zR_{13} - LEFT\_OFFSET - TOOL\_X\_OFFSET) \\ R_{21} & R_{22} & R_{23} & (-xR_{21} - yR_{22} - zR_{23} - TOOL\_Y\_OFFSET) \\ R_{31} & R_{32} & R_{33} & (-xR_{31} - yR_{32} - zR_{33}) \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

To obtain $T_{10,1}$ for RIGHT EYE CS, change LEFT_OFFSET to RIGHT_OFFSET. Hereafter, we derive the equations for LEFT EYE CS only.

Using the formula above for converting $\vec{l}$ to $\vec{e}$:

$$x_e = x_{lab}R_{11} + y_{lab}R_{12} + z_{lab}R_{13} - xR_{11} - yR_{12} - zR_{13} - LEFT\_OFFSET - TOOL\_X\_OFFSET$$
$$y_e = x_{lab}R_{21} + y_{lab}R_{22} + z_{lab}R_{23} - xR_{21} - yR_{22} - zR_{23} - TOOL\_Y\_OFFSET$$
$$z_e = x_{lab}R_{31} + y_{lab}R_{32} + z_{lab}R_{33} - xR_{31} - yR_{32} - zR_{33}$$

Having completed the conversion of the target from LAB to EYE coordinates, we are ready to return to the equations for eye pitch and yaw originally given in Section 10 of [Brown and Rimey, 1988]. With the current notation and inserting the appropriate constants we obtain the following.

$$\phi = tan^{-1}(-\frac{y_e}{z_e}) - sin^{-1}(-\frac{ALT\_OFFSET}{(y_e{}^2 + z_e{}^2)^{\frac{1}{2}}})$$
$$\theta = tan^{-1}\frac{x_e}{z_e cos(\phi) - y_e sin(\phi)}$$

We now have all the necessary equations to take a robot head position and a target in LAB coordinates, and convert it into the ϕ and θ that will point the eye directly at the target point. The next step is to compute the Jacobian (determine all the partial derivatives). This was done with Mathematica. The results are included in Section 9, but are also they exist on line in 2 formats, Mathematica and C++ code. The former can be found in /u/soong/src/math/jac/RCS while the latter can be found in /u/soong/src/jac/RCS. Pointers to the rest of the code are given in Section 6.

4 Robot and Sun Communication

The Rochester robot is controlled by a robot controller, VAL II (hereafter called VAL) and the camera motors of the robotic head are controlled by the motors library on the Sun side. The homog and matrix libraries are used for matrix computations. Robotalk is a new package for VAL $\Longrightarrow$ Sun communication.

There are two software packages that perform Sun to and/or from VAL communication, probocom (a new version of robocom) and robotalk. Probocom performs Sun $\Longleftrightarrow$ VAL communication while robotalk only performs VAL $\Longrightarrow$ Sun communication. There are constant definitions conflict between probocom and robotalk, so these packages cannot be used at the same time.

Probocom is installed as a public package. The files are

/usr/vision/lib/robot/lib/libprobocom.a, and
/usr/vision/include/robot/probocom.h.

The probocom package includes several subroutines. On the VAL side, it runs as a big program on the foreground, and it does many tests to know what command is being requested. Thus it takes longer to complete one cycle. It runs at about 15 Hz if the program on the Sun *only* requests location variables from the VAL. Since probocom only sends integral values during the communication, values received on the Sun are corrected to plus or minus 0.5 unit (units are degrees or millimeters or something else, depending on what we request from VAL).

Robotalk is a local package. The files are

/u/soong/src/getloc.c,
/u/soong/src/robotalk.c, and
/u/soong/include/robotalk.h.

The robotalk package includes only 3 subroutines. robopen() and roboclose() open and close the VAL $\Longrightarrow$ communication link respectively. getloc() receives location variables (both compound transformation and precision points) from the VAL. Since there is essentially only one data transfer subroutine in robotalk, there is no test for command type. Robotalk runs at about 20 Hz on the host computer if the program only request locations from VAL and nothing else. Robotalk sends floating point values which are corrected to plus or minus 0.1 unit. Programmers may prefer robotalk if their programs require higher precision and frequency.

There are two parts of the robotalk package, one on VAL and the other on the Sun. On the VAL side, we have subroutines, RTALKINIT (which defines all the constants), SENDCOMPOUND (which sends *compound transformations* to the Sun through the serial port) and SENDPPOINT (which sends *precision points or individual joint angles*), stored in file robotalk.pg, to send location variables back to the Sun.

First type "COMM AUTO", which will calibrate the robotic arm and define all the constants needed, and then type "PCEXECUTE SENDCOMPOUND, -1", which makes SENDCOMPOUND run continuously as a *process control program* at the background. MOVE commands, issued manually or by running a VAL program in the foreground will move the robot. *Process control programs* are guaranteed to have $\geq$ 28% of the CPU time as described in the Puma manual. Thus even if the VAL is calculating where the robot should move after a MOVE command is issued, SENDCOMPOUND (or SENDPPOINT) is still sending locations back using at least 28% of the CPU time of the robot controller.

If TOOL in VAL is to be reset, type "POINT P1", which lets you define a compound transformation, and then type "TOOL P1", which defines the TOOL to the compound transformation P1.

On the Sun side, we have a subroutine, getloc() (different from that in probocom) in the robotalk package to receive location variables from the VAL.

After having received the locations and computed the appropriate velocities (Section 5) for the camera motors, the control program on the Sun sends velocity signals to the motors using set_abs_velocity() of the motors library.

5 Gaze Stabilization Without Jacobian

Keeping a high-level "efference copy" of robot commands is not enough to allow us to compensate for (cancel out) the head motion with eye motions since we do not know the control algorithms used by VAL, and the joint trajectories are unpredictable, especially around singularities [Coombs and Brown, 1990] (see also Figure 6.). To implement gaze stabilization, we used the robot communications facilities described above to query the robot for its location, which is sent back to a control program on the Sun.

Our goal is to compute pan and tilt velocities to offset general head motions. So far we have carried through the kinematics for fixed world points but not for moving world points. Using the kinematics developed so far allows a first-order approximation to the Jacobian calculation, described in this section. Section 3.4 gives the full Jacobian calculation, which specifies pan and tilt velocities to offset (x, y, z, o, a, t) velocities.

As the robot moves, FLANGE moves correspondingly. Therefore, a static target in the LAB moves with respect to the FLANGE as the robot moves. Assume we know the position of the target in LAB and the target in LAB is static (doesn't move with respect to LAB). That is, we have two compound transformations, loc_i and loc_{i+j}. We can then calculate two positions $p_{flange,i}$ and $p_{flange,i+j}$ of the target in FLANGE, using (4) and (5).

Using (1) to (3), we get the two sets of angles, ϕ_i, $\theta_{L,i}$, and $\theta_{R,i}$, and ϕ_{i+j}, $\theta_{L,i+j}$, and $\theta_{R,i+j}$.

If the robot moves continuously, then we can get the average velocity for the PITCH motor $\overline{v_P}$ by

$$\overline{v_P} = (\frac{\phi_{i+j} - \phi_i}{\delta t}) \tag{6}$$

where δt is the time required for the robot to move from loc_i to loc_{i+j}.

This average velocity is the velocity for the PITCH motor in order to gaze at the target by compensating the motion of the robot in the ϕ direction. We do the same to $\theta_{L,i+j}$, $\theta_{L,i}$, and $\theta_{R,i+j}$, $\theta_{R,i}$, and we obtain the other average velocities for the two YAW motors, $\overline{v_{Y,L}}$ and $\overline{v_{Y,R}}$.

In an implementation, the time information is obtained by a system call to Unix. However, there are quantization problems. Since the VAL program is only sending simple floating point values (corrected to plus or minus 0.1 unit) to the Sun, and since the Sun 4 system clock we use updates only every 10 milliseconds (which means the error due to the system clock is in the range of $[0, 10)$ milliseconds), the output velocities we calculate for the camera motors fluctuate significantly. The error is in this interval, not in the [-5,5] interval. When we get a time recorded from the system clock, what we get is the time updated by the clock last time. Therefore, we get the minimum error (0 msec) if we record the time immediately after the update, and we get the maximum error (very close to 10 msec) if we record the time right before the another update. Our solution is to smooth the velocities for the three camera motors using a PID-controller [Bennett, 1988]. The equations for this controller are the following.

$$m_n = K_g(K_p e_n + K_i \sum_{k=0}^{n} c^{n-k} e_k + K_d \frac{e_n - e_{n-1}}{\Delta t})$$

where

m_n	is the control output,
K_g	the global gain,
K_p	the proportional gain,
K_i	the integral gain,
K_d	the differential gain,
c	the integral decay constant,
e_n	the error at the n^{th} iteration, and
Δ t	the sampling interval.

The controller diagram is shown in Fig. 4.

6 Gaze Stabilization using Jacobian

6.1 The Algorithm

A simple stabilization implementation was done on the robot to demonstrate the accuracy and usefulness of the Jacobian.

The program is a main loop that reads robot locations, computes robot velocities, computes eye velocities, and sends them to the eye motors. There is no way for the robot to report its velocity, so an approximate velocity is determined from positional change, as in Section 5. The Jacobian is computed from the current position and target using the math from Section 3.4. Multiplying the Jacobian by the robot velocity vector gives the desired eye velocities. The desired and current actual eye velocities are given to a software PID controller in order to smooth the eye movements. The velocity returned by the PID controller is what finally goes to the eye motors (see Fig. 4).

One drawback to this approach for VOR is that all positional error is cumulative. The program only tries to set the current eye velocities such that they offset the current head velocities. As positional error accumulates from delay in the system, noise in the position readings, problems with the robot head velocity computation, and any other source, the real place that the eyes are fixed on will wander away from the original target. The I(ntegral error) term in the controller should be taking care of this problem to some extent but it is only as good as the input data, which is actually differentiated position. This points to a need to combine positional and velocity control in the stabilization controller. It would be best if position and velocity were measured by independent mechanisms (say target centroid for position, retinal slip (optic flow or motion blur) for velocity).

6.2 The Code

The source code and an appropriate makefile can be found in */u/soong/src/jac/RCS. Jac.cc* has functions that perform all the necessary math computations as described in the previous section. Before executing the program, it is necessary to start the robot and initialize it. On the VAL console use the following commands:

load robotalk (contains programs by John Soong used during setup)

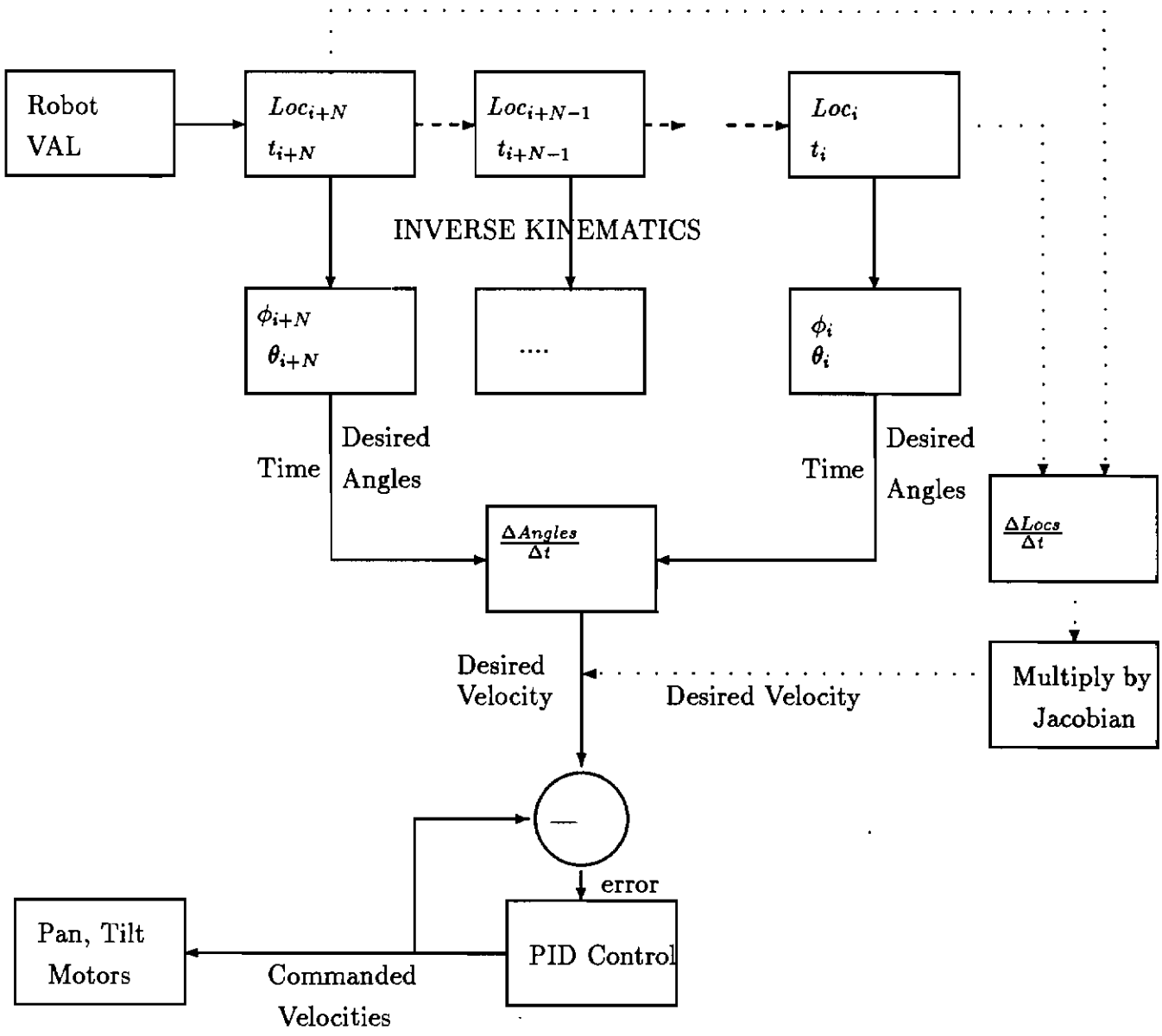


Figure 4: The control diagram for the process. Dotted lines show the Jacobian data path.

comm auto (does basic calibration, make sure there is no obstacles inside the safety circle of the robot.)

The eyes should be fixed so that they are looking straight ahead and are parallel. This can be done by manually moving them while the motors are turned off or, more accurately, by the following procedure:

VAL: ex parallel (points eyes toward two black dots on the south wall)

SUN: eyejoy (program that interactively moves the eyes)

Using eyejoy and the monitors, adjust the eyes until the image of the left dot through the left eye is at the same place as the image of the right dot through the right eye.

At this point the robot is ready. Execute the stabilizing code on the sun by typing: vor1. First it will prompt you for two sets of PID controller parameters, the first set for the pan motors and the other for the shared tilt motor (there are suitable defaults). The next prompt is for the size of the main storage arrays. The program stores every robot position that it gets from VAL. Because of this, the execution time is limited by the size of the array. Approximately 2000 locations are used for each minute of operation. Making the array circular would remove this limitation. The next prompt is for the number of previous locations that are used for computing the current robot velocity. The default is 10. There is a prompt titled "threshold". The default value is 4. The value of the threshold controls how often we send a velocity command to the motor. If we send too frequently, the input buffer for the motor controllers will overflow and will stop accepting further velocity commands. This hangs up the whole process.

Once the input parameters are set, the program will report the initial position of the robot head in LAB coordinates. Following this, it asks for the desired target position in LAB CS. From the target, it computes the necessary camera angles needed to point the camera at the target initially. If they are within the acceptable range of movement, the cameras will point to the target. The program then asks the user whether the target is acceptable. If the user responds with "no", he must choose a new target. When a satisfactory target is found and the user responds with "yes", the main loop is entered immediately and the eyes will stay fixed at the target (with some error — see Section 7.)

To test stabilization, put the robot into teach pendant mode and use the teach pendant to move the robot. As the robot head moves, the program will automatically adjust the eyes to keep them fixed on the target. A sample setup is described in the README file in directory /u/soong/src/jac/RCS. The README describes a camera geometry setup that commands the motors to fix on the 3-D spot normally occupied by a person who is standing by the console to move the robot arm with the teach pendant.

7 Experimental Results

For the experiment described here, we run a program on VAL that commands the robot head to revolve around a circular path on the zy-plane. On the host computer, we run both the position differencing and the Jacobian methods subsequently in two runs. The performances are compared.

We first input a target position for the robot to gaze at (Section 6.2), then we run the program on the host computer. In these experiments the target is at $(2140, -170, 805)$ in LAB CS. The robot arm is initially at location $(1300, -98, 780, -6.4, -90, 0)$, with TOOL = NULLTOOL during

the whole experiment. The robot arm then performs *circular* motion on the zy-plane of LAB CS with speed = 100 unit (default speed on VAL). This experimental setup is such that the target lies close to the line normal to center of the circle about which the robot revolves. VAL continuously feeds back robot locations whenever the host computer is ready. The host computer transforms a sequence of locations using either one of the methods described, to compute velocity signals, pass them through the PID controller, and sends them to the camera motors. Although it would be possible to do blob centroid analysis to get the perceived error of the resulting computations, the visual computations would slow down the sampling time of the system by a factor of two. Instead we have chosen to compare the vertical and horizontal head velocities with the pan and tilt computerd motor velocities, to which they should be very similar. Observing the image on the monitor during stabilization reveals the cameras to be pointing at the correct location to within approximately two centimeters throughout the circular path. We attribute this variation partly to delay and partly to inaccuracies of head geometry measurement.

The radius of the circular path is 150mm. The robot arm takes about 10.9 seconds to complete one revolution. That means the linear speed of the robot is about 8.6526 cm / sec. The target is about 84cm perpendicular from the center of revolution. The target is of size 5.34 x 3.8 cm^2. The region which bounds the center of the target is about 3 x 3 cm^2.

Fig. 5 and Fig. 6 are the locations and velocity of the robot calculated from the locations feedback from VAL in a run of the experiment. As we can see, the velocity of the robot doesn't form a nice circle as we should expect, and that is why we need to use schemes described in this document to implement gaze stabilization.

First consider the averaging method. Fig. 4 gives a diagrammatical explanation.

We obtain a sequence of locations from VAL. Fig. 7 is the velocity of the robot in the z-direction (We do not use this piece of infomation to compute velocities in the averaging method). Every two locations separated by ten locations in the sequence are differenced and transformed into two sets of motor angles, as described in Section 5. The interval separating the locations acts as a temporal smoother, because we want to reduce the effect of noise when the signal changes very little in magnitude. From the UNIX system clock, we obtain the time the robot takes to travel between the two locations. Eq. 6 is used for computing the motor velocities, Fig. 8. Notice that the velocity of the robot arm is transformed accurately to camera motor velocities. We use this computed velocity to obtain a PID velocity from a PID controller, Fig. 9.

For the tilt motor, (proportional gain, integral gain, differential gain, integral decay, global gain) = $(1.5, 0.1, 0.5, 0.46, 0.58)$. We use a substantial differential gain and we say why a little later in this section. We thus send the PID output velocities to the motor controllers to compensate the robot motion by moving the camera motors. Fig. 10, Fig. 11, and Fig. 12 are velocity of the robot in the y direction, velocity for the left pan *before* put it to the PID controller, and the PID output velocity for the left pan motor, respectively. For the pan motors, (proportional gain, integral gain, differential gain, integral decay, global gain) = $(1.0, 0.1, 0.5, 0.46, 0.58)$. For the same reason which will be explained later, we use a substantial differential gain. Since the graphs for the left pan motor and the right pan motor are very similar, we only present one set of them.

For the full Jacobian version, the robot performs the same circular motion as described above. We first transform the incoming sequence of locations from VAL together with the time signals from the UNIX system to a a sequence of *time derivatives* of the compound transformation, Fig. 13 is the velocity in the y direction. Then we multiply this vector of time derivatives with the Jacobian described in Section 3.4 to obtain the motor velocities to be put into the PID controller

(Fig. 14). Again the output from the PID controller, Fig. 15, is sent to the motor controllers to compensate the motion of the robot. The same sets of PID gains are used for the Jacobian module.

Although the computation of the Jacobian is much more computationally intense, both algorithms run at about 13 Hz. In simulation (no communication delay), i.e. no robot and no motors, the Jacobian takes 1.251 milliseconds to run an iteration, which is about 799 Hz, while the averaging method takes 0.312 milliseconds to run an iteration, which is about 3205 Hz. The runtime of the real process is caused by the communication delay between the host computer and VAL, and the communication delay between the host computer and the motor controllers.

Our PID gains were chosen empirically to reduce the stabilization error. The high differential component is unusual. First, This high differential gain makes the control loop, Fig. 4, very sensitive to acceleration. And this sensitive to acceleration helps the motors to catch up the motion of the robot. Of course, this differencing operation has its dangers but our data is relatively noiseless. The second reason for the particular choice of gains is somewhat obscure. In fact, with the almost-sinusoidal nature of our signal, the PID controller is implementating prediction. Fig. 16 shows that indeed the PID output anticipates the signal. To see why, remember that our signal is sinusoidal, that the integral and derivative of a sine is a cosine, and that adding sinusoids can produce a phase shift:

$$\sin(x + y) = \sin x \cos y + \cos x \sin y.$$

Thus we would not expect to realize this predictive benefit on any but sinusoidal velocities. Refer to [Brown and Coombs, 1991; Brown, 1989; Brown *et al.*, 1989; Brown, 1990a; Brown, 1990c; Brown, 1990b; Bar-Shalom and Fortmann, 1988] for more information on predictive control.

From the monitor, when we run the Jacobian module, we see that the left pan motor and the right pan motor slowly diverge from the target. This indictates the error introduced by the Jacobian for position accumulates over time. There is position information for the Jacobian module initially but not afterwards. While for the averaging method, every time we evaluate where the new target position is in the new FLANGE. Therefore we don't have cumulative error for the averaging method.

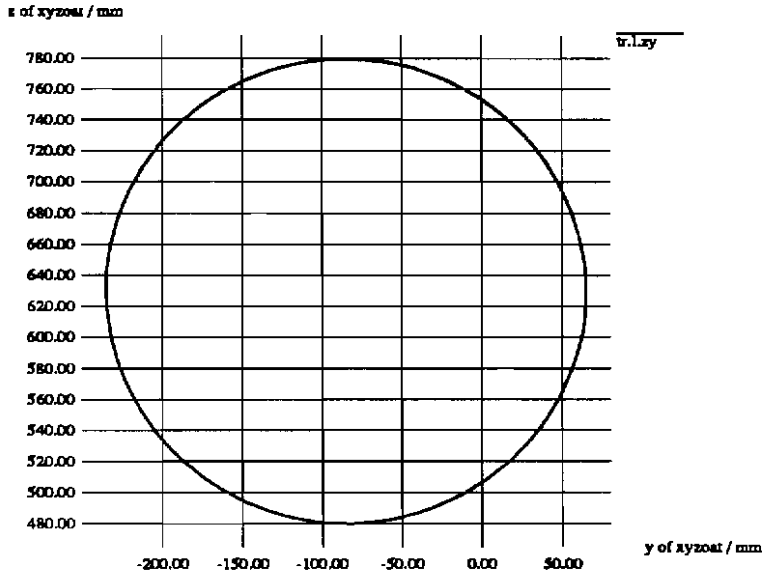


Figure 5: Robot position in Cartesian coordinates, Z versus Y.

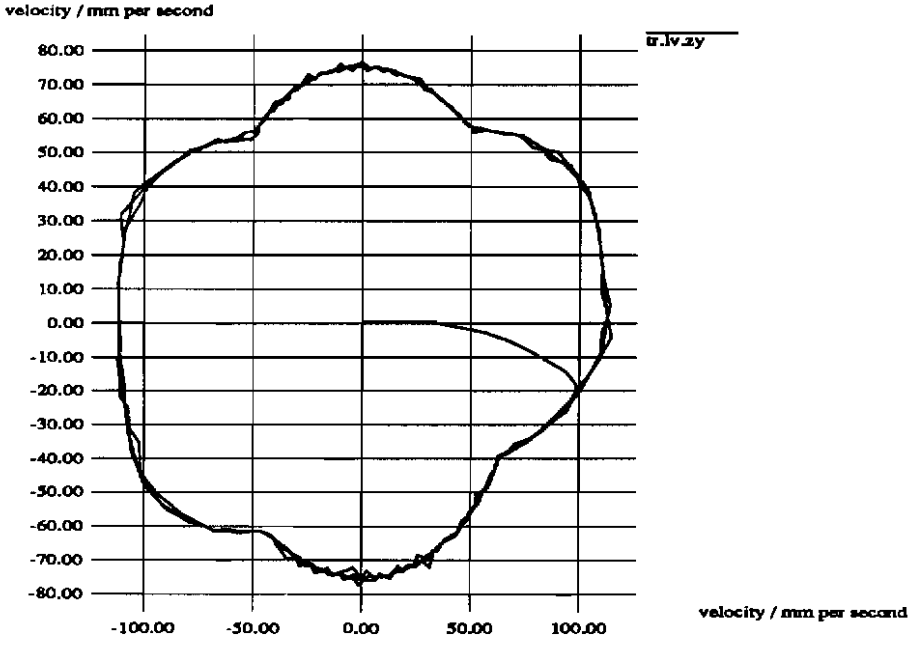


Figure 6: Velocity of the robot in Cartesian coordinates, $\frac{\delta z}{\delta t}$ verses $\frac{\delta y}{\delta t}$. Despite robot "speed" of a constant 100, the VAL controller uses non-constant velocities to implement the accurate position control shown in the previous figure.

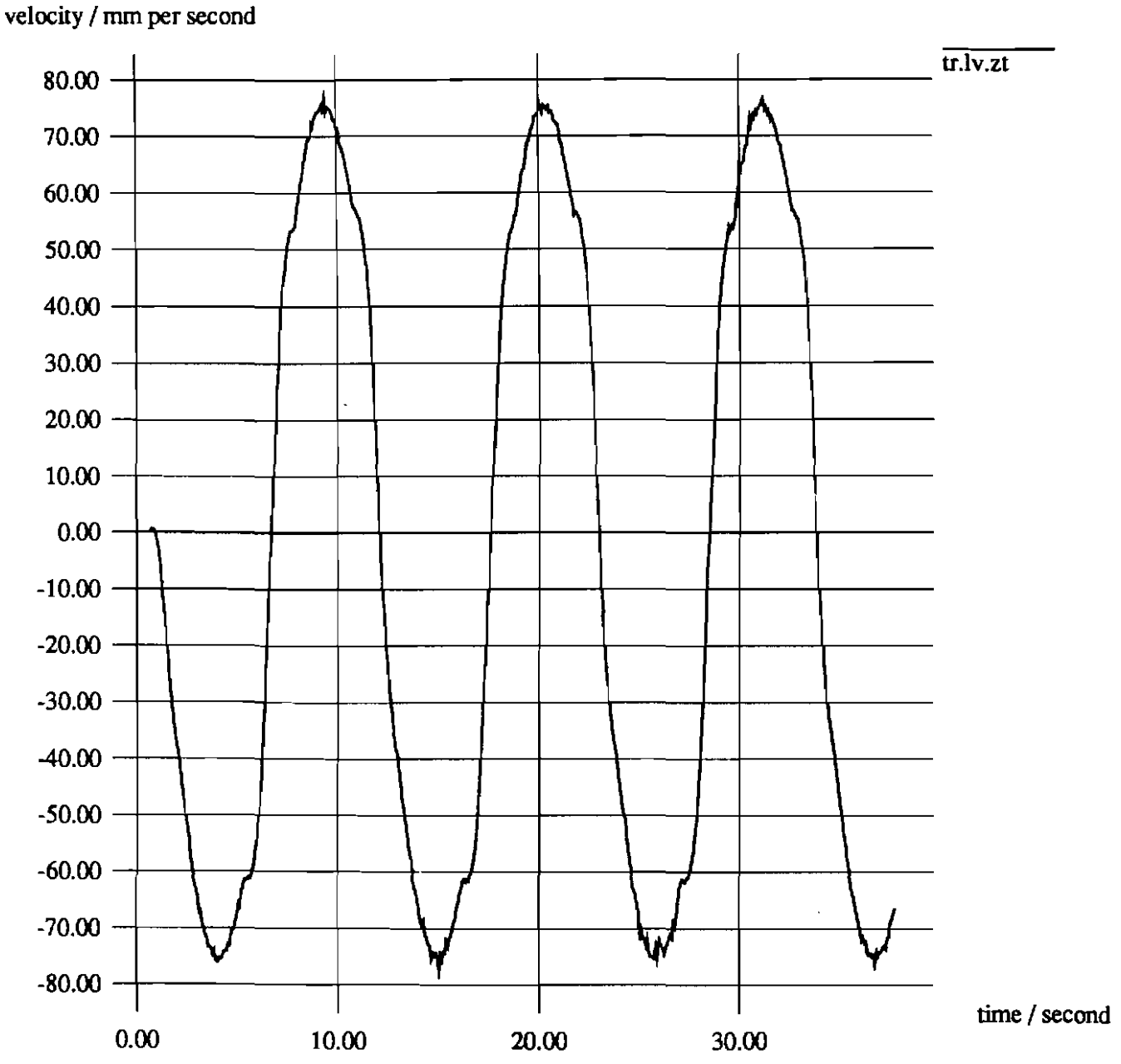


Figure 7: Averaging Method, robot velocity $\frac{\partial z}{\partial t}$ versus time

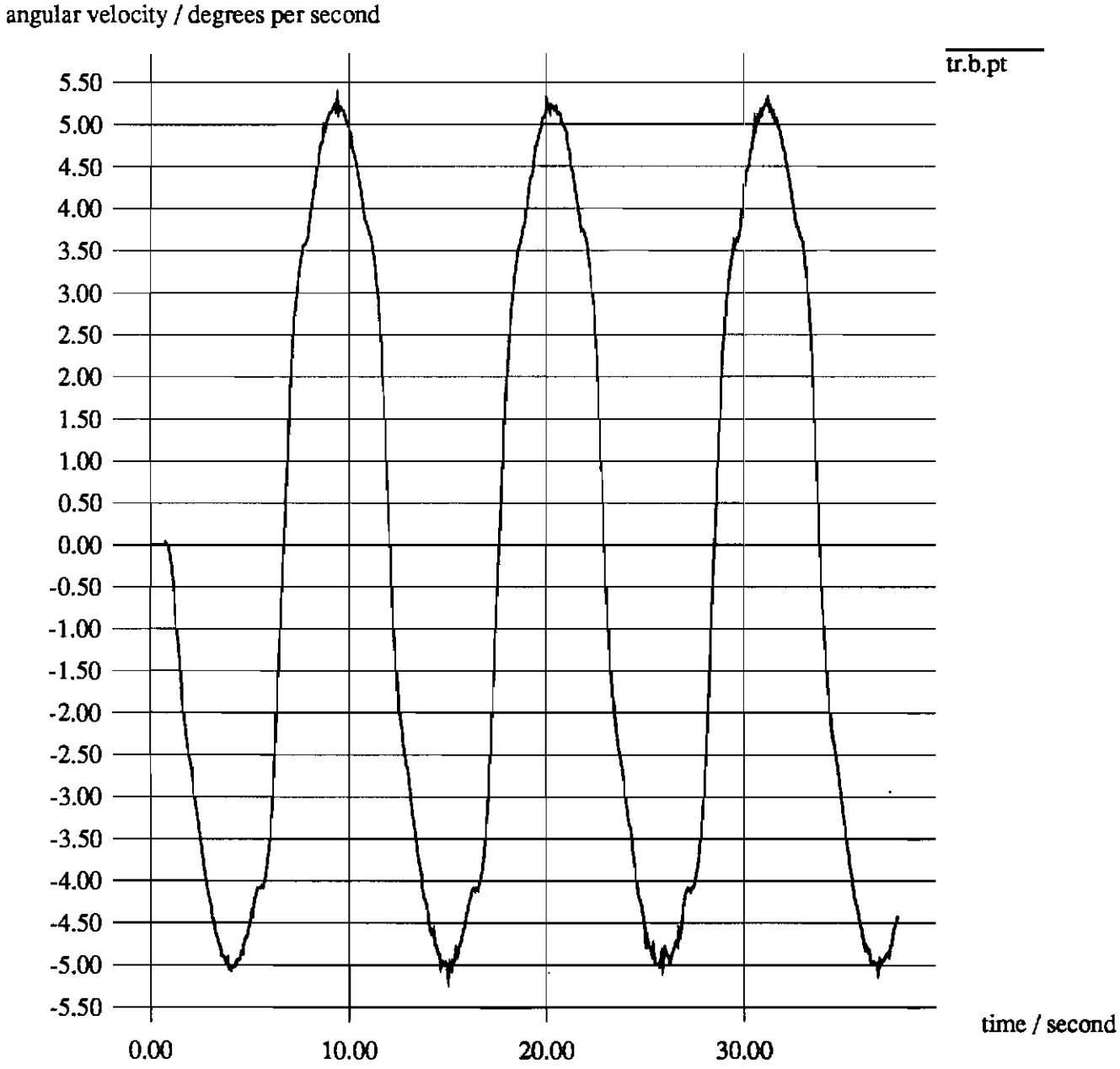


Figure 8: Averaging Method, velocity for pitch motor BEFORE PID controller versus time

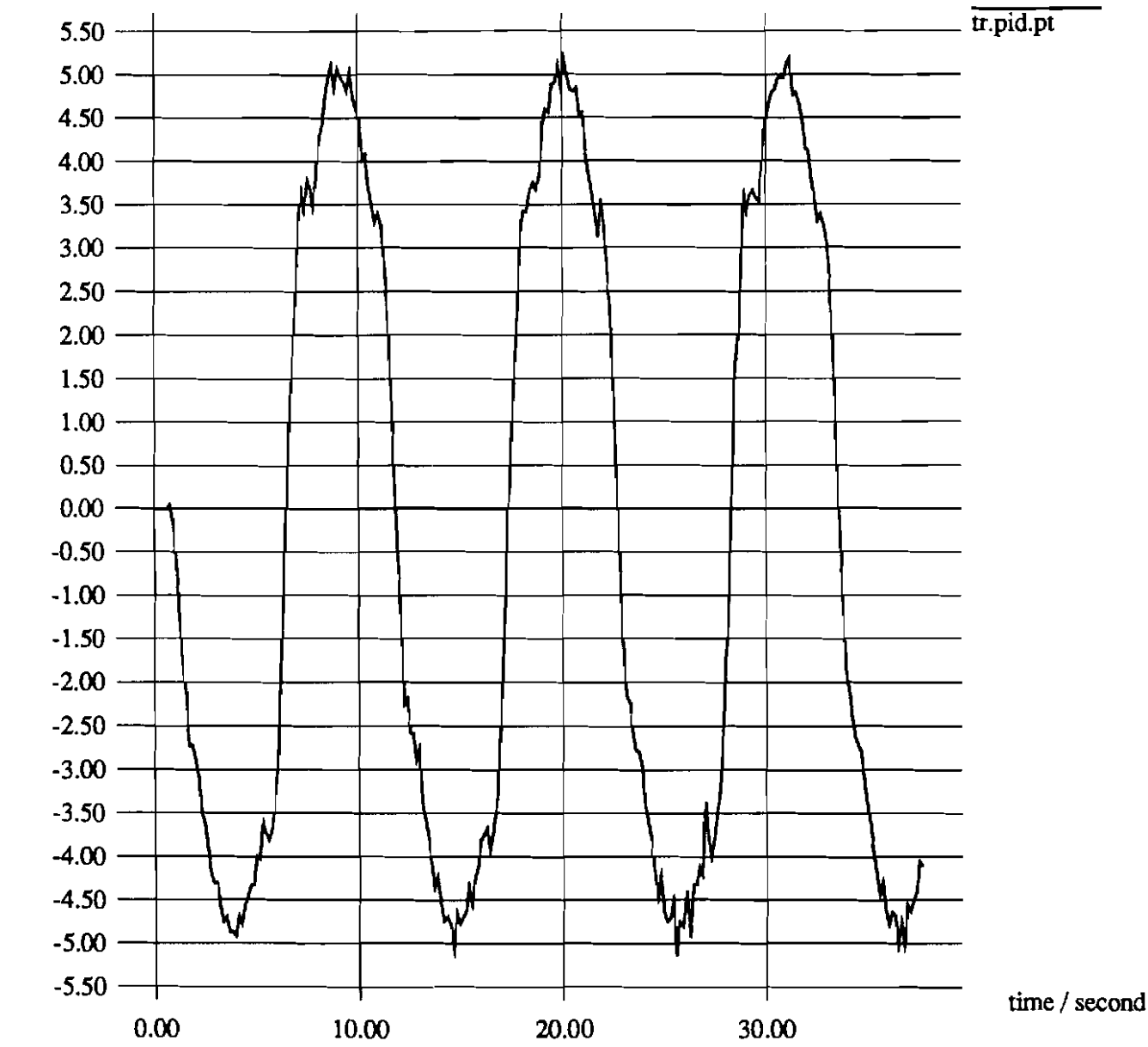


Figure 9: Averaging Method, PID velocity for the pitch motor versus time

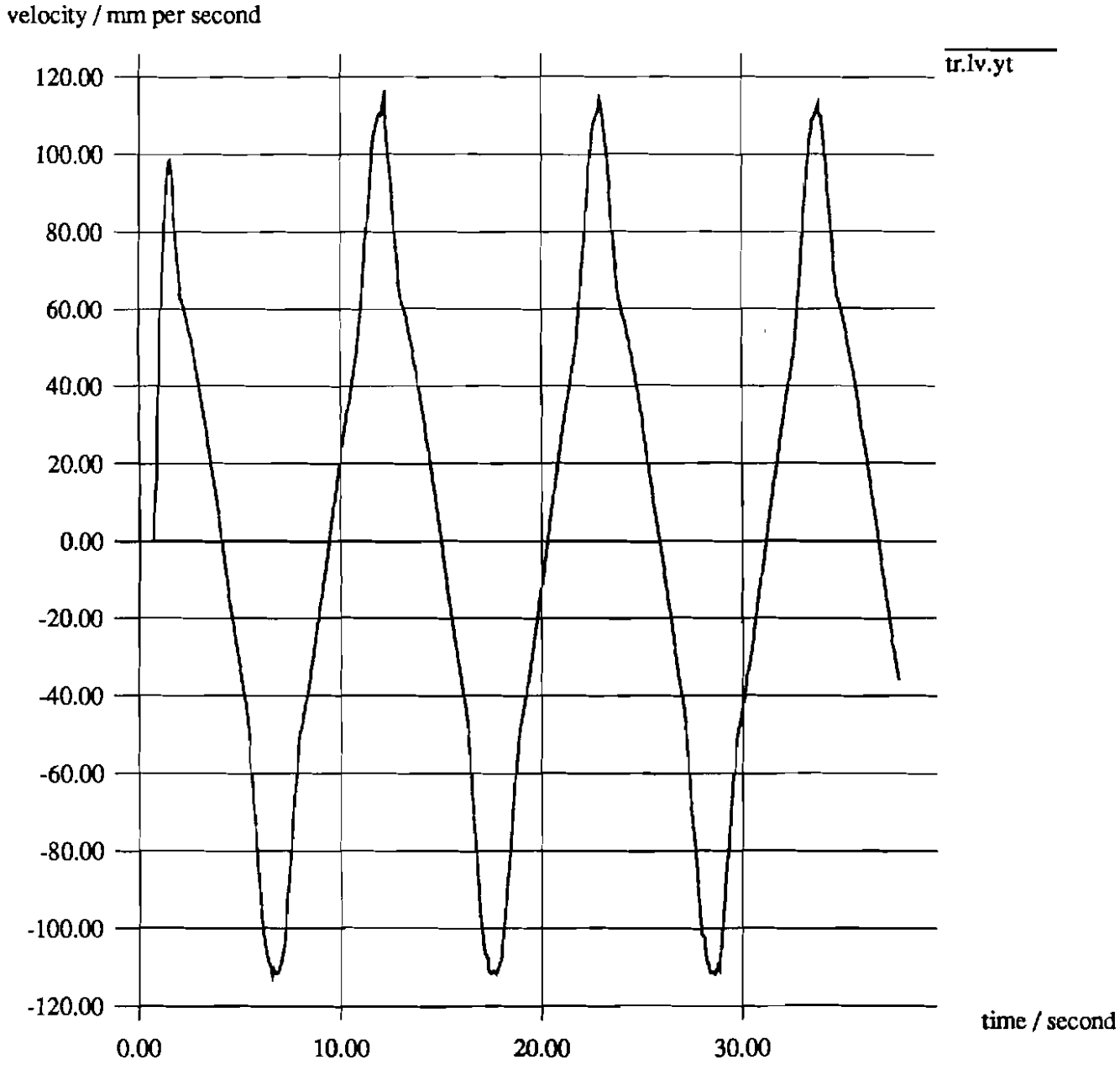


Figure 10: Averaging Method, robot velocity $\frac{\partial y}{\partial t}$ versus time

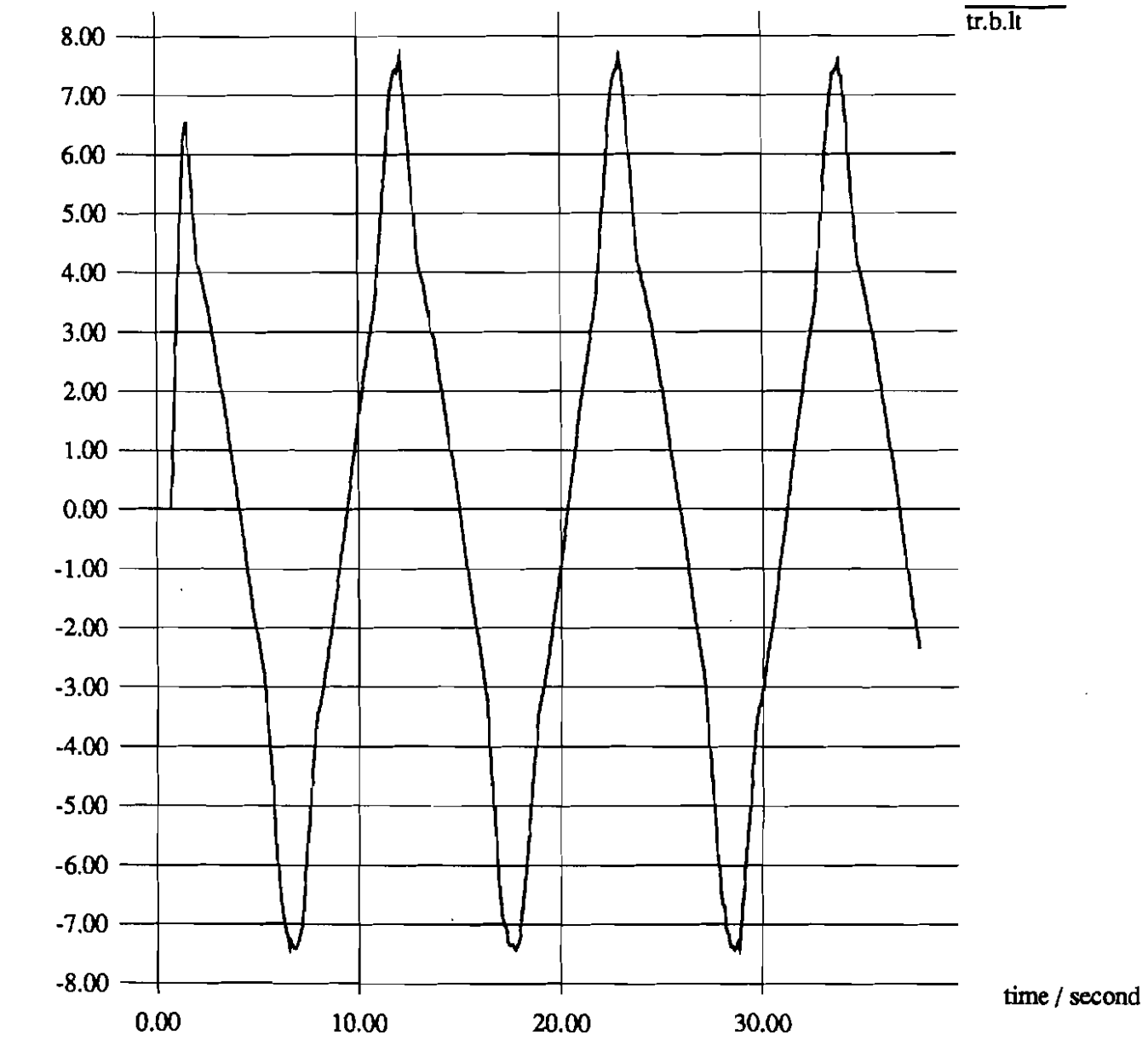


Figure 11: Averaging Method, velocity for left pan motor BEFORE PID controller versus time

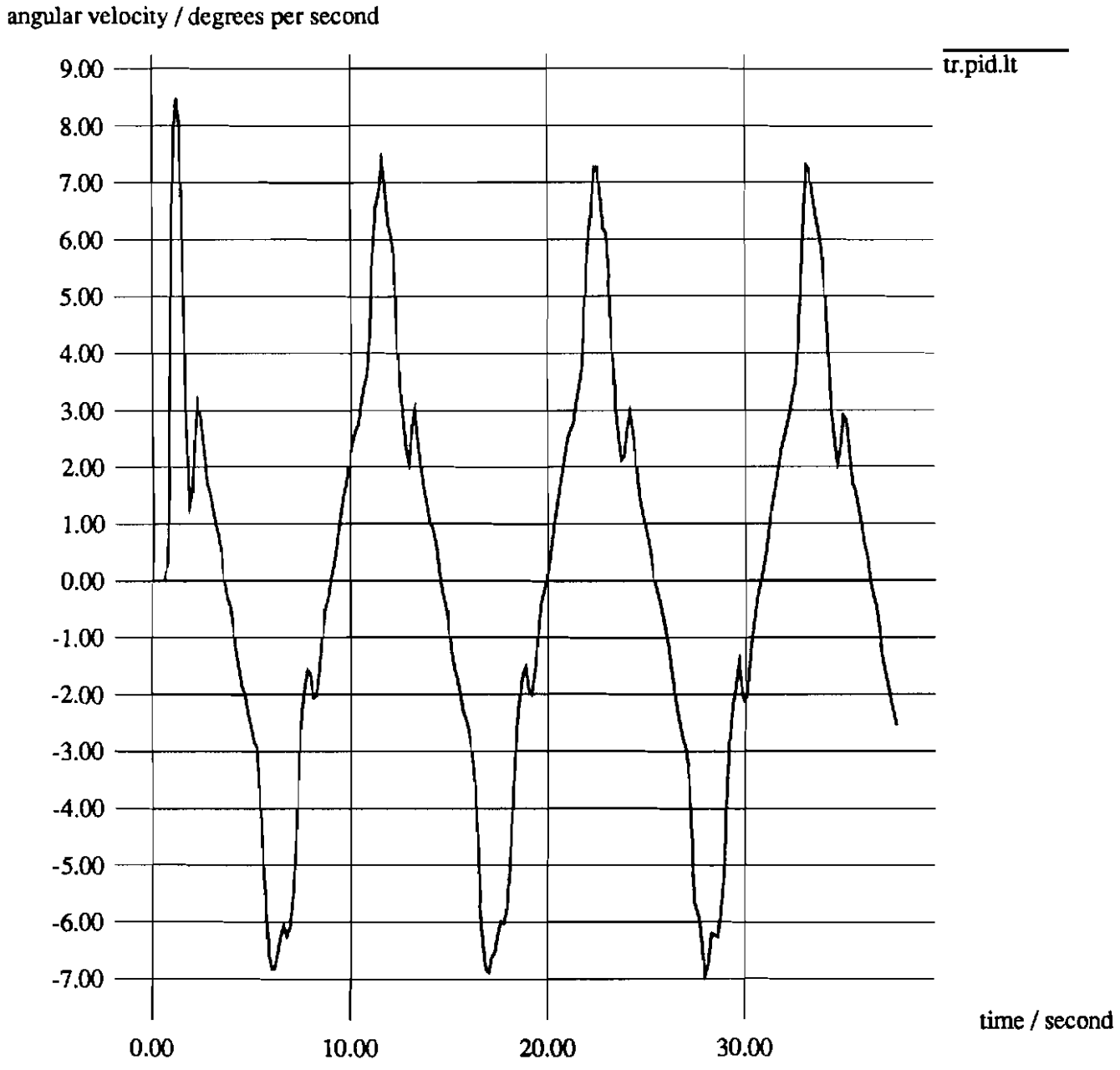


Figure 12: Averaging Method, PID velocity for left pan motor versus time

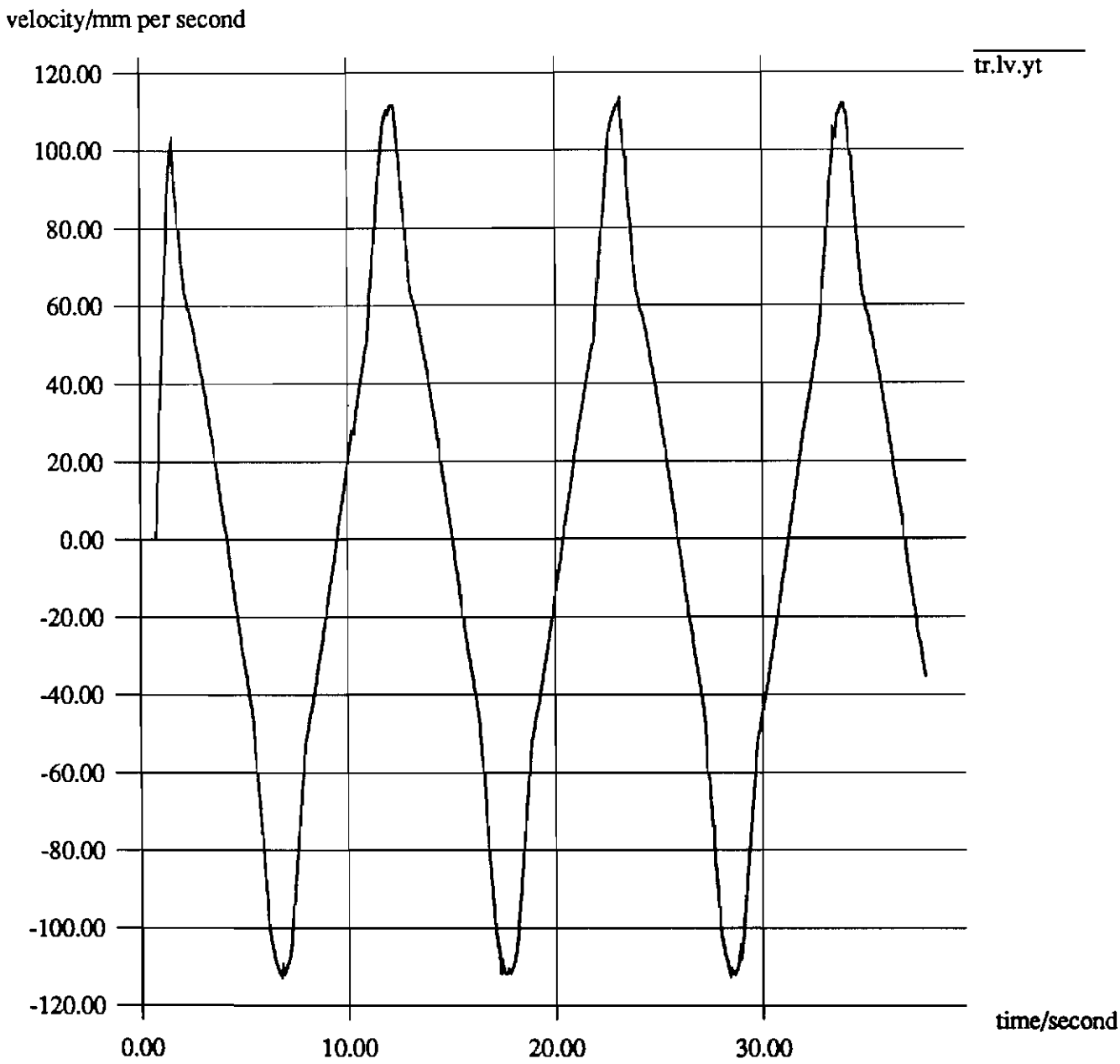


Figure 13: Jacobian, robot velocity $\frac{\partial y}{\partial t}$ versus time

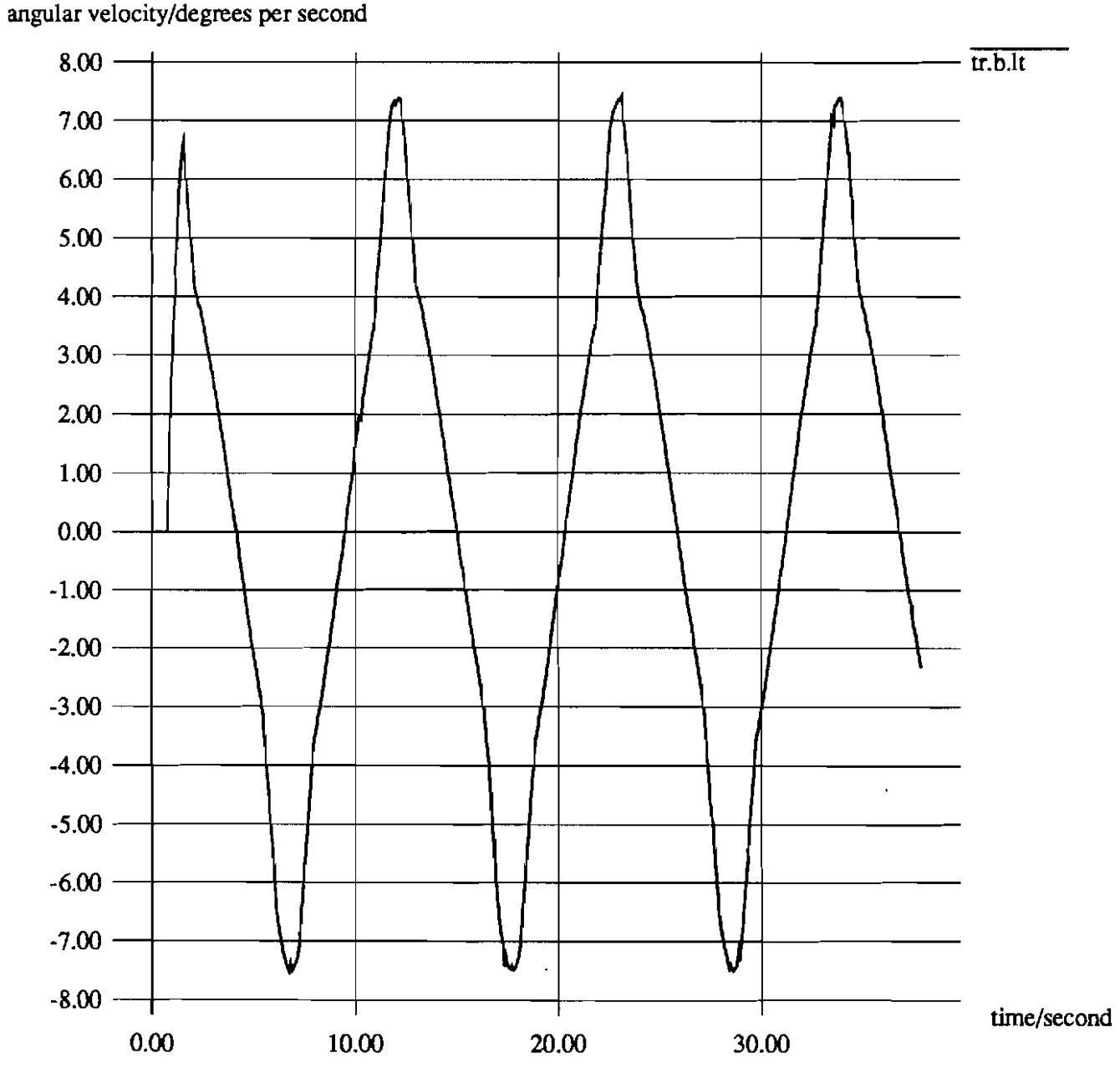


Figure 14: Jacobian, velocity for left pan motor BEFORE PID controller versus time

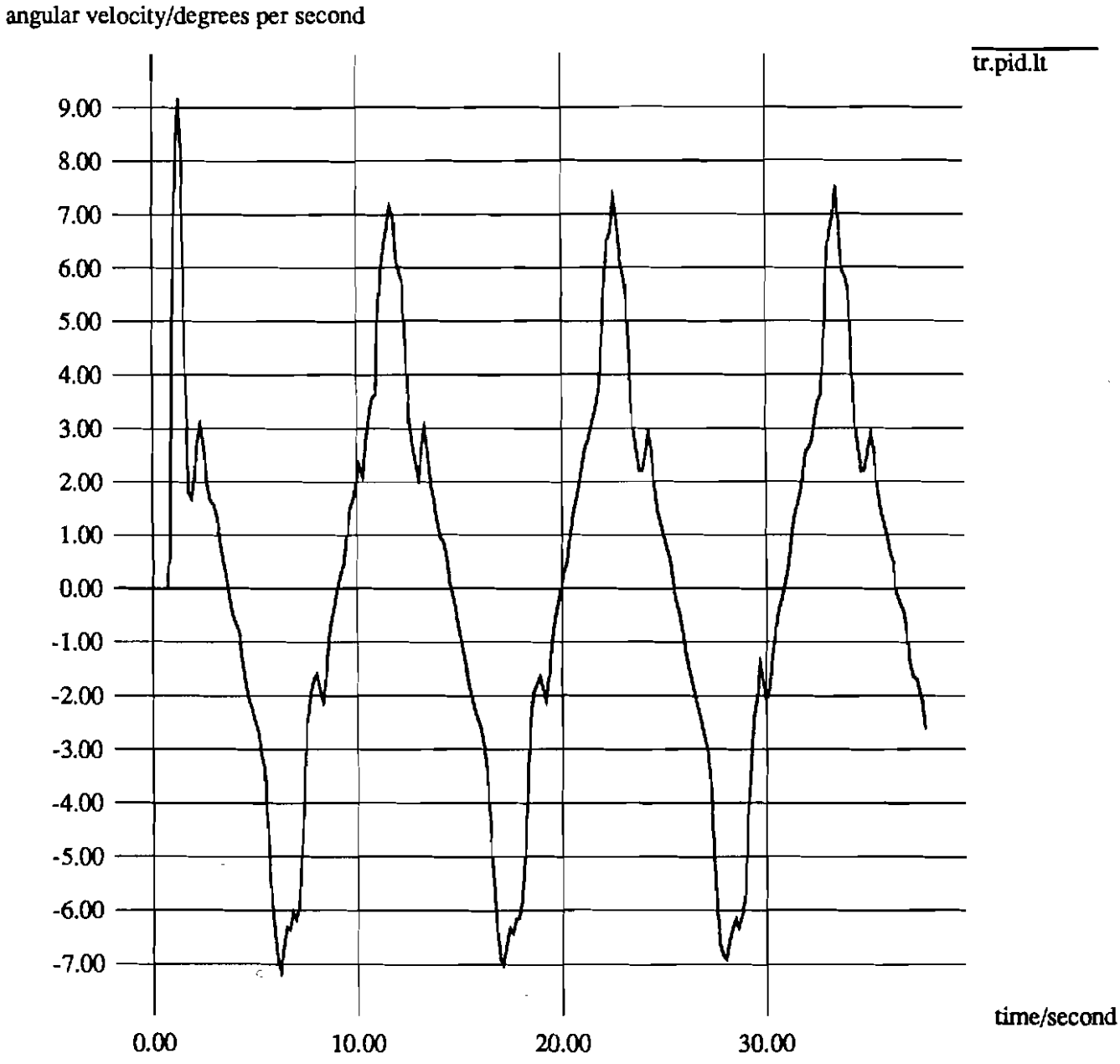
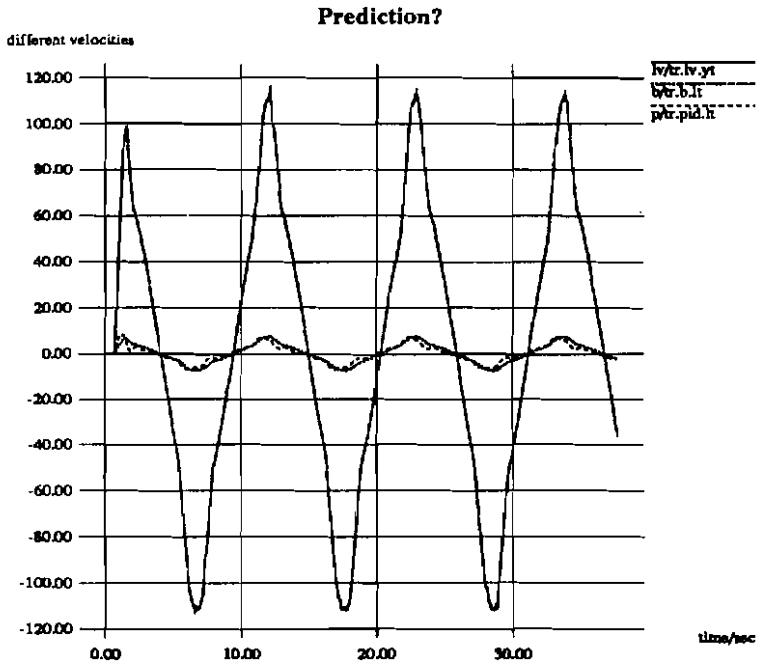
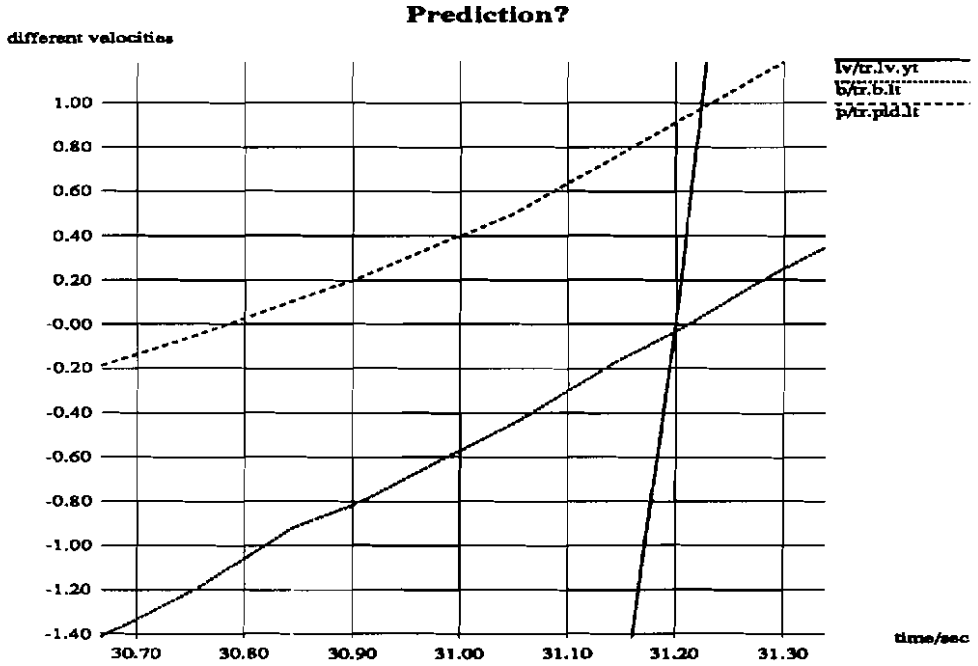


Figure 15: Jacobian, PID velocity for the left pan motor versus time



(a)



(b)

Figure 16: Averaging Method, (a) overlay of three velocity profiles, solid line — robot velocity $\frac{\partial y}{\partial t}$ versus time, dotted line — velocity for left pan motor BEFORE PID controller versus time, dashed line — PID velocity for the left pan motor versus time, (b) ZOOM at the 31^{st} sec of the 3 velocity profiles

8 Future Work

When we run either of the algorithms and observe the performance through the video output of the camera (remember we do not use any visual information), there are delays in the system which causes a shift of scene on the video output. Two common major sources of delays in these control algorithms are the *sensory delay* and the *motor delay*. The sensory delay is from the time the robot reaches a particular location to the time the sun receives the location variable for that particular location. The motor delay is from the time the sun sends a velocity signal to the camera motor to the time the camera motor actually moves at that velocity.

Delays can come from communication between the levels of software necessary to communicate with the camera controllers or with VAL, and from computational delays within control software. Our plan is to use α - β or α - β - γ filters [Bar-Shalom and Fortmann, 1988] to predict the (x, y, z, o, a, t) velocities. This technique will not account for sudden (step-function like) inputs, but once initialized they will effectively predict velocity changes that are, at the sampling resolution, effectively quadratically varying or simpler.

As mentioned, the accuracy of the signals sent from VAL using robotalk is up to 0.1 unit. In other words, we have noise in the range of [-0.1,0.1] in every component of every location feedback. For example, if the robot is currently at location (1300, 0, 1000, 0, -90, 0), and the robot moves from this location to (1300, 1000, 1000, 0, -90, 0) in one second in 1000 ticks with target at (11300, -82.55, 1214.3) (this point is set so that we can have a speed of the Pan motors move at nearly 0.1 rad per second while the Tilt motor stays stationary) in LAB, then the data set looks like the following.

TICK	X	Y	Z	O	A	T	TIME
3	1300.00	3.11	1000.00	-6.40	-90.00	0.00	0.003
4	1300.00	4.19	1000.00	-6.40	-90.00	0.00	0.004
5	1300.00	5.03	1000.00	-6.40	-90.00	0.00	0.005

We can see that noise in the range of [-0.1, 0.1] doesn't have significant effect on the Y component, because Y increments at about 1.0 mm steadily. Therefore we should be able to calculate quite clear motor velocity signals for the motor. But if we move from location (1300, 0, 1000, -6.40, -90, 0) to (1300, 0, 1000, -0.67, -90, 0) in 1 second in 1000 ticks (with velocity of about 0.1 rad / sec), then the data set looks like:

TICK	X	Y	Z	O	A	T	TIME
7	1300.00	0.00	1000.00	-6.3146	-90.00	0.00	0.007
8	1300.00	0.00	1000.00	-6.2552	-90.00	0.00	0.008
9	1300.00	0.00	1000.00	-6.3235	-90.00	0.00	0.009

Now the O component increments at about 0.000573 degrees and if noise is in the range of [-0.1, 0.1], then the output velocity signal for the left motor can span from -14 degrees / sec to 24 degrees / sec. Without noise the velocity signal should be about 5.73 degrees / second.

Therefore if we can introduce more precision to the angle components of a location, we'll have less noisy velocity signals. This can be done by altering code for robotalk both on the VAL and the Sun sides.

9 Partial Derivatives for the Jacobian

$\frac{\partial x_e}{\partial x}$ = cos(t) sin(a) sin(o) - cos(o) sin(t)

$\frac{\partial x_e}{\partial y}$ = -(cos(o) cos(t) sin(a) + sin(o) sin(t))

$\frac{\partial x_e}{\partial z}$ = cos(a) cos(t)

$\frac{\partial x_e}{\partial o}$ = y (cos(t) sin(a) sin(o) - cos(o) sin(t)) + y_{lab} (-cos(t) sin(a) sin(o) + cos(o) sin(t)) + x_{lab} (-cos(o) cos(t) sin(a) - sin(o) sin(t)) + x (cos(o) cos(t) sin(a) + sin(o) sin(t))

$\frac{\partial x_e}{\partial a}$ = -y cos(a) cos(o) cos(t) + y_{lab} cos(a) cos(o) cos(t) - z cos(t) sin(a) + z_{lab} cos(t) sin(a) + x cos(a) cos(t) sin(o) - x_{lab} cos(a) cos(t) sin(o)

$\frac{\partial x_e}{\partial t}$ = -z cos(a) sin(t) + z_{lab} cos(a) sin(t) - y (cos(t) sin(o) - cos(o) sin(a) sin(t)) + y_{lab} (cos(t) sin(o) - cos(o) sin(a) sin(t)) + x (-cos(o) cos(t) - sin(a) sin(o) sin(t)) + x_{lab} (cos(o) cos(t) + sin(a) sin(o) sin(t))

$\frac{\partial y_e}{\partial x}$ = cos(a) sin(o)

$\frac{\partial y_e}{\partial y}$ = -cos(a) cos(o)

$\frac{\partial y_e}{\partial z}$ = -sin(a)

$\frac{\partial y_e}{\partial o}$ = x cos(a) cos(o) - x_{lab} cos(a) cos(o) + y cos(a) sin(o) - y_{lab} cos(a) sin(o)

$\frac{\partial y_e}{\partial a}$ = -z cos(a) + z_{lab} cos(a) + y cos(o) sin(a) - y_{lab} cos(o) sin(a) - x sin(a) sin(o) + x_{lab} sin(a) sin(o)

$\frac{\partial y_e}{\partial t}$ = 0

$\frac{\partial z_e}{\partial x}$ = -(cos(o) cos(t) + sin(a) sin(o) sin(t))

$\frac{\partial z_e}{\partial y}$ = -cos(t) sin(o) + cos(o) sin(a) sin(t)

$\frac{\partial z_e}{\partial z}$ = -cos(a) sin(t)

$\frac{\partial z_e}{\partial o}$ = x (cos(t) sin(o) - cos(o) sin(a) sin(t)) + x_{lab} (-cos(t) sin(o) + cos(o) sin(a) sin(t)) - y (cos(o) cos(t) + sin(a) sin(o) sin(t)) + y_{lab} (cos(o) cos(t) + sin(a) sin(o) sin(t))

$\frac{\partial z_e}{\partial a}$ = y cos(a) cos(o) sin(t) - y_{lab} cos(a) cos(o) sin(t) + z sin(a) sin(t) - z_{lab} sin(a) sin(t) - x cos(a) sin(o) sin(t) + x_{lab} cos(a) sin(o) sin(t)

$\frac{\partial z_e}{\partial t}$ = -z cos(a) cos(t) + z_{lab} cos(a) cos(t) - x (cos(t) sin(a) sin(o) - cos(o) sin(t)) + x_{lab} (cos(t) sin(a) sin(o) - cos(o) sin(t)) + y_{lab} (-cos(o) cos(t) sin(a) - sin(o) sin(t)) + y (cos(o) cos(t) sin(a) + sin(o) sin(t))

$$\frac{\partial \phi}{\partial x} = \frac{y_e \frac{\partial z_e}{\partial x} - z_e \frac{\partial y_e}{\partial x} - \frac{ALT\_OFFSET(y_e \frac{\partial y_e}{\partial x} + z_e \frac{\partial z_e}{\partial x})}{(y_e^2 + z_e^2 - ALT\_OFFSET^2)^{\frac{1}{2}}}}{y_e^2 + z_e^2}$$

$$\frac{\partial \phi}{\partial y} = \frac{y_e \frac{\partial z_e}{\partial y} - z_e \frac{\partial y_e}{\partial y} - \frac{ALT\_OFFSET(y_e \frac{\partial y_e}{\partial y} + z_e \frac{\partial z_e}{\partial y})}{(y_e^2 + z_e^2 - ALT\_OFFSET^2)^{\frac{1}{2}}}}{y_e^2 + z_e^2}$$

$$\frac{\partial \phi}{\partial z} = \frac{y_e \frac{\partial z_e}{\partial z} - z_e \frac{\partial y_e}{\partial z} - \frac{ALT\_OFFSET(y_e \frac{\partial y_e}{\partial z} + z_e \frac{\partial z_e}{\partial z})}{(y_e^2 + z_e^2 - ALT\_OFFSET^2)^{\frac{1}{2}}}}{y_e^2 + z_e^2}$$

$$\frac{\partial \phi}{\partial o} = \frac{y_e \frac{\partial x_e}{\partial o} - z_e \frac{\partial y_e}{\partial o} - \frac{ALT\_OFFSET(y_e \frac{\partial y_e}{\partial o} + z_e \frac{\partial x_e}{\partial o})}{(y_e{}^2 + z_e{}^2 - ALT\_OFFSET^2)^{\frac{1}{2}}}}{y_e{}^2 + z_e{}^2}$$

$$\frac{\partial \phi}{\partial a} = \frac{y_e \frac{\partial x_e}{\partial a} - z_e \frac{\partial y_e}{\partial a} - \frac{ALT\_OFFSET(y_e \frac{\partial y_e}{\partial a} + z_e \frac{\partial x_e}{\partial a})}{(y_e{}^2 + z_e{}^2 - ALT\_OFFSET^2)^{\frac{1}{2}}}}{y_e{}^2 + z_e{}^2}$$

$$\frac{\partial \phi}{\partial t} = \frac{y_e \frac{\partial x_e}{\partial t} - z_e \frac{\partial y_e}{\partial t} - \frac{ALT\_OFFSET(y_e \frac{\partial y_e}{\partial t} + z_e \frac{\partial x_e}{\partial t})}{(y_e{}^2 + z_e{}^2 - ALT\_OFFSET^2)^{\frac{1}{2}}}}{y_e{}^2 + z_e{}^2}$$

$$\frac{\partial \theta}{\partial x} = \frac{(z_e cos(\phi) - y_e sin(\phi)) \frac{\partial x_e}{\partial x} - x_e (cos(\phi) \frac{\partial x_e}{\partial x} - y_e cos(\phi) \frac{\partial \phi}{\partial x} - sin(\phi) \frac{\partial y_e}{\partial x} - z_e sin(\phi) \frac{\partial \phi}{\partial x})}{x_e{}^2 + (z_e cos(\phi) - y_e sin(\phi))^2}$$

$$\frac{\partial \theta}{\partial y} = \frac{(z_e cos(\phi) - y_e sin(\phi)) \frac{\partial x_e}{\partial y} - x_e (cos(\phi) \frac{\partial x_e}{\partial y} - y_e cos(\phi) \frac{\partial \phi}{\partial y} - sin(\phi) \frac{\partial y_e}{\partial y} - z_e sin(\phi) \frac{\partial \phi}{\partial y})}{x_e{}^2 + (z_e cos(\phi) - y_e sin(\phi))^2}$$

$$\frac{\partial \theta}{\partial z} = \frac{(z_e cos(\phi) - y_e sin(\phi)) \frac{\partial x_e}{\partial z} - x_e (cos(\phi) \frac{\partial x_e}{\partial z} - y_e cos(\phi) \frac{\partial \phi}{\partial z} - sin(\phi) \frac{\partial y_e}{\partial z} - z_e sin(\phi) \frac{\partial \phi}{\partial z})}{x_e{}^2 + (z_e cos(\phi) - y_e sin(\phi))^2}$$

$$\frac{\partial \theta}{\partial o} = \frac{(z_e cos(\phi) - y_e sin(\phi)) \frac{\partial x_e}{\partial o} - x_e (cos(\phi) \frac{\partial x_e}{\partial o} - y_e cos(\phi) \frac{\partial \phi}{\partial o} - sin(\phi) \frac{\partial y_e}{\partial o} - z_e sin(\phi) \frac{\partial \phi}{\partial o})}{x_e{}^2 + (z_e cos(\phi) - y_e sin(\phi))^2}$$

$$\frac{\partial \theta}{\partial a} = \frac{(z_e cos(\phi) - y_e sin(\phi)) \frac{\partial x_e}{\partial a} - x_e (cos(\phi) \frac{\partial x_e}{\partial a} - y_e cos(\phi) \frac{\partial \phi}{\partial a} - sin(\phi) \frac{\partial y_e}{\partial a} - z_e sin(\phi) \frac{\partial \phi}{\partial a})}{x_e{}^2 + (z_e cos(\phi) - y_e sin(\phi))^2}$$

$$\frac{\partial \theta}{\partial t} = \frac{(z_e cos(\phi) - y_e sin(\phi)) \frac{\partial x_e}{\partial t} - x_e (cos(\phi) \frac{\partial x_e}{\partial t} - y_e cos(\phi) \frac{\partial \phi}{\partial t} - sin(\phi) \frac{\partial y_e}{\partial t} - z_e sin(\phi) \frac{\partial \phi}{\partial t})}{x_e{}^2 + (z_e cos(\phi) - y_e sin(\phi))^2}$$

10 Lab Constants

ALT OFFSET	-65.1
NECK OFFSET	-149.2
LEFT OFFSET	-85.22
RIGHT OFFSET	85.22
TOOL X OFFSET	64.00
TOOL Y OFFSET	-149.2
TOOL Z OFFSET	0.0

References

[Bar-Shalom and Fortmann, 1988] Y. Bar-Shalom and T. E. Fortmann, *Tracking and Data Association*, Academic Press, 1988.

[Bennett, 1988] Stuart Bennett, *Real-time Computer Control: An Introduction*, Prentice Hall, 1988.

[Brown, 1988] C. M. Brown, "The Rochester Robot," Technical Report 257, University of Rochester, September 1988.

[Brown, 1989] C. M. Brown, "Kinematic and 3D Motion Prediction for Gaze Control," In *Proceedings: IEEE Workshop on Interpretation of 3D Scenes*, pages 145–151, Austin,TX, November 1989.

[Brown, 1990a] C. M. Brown, "Gaze controls cooperating through prediction," *Image and Vision Computing*, 8(1):10–17, February 1990.

[Brown, 1990b] C. M. Brown, "Gaze controls with interactions and delays," *IEEE Transactions on Systems, Man, and Cybernetics, in press*, IEEE-TSMC20(2):518–527, May 1990.

[Brown, 1990c] C. M. Brown, "Prediction and cooperation in gaze control," *Biological Cybernetics*, 63:61–70, 1990.

[Brown and Coombs, 1991] C. M. Brown and D. J. Coombs, "Notes on Control with Delay," Technical Report TR 387, Department of Computer Science, University of Rochester, 1991.

[Brown *et al.*, 1989] C. M. Brown, H. Durrant-Whyte, J. Leonard, and B. S. Y. Rao, "Centralized and Noncentralized Kalman Filtering Techniques for Trackingand Control," In *DARPA Image Understanding Workshop*, pages 651–675, May 1989.

[Brown and Rimey, 1988] Christopher. M. Brown and Rimey. D. Rimey, "Coordinates, Conversions, and Kinematics for the Rochester Robotics Lab," Technical Report 259, Department of Computer Science, University of Rochester, August 1988.

[Coombs, 1991] David J. Coombs, *Real-time Gaze Holding in Binocular Robot Vision*, PhD thesis, Department of Computer Science, University of Rochester, July 1991.

[Coombs and Brown, 1990] David J. Coombs and Christopher M. Brown, "Intelligent Gaze Control in Binocular Vision," In *Proc. of the Fifth IEEE International Symposium on Intelligent Control*, Philadelphia, PA, September 1990. IEEE.

[Craig, 1986] John J. Craig, *Introduction to robotics : mechanics and control*, Addison-Wesley, 1986.

[Olson and Coombs, 1990] T.J. Olson and D.J. Coombs, "Real-time Vergence Control for Binocular Robots," In *Proceedings: DARPA Image Understanding Workshop*. Morgan Kaufman, September 1990.

[Paul, 1981] Richard P. Paul, *Robot Manipulators: Mathematics, Programming, and Control*, MIT Press, Cambridge MA, 1981.

[Wolfram, 1988] S. Wolfram, *Mathematica*, Addison-Wesley Publishing, 1988.