Control of a Cartesian Robot

Judy Franklin, Terri Noyes, Gerry Pocock

Laboratory for Perceptual Robotics
Computer and Information Science Department
University of Massachusetts
Amherst, Massachusetts 01003

The preparation of this document has been supported in part by NSF under Grant number ECS-8108818.

Abstract

The work in control applications at the Laboratory for Perceptual Robotics has been directed toward a prototype Cartesian Assembler donated to the laboratory by General Electric of Schenectady, New York. The machine and some of the hardware interfaces are described along with low level controlling schemes for point-to-point position/velocity control. An emulation of single axis controllers is shown to be an effective control method. Encoder/positional information is the basis of this low level control structure which will later be tailored for use with processors devoted to each axis. High level control issues such as adaptive learning techniques are addressed.

The prototype Cartesian Assembler (pictured in Figure 1) is a robotic manipulator which consists of three sliding (or prismatic) joint axes and a revolute joint axis. Each axis is actuated by a DC servomotor. The motor shaft for each linear axis is directly coupled to a ball screw. The ball screws for the two horizontal axes are mounted on the frame of the machine. A "carriage" which houses the vertical axis motor and screw is mounted upon the ball nuts of the two horizontal ball screws. The rotational axis motor is found at the lower end of the vertical axis.

There are two closed loop low level control systems for this machine. The first is a velocity servo-loop. This control loop is implemented completely in hardware via a set of servoamplifiers (one for each axis) which use velocity feedback from tachometers that measure the angular velocity of each motor shaft. The servoamplifiers combine the signals from these tachometers with a "desired" velocity signal received from the AAV11-A D/A converter interface on the PDP-11/23 and send an appropriate controlling signal to each axis motor. The tachometers are coupled to the motor shafts.

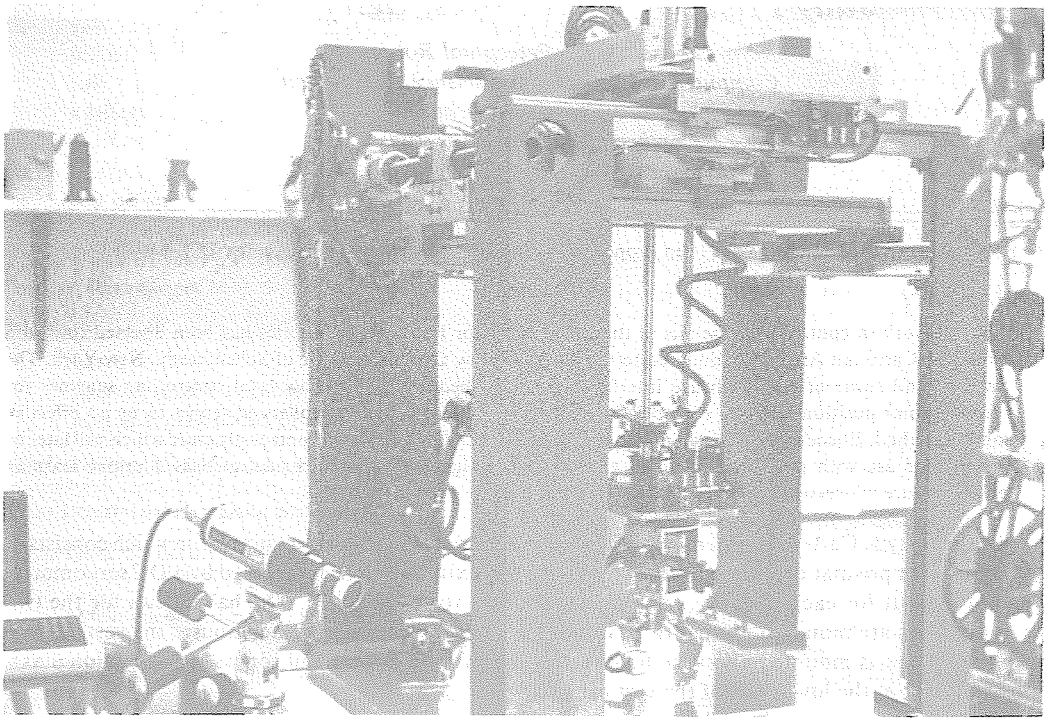
Also mounted on each motor shaft is an optical incremental encoder composed of a "slotted" disk and a lamp/phototransistor assembly. It is this encoder which provides positional feedback for the second low level control loop. The two resulting out-of-phase "sine" waves are passed through comparator circuits which square the waves and produce TTL-compatible forms. The edges (both rising and falling) of these waves are counted by a sixteen-bit binary up/down counter circuit (one for each axis). Suppose a particular encoder disk has two hundred slots. Then for one rotation of the motor shaft, each encoder output wave will complete two hundred cycles. Thus each of the square waves which reach the counter will have four hundred edges. There are two out-of-phase waves and the counter counts the edges of each one. The counter will therefore record eight hundred counts for each motor shaft rotation. Since each linear axis travels one inch per motor shaft revolution, this translates to eight hundred counts per one inch of axis travel.

The sixteen-bit output of each counter is presently sent through a multiplexer circuit and this output is sent to a DRV11 interface to a single DEC PDP-11/23. In the near future, a single (controller) processor will be devoted to each axis and the need for the multiplexer circuit eliminated. (For a more detailed description of the Laboratory for Perceptual Robotics see [1].)

At present, the individual controllers are emulated by FORTH Assembler routines running on the lab's PDP-11/23. The version used was developed at the Laboratory for Laser Energetics at the University of Rochester. One goal when constructing the low level servo routines was that they should

*Received June 1983.

Figure 1. The Prototype Cartesian Assembler
At left is the GE CID camera used for visual input. At right is a small revolute joint arm, the Rhino XR-1.



execute as quickly as possible. Since the CODE words of FORTH Assembler run at machine speed, this goal was easily attained.

The controlling signal for each axis is a velocity signal which is sent to the AAV11-A interface and delivered to the servoamplifiers. The inputs to the point-to-point position servo are the desired position, the actual position (in terms of encoder counts), and a maximum velocity allowable for each axis. Since the same controlling routine is used for all four axes, arrays indexed by register R0 of the PDP-11 were constructed for the desired position, the maximum velocity, and the current velocity being sent to the amplifiers. A state table is also used to record whether each axis has ceased to move (i.e. is within a user-chosen range of the desired position). This information is used to determine whether to position servo each axis. The code for these routines is included in the Appendix at the end of this paper and discussed below.

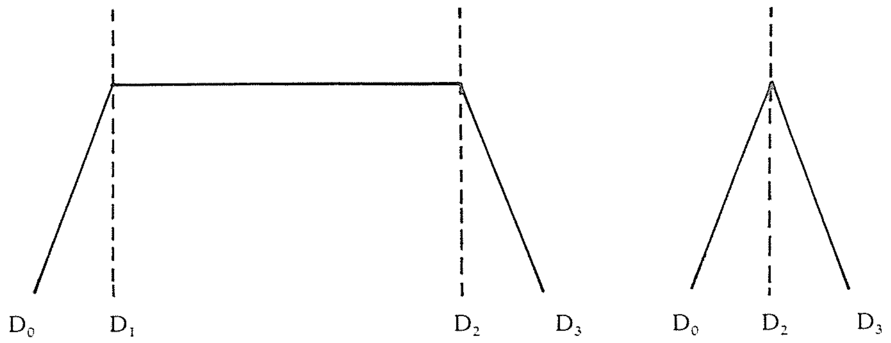
A standard speed trajectory control scheme is used with an acceleration period, a constant speed period (at the maximum velocity), and a deceleration period. The period changes do not depend on time, but on the absolute value of the difference between the desired and the actual distance. Figure 2 depicts this trajectory and also shows the trajectory which results if the distance to be travelled is relatively small.

The acceleration phase is actually a step function with a user-chosen increment (also stored in a four-element array). The control velocity is incremented at each sampling until it reaches the maximum velocity or (for a short distance) until the distance to be travelled is less than the output velocity signal. When the velocity/position relationship reaches this point (D2 in Figure 2), the velocity signal becomes proportional to the distance left to travel. In actuality, this deceleration phase is also composed of a step function since the velocity signal is reset at discrete instances.

The first step in adaptive control has been taken in the form of a routine which corrects the drift inherent in the servoamplifiers when a zero velocity is requested. If a zero is sent via the AAV11-A to the amplifier, a strictly zero velocity is not always the result, even when the amplifiers are balanced beforehand. Thus a drifting of the axis results. Part of the Change.Speed macro of blocks 1004-1007 check for such a drift using another state table called Command.Status. If it is determined via use of this table that an axis is no longer being position servoed, the routine checks to see if the axis position is still

Figure 2. Line Insertion

The velocity trajectory of the position/velocity servo. D_0 = Initial Position; D_1 = Point where Vel = Maxvel; D_2 = Point where Vel > $(D_3 - D_2)$; = Desired Position. At left is the normal trajectory. At right is the case where the distance to be travelled is so short that the maximum velocity is never reached.



within the desired range. If not, the value which (currently) represents a zero velocity is appropriately incremented or decremented by one. The same algorithm compensates for loading effects.

Similar control schemes for each axis will be implemented on PDP-11/03 systems. These single axis controllers will eliminate the need for the interrupt driver currently in use to emulate them. The code for this interrupt driver may be found in the Appendix of this paper.

The use of such interrupt routines allows the position servoing to take place while other routines are concurrently run on the FORTH level. This is necessary since both high level routines for axis movement and the low level position servo routines are being run on the same processor. The result is that the high level routines are run normally, but are interrupted at every 1/60th of a second (the system clock runs at 60Hz). The interrupt routine is the position servo routine which updates the output velocities for each of the four axes.

Routines to control the currently used two-fingered gripper which is actuated by a stepper motor have been run synchronously with the above routines.

The control algorithm given above allows only point-to-point movement of the machine. In other words, one point at a time is specified, the machine axes move to that point and stop, then the next point is specified. In a complex task, the robot will need to move continuously through various points in the work space. Continuous path motion algorithms using spline interpolation of desired trajectory segments are now being developed and are based on works such as Paul [2].

The use of tactile and visual information will be integrated with existing routines for decision-based planning and machine movement [3]. Routines which acquire such sensory information from the lab's C.I.D. camera (a solid-state, charge injection device sold by GE) and from the Overton tactile sensor [4] are running successfully. Processing of both types of sensory information is in development.

Higher level adaptive-learning control routines will also be implemented on the cartesian machine. Such routines, already written in FORTH and to be tested within the next several weeks, are based upon an Associative Search Network (ASN) designed by the Adaptive Networks Group of the University of Massachusetts [5]. The idea is to search for the correct manipulator control actions and to learn from the experience. The effect of the control signals is to move the joint variables by certain fixed amounts. The ASN receives a reinforcement which is used to determine the effects (good or bad) of its previous action(s) to accomplish learning.

Conclusion:

Some of the issues encountered in the evolution of the use of the cartesian assembler of the Laboratory for Perceptual Robotics have been discussed. Hardware feedback devices, and in particular, methods of using these devices to provide information for machine control have been developed. The implementation of these control schemes in FORTH has been a successful venture. High level control schemes such as the ASN approach are under investigation.

We would like to thank Larry Forsley and the Computer Science Team of the University of Rochester Laboratory for Laser Energetics for their continuing help in our development of a FORTH based Robot Control Language. Our thanks also go to all those at the General Electric Corporation and the Digital Equipment Corporation who have helped get the LPR started.

References

[1] Pocock, Gerry, et. al.: "System Architecture of the Laboratory for Perceptual Robotics", *Proceedings of the 1983 Rochester FORTH Applications Conference* in press.

[2] Paul, Richard P.: *Robot Manipulators Mathematics, Programming, and Control*, The MIT Press, Cambridge, Massachusetts and London, England, 1981.

[3] Noyes, Terri, et. al.: "Design of a FORTH-based Robot Control Language," *Proceedings of the 1983 Rochester FORTH Applications Conference*, in press.

[4] Overton, Kenneth J.: *The Acquisition, Processing, and Use of Tactile Sensor Data in Robot Control*, Doctoral Dissertation [to appear], COINS Department, University of Massachusetts at Amherst, 1983.

[5] Barto, Andrew G. and Sutton, Richard S.: *Goal Seeking Components for Adaptive Intelligence: An Initial Assessment*, Technical Report AFWAL-TR-81-1070, Avionics Laboratory, Air Force Wright Aeronautical Laboratories, Air Force Systems Command, Wright-Patterson Air Force Base, Ohio, 1981.

For more information on the Laboratory for Perceptual Robotics or COINS see:

Franklin, Judy A.: *Computer Interfaces and Operating Instructions for a Prototype Cartesian Robot*, COINS Technical Report Number 83-10, COINS Department, University of Massachusetts at Amherst, 1983.

Noyes, Terri: *Robot Programming Languages: A Study and Design*, COINS Technical Report Number 83-22, COINS Department, University of Massachusetts at Amherst, [to appear].

Ms. Franklin received a BA in mathematics from Clarion State College in 1980 and an MS in computer science from the University of Massachusetts in 1983. Her interests include self-replicating robots for autonomous factories and robots for space-based applications. Currently she is studying for an MA in mathematics and is working on controlling the Salisbury hand in the Laboratory for Perceptual Robotics.

Ms. Noyes graduated from the University of Massachusetts in 1983 with a BS in computer science and mathematics. She is interested in the mathematics of motion and applications for computer vision, and participated in the summer internship in robotics at MIT's Artificial Intelligence Laboratory. She is continuing her interest in robotics and sensory feedback at the Laboratory for Perceptual Robotics.

Mr. Pocock received a BA in general studies from the University of Rochester in 1981 and is in the PhD program in Computer Science at the University of Massachusetts. He is interested in robotics and neurosciences, and specifically, neural modeling, learning and motor control. He is working in the Laboratory for Perceptual Robotics and is studying adaptive arm control of the prototype cartesian robot for his PhD thesis.

Appendix

```
BLOCK 1000
( Servo Routine Variables                  GCP 5/16/83 )
4 ARRAY Desire.Pos                 ( The desired position for all 4 axes)
4 ARRAY State                      ( The state of each axis )
4 ARRAY MaxVel                     ( The maximum velocity of each axis )
4 ARRAY Velocity                   ( The current velocity of each axis )
4 ARRAY Increment                  ( Velocity increment for each axis )
4 ARRAY 0-Vel                      ( Zero Velocity output for each axis )
4 ARRAY Command.Status             ( 1=Command executing/0=Idle )
4 ARRAY Delay.Left                 ( Amount of time left for next adapt)
170440O CON VelOut                 ( Starting address of the output D/A)
167770O CON CntOut                 ( Address to write to counter )
167774O CON CntIn                  ( Address to read from counter )
      5 VAR Range                  ( Error range for stopping )
     30 VAR Delay                  ( Delay between adapt. steps )
0 CON Stop 1 CON Move              ( Internal states for servo routine)
      0 VAR Dir                    ( Direction of move)          -->

BLOCK 1001
( Servo Routine Macros                     GCP 4/7/83 )
: Save-Regs   ( <>--<>, Save used registers on return stack )
    ASSEMBLER
          RP -)      R0   MOV,  ( Save  R0 )
          RP -)       T   MOV,  ( Save  T )
          RP -)       W   MOV,  ( Save  W )
    FORTH  ;

: Restore-Regs  ( <>--<>, Restore registers from return stack )
    ASSEMBLER
              W   RP )+ MOV,  ( Restore  W )
              T   RP )+ MOV,  ( Restore  T )
             R0   RP )+ MOV,  ( Restore  R0 )
    FORTH  ;                                                     -->


BLOCK 1002
(  Stop?                                   GCP 4/7/83 )
: Stop? ( <>-<>,  Test for stop point )
    ASSEMBLER
                             R0 CLR,       ( Init. loop counter )
    BEGIN,
                          T  R0 MOV,       ( Determine table ... )
                              T ASL,       ( ...byte offset )
          W  Desire.Pos  T I) MOV,         ( Get desired position )
             CntOut  @#  R0  MOV,          ( Select axis position )
                  W  CntIn  @# SUB,        ( How far away? )
         MI IF,
                           W  NEG,         ( Get absolute value )
         THEN,                                                   -->
```

BLOCK 1003

```
( Stop? cont.                                    GCP 5/16/83 )
                  Range @# W CMP.        ( Within range of Des. Pos. )
   LT IF.                                ( Yes )
       VelOut T I) 0-Vel T I) MOV.       ( Stop Axis Movement )
            State T I) Stop # MOV.       ( Set axis state to stop )
                 Velocity T I) CLR.      ( Update velocity table )
         Command.Status T I) CLR.        ( Command completed )
   ELSE.                                 ( No )
           State T I) Move # MOV.        ( Set axis state to move )
   THEN.
                           R0 INC.       ( Increment counter )
                      4 # R0 CMP.        ( Finished? )
   GE END.                               ( Yes if greater or equal )
   FORTH ;                                              -->
```

BLOCK 1004

```
( Change.Speed?                                  GCP 4/7/83 )
: Change.Speed?  ( <>--<>. Test for speed change & do it )
  ASSEMBLER                  R0 CLR.     ( Initialize loop counter )
    BEGIN.
                          T R0 MOV.      ( Determine table ... )
                             T ASL.      ( ... byte offset )
            State T I) Stop # CMP.       ( Is axis stopped ? )
      NE IF.                             ( No )
          W Desire.Pos T I) MOV.         ( Get desired position )
              CntOut @# R0 MOV.          ( Select axis position )
                W CntIn @# SUB.          ( How far away? )
        MI IF.
                            W NEG.       ( Get absolute value )
               Dir @# -1 # MOV.          ( Set negative direction )
        ELSE.
                Dir @# 1 # MOV.          ( Set pos. dir. )   -->
```

BLOCK 1005

```
( Change.Speed cont.                             GCP 5/18/83 )
        THEN.
           Command.Status T I) TST.      ( Idle ?)
        EQ IF.                           ( Yes )
               Delay.Left T I) DEC.      ( Decrement time remaining)
          EQ IF.                         ( Ready for next adapt )
      Delay.Left T I) Delay @# MOV.      ( Reset time remaining )
           0-Vel T I) Dir @# ADD.        ( Adapt for zero offset )
          THEN.
        THEN.
             Velocity  T I) W CMP.       ( Distance left < Vel )
        LT IF.                           ( Yes )
              Velocity T I) W MOV.       ( Update velocity table )
                      Dir @# TST.        ( Which direction ?)
          MI IF.                         ( Negative direction )
                           W NEG.        ( Set for sub. )   -->
```

BLOCK 1006

```
( Change.Speed? cont.                              GCP 4/7/83 )
       THEN,
                W 0-Vel T I) ADD,      ( Add offset )
              VelOut T I) W MOV,       ( Output new vel. )
     ELSE,                             ( Distance left > Max Vel)
      MaxVel T I) Velocity T I) CMP,   ( Increase Velocity ? )
       LT IF,                          ( Yes )
    Velocity T I) Increment T I) ADD,  ( Increase Vel by Inc.)
       ELSE,
      Velocity T I) MaxVel T I) MOV,   ( Set velocity to Max )
       THEN,
             W Velocity T I) MOV,      ( Copy vel. for output )
                    Dir @# TST,        ( Which Direction ?)
        MI IF,                         ( Negative Direction ?)
                          W NEG,       ( Yes )
        THEN,                                      -->
```

BLOCK 1007

```
( Change.Speed? cont.                              GCP 5/18/83 )
                W 0-Vel T I) ADD,      ( Add Offset )
              VelOut T I) W MOV,       ( Output Velocity )
       THEN,
     THEN,
                         R0 INC,       ( Increment counter )
                   4 # R0 CMP,         ( Finished? )
  GE END,                              ( If >= to 4 )
  FORTH ;

(Extra space for expansion of routines)




                                                   -->
```

BLOCK 1008

```
( Interrupt.Servo, Install.Int.Servo, Enable & Disable Int.   )
CODE Interrupt.Servo  ( <>--<>,  Interrupt servo routine )
    Save-Regs Stop? Change.Speed? Restore-Regs RTI,
: Install.Int.Servo  ( <>--<>,  Install the servo routine )
    0 177546O !   ( Disable Clock interrupt )
    ' Interrupt.Servo 100O !  340O 102O !  (Install interrupt )
    1000 177546O !     ( Enable interrupt )  ;

0 VAR Proc.Status
CODE Restore.Processor  ( <>--<>, Restore old processor status )
              Proc.Status @# MTPS,     ( Set Processor status )
                             NEXT,     ( Sequence  )
CODE Disable.Interrupts ( <>--<>, Disable all Interrupts  )
              Proc.Status @# MFPS,     ( Save old Processor Status )
                    340O # MTPS,       ( Disable all interrupts )
                             NEXT,     ( Sequence )
                                                   -->
```

```
BLOCK 1009
( Read.Counter, Zero.Array, Abs.Move.Single    GCP 5/16/83 )
: Read.Counter ( <int1>--<int2>, read axis int1 position )
      Disable.Interrupts
      CntOut ! CntIn @
      Restore.Processor  ;

: Zero.Array   ( <array>--<>, Zero out the array )
    DUP 2DUP 4 0 DO I 2 * + 0 SWAP ! LOOP ;

: Abs.Mov.Single ( <pos,axis>--<>, Absolute single axis move )
    Disable.Interrupts
    DUP ROT SWAP 2* Desire.Pos + !  ( Set Desire Position )
    2* Command.Status + 1SET        ( Set Command Status)
    Restore.Processor  ;                       -->

BLOCK 1010
( Position and Velocity setting words               GCP 5/16/83     )
: Max.Vel.Single ( <vel,axis>--<>, Set velocity for axis )
    2 * MaxVel + ! ;

: Abs.Move ( <x,y,z,t>--<>, Absolute move on all four axes )
    0 3 DO I Abs.Move.Single -1 +LOOP ;

: Max.Vel ( <x,y,z,t>--<>, Set all four Velocities )
    0 3 DO I Max.Vel.Single -1 +LOOP ;

: Rel.Move.Single ( <Rpos,Axis>--<>, Relative single axis move )
    DUP 2* Desire.Pos + @ ROT + SWAP Abs.Move.Single ;

: Rel.Move ( <x,y,z,t>--<>, Relative move on all axes )
    0 3 DO I Rel.Move.Single -1 +LOOP ;            -->

BLOCK 1011
( Array Initialization words                        GCP 4/11/83   )
: Clear.All.Arrays   ( <>--<>, Initialize all arrays )
        Desire.Pos Zero.Array         State Zero.Array
        MaxVel Zero.Array             Velocity Zero.Array
        Command.Status Zero.Array  ;

: Initialize.Counter ( <>--<>, Initialize the counter )
      CntOut 2+ DUP 0 SWAP ! 15 SWAP ! ;

: Initialize.Zero.Velocities ( <>--<>, Init 0-Vel Array )
    40010 0-Vel !   40010 0-Vel 2+ !   40520 0-Vel 4 + !
    40000 0-Vel 6 + !   ;

: Set.Increments ( <>--<>, Init Increment Array  )
    3 DUP DUP DUP Increment ! Increment 2+ ! Increment 4 + !
    Increment 6 + ! ;                              -->
```

```
BLOCK 1012
( Set.Up.Robot, Move.Complete                     GCP 5/18/83 )
: Set.Delay ( <>--<>, Initialize Delay for Adapt. routine )
  Delay @ 4 0 DO DUP Delay.Left I 2* + ! LOOP DROP ;

: Set.Up.Robot ( <>--<>, Set up Arrays and Enable Servo Rtn. )
  Clear.All.Arrays              Initialize.Counter
  Initialize.Zero.Velocities    Set.Increments
  300 300 300 0 Max.Vel         Set.Delay
  Install.Int.Servo ;


: Move.Complete? ( <>--<>, Waits for completion of a command
                                 move on x, y, and z axes only )
  BEGIN
     3 0 DO I 2* Command.Status + @ LOOP OR OR 0=
  END ;                                                   ;S
```