An Intelligent Forth-Based Robotic Vehicle

Steven J. Formisani
AT&T Bell Laboratories
Morristown, New Jersey

Stuart D. Asakawa
Hewlett-Packard
San Diego, California

Allan K. Ronne
Watkins-Johnson
San Jose, California

Maged G. Tomeh
Harvard Business School
Boston, Massachusetts

Abstract

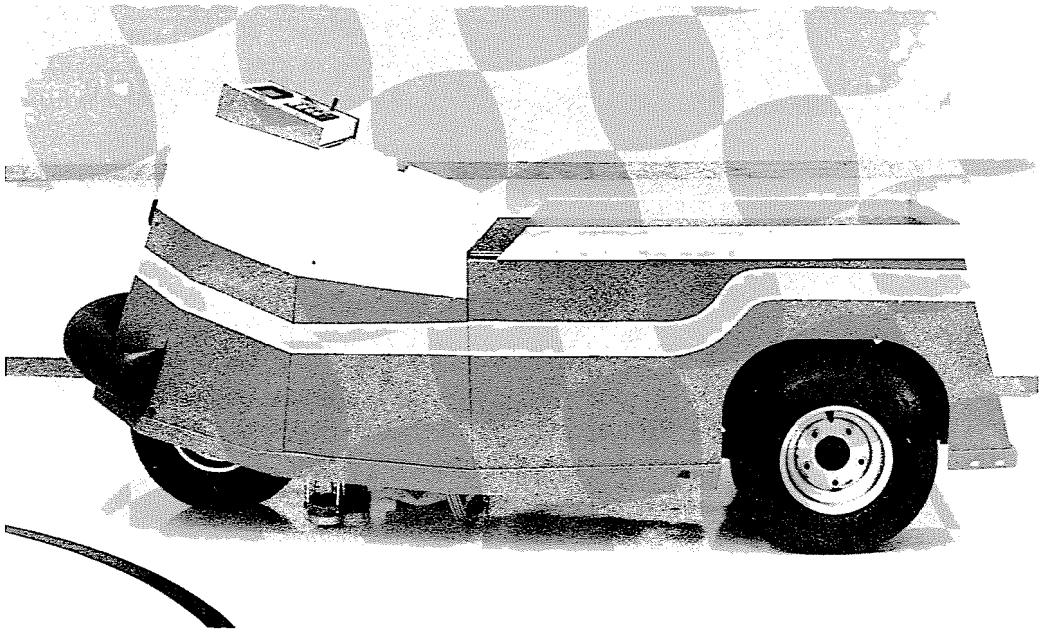
A Forth-based robotic parts delivery system was developed at Stanford University as a joint masters project. An electric personnel carrier was modified to follow a tape track from workstation to workstation, where it would stop for a preprogrammed period of time. Closed loop feedback systems were implemented to control vehicle guidance and speed, and an ultrasonic sensor was used for obstacle avoidance. The vehicle recognized thin metal plates on the floor, enabling it to be taught to wait at a workstation or change the obstacle detection range.

Introduction

The vehicle described in this paper (pictured in Photograph 1) was designed and developed by a team of four Mechanical Engineers as our masters project in the Smart Product Design Curriculum of the Mechanical Engineering Department at Stanford University. The project was presented at the 1984 Rochester Forth Applications Conference in June, and the 1984 ASME International Computers in Engineering Conference held in Las Vegas, in August. Technical descriptions of the vehicle in this paper have been taken from "Design and Development of a Microprocessor-controlled Robotic Vehicle", which was published in the proceedings of the ASME conference.[1]

The design team was confronted with the problem of transporting printed circuit boards from workstation to workstation on a low volume production line, with a minimum of human interaction. The system would have to operate in a printed circuit board assembly environment with moderate human traffic, on grounded carpet, tile, and concrete floor surfaces with 5-foot (minimum) aisles and doorways. The system would also have to be flexible to allow for changes in the order, number, and location of workstations.

The team's solution was to transport the printed circuit boards on an electric cart that would be made capable of self-guidance along a predetermined route by the use of an



Photograph 1.

embedded microcomputer. Tape was chosen as the route medium because of its low cost and ease of application and alteration. Using tape as the route medium differs from the traditional approach utilized by automated guided vehicles — most vehicles follow a signal emitted from a wire that must be embedded within the floor, and one system uses a fluorescent chemical guidepath. (See Appendix for partial list of manufacturers.)

To complete the project within the nine-month time frame, a system integration approach was taken using existing commercially available hardware when possible. The computer system development strategy involved the use of Jib Ray Forth, STD bus hardware, and the host/target communication concept, which links the development computer system (host) to the application microcomputer (target). [2]

Theory of Operation

System Overview

The microcomputer controlled cart optically follows a tape track from workstation to workstation along a closed circuit path by means of a phototransistor array sensor. The system is designed to stop at each workstation for a preprogrammed period of time to allow for manual loading and unloading of the printed circuit boards by the workstation personnel.

An ultrasonic sensor mounted at the front of the vehicle provides long and short range sensing for obstacle avoidance, enabling the cart to stop if a person or object lies within its path of motion. Long range (5-foot) sensing is used only in corridors because it would inhibit cart motion during cornering, since the sensor would see neighboring walls as within 5 feet, and thus treat them as an obstacle. For this reason, short range (1-foot) sensing is used during cornering. Two emergency stop buttons and a front-mounted air-inflated bumper will also disable the cart's motion if activated.

The user interface consists primarily of an alphanumeric display for messages and two keypads for data input. The use of in-system alterable, electrically erasable read-only memory (EEROM) allows the vehicle to be easily reprogrammed for a new workstation configuration without the need for software changes. A scheme of detecting thin metal plates using metal sensors permits workstation location and switching between long and short range obstacle sensing. Figure 1 shows the system diagram with the microprocessor and application software at the center of the system interfacing to the six subsystems.

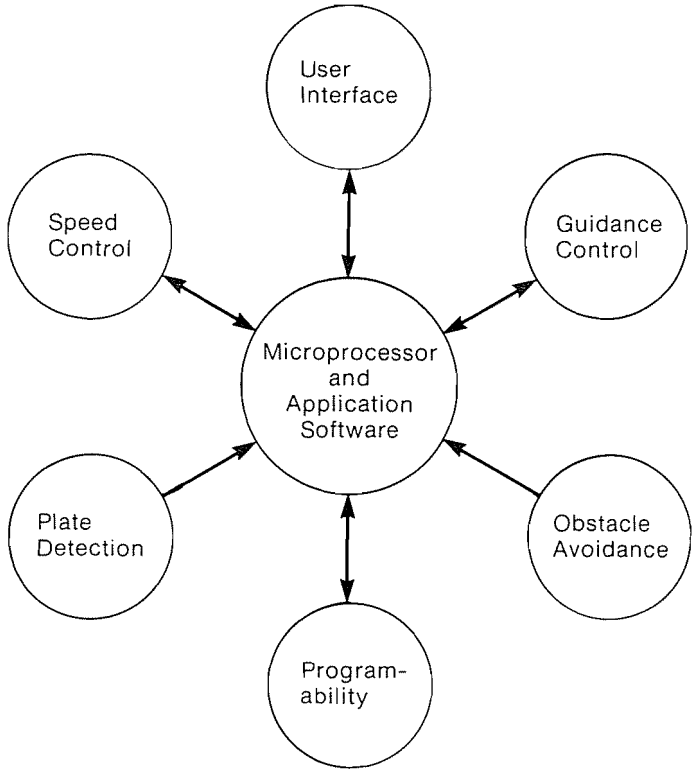


Figure 1.

Plate Concept

Although the cart uses the tape track laid by the user to navigate, the tape conveys no further information to the cart. Metal plates (thin 5-inch square pieces of metal tape) located alongside the tape track serve as markers to the cart. The plates are detected by metal proximity sensors that are mounted underneath the vehicle. As the cart passes over a metal plate while following the tape track, it will know that it:

1) has arrived at a workstation
2) should enable long range obstacle sensing
3) should enable short range obstacle sensing.

Although there are three unique tasks, each metal plate looks alike to the cart. The process of "teaching" the cart allows the user to select which one of the three tasks is to be associated with each plate. During the teaching program the software keeps track of this

plate/task association and stores this information in the EEROM, along with the distance between the plates. By starting the cart each day from the same location that the teaching sequence was started, the cart is able to associate which one of the three tasks to perform at each plate along the entire track.

System Hardware

Cart

The electric vehicle chosen was a Taylor-Dunn model SS personnel carrier, which is 77 inches long by 29 inches wide, and has a 6 square foot flatbed area with a 500 lb. capacity. The cart is a three wheeled, rear wheel driven vehicle powered by a 4.5 hp, 24 volt DC motor that is controlled by a Pwr-Tron transistor speed controller. The Pwr-Tron enables smooth speed and acceleration control as well as easy computer interface. Four 6 volt, 245 ampere-hour batteries, recharged by an on-board charger, power the system. A 60:1 right angle worm gear reducer, coupled to a 1/6 hp, 12 volt DC motor, replaced the original tiller steering bar that turned the single front wheel. A pair of 25 amp relays control the steering motor's direction of rotation.

The cart can operate in either manual or computer mode. For manual mode operation (to move the cart outside of the track circuit) a three position momentary paddle switch controls the relays in order to steer the cart. A direction switch controls the forward and reverse solenoids that carry current to the drive motor. In manual mode the cart is controlled by the steering switch, direction switch, accelerator pedal, and foot brake, all but the latter being disabled in computer mode.

Microcomputer

The microcomputer system uses a STD bus: an 8-bit microprocessor card system using a standardized pinout, a 56-pin edge connection, and a 4.5 inch by 6.5 inch card size. A 16 position card rack houses the system and a Kepco DC to DC converter, receiving +24 volts from the cart batteries and delivering regulated +5, +12, and -12 volts to the backplane powers the system. Using several commercially available STD boards greatly reduced development time. The microcomputer consists of the following nine cards:

- dy-4 Systems 4.0 MHz Z80 CPU with Serial I/O
- Pro-Log 64K Byte-Wide Memory
- Pro-Log Termination Network
- Micro/sys Counting and Timing Controller
- Enlode Keypad Interface
- Enlode Alphanumeric Display Interface
- Custom Circuit Card #1
  - Phototransistor Array Interface
  - Metal Plate Detection
  - Shaft Encoder Interface
  - Emergency Stop Detection
  - Cart Hold Detection
- Custom Circuit Card #2
  - Pwr-Tron Input Voltage (D/A Conversion)
  - Steering Feedback (A/D Conversion)
  - Steering Motor Relay Control
  - Solenoid/Pwr-Tron Control (Transistor Switch)
- Custom Circuit Card #3
- Ultrasonic Circuit Board
  - EEROM and Interface

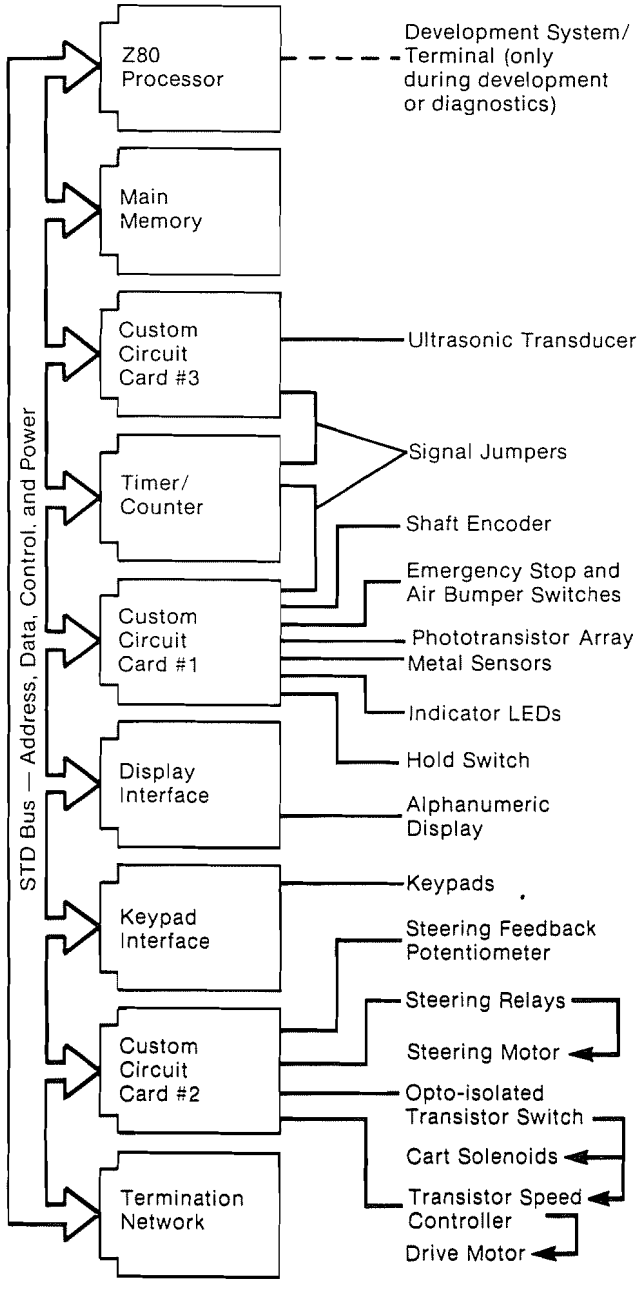


Figure 2.

Figure 2 illustrates the interconnection between the microcomputer and the various peripheral devices comprising the six subsystems.

Vehicle Guidance

Subsystems

The cart "sees" the tape path by means of an array of 16 phototransistors, recess mounted on 0.5 inch spacing, along a bar whose angle to the floor is adjusted for optimum sensitivity. The bar in turn is mounted to a larger rectangular plate supported by three runners, thus keeping the phototransistors at a constant distance from the floor surface. The array system is attached to pivoted bars on the bottom of the cart and pulled along the tape marked floor by the forward motion of the cart. An incandescent light source mounted on the array above the phototransistors illuminates both the background surface and the 1.5 inch wide tape track. A black, non-reflective tape track is required if the background surface is reflective; if the background is non-reflective, then a white, reflective tape track is required. The software also allows the use of both white and black tape within the track circuit, since it is likely that the cart will travel on adjoining reflective and non-reflective surfaces.

The phototransistor output is gated through Schmidt triggers producing a pattern of 0's and 1's that represents the current lateral position of the array with respect to the track (see Figure 3). This array signal is read as two bytes through tri-state buffers by GET-ARRAY-SIGNAL, where ARRAY-PORT1 and ARRAY-PORT2 are hardware dependent constants.

```
: GET-ARRAY-SIGNAL      ARRAY-PORT1 P@  ARRAY-PORT2 P@  ;
```

GET-ARRAY-SIGNAL is the first word called by SIGNAL-PROCESS, which is one of the highest level guidance control words that coordinates input, decision branching, and processing.

```
:  SIGNAL-PROCESS
        GET-ARRAY-SIGNAL
        LOOK-FOR-TRANSITIONS
        IF          INCREMENT-BAD-SIGNALS
        ELSE        DETERMINE-BACKGROUND
                        FIND-VALID-TRACK-SIGNAL
            IF          GENERATE-ERROR-TERMS
            ELSE        INCREMENT-BAD-SIGNALS
            THEN
        THEN    ;
```

Once the two bytes are read, then the word LOOK-FOR-TRANSITIONS checks the array signal to see that it is not all 1's (all reflected light) or 0's (no reflected light), and leaves a boolean flag on the stack. If this check is true, i.e. there are no transitions, then the cart has lost the track and the INCREMENT-BAD-SIGNALS routine is executed, thus incrementing the "bad signal" counter by one. This counter only accumulates consecutive bad array signals — once a valid signal is read the counter is reset to zero. If this counter exceeds the user's preset limit, then the microcomputer considers the cart to be off the track, stops the cart, and requests that the user examine this area of the track for problems. This feature adds robustness to the system since if a few inches of track have been removed or are severely soiled, the cart continues forward in an attempt to obtain a valid signal rather than stopping immediately.

If the LOOK-FOR-TRANSITIONS check is false, i.e. there are transitions, then the array signal will be further processed to determine if a valid track signal exists. DETERMINE-BACKGROUND determines if the background surface is reflective or non-reflective. FIND-VALID-TRACK-SIGNAL then examines the array signal for a first transition (left tape edge), a minimum two-bit wide track signal, a second transition (right tape edge), and finally checks to make sure there are no other transitions. If and only if all

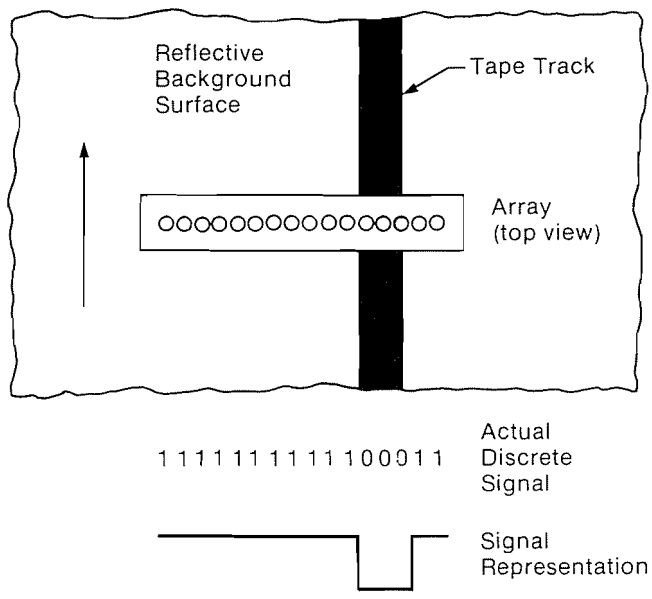


Figure 3.

these conditions are met then the current lateral position signal is considered valid and the GENERATE-ERROR-TERMS routine is executed. If all these conditions are not met, then the track signal is not valid and the INCREMENT-BAD-SIGNALS routine is executed. Therefore the end result of SIGNAL-PROCESS is either generating error signals or incrementing and checking the bad signal counter.

Thus if a valid lateral position signal is obtained, error signal values are generated by GENERATE-ERROR-TERMS for the lateral position (based on a setpoint of centering the array over the track), lateral velocity, and lateral acceleration. These values are then used by the proportional, derivative, and integral position control algorithm (CALCULATE-DELTA-SETPOINT) to produce the clockwise or counterclockwise position change in the front (steering) wheel setpoint. UPDATE-SETPOINT adds this change (variable DELTA-SETPOINT) to the past setpoint (variable STEERING-SETPOINT) to produce the new front wheel setpoint. UPDATE-SETPOINT also checks the new front wheel setpoint to prevent the wheel from turning beyond its physical limits of +/- 100 (adjusted from the actual steering feedback potentiometer range of 27 to 227 as input by the A/D) by setting the setpoint to the maximum allowable left or right value if the new setpoint value is out of bounds.

```
: UPDATE-SETPOINT
        DELTA-STEERING @ STEERING-SETPOINT @ + DUP ABS 100 <
        IF                  STEERING-SETPOINT !
        ELSE                0 <
            IF              -100 STEERING-SETPOINT !
            ELSE             100 STEERING-SETPOINT !
            THEN
        THEN ;
```

Once the new front wheel setpoint has been calculated and verified, then APPLY-STEERING-CONTROL obtains the current front wheel position (GET-WHEEL-POSITION) through the steering feedback potentiometer and an 8-bit A/D converter. WITHIN-TOLERANCE? checks if the new setpoint and the current wheel position agree within the allowed tolerance band. If this is true, then no control is applied, otherwise the wheel is proportionally turned left or right by turning ON the DC steering motor for the calculated period of time. An opto-isolated circuit controls the relays that activate the DC steering motor either clockwise or counterclockwise.

```
: APPLY-STEERING-CONTROL
        GET-WHEEL-POSITION
        WITHIN-TOLERANCE?
        IF                  ( no control applied )
        ELSE                WHEEL-POSITION @ STEERING-SETPOINT @ >
            IF              TURN-RIGHT
            ELSE            TURN-LEFT
            THEN
        THEN        ;
```

NAVIGATE is the highest level guidance control word that brings the signal processing and the steering control together. SIGNAL-PROCESS leaves a boolean flag (a true is left by GENERATE-ERROR-TERMS or a false by INCREMENT-BAD-SIGNALS) on the stack since applying steering control is dependent on obtaining a valid array signal.

```
: NAVIGATE
        SIGNAL-PROCESS
        IF          CALCULATE-DELTA-SETPOINT
                    UPDATE-SETPOINT
                    APPLY-STEERING-CONTROL
        THEN        ;
```

Figure 4 illustrates the block diagram of the lateral position control feedback system.

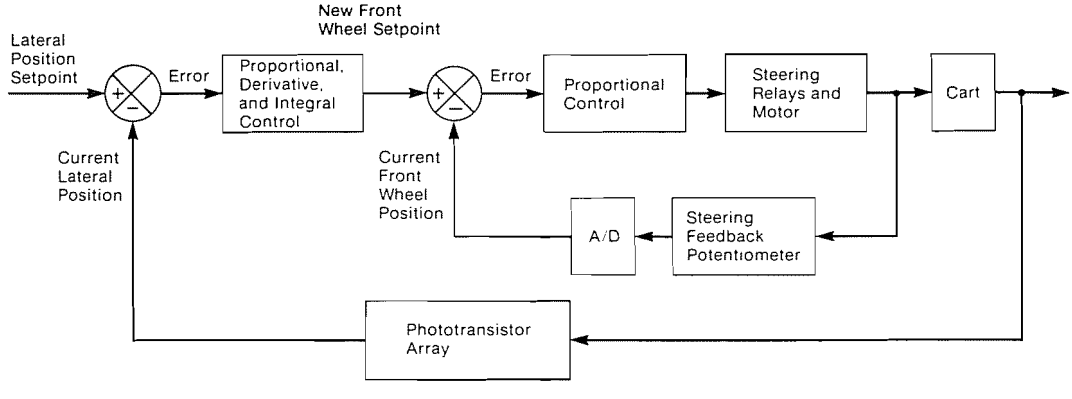


Figure 4.

Speed Control

Closed loop speed control provides smooth cart operation under varying payload, battery charge level, and operating surfaces. Signals from a Hewlett-Packard optical shaft encoder to the timer/counter card measure the cart's speed. A speedometer cable couples the shaft encoder and the cart's differential. Schmidt triggers gate the 500 pulses per revolution output from the shaft encoder to a counter of the timer/counter card. A timer is programmed to generate a continuous 1/8 second pulse waveform whose rising edge triggers the counter channel which receives the pulses from the shaft encoder. The falling edge of that waveform stops the counter, at which time the contents are automatically transferred to a hold register. The next rising edge clears the counter and the sequence repeats continuously.

The speed control software reads the current speed (pulse count) from the hold register and generates the proportional error between the current speed and the 1 mph setpoint. WITHIN-SPEED-BAND? checks if the proportional error is within the accepted tolerance band and leaves a boolean flag on the stack.

```
: WITHIN-SPEED-BAND?
        PROPORTIONAL-ERROR @    ABS
        SPEED-TOLERANCE @       <       ;
```

WITHIN-SPEED-BAND? is the first routine called by SPEED-COMMAND, which is the highest level speed control word.

```
: SPEED-COMMAND
        WITHIN-SPEED-BAND?
        IF              ( no control applied )
        ELSE            CAL-DERIV-SPEED-ERROR
                        CALCULATE-DELTA-SPEED
                        DELTA-SPEED @ PWR-TRON-INPUT @ + DUP 255 >
             IF         DROP   255 DUP PWR-TRON-INPUT !
             ELSE DUP   PWR-TRON-INPUT !
             THEN SPEED-PORT P!
        THEN    ;
```

If the WITHIN-SPEED-BAND? check is true, then no control is applied, otherwise the derivative speed error term is generated (CAL-DERIV-SPEED-ERROR) and used with the proportional error term to calculate the change in the control signal CALCULATE-DELTA-SPEED). SPEED-COMMAND updates the speed setpoint by adding the positive or negative change (DELTA-SPEED) to the previous setpoint (PWR-TRON-INPUT). This new setpoint is then checked to see that it does not exceed the limit of the 8 bit D/A (if so the maximum value of 255 is used). The setpoint is then updated with the newly calculated or maximum value, and the value is written to the D/A through the 8 bit port whose value is specified by the constant SPEED-PORT. The control signal generated by the D/A, i.e. a voltage level between 6.5 and 11.5 volts, is issued to the cart's Pwr-Tron transistor speed controller through a Darlington transistor (to boost the output current). Controlling the voltage signal to the transistor speed controller is analogous to depressing the accelerator pedal, which in turn rotates a potentiometer whose output voltage is input to the speed controller. Figure 5 illustrates the block diagram of the speed control feedback system.

User Interface

The user interface consists of indicator LEDs, a lighted hold button, two keypads, and an alphanumeric display. The LEDs indicate the status of: the teaching program, the emergency stop or air bumper switches, and the left and right metal plate sensors.

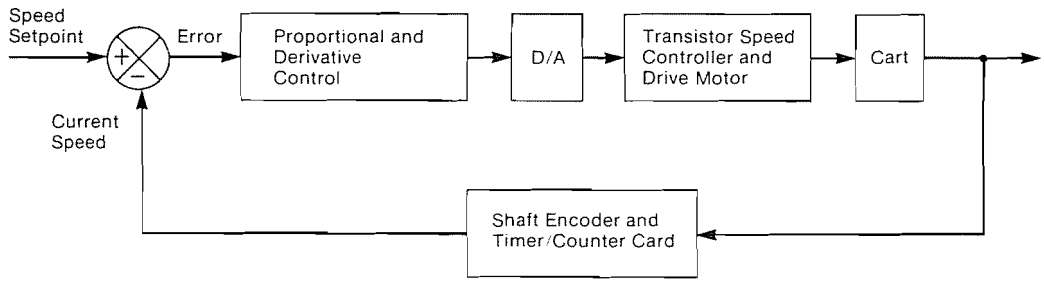


Figure 5.

The cart can be held at a workstation for any period of time by activating the hold button (maintained switch) during the preprogrammed waiting period, thus interrupting the displayed time countdown (time remaining before cart leaves workstation). The software will display a message reminding the workstation personnel that the maintained hold button must be reset before the cart can continue along the track. The maintained switch is sensed for closure through a tri-state buffered input port.

The keypads and alphanumeric display constitute the primary I/O with the user. Each is an independent, integrated system having its own dedicated interface card. The keypad interface card handles both 16-key keypads (one for data entry and one for function/program selection), performs key debouncing in hardware, and is mapped as an 8-bit input port. The software polls the input port for keystroke data only during those sections of cart operation that require user input (which always occur when the cart is not moving). KEYPAD-SCAN is the first word used by all keypad scanning routines that are looking for specific keys to be pressed on either of the two keypads.

```
: KEYPAD-SCAN
        KEYPAD-PORT P@          ( fetch status byte)
        KEY-INPUT-BYTE          ( store status byte)
        GET-BITS-6&7  0 =       ( check if bits 6 and 7 are low)
        IF      GET-BITS-4&5  KEYPAD-IDENTIFICATION
                GET-BITS-0-3  KEY-DATA !
        THEN                ;
```

Once the KEY-INPUT-BYTE is obtained, bits 6 and 7 are checked. If they are both low then a key was pressed and the word GET-BITS-4&5 obtains the code of the keypad that generated the data and leaves it on the stack (bits 4 and 5 are set differently depending on which keypad originated the data). GET-BITS-0-3 then obtains the four data bits and leaves them on the stack. If bits 6 and 7 were not both low, then no key was pressed. KEYPAD-SCAN is always used in a polling loop which is repeated until the requested key or keys have been pressed.

Various keypad scanning routines were developed which accept only requested data, thus preventing the user from entering invalid data into a program. An example of this is RESET-SCAN, which is a word that waits until the RESET key is pressed (following cart disabling due to obstacle avoidance device activation). When the keypad identification check is true, i.e. a keystroke was entered on the desired keypad, then the data is checked to see if the desired key was pressed. When both are true the RESET key has been pressed and the routine is exited.

```
: RESET-SCAN
        0 EXIT-FLAG !
        BEGIN   KEYPAD-SCAN
                KEYPAD-IDENTIFICATION @ 32 =
            IF              KEY-DATA @ 7 =
                IF          1 EXIT-FLAG !
                THEN
            THEN
        EXIT-FLAG @ UNTIL       ;
```

The 28 character alphanumeric display is used extensively to display messages and prompt the user for input. The interface card is mapped as two sequential 8-bit output ports: one for the character position in the display and the other for the ASCII character code. Each static message, i.e. those without changing values, is stored separately in the microcomputer's memory in ASCII form as 28 character strings using ." (dot-quote).

```
: MESSAGE-1     ." HELLO...I AM THE HI-TEK CART"     ;
```

To display a message the display software must first obtain the address of the message to be sent. Once stored, the base message address (address of first character) can easily be obtained by adding 3 to the parameter field address generated by ' (tick) (this method may be Forth version dependent). SEND-MESSAGE-1 is used to display the message by passing the base message address to DISPLAY-LINE to do the actual writing to the display.

```
: SEND-MESSAGE-1
        '   MESSAGE-1  3 +  ( leave base message address on stack)
        DISPLAY-LINE    ;
```

Since each message must be written one character at a time, DISPLAY-LINE executes a loop whose limit is equal to 28. DISPLAY-LINE first writes the position code (loop index value) to the position port (SET-POSITION), then fetches the ASCII character byte from memory (GET-CHARACTER), and finally writes it to the character code port (SEND-CHARACTER).

For messages that are not 28 characters in length, the loop limit is a variable that is set equal to the message length at display time by the PARTIAL-DISPLAY routine. This routine also enables variable placement of the partial message within the 28 character display field by setting a variable for the display position, rather than using the loop index.

Obstacle Avoidance

The cart has three safety mechanisms: rear and mid-mounted emergency stop buttons, a front-mounted air-inflated bumper, and a front-mounted ultrasonic sensor. The two emergency stop buttons (normally open switches) and the air bumper (air-actuated normally open switch) are connected in parallel to an edge triggered latched circuit that is polled by the obstacle avoidance software. If any of the three switches is closed a flag is set and the circuit is reset.

A modified Polaroid Ultrasonic Ranging System (transducer and ultrasonic circuit) is used with the timer/counter card to detect the presence of a person or object in the path of the cart. The transducer serves as both the high frequency transmitter and receiver. A continuous sequence of transmit/receive cycles is accomplished by supplying the ultrasonic circuit power section with a 200 millisecond timer generated square wave. The rising edge of the square wave initiates the transmit signal which consists of eight cycles at 60 KHz, eight cycles at 57 KHz, sixteen cycles at 53 KHz, and twenty-four cycles at 50 KHz [3]. The beginning of the transmit sequence triggers a counter which continues until the first echo signal is received, at

which time the measured time is automatically transferred to a hold register on the timer/counter card. The measured echo time is proportional to the distance between transducer and object.

POLAROID-FLAG? is the highest level ultrasonic sensor word that leaves a boolean flag on the stack reflecting the range of the detected object and the stopping threshold.

```
: POLAROID-FLAG?
        READ-POLAROID-RANGE
        VALIDATE-RANGE
        IF      RANGE-FILTER
                FILTERED-RANGE @  CURRENT-THRESHOLD @ <
        ELSE    0
        THEN    ;
```

Once the latest echo time (CURRENT-RANGE) has been read from the hold register by READ-POLAROID-RANGE, the value is then checked to see if it is within the operational limits of the sensor by VALIDATE-RANGE. If this is true then the current range is input to a weighted average filter. RANGE-FILTER takes 1/4 the n-2 range value (RANGE-n-2), 1/2 the n-1 range value (RANGE-n-1), and 1/4 the current value (CURRENT-RANGE) to produce the FILTERED-RANGE. (It also stores the n-1 value into n-2 and the current range into n-1 for the next time RANGE-FILTER is executed.)

```
: RANGE-FILTER
        RANGE-n-2 @ 4 /             ( 1/4 n-2 value)
        RANGE-n-1 @  DUP            ( copy n-1 value)
        RANGE-n-2 !                 ( update n-2 value)
        2/ +                        ( add 1/2 n-1 value)
        CURRENT-RANGE @  DUP        ( copy current value)
        RANGE-n-1 !                 ( update n-1 value)
        2 / +                       ( add 1/4 current value)
        FILTERED-RANGE !                        ;
```

The filtered range is then checked to see if it is less than the current stopping threshold value (either the long or short range value). If this is true then the obstacle detected is within the stopping threshold, and a true is left on the stack. If the filtered range is greater than the stopping threshold, then a false is left on the stack.

EMERGENCY-STOP-FLAG? polls the combined emergency stop button and air bumper switch latched circuit port (EMERGENCY-STOP-PORT) and masks off the unwanted data lines (4 AND). This word leaves a boolean flag on the stack reflecting if any of the emergency stop devices are activated.

```
: EMERGENCY-STOP-FLAG?
        EMERGENCY-STOP-PORT P@  4 AND           ;
```

The results of the switch closure flag and the ultrasonic sensor flag that have been left on the stack are logically OR'ed by the highest level obstacle avoidance word, STOP-CHECK?. If any obstace avoidance device has been activated the cart will stop, display the reset message, and will remain inactive until the RESET key is pressed, at which time the software (STOP-CHECK?) will display a message that the cart is about to resume motion.

```
: STOP-CHECK
        EMERGENCY-STOP?  POLAROID-FLAG?  OR
        IF          STOP-CART
                    SCAN-FOR-RESET
                    RESTART-INITIALIZATION
                    ACTIVATE-DRIVE-MOTOR
        THEN        ;
```

STOP-CART stops the cart by opening the forward solenoid and the isolator (main motor circuit breaker) solenoid, and shutting off power to the transistor speed controller. These devices are controlled by supplying them with either +24 volts (ON) or 0 volts (OFF) through a latched 8-bit output port, by an Opto 22 opto-isolated transistor switch. ACTIVATE-DRIVE-MOTOR turns back ON the forward and isolator solenoids, and the transistor speed controller. However, the cart will not resume motion until a control signal is issued to the controller by the SPEED-CONTROL routine.

Plate Detection

As earlier discussed, proximity sensors mounted beneath the cart detect the thin metal plates as the cart passes over them. These Micro Switch sensors operate on the Eddy Current principle and are interfaced directly to an edge triggered latched circuit that is polled by the plate detection software. The two sensors are mounted directly across from each other underneath the vehicle at a height of approximately 1 inch above the floor. They are flexibly mounted to deflect if they collide with an object inadvertently left on the floor, that would not be detected by the ultrasonic sensor.

In order to avoid false detection, due to metal objects on or in the floor such as door jams (rather than actual metal plates), two steps were taken. The first uses a system of alternating the plates on either side of the track, and the second involves measuring the distance between the plates with the shaft encoder. (Using metal plates and the shaft encoder eliminates cumulative error problems associated with only using a shaft encoder, since the encoder counter is set to zero when each plate is found.) By using two metal sensors, the first plate is located on the right-hand side of the tape track, the next plate on the left-hand side, and so on for as many plates as are required in the track circuit (up to 99).

READ-METAL-SENSOR polls the latched metal sensor circuit port (METAL-SENSOR-PORT) for the sensor status byte and masks off the unwanted data lines (3 AND).

```
: READ-METAL-SENSOR     METAL-SENSOR-PORT  P@  3 AND     ;
```

This word is the first routine called by METAL-SEQUENCE, which is the highest level metal sensor definition that determines if a plate has been found.

```
: METAL-SEQUENCE          READ-METAL-SENSOR  DUP  0 =
      IF                  DROP ( metal plate not detected )
      ELSE                CURRENT-SIDE @ =
          IF              TEACH-FLAG @  0 =
              IF          GET-DISTANCE
                          WITHIN-TOLERANCE?
                  IF      RESET-ENCODER-COUNTER
                          PERFORM-PLATE-TASK
                          SWITCH-SIDES
                          INCREMENT-PLATE-COUNTER
                  THEN    RESET-METAL-SENSOR
              ELSE        EXECUTE-TEACH-SEQUENCE
              THEN
          ELSE            RESET-METAL-SENSOR
          THEN
      THEN      ;
```

Once the sensor status has been read by READ-METAL-SENSOR, it is compared to zero. If the result of this check is true then neither the left or right sensor has been activated and the word is exited. If a sensor has been activated (zero check was false) then the status is compared with the CURRENT-SIDE variable, whose value is 1 if the next plate is on the right, or 2 if it is on the left. If the sensor status does not agree with the value of CURRENT-SIDE then either the activated sensor is not the correct side, or both sensors are ON. In either case the metal sensor circuit is reset by RESET-METAL-SENSOR (when execution branches to the corresponding ELSE statement) and the word is exited. If the sensor status does agree with CURRENT-SIDE then the correct sensor is on, i.e. the detected plate is on the expected side, and execution proceeds to the 3rd IF statement.

The 3rd IF statement is checking to see if the variable TEACH-FLAG? is set to zero. If this is true then the RUN program (user selected program used to put the cart into automatic operation in which it repeats the track layout continuously) was selected and execution proceeds to GET-DISTANCE. If the TEACH-FLAG? was not equal to zero then the TEACH program (user selected program used to teach the cart the track/plate layout) was selected and the EXECUTE-TEACH-SEQUENCE is executed. (This sequence first stops the cart and asks the user if a valid plate was found, and if so, the user is prompted to program the task at that plate. This is continued until all the plates have been programmed.)

Thus if the TEACH-FLAG? was equal to zero, then execution proceeds to the GET-DISTANCE routine which obtains the distance from the last plate as recorded by the shaft encoder. This distance is then compared with the distance between the plates that was recorded during teaching. If these values agree within the allowed tolerance band (WITHIN-TOLERANCE?) then a valid plate has been found, and execution proceeds to the 4th (innermost) IF statement. The encoder counter is then set to zero by RESET-ENCODER-COUNTER, PERFORM-PLATE-TASK looks up in memory which task (workstation, long range sensing, or short range sensing) to execute and does so, SWITCH-SIDES updates the CURRENT-SIDE variable, and finally the plate counter is incremented by INCREMENT-PLATE-COUNTER. However, if the distance values do not agree within the allowed tolerance band as checked by WITHIN-TOLERANCE?, then a valid metal plate was not found (False detection). The latched circuit is then reset whether or not a valid plate was found since in either case a metal sensor was activated.

Programability

The ability to reprogram the cart for a new track and plate layout is made possible by the use of a SEEQ Technology EEROM — a 5 volt non-volatile memory with in-system

write and erase capability. The number of plates, task at each plate, distance between plates, number of workstations, and time at each workstation are written into the EEROM during the teaching sequence. The EEROM also contains one dozen system constants for the speed, guidance, plate detection, and obstacle avoidance algorithms. These user "alterable constants" allow for fine tuning of the system on location. At the end of each day when the cart is powered down for recharging, all of this data remains intact. When teaching the cart a new track/plate layout, the software must first erase the old track/plate data (by writing an FF hex to each byte) before the new data can be written into the EEROM.

The EEROM requires additional circuitry (as compared to conventional memory) to hold the data, address, and control lines for 10 milliseconds during the erase and write operations. This is accomplished by generating a busy signal to the processor. Since the cart is not moving while the user is being prompted for data input, having the processor hold the data, address, and control lines rather than using a latched circuit does not penalize system performance.

ERASE-WORD and SEEQ! are definitions used to store a word (versus a single byte) of data in the EEROM. ERASE-WORD erases the word in the EEROM whose address is on the top of the stack by separately writing an FF hex (255 decimal) to the two consecutive bytes that comprise the word.

```
: ERASE-WORD
        DUP             ( copy address)
        255  SWAP        ( place data byte under address)
        C!              ( erase low address)
        255  SWAP        ( place data byte under address)
        1+              ( produce high address)
        C!              ( erase high address)       ;
```

SEEQ! stores the word of data (second entry on stack) to the address in the EEROM that is on the top of the stack. SEEQ! first calls ERASE-WORD and then writes the data word one byte at a time into the EEROM. This is accomplished by reading the high byte of the data word which resides at the address of the stack pointer, and writing it to the low address of the word. The low byte is then read from the address of the stack pointer plus one, and written to the high address of the word.

```
: SEEQ!
        DUP                     ( copy address)
        ERASE-WORD
        WORD-ADDRESS !          ( store address)
        SP@                     ( fetch address of high data byte)
        C@                      ( fetch high byte)
        WORD-ADDRESS @          ( fetch low address of word)
        C!                      ( store high byte at low address)
        SP@ 1+                  ( produce address of low byte)
        C@                      ( fetch low byte)
        WORD-ADDRESS @ 1+       ( produce high address of word)
        C!                      ( store low byte at high address)
        DROP                    ( drop data word)       ;
```

Software

System Programs

The 16K byte application software consists of hardware drivers, control and signal processing algorithms, user messages, hardware diagnostics, and the program logic. The user

has three program choices after the power-up sequence.

Upon power-up the processor begins execution at the lowest address (0000H) of the Forth kernel. Forth first executes COLD which sets all the system variables and then executes ABORT. ABORT then clears the parameter and return stacks, prints the sign-on message, and executes QUIT. At the beginning of QUIT (before the infinite loop that calls the Forth interpreter repeatedly) the constant DRIVER is executed, which returns the address of a NOOP (no-operation) at a point within QUIT where the address of a driver definition can be inserted. In order for the target application to start automatically upon power up, the address of the NOOP has been replaced by the code field address of the driver definition (CART-DRIVER). Thus upon power up CART-DRIVER is executed rather than passing control over to the Forth interpreter by entering the infinite loop within QUIT. [4]

```
: CART-DRIVER
        START-UP-INITIALIZATION
        GREETINGS-MESSAGE
        ?TERMINAL
        IF      CR ." WELCOME TO THE LAND OF INTERACTIVE
                      DIAGNOSTICS "
                ( exit driver definition and get back to the interpreter in order to
                  perform diagnostics)
        ELSE    ( execute) MAIN
        THEN        ;
```

CART-DRIVER first executes the application's variable and hardware initialization routine (START-UP-INITIALIZATION) and then displays the introductory messages (GREETINGS-MESSAGE). After the messages are displayed (which takes approximately 30 seconds) the terminal input port is then checked to see if a key has been hit on a terminal (not one of the two keypads) connected to the target's serial channel. If ?TERMINAL is true, then the driver definition relinquishes control to the Forth interpreter (exits the driver definition and enters the infinite loop of QUIT), enabling the user to perform interactive hardware diagnostics. If ?TERMINAL is false, then MAIN is executed, which prompts the user to choose one of the three system programs: Teach, Run, or Reprogram.

Teach. This program allows the user to teach the cart which of the three tasks (workstation, long range sensing, or short range sensing) to perform at each metal plate along the tape track. Once the track and plates have been laid, the user drives the cart (in manual mode) to the location that will be the daily starting point on the circuit. With the user on board (in computer mode), the cart travels once around the track stopping at each plate. The user is prompted to enter the task at that plate. If the task is a workstation, then the user is prompted to enter the time at the workstation. Each time data is entered, the user is prompted to confirm the entry. During this process the software is storing the task at each plate, the time at each workstation, and the distance between plates in the EEROM. When there are no more plates to be programmed the user responds affirmatively to the, "Is this the last plate?" prompt. Once the teaching sequence is finished the program proceeds to the main loop where it waits for the user to select one of the three programs.

Run. This program is selected each day to put the cart into automatic operation in which it repeats the track layout continuously. When a valid plate is detected, the program looks up the task in memory. If the task is a workstation, then the cart stops and waits the preprogrammed period of time. (If zero minutes have been programmed, then the cart does not stop.) As previously explained, the cart can be held indefinitely at a workstation by activating the hold button. If the task is to change the obstacle sensing range, then the program merely stores the long or short range value into the current obstacle range threshold

variable. This program is an infinite loop which can only be exited by powering down the computer.

Reprogram. This program only allows the user to increase or decrease the time that the cart will wait at each workstation (preprogrammed stop time), as well as eliminate a workstation stop by programming a 0-minute stop time. As in the Teach program, the user is prompted to confirm each data entry. During this program the cart remains completely stationary. Once the reprogramming sequence is finished the program proceeds to the main loop where it waits for the user to select one of the three programs.

The heart of the software is the MASTER-CONTROL-PROGRAM routine which is used in both the Teach and Run programs. This is the continuous main loop that links the obstacle avoidance, speed control, guidance control, and plate detection sequences.

```
: MASTER-CONTROL-PROGRAM
        BLANK-DISPLAY
        ACTIVATE-DRIVE-MOTOR
        BEGIN
                STOP-CHECK
                SPEED-CONTROL
                NAVIGATE
                METAL-SEQUENCE
        0 UNTIL         ;
```

Figure 6 shows a simplified flowchart for the Master Control Program as executed within the Run program (the plate detection and obstacle avoidance sequences are slightly different when executed for the Teach program).

Diagnostics

As previously mentioned, interactive hardware diagnostics can be run by attaching a terminal to the target's serial channel and pressing any key on the terminal within thirty seconds of turning the cart's computer ON. This feature permits the user to execute diagnostics that test the phototransistor array, steering control, ultrasonic sensor, metal sensors, emergency stop switches, and other hardware. The serial maintenance link is also the only means by which to change the user alterable constants that reside in the EEROM.

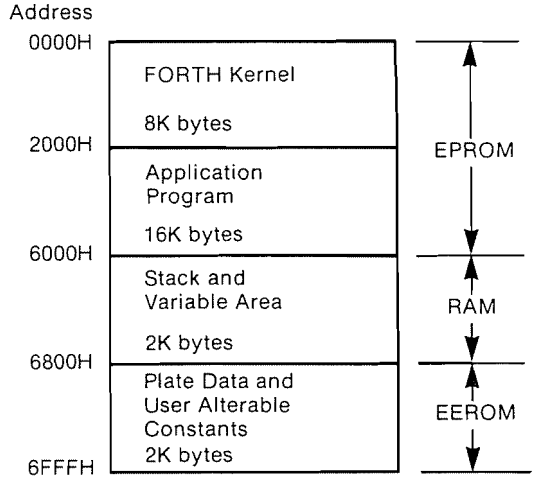


Figure 6.

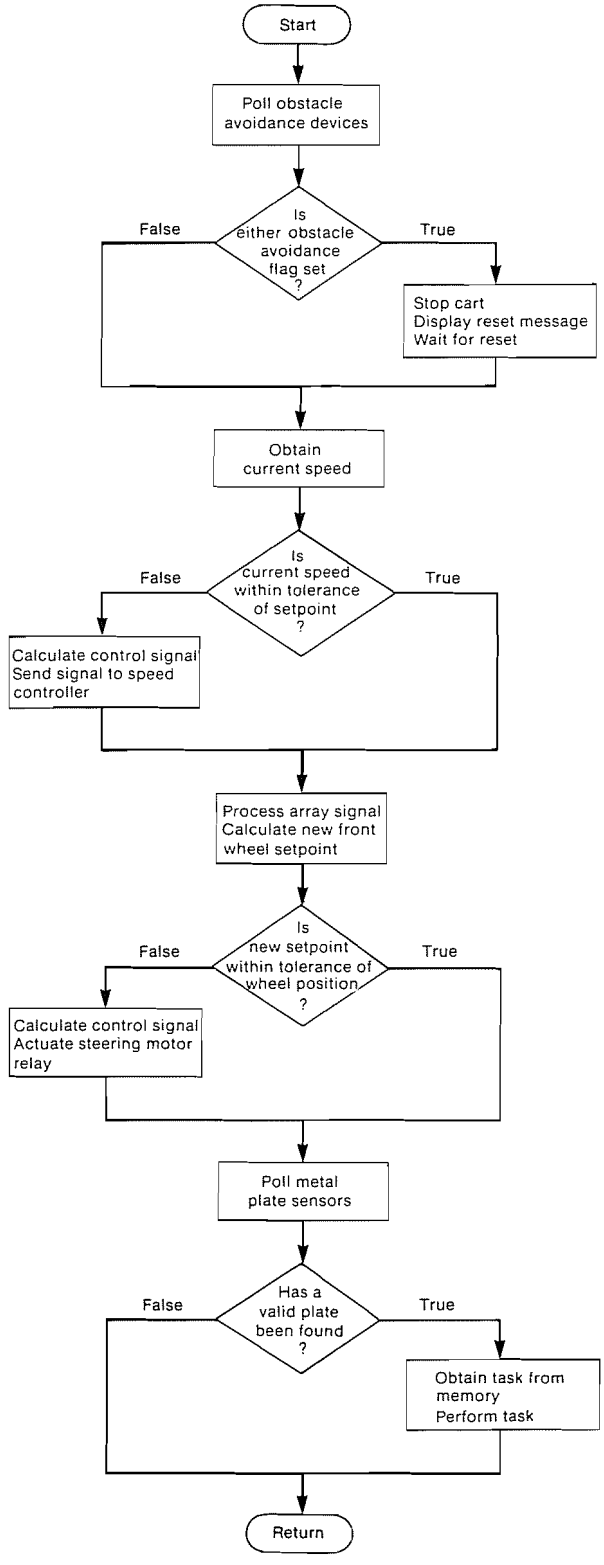

Start
Poll obstacle avoidance devices
Is either obstacle avoidance flag set ?
False
True
Stop cart
Display reset message
Wait for reset
Obtain current speed
Is current speed within tolerance of setpoint ?
False
True
Calculate control signal
Send signal to speed controller
Process array signal
Calculate new front wheel setpoint
Is new setpoint within tolerance of wheel position ?
False
True
Calculate control signal
Actuate steering motor relay
Poll metal plate sensors
Has a valid plate been found ?
False
True
Obtain task from memory
Perform task
Return

Figure 7.

Development Environment

Use of host/target communication linked the power and resources of a development system (host) to the application microcomputer (target). The two systems are connected through a serial interface and linked by the Jib Ray Forth host/target communication software. During development the software modules were coded, stored on disk, edited, printed, simulated, and test compiled using an Intel Development System running Jib Ray Forth. The stored source programs on disk were downloaded from the development system via the serial link to the target Forth computer, where they were compiled by the target computer into RAM memory.

At this stage the target computer's memory consisted of the Forth kernel (8K EPROM) and RAM memory for the application program and stack/variable area. The program could now be tested running on the target computer using the actual hardware. After several iterations of modifying code on the development system, downloading it to the target computer, and testing with the cart, a final version was obtained. Once this version was downloaded to the target, the compiled RAM image was uploaded to the development system so that EPROMs could be burned [1]. The final stand-alone target memory configuration can be seen in figure 7.

Summary

The revolution in microprocessor technology has enabled the design of stand-alone intelligent systems. The development of the vehicle described was a challenging task that combined engineering skills from the mechanical, electrical, and computer science disciplines. The key to the design team's success in completing a functioning prototype was twofold: the system integration approach using existing commercially available hardware when possible, and the three point development strategy using the Forth language, STD bus hardware, and the host/target communication link.

Acknowledgments

The design team would like to extend thanks to Tektronix, for sponsoring the project, Brent Anderson, our liaison with Tektronix, and Stefan Michalowski, our project advisor at Stanford. Special thanks go to David L. Jaffe of the Palo Alto Veterans Administration, Rehabilitation Research and Development Center, without whose technical assistance the project's success would not have been possible.

References

[1] Formisani, S. J., et al, "The design and Development of a Microprocessor Controlled Robotic Vehicle:, *Proc. of the 1984 ASME Computers and Engineering Conference*, Las Vegas, NV, August, 1984.

[2] Jaffe, D. L., "A Design/Development Methodology for Rehabilitation Devices Using Embedded Microcomputers," *Proceedings of the Sixth Annual Conference on Rehabilitation Engineering*, San Diego, CA, June 1983.

[3] *Ultrasonic Ranging System Manual*, Polaroid Corp., Ultrasonic Ranging Marketing, Norwood, MA, pg. 2.

[4] *Jib Ray FORTH Manual*, Jib Ray Inc., Santa Barbara, CA, 1982, pp. 80–88.

Manuscript received June 1984.

Appendix

Barrett Electronics Corp., Electronic Systems Division, Northbrook, Il.
Bell & Howell, Automated Systems Division, Zeeland, MI.
Clark Equipment Co., Handling Systems Division, Battle Creek, MI.
Conoco-Tellus Inc., Mendota, IL.
Control Engineering Co., Farmington Hills, MI.
Eaton-Kenway, Automated Systems, Salt Lake City, UT.
SPS Technologies, Automated Systems Division, Hatfield, PA.

Mr. Formisani received a BS from the University of Lowell in 1982 and a MS from Stanford University in 1983, both in Mechanical Engineering. Currently he is working on robotics systems applications and pursuing a MS in computer science at Stevens Institute of Technology.

Mr. Asakawa received a BS from the University of California at Berkeley in 1982 and a MS from Stanford University in 1983, both in Mechanical Engineering. Currently he is designing and developing next generation hard copy graphic devices.

Mr. Ronne received a BS from the University of Utah in 1982 and a MS from Stanford University in 1983, both in Mechanical Engineering. Currently he is a program manager in microwave communications systems.

Mr. Tomeh received a BS from Princeton University in 1982 and a MS from Stanford University in 1983, both in Mechanical Engineering. Currently he is pursuing an MBA at Harvard Business School.