Val - the Valet Robot
Intelligent Machine Design Lab
EEL 5666
Daniel Copeland

Table of Contents

Abstract

Val is a valet robot whose purpose is to respond to voice command by carrying candy (or any other small objects) from its docking station to different locations, and then return to recharge and wait for its next instruction. Val records how to get to these locations by human direction via remote control. As Val's master directs it to the target location, Val is busy monitoring the distance traveled and turns made. After the location is determined and Val is returned to the docking station, the recorded instructions are saved on a PC under a voice command which, when heard by the PC's voice recognition software, will trigger Val to go to the determined location autonomously and return. In this way, Val's master is able to teach it where it should go, and assign a meaningful voice command to the location like "go to the refrigerator." With this scheme, Val is a reliable and teachable valet with no other agenda but to serve.

Introduction

The first ideas for Val came with the thought of a coffee-serving robot set in an office. A robot that could respond to workers requests when needed and be out of the way when not. It would be a self-sufficient robot that hardly needs maintenance or attention. Several obstacles lay in the way of realizing that idea. How would it know where it is or where to go? How would it receive commands from its users in a way that is natural to them? The answers to these questions materialized in the form of Val. *How would it know where it is?* By dead reckoning, Val uses two potentiometers mounted on the sides about its center axis. Connected to these pots are two wheels, heavy and free to spin, that move as Val moves and work for both forward/backward motion and left/right turns. *How would it know where to go?* Always launching from the same location (its docking station), Val is commanded exactly where to go with remote control and records the directions in memory. Then, after docking, the recorded information is saved to a file on a PC. *How would it receive commands from its users in a way that was natural to them?* What is more natural than speech? Val connects to a PC via the docking station. The PC listens for voice commands using Microsoft Voice—a free voice recognition program. After receiving a command, a program is launched which sends the appropriate record file to Val. Then more questions came: How will the robot dock and make reliable electrical connections? If it is to be low maintenance, how will it stay running? How will it be remotely controlled? What if there is an obstruction in its path? What if it gets off course? The answers followed as Val developed. *How will the robot dock and make reliable electrical connections?* The docking station is made in a "Y" shape to help Val find its way. Also, rolling lever microswitches are mounted two on each side, which allow for a tight fit. Consequently, it can make some precise electrical connections. *If it is to be low maintenance, how will it stay running?* While docked, Val recharges both of its battery packs. *How will it be remotely controlled?* Val uses the serial connector used with the docking station to connect to an hp48g calculator. With it, Val is remotely controlled. *What if there is an obstruction in its path?* Val has infrared detectors and emitters as well as bump switches which may be used for object avoidance and detection. *What if it gets off course?* Val has the ability to wall follow, which helps with recalibrating itself on long-distance trips. The final product, after a semester of working through the details, is shown below in figure 1, followed by a more detailed description of Val and its sub-systems.

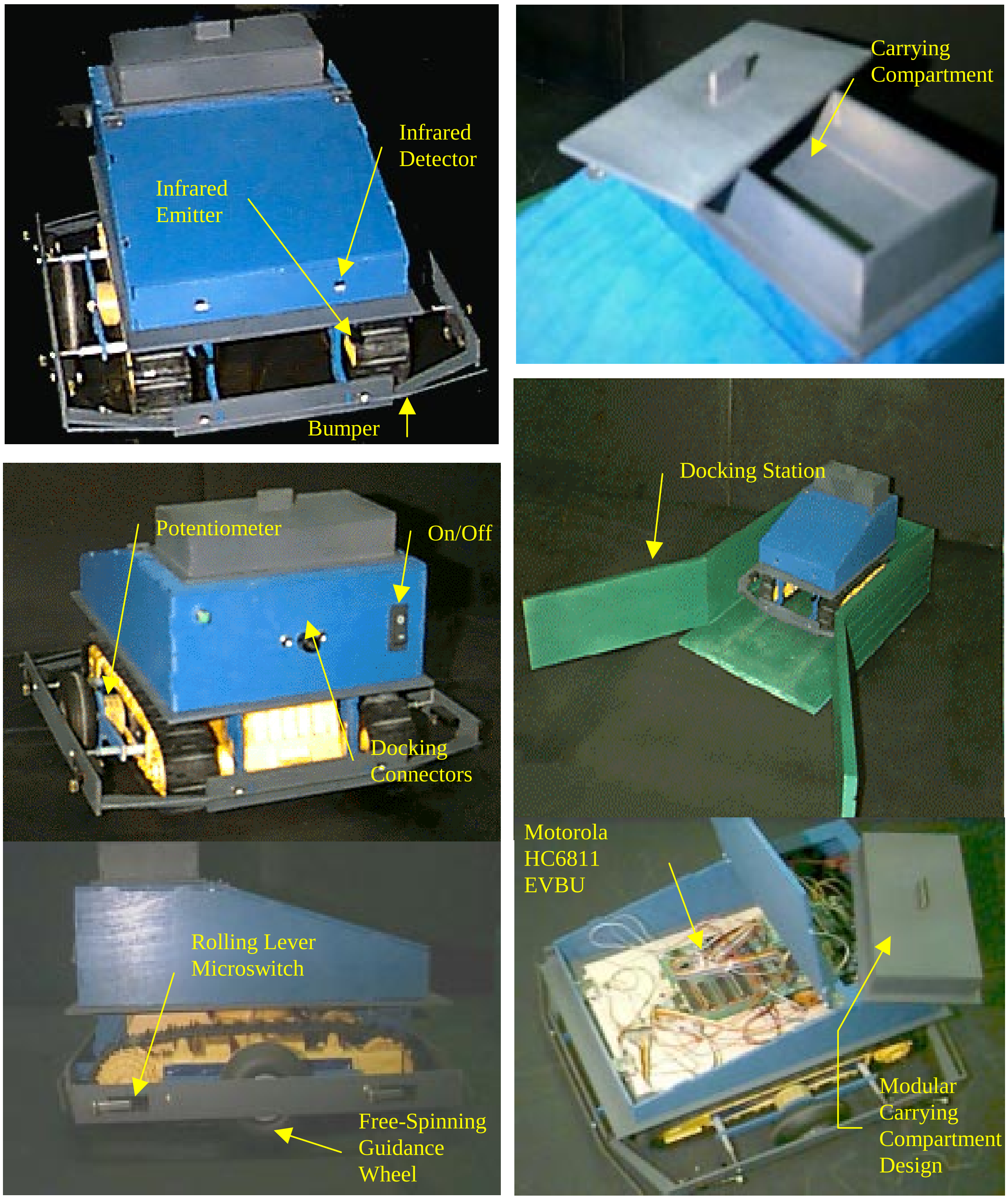


Figure 1: Introduction to Val

Mobile Platform

For the robot platform, I needed a design that could promote forward movement rather reliably as well as 360-degree turns. I wanted the platform to be a tracked vehicle because of the high surface area contact of the track to the floor. I felt a tracked vehicle would have less of a tendency to spontaneously slip on the floor, and would probably be stronger—thus more capable of carrying a small load. Because I wanted to avoid the details of building my own platform, I looked for a toy with these qualities. At Toy's R Us, I found the "Excavator" by New Bright. The toy was a wire remote controlled unit with four motors—one for each track to propel the vehicle and two to actuate the excavator arm- all of which were powered by a single 6.3V rechargeable battery pack. I purchased the Excavator for $40 dollars and the rechargeable battery pack with recharger for $10. After removing the top of the Excavator, I began to build on to the base shown in figure 2 below. The rest of Val's body was drawn in AutoCAD and cutout by a t-tech-milling machine at the Machine Intelligence Lab of the University of Florida.

Figure 2: The Base of the Excavator by New Bright

Actuation

The only actuation provided by Val is that of its two tank trends actuated by the two motors and a few gears to gear up the motors all provided with the Excavator toy. The motors are simple, electric motors with unknown speed and torque characteristics. To control the motors, I use a motor driver circuit (shown as Fig. 3) designed by Drew Bagnell that enables me to control the motors with the HC6811. The circuit provides for four signals from the HC6811: Left Motor Forward/Reverse, Right Motor Forward/Reverse, Left Motor On/Off, and Right Motor On/Off. For speed control of the motors, I simply pulse the On/Off signals to create various duty cycles. For direction control, I either bias one of the motors for a slight turn to the right or left or reverse it to turn.

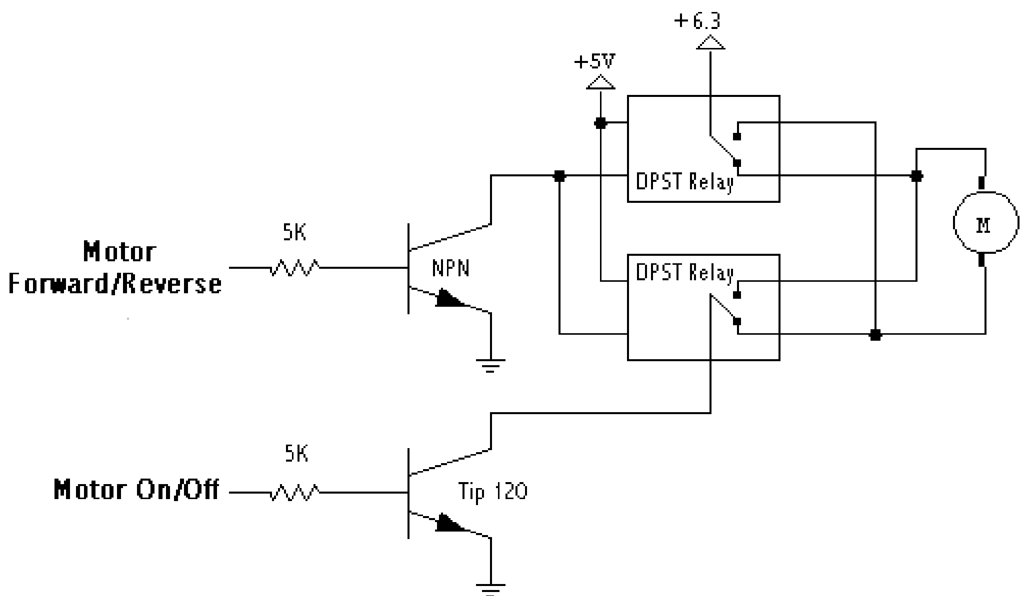


Figure 3: Motor Driver Circuit

Actuation Algorithms

The pulsing of the motors is done through the output compare system of the HC6811. I use output compare 1 to always set output compare 3 and 4 to high when the timer is 0. Then, I use OC3 and OC4 to turn themselves off. Thus, if a number between 0 and 2^{16} is written to OC3 or OC4 that number represents the amount of time the pulse signal is high, (see Fig.4). For my system, the pulse wave period of 2^{16} clock cycles = 32ms.

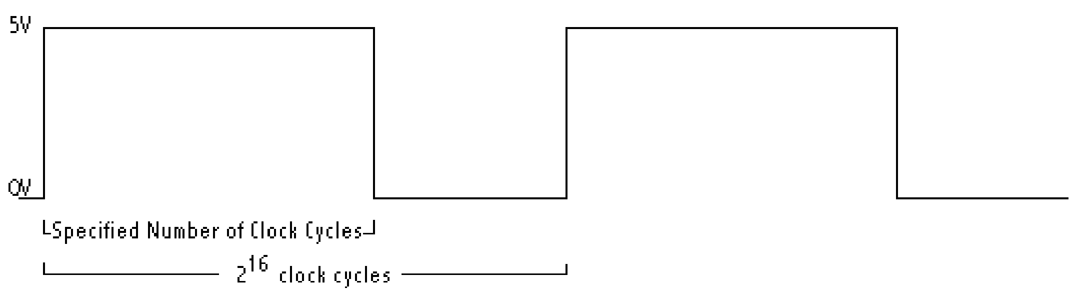


Figure 4: Pulse Width Modulation

To start the vehicle motion, I enable this system and then wait 160ms. Likewise, to stop the vehicle motion, I disable the output capture system and wait 160ms. With this start/stop system, I ensure that there is at least a period of 320ms before the motor can change its direction—a technique intended to prolong the life of the motors. To change a motor's speed while it is in motion, I only need to write a different number to the appropriate output compare register. For turning in place, one motor must change its direction. To do this, I first stop the motors. Then, I set the signal to change the appropriate motor direction. Finally, I start the motors again. This technique provides for turning right or left as well as moving backwards.

Sensors

Bump Switches

There are three major reasons why I added bump switches: to detect when Val bumps something, to guide it into the docking station, and to detect walls for wall following. To meet these goals I used two different types of switches (see fig 5): four roller microswitches–two on each side—to help with wall following and docking and four momentary tactile switches—two in front and back—to sense forward and backward bumps. Across the front and back switches, I epoxied a piece of a clothes hanger that allowed for sensing bumps over the whole front and back area.

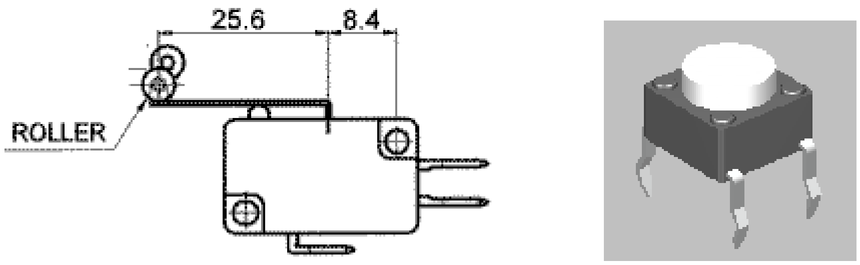


Figure 5: Microswitch (Left) and Momentary Tactile Switch (Right)

Infrared Emitters and Detectors

For more long distance sensing, I added two infrared emitter/detector pairs to the front (see fig. 6). The emitters are collimated with shrink-wrap tubing and the detectors are standard digital 40KHz sensors sold by Sharp. They have been hacked to give an analog signal. See Sharp Sensor Hack for Analog Distance Measurement at the following web site www.mil.ufl.edu/imdl/handouts/sharphack.pdf for more details.

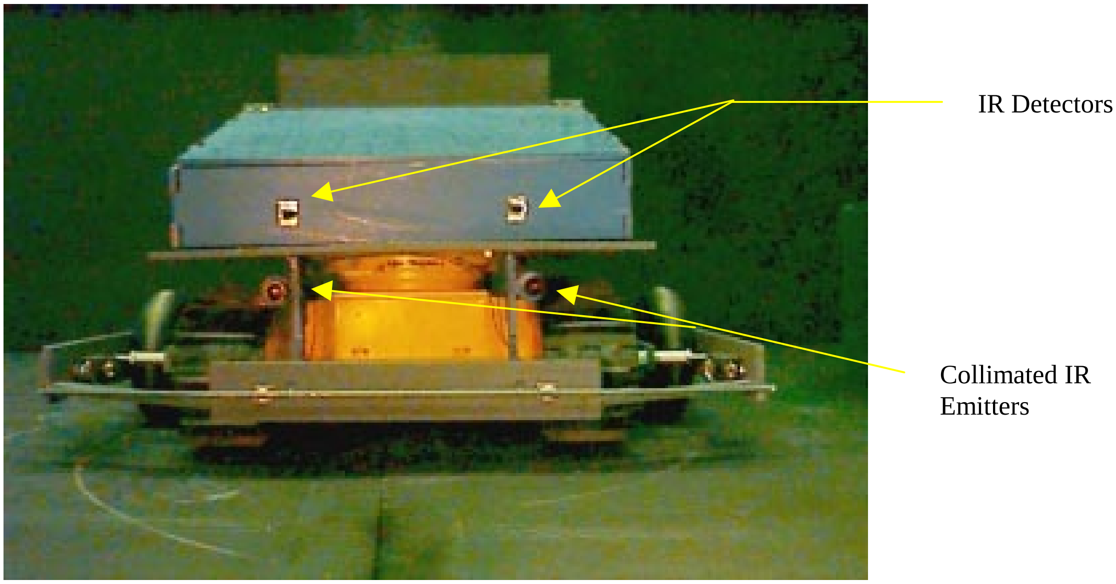


Figure 6: Infrared Emitters/Detectors

Continuously Turning Potentiometer

To keep track of Val's location using dead reckoning, Val must measure three things: distance traveled forward, angle of point-turns, and amount of drift. To do this, I originally planned to use three different sensors, but the idea of using two potentiometers mounted on the side replaced those plans. With the potentiometers mounted with free spinning wheels about the center axis, I could measure both distance and angle. Furthermore, by comparing the speed of both wheels I could get an idea if Val was drifting to one side and compensate. After looking for good continuously turning pots, I contacted Spectrol who graciously donated four 5K pots as shown in fig 7 and 8. These pots have a life of around 2 million turns and an electrical freedom of about 352 degrees. This means that there is about an eight-degree space in which no sure signal is given.

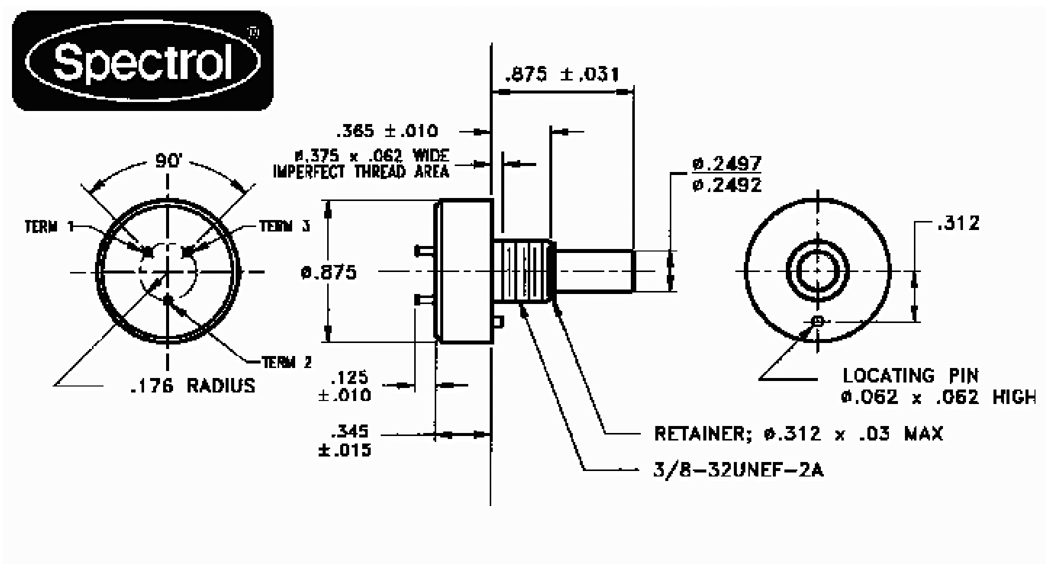
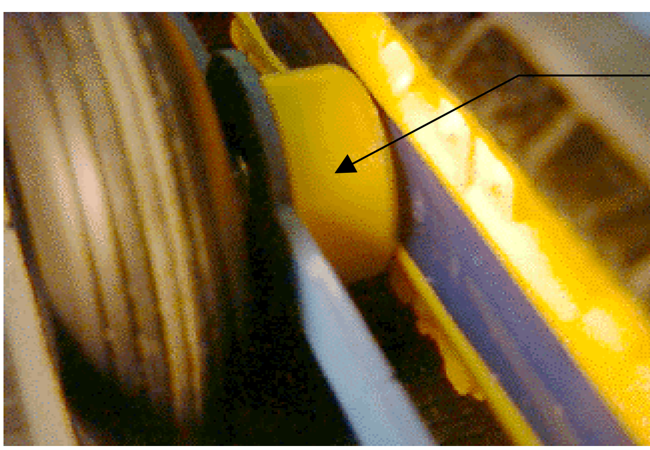


Figure 7: Spectrol Potentiometer



Spectrol Potentiometer

Figure 8: Potentiometer and Wheel

PC-HC6811 Feedback Program

In troubleshooting Val's code, which is all written in assembly, I found it difficult to see what was going on with its sensors, and whether it was executing its code properly. To help with this problem, I wrote a Windows 95 based program in Visual Basic that cooperated serially with Val.

The PC program, PC-HC6811 Link, allows the user to request to see any variety of memory locations in Val real time. The information is displayed real time in three possible ways: as a graph, as a numerical value (in hex or decimal), or as an "LED." The user can change the memory requests at any time, and the sessions can be saved to disk as desired. An example session is shown as figure 9.

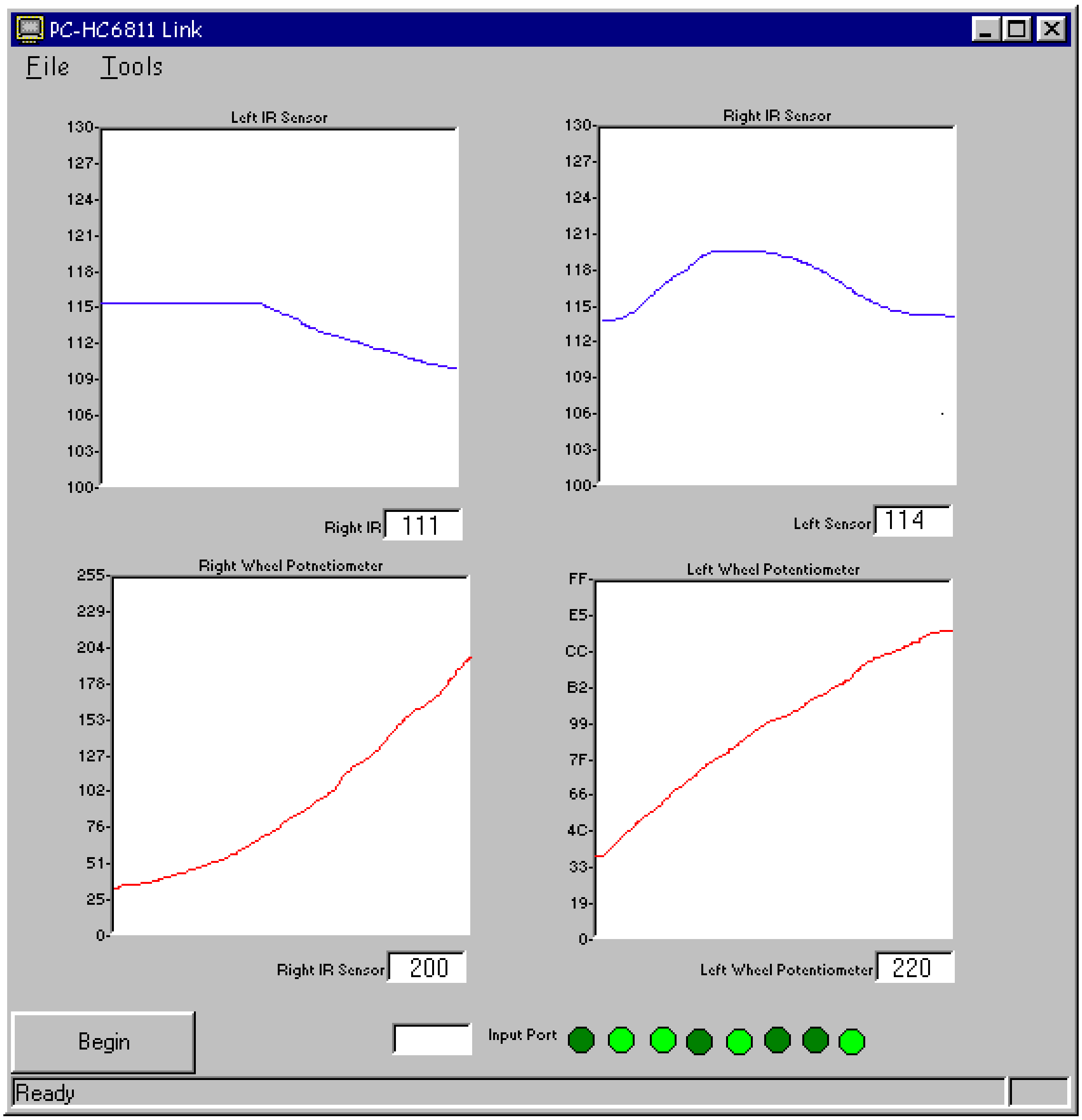


Figure 9: PC-HC6811 Link Program

Docked Computer Communication Link and Recharger

When Val docks, it makes five electrical connections: Ground, RS232 Transmit, RS232 Receive, +6.3V Recharge, and +9.6V Recharge – see Fig. 10. With these connections, Val recharges both of its battery packs while maintaining a serial communication connection with a computer. To recharge the batteries, I used the transformers that were made to recharge them. For the connectors I used two headphone type connectors and one probe type between them.

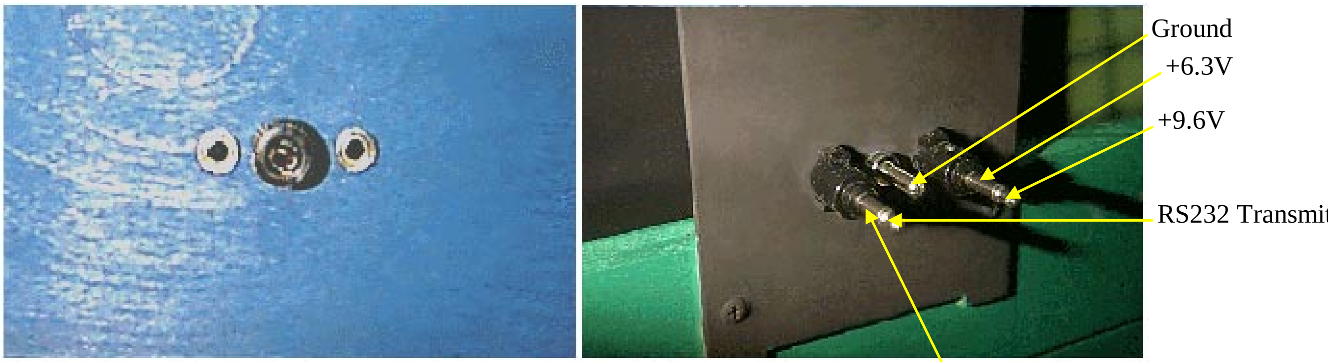


Figure 10: Docking Electrical Connectors. Val (Left) and Docking Station (Right)

Remote Control Link

To communicate with Val while in its recording mode, I decided to use the same serial port used by the docking station. The advantage of using a serial connection is the ease with which I can add new commands for Val to receive—no additional hardware is required. I wrote a program for my HP48g calculator that allowed me to send the serial commands (see fig. 11).

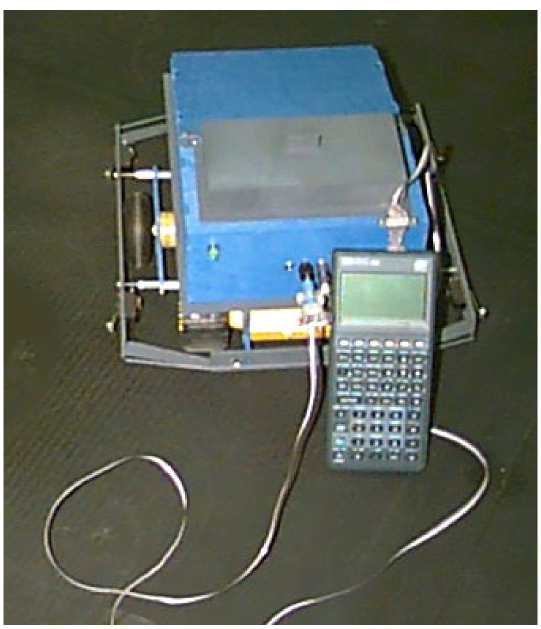


Figure 11: Remote Controller Linked to Val

Behaviors

Record

When Val is in Record mode, its behavior is simply to wait for commands. Then, when a command is received, it is recorded in a table. If the instruction involves movement, then Val begins keeping track of the distances traveled by the potentiometers and continues until a new command is given. Then, the distances traveled are recorded and the new instruction is handled. Below is a list of the current possible commands Val responds to.

- -Go Forward
- -Turn left
- -Turn Right
- -Stop
- -Follow Left Wall
- -Follow Right Wall
- -Wait for Bump (This is for user interaction. Val waits to be bumped)
- -Dock (Val assumes the docking station is some where close behind it)

In terms of how the commands are represented, each command is one byte long. This makes possible 256 commands. Each distance measurement is two bytes long: one byte that is the number of pot revolutions and the other byte for the final pot position. I only use one pot to measure at a time and usually the one that is moving forward. For example, I use the left pot to measure forward and right turn movements and the right pot to measure left turns.

Execution

The Execution mode is very similar to the record mode. It is assumed that the instructions to be executed have already been downloaded. Val executes each instruction one after another as they are read. If the instruction involves a certain distance requirement—like go straight for 5 and a half revolutions of the left potentiometer—then Val will read the requirement and will not move to the next instruction until the current one is satisfied. In this way each instruction is played back as it was recorded.

Docking

In the Docking mode, Val assumes the docking station is somewhere closely behind it. Because the docking station is shaped in a "Y," it slowly leads Val to the connectors in an iterative trial-and-error process. Val begins with moving backward, and continues backward until either the back left or right bumper is hit, all four side switches are closed, or a timeout occurs. If the back left bumper is hit, then Val moves a little forward, turns right, and tries again. Likewise, if the back right bumper is hit, then Val moves forward, turns left, and starts backing again. If all four side-switches close, then Val has worked its way into the narrow part of the docking station and is almost connected. When this happens, Val continues backing until both back bumper switches close. This signifies that Val is connected—docking is complete. If, however, a timeout occurs in any of these steps, then Val assumes it is stuck and moves forward, turns a little, and tries again.

Wall Following
For the wall following behavior, the roller switches on each side were very useful. I can simply bias the appropriate motor and Val veers to the appropriate side until a wall is reached. When a wall is reached, then Val straightens out against the wall, using the roller switches as a guide, and the bias helps keep Val against the wall. While this goes on, Val also measures the distance traveled just as if it was going straight.

Computer Communication/Voice Recognition Response

I wrote a Visual Basic program called *Val Link* that would work with Microsoft Voice. This program is launched by a batch file with one of two possible command line parameters. If you pass it "-r" then goes into record mode. Val is sent a special character to see if it is already online. If it is, then Val will respond with a special character. This allows the PC program to immediately command Val to begin the Record behavior. If Val does not respond, then the PC program waits until Val is turned on, then it downloads the main code to Val and starts it running. Then the command to begin recording is sent. After the command to record is sent, the PC program waits for Val to return. When Val returns, another set of special characters is exchanged and Val begins to send the recorded instructions to the computer. When the exchange is complete, the PC program asks for a filename by which to save the recorded set of instructions. Then it asks for a voice command by which these instructions will be executed. Finally, a batch file is created by the name of the voice command given. The batch file is stored in a special folder set aside by Microsoft Voice. Files in this folder are executed when their names are detected by Voice. The batch file is written to simply call the PC program with the record filename as its parameter. The second parameter type accepted by the PC program then is a filename that is the name of a text file containing the commands Val is to execute. When a filename is passed, the PC program again detects if Val is online—sending the main program if it is not. When Val is online, the list of commands are sent and stored in Val. When that it done, Val is commanded to begin execution.

Recharging

Recharging isn't much of a behavior for Val, because it is really a function of its docked connection. However, it warrants mentioning. Val recharges both battery packs while docked. The circuitry used to recharge the batteries are those that came with the battery packs.

Experimental Results

To test Val, I simply walked it through a simple recording procedure: go straight a while, stop, turn left about 180 degrees, go straight a while, turn right 180 degrees, and dock.

Then, Val would replay these steps by itself. Doing this several times, Val was mostly successful. Occasionally, Val's potentiometers would slip and it would get off course a little. The cleaner the floor, the less this happened. I also did some tests of wall following which showed good success. The only problem I had was knowing when to command the wall following to begin. If Val was too far from the wall when the command was given, then it would drift right/left too much and bump straight into the wall. This could be overcome, however, with more intelligent code.

Conclusion

Most of the original goals for Val were accomplished. Docking, wall following, recording, dead reckoning, and voice activation were all a great success. Of course, Val is only suited for tiled floors and rooms without steps, but that was an expected limitation. I did not have time to concentrate on monitoring whether Val was going straight or not as I intended to. However, I found that Val's motors were rather consistent; and, after some trial-and-error measurements, I was able to balance its motors so that it moved straight forward rather well. Also, I have not yet enabled Val to avoid obstacles in its recorded path. The ideas that made Val successful, however, I would recommend to anyone attempting to solve the same problems. The docking station design coupled with the rolling bump switches was a very reliable and robust concept. Of course, care must be take to ensure the robot fits the docking station snuggly, but otherwise it is simple and easy. I would also recommend the side-mounted potentiometers carrying the free-spinning wheels for dead reckoning. The fact that the potentiometers were not mounted to the actual drive track wheels helped eliminate a lot of potential error through slippage. The wheels that drive a vehicle will always slip, because they have to propel the vehicle by the force of friction against the floor—there will be slippage. Therefore, they are more unreliable to measure distances traveled by the robot than wheels whose purpose alone it to measure that. I would also recommend Microsoft Voice (which can be found at www.research.microsoft.com) to anyone wanting to deal with simple-command voice recognition on a PC, and say that the hp48g calculator serves as an excellent remote control. Lastly, I would recommend a reliable method for feedback. My program, which graphed in real time what my robot was seeing, saved me a ton of debugging time. Anyone interested in this program can contact me at daniel@mil.ufl.edu. Finally, I would like to thank all those whose help and support enabled me to finish this project—especially my wife, Avery Suzanne.

Appendix

Assembly Code

```
********************************************************************************
*CONSTANTS                                                         *
********************************************************************************
ADCTL           EQU     $1030
ADR1            EQU     $1031
ADR2            EQU     $1032
ADR3            EQU     $1033
ADR4            EQU     $1034
BAUD            EQU     $102B
BPROT           EQU     $1035
CFORC           EQU     $100B
CONFIG          EQU     $103F
COPRST          EQU     $103A
DDRC            EQU     $1007
DDRD            EQU     $1009
EPROG           EQU     $1036
HPRIO           EQU     $103C
OC1D            EQU     $100D
OC1M            EQU     $100C
OPTION          EQU     $1039
PACNT           EQU     $1027
PACTL           EQU     $1026   ; RTI Timer control
PORTA           EQU     $1000
PORTB           EQU     $1004
PORTC           EQU     $1003
PORTCL          EQU     $1005
PORTD           EQU     $1008
PORTE           EQU     $100A
PPROG           EQU     $103B
SCCR1           EQU     $102C
SCCR2           EQU     $102D
SCSR            EQU     $102E
SCDR            EQU     $102F
SPCR            EQU     $1028
SPDR            EQU     $102A
SPSR            EQU     $1029
TCNT            EQU     $100E
TCTL1           EQU     $1020
TCTL2           EQU     $1021
TFLG1           EQU     $1023
TFLG2           EQU     $1025
TIC1            EQU     $1010
TIC2            EQU     $1012
TIC3            EQU     $1014
TIC4            EQU     $101E
TMSK1           EQU     $1022
TMSK2           EQU     $1024   ; RTII enable flag
TOC1            EQU     $1016
TOC2            EQU     $1018
TOC3            EQU     $101A
TOC4            EQU     $101C
TOC5            EQU     $101E

********************************************************************************
*ISR_VECTORS                                                       *
```

```
*****************************************************************************
                         ORG      $00EB
                         JMP      RTI_ISR
                         org      $00c4
                         jmp       ISR_SCI

*****************************************************************************
*VARIABLES                                              *
*****************************************************************************
                         ORG    $2000
                         JMP      INITIALIZE
LEFT_IR                  FCB      0
RIGHT_IR                 FCB      0
RIGHT_POT                FCB      0
LEFT_POT                 FCB      0
OLD_LEFTPOT              FCB      0
OLD_RIGHTPOT             FCB      0
SAVE_LEFT_ROT            FCB      0
SAVE_LEFT_CNT            FCB      0
LOVERFLAG                FCB      0
SaveA000                 FCB      0
DO_POINT                 FDB      DO_TABLE
TEMP                     FCB      0
SAVE_RIGHT_ROT           FCB      0
SAVE_RIGHT_CNT           FCB      0
ROVERFLAG                FCB      0
RTI_CNT                  FCB      0
MODE                     FCB      0
DOCK_COUNT               FDB      0
SAVE_STUCK               FCB      0
STUCK_ADDR               FDB      0
IR_MODE                  FCB      0
TEMP_IR                  FDB      0
SPEED_WAIT               FCB      0
LAST_LEFT_CNT            FCB      0
LAST_LEFT_ROT            FCB      0
LAST_RIGHT_CNT           FCB      0
LAST_RIGHT_ROT           FCB      0
RIGHT_SPEED              FCB      0
LEFT_SPEED               FCB      0
SPEED_FLAG               FCB      0

*****************************************************************************
*INITIALIZATION                                      *
*****************************************************************************

INITIALIZE               LDS      #$0041
                         LDAA     #1
                         STAA     OLD_LEFTPOT
                         STAA     OLD_RIGHTPOT
                         LDAA     #EDGE_MODE
                         STAA     TCTL1
                         LDAA     #OC1M_MASK
                         STAA     OC1M
                         LDAA     #OC1D_MASK_SP
                         STAA     OC1D
                         LDD      #$8000
                         STD      RIGHT_FWD_PWM
                         LDD      #$7800
                         STD      LEFT_FWD_PWM
                         LDD      #0
```

```
                STD     TOC1
                STD     TOC4
                STD     TOC3
                STD     DOCK_COUNT
                CLR     SaveA000
                CLR     MODE
                LDD     #DO_TABLE
                STD     DO_POINT
                JSR     InitAtD
                JSR     InitRTI
                JSR     InitSCI
                JSR     TurnLEDOn

***************************************************************************
*MAIN PROGRAM                                                            *
***************************************************************************

MAIN            CLI
COM_WAIT        JSR     WAIT_FOR_COM
                CMPA    #'G'
                BEQ     EXEC_RECORD
                CMPA    #'R'
                BEQ     JUMP_RECORD
                CMPA    #'H'
                BEQ     SEND_IM_HERE
                BRA     COM_WAIT

HERE            JSR     CHKIR
                JSR     STRCHK
                BRA     HERE

JUMP_RECORD     JMP     RECORD_MODE

SEND_IM_HERE    LDAA    #%00110101
                STAA    BAUD
                LDAA    #%00001100
                STAA    SCCR2
                LDAA    #'H'
                JSR     OutChar
                JSR     InitSCI
                JMP     COM_WAIT
***************************************************************************
*WAIT FOR COMMAND                                              *
***************************************************************************
WAIT_FOR_COM    JSR     STRCHK
                LDAA    MODE
                BEQ     WAIT_FOR_COM
                CLR     MODE
                RTS

***************************************************************************
*EXECUTE RECORD TABLE                                                    *
***************************************************************************
FWD_STOP        EQU     0
FWD_GO          EQU     1
RGHT_GO         EQU     2
LFT_GO          EQU     3
BWAIT           EQU     6


NOW_LWF         JSR     SET_GO_STRAIGHT
```

```
                    LDD     #$B000
                    STD     RIGHT_FWD_PWM
                    LDD     #$4800
                    STD     LEFT_FWD_PWM
                    LDAA    0,X
                    INX
                    LDAB    0,X
                    INX
                    JSR     FORWARD_START
                    JSR     GO_TIL_STR
                    JSR     FORWARD_STOP
                    JMP     NOW_BACK

NOW_RWF             JSR     SET_GO_STRAIGHT
                    LDD     #$5000
                    STD     RIGHT_FWD_PWM
                    LDD     #$B000
                    STD     LEFT_FWD_PWM
                    LDAA    0,X
                    INX
                    LDAB    0,X
                    INX
                    JSR     FORWARD_START
                    JSR     GO_TIL_STR
                    JSR     FORWARD_STOP
                    JMP     NOW_BACK

NOW_BUMP_WAIT       JSR     BUMP_WAIT
                    BRA     NOW_BACK


EXEC_RECORD         JSR     UNDOCK
DO_TBLE_LOOP        LDX     DO_POINT
                    LDAA    0,X
                    INX
                    CMPA    #7
                    BEQ     NOW_LWF
                    CMPA    #8
                    BEQ     NOW_RWF
                    CMPA    #0
                    BEQ     NOW_FWDSTOP
                    CMPA    #1
                    BEQ     NOW_FORWARD
                    CMPA    #2
                    BEQ     NOW_RIGHT
                    CMPA    #3
                    BEQ     NOW_LEFT
                    CMPA    #4
                    BEQ     NOW_STRAIGHT
                    CMPA    #5
                    BEQ     NOW_IR_FOLLOW
                    CMPA    #6
                    BEQ     NOW_BUMP_WAIT
                    JSR     DOCK
                    JMP     INITIALIZE
NOW_BACK            STX     DO_POINT
                    BRA     DO_TBLE_LOOP
NOW_FWDSTOP         JSR     FORWARD_STOP
                    BRA     NOW_BACK
NOW_RIGHT           JSR     SET_TURN_RIGHT
                    JSR     SET_TURN_SPEED
```

```
                LDAA    0,X
                INX
                LDAB    0,X
                INX
                JSR     RIGHT_START
                JSR     GO_TIL_RIGHT
                JSR     RIGHT_STOP
                JMP     NOW_BACK
NOW_LEFT        JSR     SET_TURN_LEFT
                JSR     SET_TURN_SPEED
                LDAA    0,X
                INX
                LDAB    0,X
                INX
                JSR     LEFT_START
                JSR     GO_TIL_LEFT
                JSR     LEFT_STOP
                JMP     NOW_BACK
NOW_FORWARD     JSR     SET_GO_STRAIGHT
                JSR     SET_FORW_SPEED
                LDAA    0,X
                INX
                LDAB    0,X
                INX
                JSR     FORWARD_START
                JSR     GO_TIL_STR
                JSR     FORWARD_STOP
                JMP     NOW_BACK

NOW_STRAIGHT    JSR     SET_GO_STRAIGHT
                BRA     NOW_BACK

NOW_IR_FOLLOW   JSR     CHKIR
                LDAA    LEFT_IR
                JSR     IRSTRONG
                TAB
                LSLB
                STAB    TEMP
                LDAA    RIGHT_IR
                JSR     IRSTRONG
                ADDA    TEMP
                CMPA    IR_MODE
                BEQ     NOW_IR_FOLLOW
                STAA    IR_MODE
                CMPA    #0
                BEQ     IR_STOP
                CMPA    #1
                BEQ     IR_RIGHT
                CMPA    #2
                BEQ     IR_LEFT
                CMPA    #3
                BEQ     IR_AHEAD
                BRA     NOW_IR_FOLLOW

IR_STOP         JSR     FORWARD_STOP
                JSR     SET_GO_STRAIGHT
                JMP     NOW_IR_FOLLOW
IR_AHEAD        JSR     FORWARD_STOP
                JSR     SET_GO_STRAIGHT
                JSR     FORWARD_START
                JMP     NOW_IR_FOLLOW
```

```
IR_RIGHT        JSR     FORWARD_STOP
                JSR     SET_TURN_RIGHT
                JSR     RIGHT_START
                JMP     NOW_IR_FOLLOW
IR_LEFT         JSR     FORWARD_STOP
                JSR     SET_TURN_LEFT
                JSR     LEFT_START
                JMP     NOW_IR_FOLLOW



****************************************************************************
*MONITOR IRS AND POTS                                    *
****************************************************************************
STRCHK          PSHA
                PSHB
                JSR     GETPOT
                STAA    RIGHT_POT
                STAB    LEFT_POT
                JSR     LEFT_CHANGE
                JSR     LEFT_CNTR
                JSR     LEFT_ROT
                JSR     RIGHT_CHANGE
                JSR     RIGHT_CNTR
                JSR     RIGHT_ROT
                PULB
                PULA
                RTS

BKCHK           PSHA
                PSHB
                JSR     GETPOT
                STAA    RIGHT_POT
                STAB    LEFT_POT
                JSR     BKLEFT_CHANGE
                JSR     LEFT_CNTR
                JSR     LEFT_ROT
                JSR     BKRT_CHANGE
                JSR     RIGHT_CNTR
                JSR     RIGHT_ROT
                PULB
                PULA
                RTS

LTCHK           PSHA
                PSHB
                JSR     GETPOT
                STAA    RIGHT_POT
                STAB    LEFT_POT
                JSR     BKLEFT_CHANGE
                JSR     LEFT_CNTR
                JSR     LEFT_ROT
                JSR     RIGHT_CHANGE
                JSR     RIGHT_CNTR
                JSR     RIGHT_ROT
                PULB
                PULA
                RTS
```

```
RTCHK               PSHA
                    PSHB
                    JSR     GETPOT
                    STAA    RIGHT_POT
                    STAB    LEFT_POT
                    JSR     LEFT_CHANGE
                    JSR     LEFT_CNTR
                    JSR     LEFT_ROT
                    JSR     BKRT_CHANGE
                    JSR     RIGHT_CNTR
                    JSR     RIGHT_ROT
                    PULB
                    PULA
                    RTS

************************************************************************
*MOTOR DIRECTION ROUTINES                              *
************************************************************************

SET_TURN_RIGHT      PSHA
                    LDAA    SaveA000
                    ORAA    #%00100000
                    ANDA    #%01111111
                    STAA    $A000
                    STAA    SaveA000
                    PULA
                    RTS

SET_TURN_LEFT       PSHA
                    LDAA    SaveA000
                    ORAA    #%10000000
                    ANDA    #%11011111
                    STAA    $A000
                    STAA    SaveA000
                    PULA
                    RTS

SET_GO_STRAIGHT     PSHA
                    LDAA    SaveA000
                    ANDA    #%01011111
                    STAA    $A000
                    STAA    SaveA000
                    PULA
                    RTS

SET_GO_BACK         PSHA
                    LDAA    SaveA000
                    ORAA    #%10100000
                    STAA    $A000
                    STAA    SaveA000
                    PULA
                    RTS


****************************************************************************
*Turn On the LED
*at Port a000
*Requires Memlocation SaveA000 for Data at that port
****************************************************************************
LED_ON              equ     $10
```

```
TurnLEDOn               psha
                        ldaa    SaveA000
                        oraa    #LED_ON
                        staa    $a000
                        staa    SaveA000
                        pula
                        rts
*****************************************************************************
*****************************************************************************
*Turn Off the LED
*at Port a000
*Requires Memlocation SaveA000 for Data at that port
*****************************************************************************
LED_OFF                         equu    $EF
TurnLEDOff              psha
                        ldaa    SaveA000
                        anda    #LED_OFF
                        staa    $a000
                        staa    SaveA000
                        pula
                        rts
*****************************************************************************
*************************************************************************
*LEFT POTENTIOMETER ROUTINES                                    *
*************************************************************************

LEFT_CHANGE     LDAB    #0
                LDAA    OLD_LEFTPOT             ;SEE IF POT HAS CHANGED BY ONE
KOOLJ1          CMPA    LEFT_POT
                BEQ     SEND_1
                CMPB    #22
                BEQ     DUD
                INCB
                INCA
                BRA     KOOLJ1
DUD             LDAA    #0
                RTS
SEND_1          STAA    OLD_LEFTPOT
                TBA
                RTS

LEFT_CNTR       TSTA                            ;IF CHANGE THEN INCREMENT LEFT COUNTER
                BNE     DOTHENEXT
                RTS
DOTHENEXT       TAB
                CLRA
                ADDD    SAVE_LEFT_ROT
                STD     SAVE_LEFT_ROT
                RTS

LEFT_ROT        RTS


BKLEFT_CHANGE           LDAB    #0
                        LDAA    OLD_LEFTPOT             ;SEE IF POT HAS CHANGED BY ONE
KOOLJ2                  CMPA    LEFT_POT
                        BEQ     SEND_2
                        CMPB    #22
                        BEQ     DUD2
                        INCB
```

```
                        DECA
                        BRA     KOOLJ2

DUD2                    LDAA    #0
                        RTS
SEND_2                  STAA    OLD_LEFTPOT
                        TBA
                        RTS

***************************************************************************
*RIGHT POTENTIOMETER ROUTINES                              *
***************************************************************************


RIGHT_CHANGE LDAB   #0
             LDAA   OLD_RIGHTPOT          ;SEE IF POT HAS CHANGED BY ONE
KOOLJ3       CMPA   RIGHT_POT
             BEQ    SEND_3
             CMPB   #22
             BEQ    DUD3
             INCB
             INCA
             BRA    KOOLJ3

DUD3         LDAA   #0
             RTS
SEND_3       STAA   OLD_RIGHTPOT
             TBA
             RTS

RIGHT_CNTR   TSTA                         ;IF CHANGE THEN INCREMENT LEFT COUNTER
             BNE    DOTHENEXT2
             RTS
DOTHENEXT2   TAB
             CLRA
             ADDD   SAVE_RIGHT_ROT
             STD    SAVE_RIGHT_ROT
             RTS

RIGHT_ROT    RTS



BKRT_CHANGE  LDAB   #0
             LDAA   OLD_RIGHTPOT          ;SEE IF POT HAS CHANGED BY ONE
KOOLJ4       CMPA   RIGHT_POT
             BEQ    SEND_4
             CMPB   #22
             BEQ    DUD4
             INCB
             DECA
             BRA    KOOLJ4
DUD4         LDAA   #0
             RTS
SEND_4       STAA   OLD_RIGHTPOT
             TBA
             RTS

*******************************************************************************
*INITIALIZE THE A TO D CONVERTER                  *
*******************************************************************************
```

```
AtD_LOWER       EQU     %00010000
AtD_HIGHER      EQU     %00010100
InitAtD   psha
                ldaa    #%10010000
                staa    OPTION
                ldaa    #AtD_LOWER
                staa    ADCTL
                pula
                rts

*******************************************************************************
*INITIALIZE THE RTI TO 32 MS                                         *
*******************************************************************************
InitRTI   PSHA
                LDAA            #%00000011
                STAA            PACTL
                LDAA            #%01000000
                STAA            TMSK2
                PULA
                RTS

*******************************************************************************
*IR ROUTINES
*******************************************************************************
GETIR           PSHX                            ;GET THE IR READINGS FROM THE AtD
CONVERTER
                ldaa    #AtD_LOWER
                staa    ADCTL
                LDX     #ADCTL

LOOPY           BRSET 0,x %10000000 LDAD
                Bra     LOOPY
LDAD            LDAA    ADR3
                LDAB    ADR4
                PULX
                RTS

IRSTRONG        CMPA    #119            ;TEST VAL IN REG A TO SEE IF CONSIDERED STRONG
                BGE     STRONG
                LDAA    #0
                RTS
STRONG          LDAA    #1
                RTS

CHKIR           JSR     GETIR
                PSHA
                CLRA
                STD     TEMP_IR
                LDAA    #58
                LDAB    RIGHT_IR
                MUL
                ADDD    TEMP_IR
                ADDD    TEMP_IR
                ADDD    TEMP_IR
                ADDD    TEMP_IR
                ADDD    TEMP_IR
                ADDD    TEMP_IR
                LSRD
                LSRD
                LSRD
                LSRD
```

```
                LSRD
                LSRD
                STAB    RIGHT_IR
                PULB
                CLRA
                STD     TEMP_IR
                LDAA    #58
                LDAB    LEFT_IR
                MUL
                ADDD    TEMP_IR
                ADDD    TEMP_IR
                ADDD    TEMP_IR
                ADDD    TEMP_IR
                ADDD    TEMP_IR
                ADDD    TEMP_IR
                LSRD
                LSRD
                LSRD
                LSRD

                LSRD
                LSRD
                STAB    LEFT_IR
                RTS

****************************************************************************
*GET POT VALUES                                                            *
****************************************************************************
GETPOT          PSHX
                ldaa    #AtD_HIGHER
                staa    ADCTL
                LDX     #ADCTL

POTLOOP         BRSET 0,x %10000000 POTLDAD
                Bra     POTLOOP
POTLDAD         LDAA    ADR1
                LDAB    ADR2
                PULX
                RTS

INITPOT         PSHB
                LDAB    #50
                CBA
                BLT     INITPOTDN
                LDAB    #100
                CBA
                BLT     INITPOTDN
                LDAB    #150
                CBA
                BLT     INITPOTDN
                LDAB    #200
                CBA
                BLT     INITPOTDN
                LDAB    #250
                CBA
                BLT     INITPOTDN
                LDAB    #1
INITPOTDN       TBA
                PULB
                RTS
****************************************************************************
```

```
*MOTOR ACTUATION ROUTINES                         *
******************************************************************************
OC_ON_MASK          EQU     %11100000
EDGE_MODE           EQU     %00101000
OC1M_MASK           EQU     %00110000
OC1D_MASK_SP        EQU     %00000000
OC1D_MASK_GO        EQU     %00110000
NM_TIMES_WT         EQU      5
PWD_INC             EQU     1
pwd_inc2            EQU     2*PWD_INC
max                 EQU     $3000
compare_val         EQU     max-pwd_inc2

RIGHT_FWD_PWM       FDB     0
LEFT_FWD_PWM                FDB     0
RIGHT_BKWD_PWM      FDB     0
LEFT_BKWD_PWM       FDB     0
RIGHT_FWD_INC       FDB     0
LEFT_FWD_INC        FDB     0
LEFT_BKWD_INC               FDB     0
RIGHT_BKWD_INC      FDB     0
RIGHT_FWD_CMP               FDB     0
LEFT_FWD_CMP        FDB     0

******************************************************************************
*SLOWLY STOP YOUR FORWARD MOTION                                  *
******************************************************************************
STOP                PSHA
                    PSHB
                    PSHX
                    LDAA    #OC1D_MASK_SP
                    STAA    OC1D
                    LDD     #0
                    STD     TOC4
                    STD     TOC3
                    LDX     #TFLG2
FWD_ST_START  LDAA  #NM_TIMES_WT
FWDSTLOOP           JSR     0,Y
                    BRCLR 0,X %10000000 FWDSTLOOP
                    BCLR    0,X $7F
                    DECA
                    BNE     FWDSTLOOP



FINAL_STP1          PULX
                    PULB
                    PULA
                    RTS

FORWARD_STOP        LDY     #STRCHK
                    JMP     STOP

LEFT_STOP           LDY     #LTCHK
                    JMP     STOP

RIGHT_STOP          LDY     #RTCHK
                    JMP     STOP

BACK_STOP           LDY     #BKCHK
                    JMP     STOP
```

```
******************************************************************************
*SLOWLY START YOUR FORWARD MOTION                                            *
******************************************************************************

START             PSHA
                  PSHB
                  PSHX
                  PSHY
                  LDX     #TFLG2
                  LDAA    #NM_TIMES_WT
FWDSRTLOOP        JSR     0,Y
                  BRCLR 0,X %10000000 FWDSRTLOOP
                  BCLR    0,X $7F
                  DECA
                  BNE     FWDSRTLOOP
                  LDAA    #OC1D_MASK_SP
                  STAA    OC1D
                  LDD     LEFT_FWD_PWM
                  STD     TOC4
                  LDD     RIGHT_FWD_PWM
                  STD     TOC3
                  LDAA    #OC1D_MASK_GO
                  STAA    OC1D
                  PULY
                  PULX
                  PULB
                  PULA
                  RTS

FORWARD_START     LDY     #STRCHK
                  JMP     START

LEFT_START        LDY     #LTCHK
                  JMP     START

RIGHT_START       LDY     #RTCHK
                  JMP     START

BACK_START        LDY     #BKCHK
                  JMP     START



SET_FORW_SPEED    PSHA
                  PSHB
                  LDD     #$8000
                  STD     RIGHT_FWD_PWM
                  LDD     #$7800
                  STD     LEFT_FWD_PWM
                  PULB
                  PULA
                  RTS

SET_TURN_SPEED    PSHA
                  PSHB
                  LDD     #$6800
                  STD     RIGHT_FWD_PWM
```

```
                LDD     #$6400
                STD     LEFT_FWD_PWM
                PULB
                PULA
                RTS

*****************************************************************************
*BUMP SWITCH ROUTINES                                              *
*****************************************************************************
BUMP_PORT     EQU     $A400
SIDE_R_BACK   EQU     %00010000
SIDE_R_FRONT  EQU     %00000010
SIDE_L_FRONT  EQU     %10000000
SIDE_L_BACK   EQU     %00000001
FRONT_LEFT    EQU     %00001000
FRONT_RIGHT   EQU     %00100000
BACK_LEFT     EQU     %00000100
BACK_RIGHT    EQU     %01000000
BACK          EQU     %01000100
FRONT         EQU     %00101000
SIDE_R        EQU     %00010010
SIDE_L        EQU     %10000001
BOTH_SIDES    EQU     %10010011

GET_BACK      LDAA    BUMP_PORT
              ANDA    #BACK
              RTS

GET_FRONT     LDAA    BUMP_PORT
              ANDA    #FRONT
              RTS

GET_SIDE_R    LDAA    BUMP_PORT
              ANDA    #SIDE_R
              RTS

GET_SIDE_L    LDAA    BUMP_PORT
              ANDA    #SIDE_L
              RTS

GET_BOTH_SD   LDAA    BUMP_PORT
              ANDA    #BOTH_SIDES
              RTS

GET_SRB       LDAA    BUMP_PORT
              ANDA    #SIDE_R_BACK
              RTS

GET_SRF       LDAA    BUMP_PORT
              ANDA    #SIDE_R_FRONT
              RTS

GET_SLB       LDAA    BUMP_PORT
              ANDA    #SIDE_L_BACK
              RTS

GET_SLF                 LDAA    BUMP_PORT
              ANDA    #SIDE_L_FRONT
              RTS

GET_BL        LDAA    BUMP_PORT
```

```
                ANDA    #BACK_LEFT
                RTS

GET_BR          LDAA    BUMP_PORT
                ANDA    #BACK_RIGHT
                RTS

GET_FL          LDAA    BUMP_PORT
                ANDA    #FRONT_LEFT
                RTS

GET_FR          LDAA    BUMP_PORT
                ANDA    #FRONT_RIGHT
                RTS

ALL_OPEN        LDAA    BUMP_PORT
                CMPA    #%11111111
                BEQ     ALL_OPEN_CL
                RTS
ALL_OPEN_CL     LDAA    #0
                RTS

BUMP_WAIT       JSR     STRCHK
                JSR     ALL_OPEN
                BEQ     BUMP_WAIT
                RTS

*******************************************************************************
*UNDOCK ROUTINE                                        *
*******************************************************************************
UNDOCK                  PSHA
                PSHB
                JSR     GETPOT
                JSR     INITPOT
                STAA    OLD_RIGHTPOT
                TBA
                JSR     INITPOT
                STAA    OLD_LEFTPOT
                JSR     FORWARD_START
UNDK_LP         JSR     STRCHK
                JSR     ALL_OPEN
                BEQ     UNDK_NXT
                BRA     UNDK_LP
UNDK_NXT        LDAA    #0
                LDAB    #20
                JSR     FORWARD_START
                JSR     GO_TIL_STR
                JSR     FORWARD_STOP
                CLR     SAVE_LEFT_CNT
                CLR     SAVE_LEFT_ROT
                CLR     SAVE_RIGHT_CNT
                CLR     SAVE_RIGHT_ROT
                PULB
                PULA
                RTS
*******************************************************************************
*ADJUST IF STUCK ROUTINE                                       *
*******************************************************************************
ADJ_STUCK       JSR     BACK_STOP
                JSR     SET_GO_STRAIGHT
                LDAA    #0
```

```
                LDAB    #10
                JSR     FORWARD_START
                JSR     GO_TIL_STR
                JSR     FORWARD_STOP
                CLR     RTI_CNT
                LDY     STUCK_ADDR
                JMP     0,Y


*******************************************************************************
*DOCK ROUTINE                                                  *
*******************************************************************************
DOCK            LDD     #$8000
                STD     RIGHT_FWD_PWM
                LDD     #$7800
                STD     LEFT_FWD_PWM
                LDD     #DOCK_LP
                STD     STUCK_ADDR
DOCK_LP         JSR     SET_GO_BACK
                JSR     BACK_START
DOCK_LP2        JSR     BKCHK
                JSR     GET_BR
                BEQ     ADJ_RIGHT
                JSR     GET_BL
                BEQ     ADJ_LEFT
                JSR     GET_BOTH_SD
                BEQ     DOCK_SET
                JSR     STUCK
                BEQ     ADJ_STUCK
                BRA     DOCK_LP2
ADJ_LEFT        JSR     BACK_STOP
                JSR     SET_GO_STRAIGHT
                JSR     FORWARD_START
                LDAA    #0
                LDAB    #50
                JSR     GO_TIL_STR
                JSR     FORWARD_STOP
                JSR     SET_TURN_LEFT
                LDAA    #0
                LDAB    #20
                JSR     LEFT_START
                JSR     GO_TIL_LEFT
                JSR     LEFT_STOP
                CLR     RTI_CNT
                JMP     DOCK_LP
ADJ_RIGHT       JSR     BACK_STOP
                JSR     SET_GO_STRAIGHT
                LDAA    #0
                LDAB    #50
                JSR     FORWARD_START
                JSR     GO_TIL_STR
                JSR     FORWARD_STOP
                JSR     SET_TURN_RIGHT
                LDAA    #0
                LDAB    #20
                JSR     RIGHT_START
                JSR     GO_TIL_RIGHT
                JSR     RIGHT_STOP
                CLR     RTI_CNT
                JMP     DOCK_LP
DOCK_SET        JSR     FORWARD_STOP
```

```
                LDD     #$A000
                STD     RIGHT_FWD_PWM
                LDD     #$A000
                STD     LEFT_FWD_PWM
                LDD     #DOCKDUDE
                STD     STUCK_ADDR
DOCKDUDE        JSR     SET_GO_BACK
                JSR     FORWARD_START
DOCK_NOW_IN     JSR     BKCHK
                JSR     STUCK
                BEQ     DOCK_STUCK
                JSR     GET_BACK
                BNE     DOCK_NOW_IN
                JSR     FORWARD_STOP
                JSR     SET_GO_STRAIGHT
                RTS
DOCK_STUCK      JMP     ADJ_STUCK
******************************************************************************
*GO TIL DISTANCE ROUTINE (FOR THE LEFT POT)(GOING BK)                        *
******************************************************************************
GO_TIL_BK       CLR     SAVE_LEFT_CNT
                CLR     SAVE_LEFT_ROT
GO_TIL_LP3      JSR     BKCHK
                CMPA    SAVE_LEFT_ROT
                BNE     GO_TIL_LP3
                CMPB    SAVE_LEFT_CNT
                BHI     GO_TIL_LP3
                RTS

******************************************************************************
*GO TIL DISTANCE ROUTINE (FOR THE LEFT POT)(GOING STRAIGHT)                            *
******************************************************************************
GO_TIL_STR      CLR     SAVE_LEFT_CNT
                CLR     SAVE_LEFT_ROT
GO_TIL_LP1      JSR     STRCHK
                CMPA    SAVE_LEFT_ROT
                BNE     GO_TIL_LP1
                CMPB    SAVE_LEFT_CNT
                BHI     GO_TIL_LP1
                RTS



******************************************************************************
*GO TIL DISTANCE ROUTINE (FOR THE LEFT POT)(RIGHT TURN)                      *
******************************************************************************
GO_TIL_RIGHT    CLR     SAVE_LEFT_CNT
                CLR     SAVE_LEFT_ROT
GO_TIL_LP       JSR     RTCHK
                CMPA    SAVE_LEFT_ROT
                BNE     GO_TIL_LP
                CMPB    SAVE_LEFT_CNT
                BHI     GO_TIL_LP
                RTS

******************************************************************************
*GO TIL DISTANCE ROUTINE (FOR THE RIGHT POT)(LEFT TURN)                      *
******************************************************************************
GO_TIL_LEFT     CLR     SAVE_RIGHT_CNT
                CLR     SAVE_RIGHT_ROT
```

```
GO_TIL_LP2      JSR     LTCHK
                CMPA    SAVE_RIGHT_ROT
                BNE     GO_TIL_LP2
                CMPB    SAVE_RIGHT_CNT
                BHI     GO_TIL_LP2
                RTS

*****************************************************************************
*CHECK IF STUCK                                              *
*****************************************************************************
STUCK           LDAA    RTI_CNT
                CMPA    #255
                BEQ     IS_STUCK
                LDAA    #1
                RTS
IS_STUCK        LDAA    #0
                RTS
*****************************************************************************
*WAIT ROUTINE                                                *
*****************************************************************************
NUM_TIMES_WT            FCB     0
WAITING                 psha
                        pshx
                        LDX    #TFLG2
                        LDAA   NUM_TIMES_WT
WAIT_LOOP               BRCLR 0,X %01000000 WAIT_LOOP
                        BCLR   0,X $BF
                        DECA
                        bne    WAIT_LOOP
                        pulx
                        pula
                        rts

REC_BUMP                LDAA    #6
                        JSR     REC_INSTR
                        JSR     BUMP_WAIT
                        JMP     RECORD_WAIT

REC_LWF                 LDAA    #7
                        JSR     REC_INSTR
                        LDD     #$B000
                        STD     RIGHT_FWD_PWM
                        LDD     #$4800
                        STD     LEFT_FWD_PWM
                        JSR     FORWARD_START
                        LDAA    #’L’
                        STAA    FOLL_FLAG
                        JMP     RC_FOR_WAIT

REC_RWF                 LDAA    #8
                        JSR     REC_INSTR
                        LDD     #$5000
                        STD     RIGHT_FWD_PWM
                        LDD     #$B000
                        STD     LEFT_FWD_PWM
                        JSR     FORWARD_START
                        LDAA    #’R’
                        STAA    FOLL_FLAG
                        JMP     RC_FOR_WAIT

FOLL_FLAG               FCB     0
```

```
RECORD_MODE          JSR     UNDOCK
                     LDAA    #%00110011
                     STAA    BAUD
RECORD_WAIT          JSR     WAIT_FOR_COM
                     CMPA    #'I'
                     BEQ     REC_BUMP
                     CMPA    #'W'
                     BEQ     REC_LWF
                     CMPA    #'X'
                     BEQ     REC_RWF
                     CMPA    #'L'
                     BEQ     TOGGLE_LIGHT
                     CMPA    #'F'
                     BEQ     REC_FORWARD
                     CMPA    #'S'
                     BEQ     REC_STOP
                     CMPA    #'T'
                     BEQ     REC_LEFT
                     CMPA    #'U'
                     BEQ     REC_RIGHT
                     CMPA    #'D'
                     BEQ     REC_DOCK
                     BRA     RECORD_WAIT

TOGGLE_LIGHT         LDAA    SaveA000
                     ANDA    #LED_ON
                     BEQ     TOGGLE_ON
                     JSR     TurnLEDOff
                     BRA     TOGGLE_DONE
TOGGLE_ON            JSR     TurnLEDOn
TOGGLE_DONE          JMP     RECORD_WAIT

REC_FORWARD          LDAA    #1
                     JSR     REC_INSTR
                     JSR     SET_GO_STRAIGHT
                     JSR     SET_FORW_SPEED
                     JSR     FORWARD_START
RC_FOR_WAIT          JSR     STRCHK
                     LDAA    MODE
                     BEQ     RC_FOR_WAIT
                     JSR     FORWARD_STOP
                     JSR     REC_STR_LEFT
                     JMP     RECORD_WAIT
REC_STOP             JSR     FORWARD_STOP
                     JMP     RECORD_WAIT
REC_LEFT             LDAA    #3
                     JSR     REC_INSTR
                     JSR     SET_TURN_LEFT
                     JSR     SET_TURN_SPEED
                     JSR     LEFT_START
RC_LEF_WAIT          JSR     LTCHK
                     LDAA    MODE
                     BEQ     RC_LEF_WAIT
                     JSR     LEFT_STOP
                     JSR     REC_STR_RIGHT
                     JMP     RECORD_WAIT

REC_RIGHT            LDAA    #2
                     JSR     REC_INSTR
                     JSR     SET_TURN_RIGHT
                     JSR     SET_TURN_SPEED
```

```
                    JSR     RIGHT_START
RC_RGH_WAIT         JSR     RTCHK
                    LDAA    MODE
                    BEQ     RC_RGH_WAIT
                    JSR     RIGHT_STOP
                    JSR     REC_STR_LEFT
                    JMP     RECORD_WAIT

REC_DOCK            LDAA    #255
                    JSR     REC_INSTR
                    LDAA    #%00110101
                    STAA    BAUD
                    LDAA    #%00001100
                    STAA    SCCR2
                    JSR     BUMP_WAIT
                    JSR     DOCK
                    LDAA    #'H'
                    JSR     OutChar
                    LDD     DO_POINT
                    TBA
                    JSR     OutChar
                    LDX     #DO_TABLE
REC_SEND_DATA               CMPB    #0
                    BEQ     REC_SND_DTDN
                    LDAA    0,X
                    JSR     OutChar
                    INX
                    DECB
                    BRA     REC_SEND_DATA
REC_SND_DTDN        LDD     #DO_TABLE
                    STD     DO_POINT
                    JMP     INITIALIZE


*****************************************************************************
*RECORD INSTRUCTION                                  *
*****************************************************************************
REC_INSTR           PSHX
                    LDX     DO_POINT
                    STAA    0,X
                    INX
                    STX     DO_POINT
                    CLR     SAVE_LEFT_CNT
                    CLR     SAVE_LEFT_ROT
                    CLR     SAVE_RIGHT_CNT
                    CLR     SAVE_RIGHT_ROT
                    PULX
                    RTS

REC_STR_LEFT        PSHX
                    LDX     DO_POINT
                    LDAA    SAVE_LEFT_ROT
                    STAA    0,X
                    INX
                    LDAA    SAVE_LEFT_CNT
                    STAA    0,X
                    INX
                    STX     DO_POINT
                    PULX
                    RTS
```

```
REC_STR_RIGHT           PSHX
                        LDX     DO_POINT
                        LDAA    SAVE_RIGHT_ROT
                        STAA    0,X
                        INX
                        LDAA    SAVE_RIGHT_CNT
                        STAA    0,X
                        INX
                        STX     DO_POINT
                        PULX
                        RTS

RTI_ISR                 LDAA            #%01000000
                        STAA            TFLG2                   ;CLEAR RTI FLAG
                        INC             RTI_CNT
                        RTI


***********************************************************************
*Variables
***********************************************************************
SCI_STATE       FCB     0
DMEM            FCB     0
RCV_FLAG        FCB     0
TRANS_FLAG      FCB     0
POKE_TABLE      RMB     4
POKE_point      RMB     2
SCI_TBPT        FDB     SCI_TBLE
SCI_TBLE        RMB     200

***********************************************************************
*Initialization for SCI interrupt service routine
* 1. Init State Variable
* 2. Setup sci system
*
***********************************************************************
InitSCI PSHA                            ; Save contents of A register
        PSHX
    LDAA    #%00110101          ; Set BAUD rate to 300
    STAA    BAUD
    LDAA    #%00000000          ; Set SCI Mode to 1 start bit,
    STAA    SCCR1               ;    8 data bits, and 1 stop bit.
    LDAA    #%00101100          ; Enable SCI Transmitter
    STAA    SCCR2
    clr     SCI_STATE
    LDX     #SCI_TBLE
    STX     SCI_TBPT
    LDX     #POKE_TABLE
    STX     POKE_point
    PULX
    PULA                        ; Restore A register
    RTS                 ; Return from subtoutine

***********************************************************************
*SCI interrupt service routine
*if      (Receive Buffer Full?)
*        then--if (state=download Data Info)
*             then--store byte in table and increment tableincounter
*             else--if (byte="G") then set state to Send
```

```
*                       if (byte="S") then set state to Stop and reset table pointer
*                 if (byte="X") then disable SCI interrupt
*                 if (byte="P") then set state to download Data Info and return
*
*         else--if (Transmit Ready?)
*             then--if (State=Send?)
*                       then--Load Table Address
*                                 Load byte number flag
*                                 if (byte number = single byte)
*                                 then--load address
*                                     load data
*                                           send data
*                                 else--load address
*                                           load double data
*                                           send first byte
*                                           save second byte
*                                           increment table counter
*                       else  if (State=Send2D?)
*                                 then--load saved data
*                                           send data
*                                           set state to Send
*                                 else--Clear Flag and Return
*             else--Return
*Return
***************************************************************************
*STATE DEFINITIONS
*STOPPED                                   0
*SEND                                      1
*RECEIVING DATA INSTRUCTIONS               2
*SEND SECOND PART OF DOUBLE DATA  3
*Receiving Poke Mem                        4
***************************************************************************

ISR_SCI LDAA    SCSR
          BITA    #%00100000
          BEQ     IfTrans
          LDAA    SCDR
          LDAB    SCI_STATE
          CMPB    #2
          BEQ     RecvData
          cmpb    #4
          beq     RecvPoke
          CMPA    #'G'
          BEQ     SEND_START
          CMPA    #'S'
          BEQ     SEND_STOP
          CMPA    #'X'
          BEQ     QUIT_SCI
          CMPA    #'P'
          BEQ     RD_START
          CMPA    #'M'
          beq     PK_START
          CMPA    #'D'
          beq     PK_DO
          LDAA    #1
          Staa    RCV_FLAG
          RTI
RecvDataLDX     SCI_TBPT
          STAA    0,X
          INX
          STX     SCI_TBPT
```

```
                clr     SCI_STATE
                RTI
RecvPoke        LDX     POKE_point
                STAA    0,X
                INX
                STX     POKE_point
                clr     SCI_STATE
                RTI
IfTrans   bra   IfTransReady
SEND_START      LDAA    #1
                STAA    SCI_STATE
                LDX     #SCI_TBLE
                STX     SCI_TBPT
                LDAA    SCCR2
                ORAA    #%10001000
                STAA    SCCR2
                RTI
SEND_STOP       CLR     SCI_STATE
                LDX     #SCI_TBLE
                STX     SCI_TBPT
                LDAA    SCCR2
                ANDA    #%01111111
                STAA    SCCR2
                RTI
QUIT_SCI        LDAA    SCCR2
                ANDA    #%01011111
                STAA    SCCR2
                RTI
RD_START        LDAA    #2
                STAA    SCI_STATE
                RTI
PK_START        LDAA    #4
                STAA    SCI_STATE
                RTI
PK_DO           ldx     #POKE_TABLE
                ldy     0,x
                ldaa    2,x
                staa    0,y
                stx     POKE_point
                rti
IfTransReady    BITA    #%10000000
                BEQ     SCI_NULL
SEEHERE         ldaa    #1
                staa    TRANS_FLAG
                LDAB    SCI_STATE
                CMPB    #1
                BEQ     SSB
                CMPB    #3
                BEQ     SDB
SCI_NULL        RTI
SDB             LDAA  DMEM
                STAA    SCDR
                LDAA    #1
                STAA    SCI_STATE
                RTI
SSB             LDX     SCI_TBPT
                LDAA    0,X
                INX
                CMPA    #'1'
                BEQ     SMB
                CMPA    #'2'
```

```
                BEQ     DMB
                CMPA    #'3'
                BEQ     REGA
                CMPA    #'4'
                BEQ     REGB
                CMPA    #'5'
                BEQ     REGX
                CMPA    #'6'
                BEQ     REGY
                CMPA    #'7'
                BEQ     TABLE_END
                RTI
TABLE_END       LDX     #SCI_TBLE
                STX     SCI_TBPT
                JMP     SEEHERE
SMB             LDY   0,X
                LDAA    0,Y
                INX
                INX
                BRA     SCI_DONE
DMB             LDY     0,X
                LDD     0,Y
                INX
                INX
                STAB    DMEM
                LDAB    #3
                STAB    SCI_STATE
                clr     TRANS_FLAG
                BRA     SCI_DONE
REGA            TSY
                LDAA    2,Y
                BRA     SCI_DONE
REGB            TSY
                LDAA    1,Y
                BRA     SCI_DONE
REGX            TSY
                LDD     3,Y
                STAB    DMEM
                LDAB    #3
                STAB    SCI_STATE
                clr     TRANS_FLAG
                BRA     SCI_DONE
REGY            TSY
                LDD     5,Y
                STAB    DMEM
                LDAB    #3
                STAB    SCI_STATE
                clr     TRANS_FLAG
SCI_DONE        STAA    SCDR
                STX     SCI_TBPT
                RTI

***************************************************************************
*                SUBROUTINE -  OutChar
* Description: Outputs the character in register A to the screen after
*           checking if the Transmitter Data Register is Empty
* Input       : Data to be transmitted in register A.
* Output      : Transmit the data.
* Destroys    : None.
* Calls       : None.
***************************************************************************
```

```
OutChar         PSHB                            ; Save contents of B register
                PSHX
                LDX       #SCSR
Loop1           BRSET 0,x %10000000 READY       ; Check status reg (load it into B reg)
                BRA       Loop1                 ; Wait until empty
READY                STAAA     SCDR                  ; A register ==> SCI data
                pulx
                PULB                            ; Restore B register
                RTS                             ; Return from subtoutine

          org   $6000

DO_TABLE  FCB   1,255,5,255
```