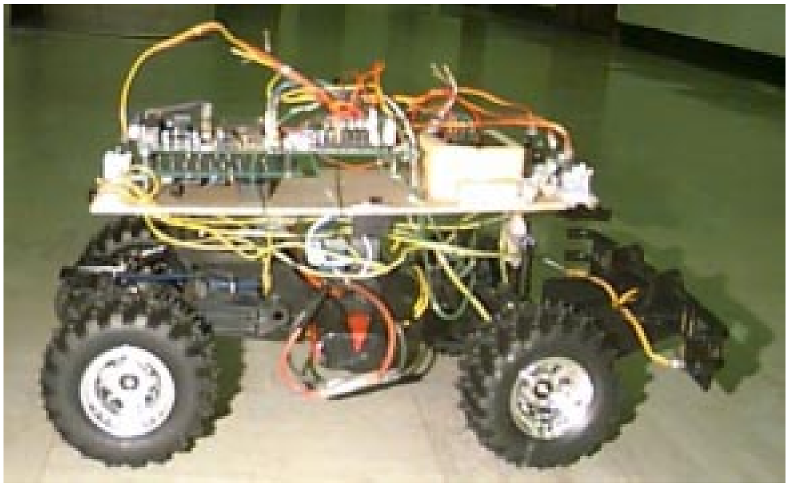
Speedy



by
Josue Peña

Keith L. Doty
EEL 5666
Intelligent Machines Design Laboratory

University of Florida
December 11, 1997

Table Of Contents

Abstract

Following is a discussion of the development of an obstacle avoidance system for an autonomous high speed RC car using infrared emitters and sensors. The car, known as Speedy, has two basic modes of operation. Speedy can operate in autonomous mode or full radio control mode. Speedy's high speed requires unique circuitry to power the high current motor. In addition the radio control mode is complicated by RF noise produced by the motors and the micro-controller which dictates Speedy's behavior.

Executive Summary

Speedy is an autonomous car designed to travel at high speeds. A powerful rear motor and differential gears provide good traction at high speeds. This motor is controlled and powered by a special high current electronic speed control. A high current delivering battery is used to operate at fast speeds with long run time. The original steering mechanism was replaced with a standard servo and torque coupling to provide tighter steering and reliable control.

Speedy's navigational system consists of IR emitters and receivers. Since Speedy will travels at high speeds, obstacle avoidance becomes a challenge. Upon detecting an obstacle Speedy will reduce its speed while adjusting the steering to avoid the obstacle. Speedy will reverse the motor for a fraction of a second if necessary to slow down. The integrated IR system can provide 140 degrees of frontal visibility up to 24 inches ahead. Even though an IR sensor is located on the back of Speedy, rear visibility is not a priority since Speedy will generally travel in the forward direction.

Speedy can also run in full radio control mode. Upon receiving valid RF signals from an FM transmitter, Speedy will switch from autonomous mode to full radio control mode. Upon losing the RF signal Speedy will switch back to full autonomous mode. Speedy can switch between both modes as often as desired.

Speedy uses the Motorola 68HC11 micro-controller to analyze all sensors and then control the speed and direction accordingly.

Introduction

My primary goal for Speedy is to be able to switch to radio control automatically and then maintain radio control at a reasonable distance. Prior RC designs have been unsuccessful due to EMF noise produced primarily by the Motorola 68HC11 micro-controller. Speedy can operate in two modes which are selected automatically. Mode one is full RC control. One will be able to control Speedy’s movements from a far distance (about 50 feet). Mode two is full autonomous mode. In this mode Speedy will roam around the room avoiding obstacles looking busy. Speedy will increase speed whenever he detects that there is enough room to navigate in safely without crashing.

Following are separate sections describing Speedy. The *Integrated System* section gives an overview of the separate systems which makes Speedy come to life. The *Mobile Platform* section describes the physical structure of Speedy and what makes it moves around. The *Actuation* section discusses the steering mechanism and throttle mechanism. The *Sensors* section lists and describes the different sensors used in determining Speedy’s direction and speed of travel. The *Behaviors* section describes the behaviors and modes which Speedy exhibits. The *Experimental Layout and Results* section discusses the experiments and results conducted in getting Speedy to work properly. The *Conclusion* section is a brief summary of what was accomplished and what was not accomplished. The *Documentation* section lists the books and papers used as a reference or guide while designing Speedy. Finally, the Appendix A contains the behavioral code which makes Speedy run around looking important.

Integrated System

Speedy's control system is a rather simple one as shown in Figure 1 below. Speedy first initializes the control of the steering and motor. This initialization makes sure Speedy does not ram into an obstacle at startup. Speedy then cycles through four processes continuously. The first process calculates which direction Speedy should go. The second process determines at what speed to travel in. The third process determined whether to be in autonomous or full radio control mode. Finally, the fourth process is the arbitrator which analyzes all the information generated by the other processes and determines what Speedy will actually do.

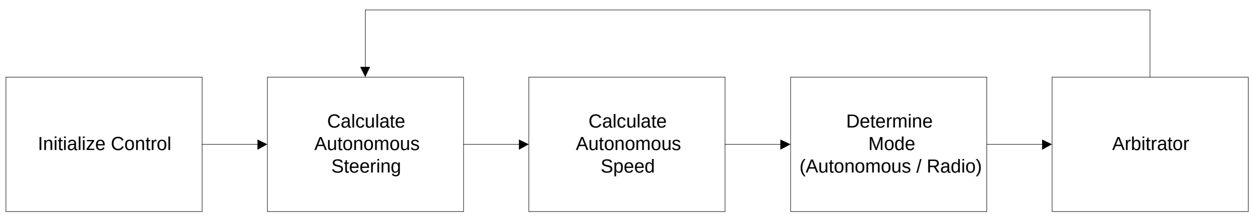


Figure 1: Control Flow

The autonomous steering and the autonomous speed data are calculated directly from the IR sensor readings. When the arbitrator is invoked it actuates the steering and throttle mechanisms which moves the motors. When in radio control mode the arbitrator simply passes the signals coming in from the receiver to the steering servo and the electronic speed control.

Mobile Platform

Due to Speedy's fast speed a strong and crash resistant platform is needed. The "Black Phantom II" RC truck by Radio Shack was therefore used. The rear bumper was removed and the top half of the front bumper was removed. Only the frame, wheels assembly with shocks, front bumper, and rear motor assembly were kept. The steering steeper motor was removed because it did not provide the proportional steering needed for good obstacle avoidance. The battery and battery casing were removed because it was too small to provide enough power for a successful run. The front bumper was trimmed low in order to increase the IR sensor range. The rear bumper was removed because it was too heavy and dragged occasionally slowing Speedy down. Finally the RF electronics were removed because they were cheaply designed and very prone to interference. After many crashes (mostly accidental) Speedy's frame has proven to be very reliable and sturdy. The excellent forward and floating rear suspension maintains Speedy from flipping over is mostly any driving condition. Since Speedy does not have a rear bumper a rear collision can cause damage to the rear motor assembly. This is not a major concern since Speedy mostly travels in a forward direction and is allowed to travel only slowly in reverse.

Actuation

Front Steering Mechanism

Speedy's original steering mechanisms was very problematic. The wheels were turned left or right by applying a voltage to the appropriate coil on the steeper motor. The range of motion was very limited and very hard to control in this design. The steeper motor was replaced with a standard servo form the IMDL lab. A couple modifications to the car body were needed to accommodate and secure the new servo. The torque coupling from the old motor was removed and attached to the new servo. This torque coupling allows the wheels to give a little if the turn is too tight. This has a couple of advantages. First, if the wheels are stuck then the servo won't overheat trying to turn the wheels. The servo will simply turn and the torque coupling will give a bit. The second advantage is that when making a sharp turn this little bit of give keeps the car from flipping over. The only disadvantage is that the tighter the curve the wider the turn. Even with these modifications Speedy still has a wide turning diameter of 20 inches as shown in Figure 2 below. This wide turning radius is due to Speedy's over sized tires and wide axles.

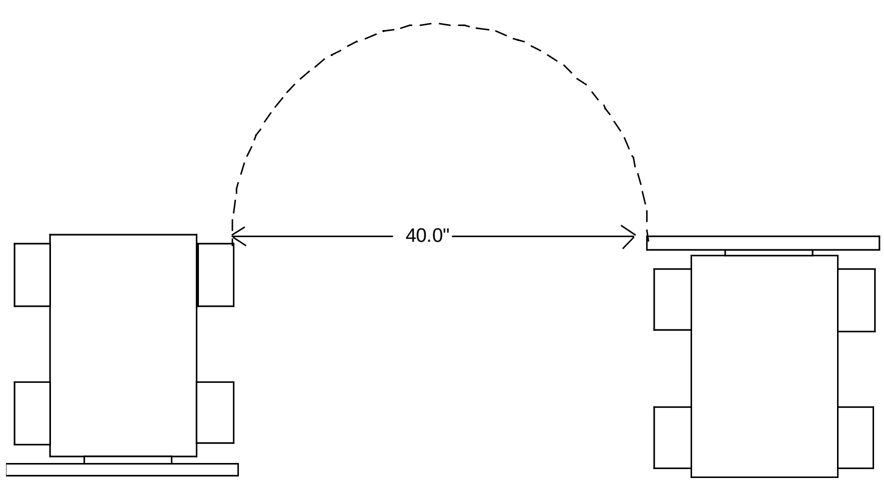


Figure 2: Turning Diameter

Rear Drive Motor

Speedy rear wheels were originally powered by a 9.6 volt bi-directional DC motor. This motor is connected to a differential gear assembly which delivers good traction on sharp turns and different road surfaces. The motor typically draws 0.45 Amps at maximum speed. The stall current of the motor is around 11 Amps. This high stall current requires a special electronic speed control discussed in the sensor section of this report. Even though this motor originally was powered with a 9.6 volt battery pack, it is now powered with a 7.2 volt Piranha battery pack. This was done to meet the electronic speed control input voltage maximum and also because this battery pack can supple twice the original power (1.4 Amp-Hrs). In order to accommodate this larger battery pack the bottom battery compartment was widened and the new battery was fastened using bell wire and electrical tape.

Sensors

Sharp IR Detectors

Speedy uses IR emitters and detectors as it's main form of obstacle avoidance. There are four Sharp IR sensors located on Speedy. Three in front and one in the back as shown in Figure 3 below. Directly below these sensors are four IR emitters calumniated inside a black tube. In addition to these four emitters there are four uncalumniated emitters. Two of the emitters point forward and two point to each side. These last four emitters were added experimentally to improve Speedy's IR blind spots on it's sides. The IR emitters are connected in series with a 330 ohm resistance to the ME11 board which produces a 40kHz signal suitable for the Sharp sensors.

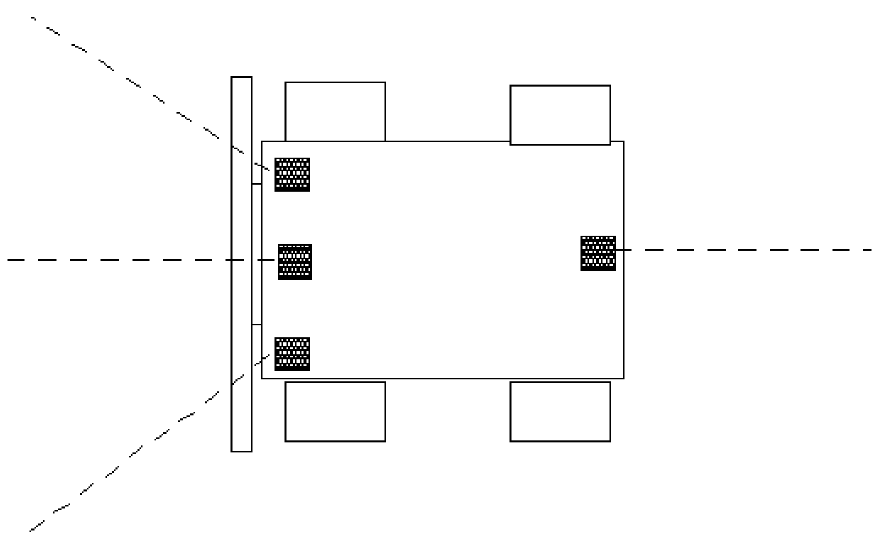


Figure 3: Sharp IR Sensors

These IR emitter/sensor combination gives Speedy 24 inches of obstacle detection. This IR short range is a problem at high speeds because obstacle avoidance is sometimes unavoidable. Future modifications might include ultrasonic emitters and detectors to increase range.

Bump Sensors

Speedy was equipped with a couple of bump sensors behind a fender attached to the front bumper. This enabled Speedy to determine when a low or dark obstacle not detectable by IR was in its path. After a couple of test runs and crashes, the bumper shattered. Therefore, the bump sensor no longer plays a role in obstacle avoidance.

Electronic Speed Control

A Traxxas XL-1 electronic speed control (ESC) is used to in order to power Speedy's high speed motor. This speed control can with stand stall currents of about 75 Amps. This speed control consists mainly of two rows of 8 MOSFETS in parallel. Even thought this electronic speed control (ESC) is very powerful it has one major disadvantage in its control. If this ESC does not receive a 20ms period waveform with a pulse width between 1.2 and 1.8 ms it will start oscillating. Once the ESC starts to oscillate, the motor as well as the speed control begin to overheat. Table 1 below shows how the ESC responds to pulse widths between 1.2ms to 1.8 ms.

Throttle	Pulse Width (ms)	68HC11 FRC Counts
Maximum Forward	1.20	2400
	1.24	2480
	1.28	2560
	1.32	2640
	1.36	2720
	1.40	2800
	1.44	2880
Stopped	1.48	2960
Stopped	1.52	3040
	1.56	3120
	1.60	3200
	1.68	3360
	1.72	3440
	1.76	3520
Maximum Reverse	1.80	3600

Table 1: Throttle Control

The 68HC11 FRC column indicates how long the free running counter on the 68HC11 must run to produce the desired pulse width.

The electronic speed control power line was tapped to power the steering servo and the FM receiver. A resistor is placed in between the electronic speed control and the 68HC11. This resistor serves two purposes. The first purpose of the resistor is to reduce the signal voltage to about 4.2 volts which is the expected signal input voltage of the ESC. The second purpose is to reduce voltage spikes cause by the motor. When observing the signal on the oscilloscope a spike is visible feeding into the 68HC11 output compare pin. After installing this resistor the spike disappeared. With out this resistor the speed control does not work, and will begin to oscillate. This resistor caused two months of trouble shooting and patience to figure out.

Futaba RF Receiver

Speedy originally came with a Radio Shack 27 MHz AM radio and receiver. The maximum control range was about 30 feet. After applying power to the 68HC11 complete control of Speedy was lost. This lost of control was due primarily for two reasons. First, the 68HC11 micro-controller emits a lot of RF/EMF noise which floods out the radio signal. Second, the cheap Radio Shack receiver used, responds to a wide band and doesn’t zero in on the 27MHz signal. A Traxxas 27 AM MHz radio and receiver from an old Grasshopper RC car was then tried. With this receiver Speedy can be controlled from 50 feet. Once again after applying power to the 68HC11 micro-controller 5 second continuous glitches occurred. These glitches usually leads to Speedy ramming into a wall.

Finally a FM model airplane receiver was used. This receiver is a Futaba FP-R127DF FM 7-channel receiver. The crystals were changed from Channel 44 (76.670 MHz) to Channel 52 (72.830 MHz) to eliminate RF noise bled on by local radio stations. The Futaba receiver has a narrow response band of only 20kHz. This receiver gives Speedy an RF range of about 50 feet with very few glitches.

Custom Sensor Interconnections

The major components making up the customs sensors is shown Figure 4 below. The 68HC11 receives data from the electronic speed control and the radio receiver. The rear wheel motor is powered by the electronic speed control which is under direct control of the 68HC11. The steering servo is powered by the radio receiver and is under direct control of the 68HC11 also.

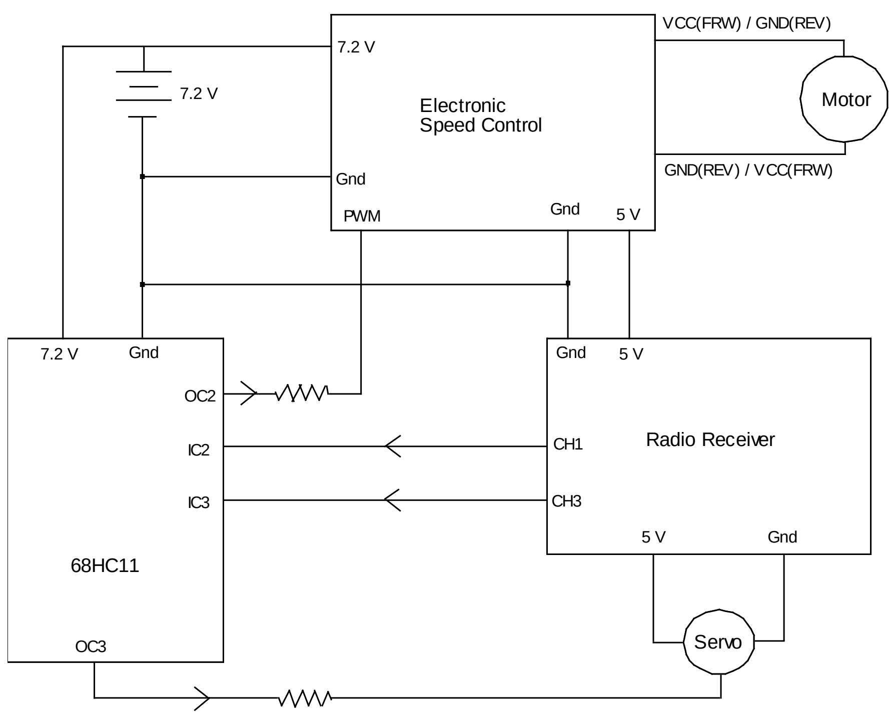


Figure 4: Sensor Wiring

Behaviors

Obstacle Avoidance

Speedy's high speed and high turning radius makes it hard to avoid obstacles. The A/D converters are put in multiple channel scanning mode to provide up to date IR data. Speedy analyzes the difference between the left and right sensors to determine how hard and which way to steer. If the difference is not significant then Speedy will remain in its current course. This prevents Speedy from jerky steering. Due to Speedy's high speed, obstacle avoidance was not easy and not yet full proof.

Maximum Speed Navigation

Speedy relies heavily on the IR sensor reading to determine his speed. Depending on how much headway room there is in front of Speedy he will roam at different speeds. If Speedy is going down a tight passage then he will set his throttle to the slowest speed possible. Setting the speed accordingly will give Speedy enough time to turn and avoid obstacles. If Speedy goes into a tight passage way which is too tight he will simple stop. Since speedy only has one sensor in the back he cannot travel in reverse and get out of a windy situation. However if Speedy is approaching an obstacle at a high speed he will apply the reverse (unless the rear is blocked) to slow him down enough to clear the turn.

Automatic Radio Mode

Speedy is constantly using the RF receiver to monitor the airways for any valid data. Whenever Speedy determines that there is valid RF data coming in he switches over from autonomous mode to full radio control mode. Whenever Speedy stops receiving a good signal he sets a timer and waits for about 30 ms. If another good RF signal does not come in then Speedy switches over to autonomous mode. Due to all the noise produced by the motor and the 68HC11 micro-controller the RF signal coming in varies a bit. Speedy determines if the signal is good by checking the period of the signal and also the duty cycles of the signals coming in. Since this will vary a bit due to noise and other factors such as the transmitter/receiver distance Speedy does not compare it to a fixed number but makes sure it's within certain limits. By adjusting these limits most glitches can be eliminated. If one makes these limits too tight them all glitches will be eliminated but then Speedy will never switch into radio control mode.

Radio Mode with Autonomous Override

Due to time constraints and RF problems I was not able to implement this mode successfully. It's goal was to be able to maneuver Speedy remotely and still leave some autonomous routines. If I were to ram into a wall then Speedy should of either stopped or turned to avoid the wall. Also if I were to travel at a higher speed than Speedy thought was safe to maneuver in, then he would of reduced the speed. The major setback in implementing this mode was in the ability to enable it remotely with the RF radio. Even though the radio/receiver has an extra channel to send data over when I tried to send data, there was too much noise to make sense of it. I believe that this is due to the way I am powering the RF receiver.

Experimental Layout and Results

IR Testing

The first experiment conducted on Speedy was too get some base data for the IR sensors. All the IR emitters were turned on and then an object was placed in front of each of the IR sensors at different distances ranging from 24 inches to 0 inches. After the object was placed in front of the emitter and the distance was measured, the appropriate sensor output voltage was measured using the 68HC11's analog-to-digital converter. The data readings from the A/D converter is shown below in Table 2.

Distance	Analog 1 (Left)	Analog 1 (Center)	Analog 2 (Right)	Analog 3 (Rear)
0	129	130	130	130
1	130	130	129	130
2	130	131	128	130
3	129	130	124	130
4	126	127	123	126
5	122	123	119	123
6	118	120	118	119
7	113	117	115	116
8	108	113	112	112
9	104	110	108	107
10	99	106	104	104
11	96	102	100	101
12	93	98	97	98
13	91	96	94	95
14	90	94	92	93
15	88	93	91	93
16	87	92	89	92
17	87	90	88	92
18	86	89	87	91
20	86	88	86	91
21	86	87	86	91
24	86	86	86	91

Table 2: IR Sensor Data

As can be seen by the IR sensor data plot, shown in Figure 5 below, all four of the IR sensors readings have a similar format. Between 5 and 13 inches the sensors are relatively linear. Below 5 inches and above 13 inches the sensors begin to exponentially saturate at a given voltage value.

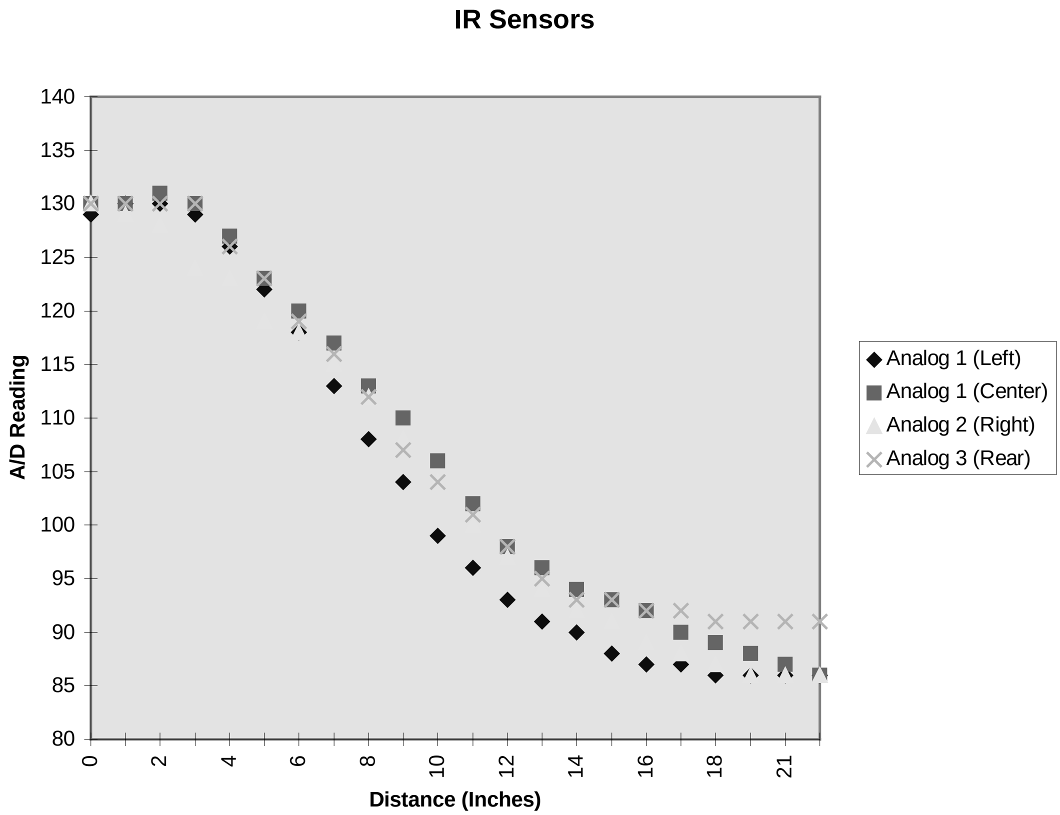


Figure 5: Analog IR Readings

Different resistors were tried in series with the IR emitters. Resistors ranging from 10 ohms to 500 ohms were used. After testing various resisters a resistance of 330 ohms produced the maximum range. For some reason, resistors greater than 330 ohms seemed to decrease the range of the IR emitter/sensor combination. This lack of resistance in the latch output introduced some noise in the A/D system which hindered consistent readings. Therefore, a 330 ohm resistor pack was placed on the latch output for the IR emitters.

RF Testing

Testing the RF components was a bit tricky. The first test was to reduce RF noise emissions. This was done mostly by trial and error. Every time a new cable was added, the RF signal was observed on the oscilloscope and checked for noise. Simple things such as long power wires caused noise in the RF signal. Another thing that eliminates RF noise was using shielded wire to power the 68HC11. After much of the RF noise was eliminated, observation of the actual RF signal was done. Even though the signal appears pretty constant on the oscilloscope, as the distance between Speedy and the RF transmitter changes the RF signal varies slightly. Different periods and pulse widths were recorded as Speedy moved along in radio control mode. Even though an ideal RF signal has a period of 20 ms and pulse width between 1 ms and 2 ms this is not usually the case. Therefore waveforms with a period of 20 ms $\pm$ 0.05 ms and pulse widths between 2.025 ms and 0.975 ms are considered good and valid.

Electronic Speed Control (ESC) Testing

The electronic speed control had one problem which took a lot of time to figure out. Even though the ESC takes in a standard servo signal, it did not work and began to oscillate. After many trials and error I determined that a resistor between the 68HC11 output compare and the ESC was needed. Once again different resistor values were tried. Even though a resistance value of 3000 ohms stabilized the ESC and made it usable when I observed the output compare signal on the oscilloscope I noticed a faint voltage spike on the center of the waveform. When I increased the resistance to 6600 ohms the voltage spike went away. As a result of the added resistance, when the motor is stopped is does not turn on or off as harsh and noisy as it did with the 3000 ohm resistor.

Conclusion

Building Speedy has taught me many things. The most important lesson I learned is the difference between theory and reality. In a theoretical or ideal world it's easy to design a robot. In an actual world sensors, motors, and devices work very differently. All the non-ideal characteristics of motors and sensors complicate the designing of a robot. Speedy is now a high speed autonomous car which avoids obstacles at high speeds. In addition to begin fully autonomous Speedy can switch automatically to radio control mode upon receiving a valid RF signal.

The current distance limitations of the IR sensors limit Speedy's performance in obstacle avoidance. Future designs of this robot will include more powerful proximity sensors such as ultrasonic transducers. Future design of this robot will also include a third mode which mix autonomous mode with radio control mode.

Documentation

Books and Papers

[1] Joseph Jones & Anita Flynn, Mobile Robots: Inspiration to Implementation, A.K. Peters Publishers, Wellesley, MA, 1993.

[2] Gene Miller, Microcomputer Engineering, Prentice-Hall Inc., Englewood Cliffs, NJ, 1993

[3] Fred Martin, “The 6.270 Robot Builder’s Guide”, The MIT Press, c.1992

Appendix A: Behavior Code

```
* 68HC11 Registers
TIC2            EQU     $1012           ; Timer Input Capture 2 Register
TIC3            EQU     $1014           ; Timer Input Capture 3 Register
TOC2            EQU     $1018           ; Output compare 2 register
TOC3            EQU     $101A           ; Output compare 3 register
TOC4            EQU     $101C           ; Output compare 4 register
TCTL1           EQU     $1020           ; Timer control register 1
TCTL2           EQU     $1021           ; Timer control register 2
TMSK1           EQU     $1022           ; Timer mask register
TFLG1           EQU     $1023           ; Timer flag register
TCNT            EQU     $100E           ; Timer Counter Register
SCSR            EQU     $102E           ; Serial communication status register
SCDR            EQU     $102F           ; Serial communication data register
OPTION          EQU     $1039           ; Hardware option control register
ADCTL           EQU     $1030           ; A/D control register
ADR1            EQU     $1031           ; A/D first result register
ADR2            EQU     $1032           ; A/D second result register
ADR3            EQU     $1033           ; A/D third result register
ADR4            EQU     $1034           ; A/D fourth result register
BAUD            EQU     $102B           ; Baud Rate Register
SCCR1           EQU     $102C           ; SCI Control 1 Register
SCCR2           EQU     $102D           ; SCI Control 2 Register

* Masks
BIT0            EQU     %00000001       ; Bit 0 mask used to isolate bit
BIT1            EQU     %00000010       ; Bit 1 mask used to isolate bit
BIT2            EQU     %00000100       ; Bit 2 mask used to isolate bit
BIT3            EQU     %00001000       ; Bit 3 mask used to isolate bit
BIT4            EQU     %00010000       ; Bit 4 mask used to isolate bit
BIT5            EQU     %00100000       ; Bit 5 mask used to isolate bit
BIT6            EQU     %01000000       ; Bit 6 mask used to isolate bit
BIT7            EQU     %10000000       ; Bit 7 mask used to isolate bit
IBIT0           EQU     %11111110       ; Bit 0 inverse mask used to isolate bit
IBIT1           EQU     %11111101       ; Bit 1 inverse mask used to isolate bit
IBIT2           EQU     %11111011       ; Bit 2 inverse mask used to isolate bit
IBIT3           EQU     %11110111       ; Bit 3 inverse mask used to isolate bit
IBIT4           EQU     %11101111       ; Bit 4 inverse mask used to isolate bit
IBIT5           EQU     %11011111       ; Bit 5 inverse mask used to isolate bit
IBIT6           EQU     %10111111       ; Bit 6 inverse mask used to isolate bit
IBIT7           EQU     %01111111       ; Bit 7 inverse mask used to isolate bit

PERIOD          EQU     40000           ; 20ms PWM period for servo & ESC

STRAIGHT_PWM    EQU     3000            ; Straight PWM     (3000)
SOFTLEFT_PWM    EQU     3500            ; Soft Left PWM    (3500)
SOFTRIGHT_PWM   EQU     2500            ; Soft Right PWM   (2500)
MEDLEFT_PWM     EQU     3750            ; Medium Left PWM  (3750)
MEDRIGHT_PWM    EQU     2250            ; Medium Right PWM (2250)
HARDLEFT_PWM    EQU     3875            ; Hard Left PWM    (3875)
HARDRIGHT_PWM   EQU     2125            ; Hard Right PWM   (2125)

STOP_PWM        EQU     3000            ; Stopped PWM                 (3000)
MAXFRW_PWM      EQU     2500            ; Maximum Forward Speed PWM (2500)
MAXREV_PWM      EQU     3500            ; Maximum Reverse Speed PWM (3500)
MEDFRW_PWM      EQU     2650            ; Medium Forward Speed PWM  (2750)
MEDREV_PWM      EQU     3350            ; Medium Reverse Speed PWM  (3250)
SLOWFRW_PWM     EQU     2800            ; Slow Forward Speed PWM    (2900)
SLOWREV_PWM     EQU     3250            ; Slow Reverse Speed PWM    (3150)

T_IR            EQU     87              ; IR Threshold
S_DIFF          EQU     10              ; Small IR difference
M_DIFF          EQU     20              ; Medium IR difference
L_DIFF          EQU     30              ; Large IR difference
```

```
CS              EQU     0               ; Collision straight
CHL             EQU     1               ; Collision hard left
CHR             EQU     2               ; Collision hard right
CML             EQU     3               ; Collision medium left
CMR             EQU     4               ; Collision medium right
CSL             EQU     5               ; Collision soft left
CSR             EQU     6               ; Collision soft right
CSTOP           EQU     0               ; Collision stop
CMAXF           EQU     1               ; Collision maximum forward
CMAXR           EQU     2               ; Collision maximum reverse
CMEDF           EQU     3               ; Collision medium forward
CMEDR           EQU     4               ; Collision medium reverse
CSLOWF          EQU     5               ; Collision slow forward
CSLOWR          EQU     6               ; Collision slow reverse

                ORG     $100
TOTAL           RMB     2

PWMSERVO        RMB     2               ; Pulse width for steering servo
PWMESC          RMB     2               ; Pulse width for electronic speed control
(ESC)
PWMRSERVO       RMB     2               ; Pulse with from radio receiver for steering
PWMRESC         RMB     2               ; Pulse width from radio receiver for ESC

LAST_TIC2       RMB     2               ; Last timer input capture 2 value
LAST_TIC3       RMB     2               ; Last timer input capture 3 value

COLLISION_DIR   RMB     1               ; Collision Direction
COLLISION_SPEED RMB     1               ; Collision Speed

RADIO_MODE      RMB     1               ; Are we in radio controlled mode?
L_TIC2          RMB     2

REVTIME         RMB     1               ; Time to stay in reverse

* Interrupt Vectors

                ORG     $FFE6           ; Output Compare 2 Interrupt Vector
                FDB     OC2ISR

                ORG     $FFE4           ; Output Compare 3 Interrupt Vector
                FDB     OC3ISR

                ORG     $FFE2           ; Output Compare 4 Interrupt Vector
                FDB     OC4ISR

                ORG     $FFEC           ; Input Capture 2 Interrupt Vector
                FDB     IC2ISR

                ORG     $FFEA           ; Input Capture 3 Interrupt Vector
                FDB     IC3ISR

                ORG     $FFFE           ; Reset Vector
                FDB     START

* Start of program

                ORG     $8000           ; Starting RAM Address

START           lds     #$47            ; Initialize stack pointer
                ldaa    #%10100000      ; Set OC2 & OC3 pin to low
                staa    TCTL1           ; on successful compare
                ldaa    #%00000101      ; Initialize IC2 & IC3 for
                staa    TCTL2           ; low-to-high capture
                ldaa    #%01110011      ; Enable OC2, OC3, OC4,
                staa    TMSK1           ; IC2 and IC3 interrupts
```

```
                ldaa    #$30            ; baud = 9600
                staa    BAUD
                clr     SCCR1           ; 1 start 1 stop 8 data bits
                ldaa    #$0c
                staa    SCCR2           ; enable Tx and Rx

                ldaa    #$FF
                staa    $7000           ; Turn on IR emitters

                cli                     ; Turn on interrupts

                ldaa    #BIT7           ; Power-Up A/D using E-clock
                staa    OPTION
                ldaa    #%00110000      ; Scanning multiple channels
                staa    ADCTL           ; START A/D CONVERSION

                jsr     INIT_CONTROL    ; Set initial speed & steering

MAIN            jsr     DIRECTION       ; Check steering for collision
                jsr     SPEED           ; Check speed for collision
                jsr     MODE            ; Check if in Radio control mode
                jsr     ARBITRATOR      ; Carry out steering and throttle
                bra     MAIN            ; Repeat cycle forever

                sei                     ; disable interrupts
                swi                     ; Terminate program execution


********************************************************************************
* SUBROUTINE:   ARBITRATOR                                                     *
* FUNCTION:     Analyzes all information and decides what direction and speed  *
*               robot should go.                                               *
* INPUT:        None.                                                          *
* OUTPUT:       None.                                                          *
* DESTROYS:     None.                                                          *
* CALLS:        None.                                                          *
********************************************************************************
ARBITRATOR      psha                    ; Save register
                ldaa    RADIO_MODE      ; Check if in radio control mode
                beq     AUTONOMOUS

RADIO_CONTROL   ldd     PWMRSERVO       ; Get steering from radio receiver
                std     PWMSERVO        ; Set steering accordingly
                ldd     PWMRESC         ; Get throttle from radio receiver
                std     PWMESC          ; Set throttle accordingly
                bra     RARBITRATOR

AUTONOMOUS      ldaa    COLLISION_DIR
GOSTRAIGHT      cmpa    #CS
                bne     GOHARDLEFT
                jsr     Straight        ; Set steering to straight
                bra     GOSTOP
GOHARDLEFT      cmpa    #CHL
                bne     GOHARDRIGHT
                jsr     HardLeft        ; Set steering to hard left
                bra     GOSTOP
GOHARDRIGHT     cmpa    #CHR
                bne     GOMEDLEFT
                jsr     HardRight       ; Set steering to hard right
                bra     GOSTOP
GOMEDLEFT       cmpa    #CML
                bne     GOMEDRIGHT
                jsr     MedLeft         ; Set steering to medium left
                bra     GOSTOP
GOMEDRIGHT      cmpa    #CMR
                bne     GOSOFTLEFT
                jsr     MedRight        ; Set steering to medium right
```

```
                bra     GOSTOP
GOSOFTLEFT      cmpa    #CSL
                bne     GOSOFTRIGHT
                jsr     SoftLeft        ; Set steering to soft left
                bra     GOSTOP
GOSOFTRIGHT     cmpa    #CSR
                bne     GOSTOP
                jsr     SoftRight       ; Set steering to soft right

GOSTOP          ldaa    COLLISION_SPEED
                cmpa    #CSTOP
                bne     GOSLOWR
                jsr     Stop
                bra     RARBITRATOR
GOSLOWR         cmpa    #CSLOWR
                bne     GOSLOWF
                jsr     SlowRev
                bra     RARBITRATOR
GOSLOWF         cmpa    #CSLOWF
                bne     GOMEDF
                jsr     SlowFrw
                bra     RARBITRATOR
GOMEDF          cmpa    #CMEDF
                bne     GOMAXF
                jsr     MedFrw
                bra     RARBITRATOR
GOMAXF          cmpa    #CMAXF
                bne     RARBITRATOR
                jsr     MaxFrw

RARBITRATOR     pula                    ; restore register
                rts                     ; return from subroutine

********************************************************************************
* Subroutine:   MODE                                                           *
* Function:     Determines whether we are in radio control mode by analyzing   *
*               signals coming out of the radio receiver and checking if it    *
*               is a valid pulse modulated signal.                             *
* Input:        None.                                                          *
* Output:       None.                                                          *
* Destroys:     None.                                                          *
* Calls:        None.                                                          *
********************************************************************************
MODE            psha                    ; Save register
                pshb                    ; Save register
                clr     RADIO_MODE      ; Disable radio mode
                ldd     PWMRSERVO
                cpd     #4050           ; Is the servo pulse to wide?
                bhi     RMODE
                cpd     #1950           ; Is the servo pulse to narrow?
                blo     RMODE
                ldd     PWMRESC
                cpd     #4050           ;Is the ESC pulse to wide?
                bhi     RMODE
                cpd     #1950           ;Is the ESC pulse to narrow?
                blo     RMODE
                ldd     TOTAL           ;Is the total period around 20ms?
                cpd     #39900
                blo     RMODE
                cpd     #40100
                bhi     RMODE
                ldaa    #1              ; Enable radio mode
                staa    RADIO_MODE
RMODE           pulb                    ; restore register
                pula                    ; restore register
                rts                     ; Return from subroutine
```

```
****************************************************************************
* SUBROUTINE:   DIRECTION                                                  *
* FUNCTION:     Read IR sensors and determine whether robot should go straight,*
*               left, or right. This function does not actually move the wheels*
*               but sets a direction variable for later analyzing.         *
* INPUT:        None.                                                      *
* OUTPUT:       None.                                                      *
* DESTROYS:     None.                                                      *
* CALLS:        None.                                                      *
****************************************************************************
DIRECTION       psha                    ; Save register
                pshb                    ; Save register

STRAIGHT_TEST   ldaa    ADR1            ; Read Left IR Sensor
                cmpa    #90             ; Check if path is clear to left
                bhs     HARD_LEFT_TEST  ; Test failed so do next test
                ldaa    ADR2            ; Read Center IR Sensor
                cmpa    #90             ; Check if path is clear in front
                bhs     HARD_LEFT_TEST  ; Test failed so do next test
                ldaa    ADR3            ; Read Right IR Sensor
                cmpa    #90             ; Check if path is clear to right
                bhs     HARD_LEFT_TEST  ; Test failed so do next test
                ldaa    #CS             ; Test passed and therefore path is clear
                staa    COLLISION_DIR   ; Set Collision direction to straight
                bra     RDIRECTION      ; Done with collision direction

HARD_LEFT_TEST  ldaa    ADR3            ; Read Right IR Sensor
                suba    #L_DIFF         ; Large difference in left & right sensors
                cmpa    ADR1            ; Read Left IR Sensor
                bls     HARD_RIGHT_TEST ; Test failed so do next test
                ldaa    #CHL            ; There is a close object on right side
                staa    COLLISION_DIR   ; Set Collision direction to hard left
                bra     RDIRECTION      ; Done with collision direction

HARD_RIGHT_TEST ldaa    ADR1            ; Read Left IR Sensor
                suba    #L_DIFF         ; Large difference in left & right sensors
                cmpa    ADR3            ; Read Right IR Sensor
                bls     MED_LEFT_TEST   ; Test failed so do next test
                ldaa    #CHR            ; There is a close object on left side
                staa    COLLISION_DIR   ; Set Collision direction to hard right
                bra     RDIRECTION      ; Done with collision direction

MED_LEFT_TEST   ldaa    ADR3            ; Read Right IR Sensor
                suba    #M_DIFF         ; Medium diff in left & right sensors
                cmpa    ADR1            ; Read Left IR Sensor
                bls     MED_RIGHT_TEST  ; Test failed so do next test
                ldaa    #CML            ; Somewhat close object on right side
                staa    COLLISION_DIR   ; Set Collision direction to medium left
                bra     RDIRECTION      ; Done with collision direction

MED_RIGHT_TEST  ldaa    ADR1            ; Read Left IR Sensor
                suba    #M_DIFF         ; Medium diff in left & right sensors
                cmpa    ADR3            ; Read Right IR Sensor
                bls     SOFT_LEFT_TEST  ; Test failed so do next test
                ldaa    #CMR            ; Somewhat close object on left side
                staa    COLLISION_DIR   ; Set Collision direction to medium right
                bra     RDIRECTION      ; Done with collision direction

SOFT_LEFT_TEST  ldaa    ADR3            ; Read Right IR Sensor
                suba    #S_DIFF         ; Small difference in left & right sensors
                cmpa    ADR1            ; Read Left IR Sensor
                bls     SOFT_RIGHT_TEST ; Test failed so do next test
                ldaa    #CSL            ; There is an object on right side
                staa    COLLISION_DIR   ; Set Collision direction to soft left
                bra     RDIRECTION      ; Done with collision direction

SOFT_RIGHT_TEST ldaa    ADR1            ; Read Left IR Sensor
```

```
                suba    #S_DIFF         ; Small difference in left & right sensors
                cmpa    ADR3            ; Read Right IR Sensor
                bls     RDIRECTION      ; Test failed so do next test
                ldaa    #CSR            ; There is an object on left side
                staa    COLLISION_DIR   ; Set Collision direction to soft right
                bra     RDIRECTION      ; Done with collision direction

RDIRECTION      pulb                    ; Restore register
                pula                    ; Restore register
                rts                     ; return from subroutine

********************************************************************************
* SUBROUTINE:   SPEED                                                          *
* FUNCTION:     Read IR sensors and determine what speed the robot should be   *
*               going.  Also determines whether it should be going forward or  *
*               reverse.  This function does not actually move the wheels but  *
*               sets a direction variable for later analyzing.                 *
* INPUT:        None.                                                          *
* OUTPUT:       None.                                                          *
* DESTROYS:     None.                                                          *
* CALLS:        None.                                                          *
********************************************************************************
SPEED           psha                    ; Save register
                pshb                    ; Save register

                ldaa    COLLISION_SPEED
                cmpa    #CSLOWR
                bne     STOP_TEST
                inc     REVTIME
                ldaa    REVTIME
                cmpa    #$FFFF
                beq     STOP_TEST
                jmp     RSPEED

STOP_TEST       ldaa    ADR1            ; Read Left IR Sensor
                cmpa    #127            ; Check if path is blocked to left
                blo     MAX_FRW_TEST   ; Test failed so do next test
                ldaa    ADR2            ; Read Center IR Sensor
                cmpa    #127            ; Check if path is blocked in front
                blo     MAX_FRW_TEST   ; Test failed so do next test
                ldaa    ADR3            ; Read Right IR Sensor
                cmpa    #127            ; Check if path is blocked to right
                blo     MAX_FRW_TEST   ; Test failed so do next test
                ldaa    ADR4            ; Read Rear IR Sensor
                cmpa    #127            ; Check if path is blocked to the rear
                blo     MAX_FRW_TEST   ; Test failed so do next test
                ldaa    #CSTOP          ; Test passed and so path is totally blocked
                staa    COLLISION_SPEED ; Set speed to stop
                bra     RSPEED          ; Done with collision speed

MAX_FRW_TEST    ldaa    ADR1            ; Read Left IR Sensor
                cmpa    #90             ; Check if path is clear to left
                bhi     MED_FRW_TEST    ; Test failed so do next test
                ldaa    ADR2            ; Read Center IR Sensor
                cmpa    #90             ; Check if path is clear in front
                bhi     MED_FRW_TEST    ; Test failed so do next test
                ldaa    ADR3            ; Read Right IR Sensor
                cmpa    #90             ; Check if path is clear to right
                bhi     MED_FRW_TEST    ; Test failed so do next test
                ldaa    #CMAXF          ; Test passed and therefore path is clear
                staa    COLLISION_SPEED ; Set speed to maximum
                bra     RSPEED          ; Done with collision speed

MED_FRW_TEST    ldaa    ADR1            ; Read Left IR Sensor
                cmpa    #105            ; Check if path is somewhat clear to left
                bhi     SLOW_FRW_TEST   ; Test failed so do next test
                ldaa    ADR2            ; Read Center IR Sensor
```

```
                cmpa    #105            ; Check if path is somewhat clear in front
                bhi     SLOW_FRW_TEST   ; Test failed so do next test
                ldaa    ADR3            ; Read Right IR Sensor
                cmpa    #105            ; Check if path is somewhat clear to right
                bhi     SLOW_FRW_TEST   ; Test failed so do next test
                ldaa    #CMEDF          ; Test passed and so path is somewhat clear
                staa    COLLISION_SPEED ; Set speed to medium
                bra     RSPEED          ; Done with collision speed

SLOW_FRW_TEST   ldaa    ADR1            ; Read Left IR Sensor
                cmpa    #129            ; Check if path is somewhat blocked to left
                bhi     SLOW_REV_TEST   ; Test failed so do next test
                ldaa    ADR2            ; Read Center IR Sensor
                cmpa    #129            ; Check if path is somewhat blocked in front
                bhi     SLOW_REV_TEST   ; Test failed so do next test
                ldaa    ADR3            ; Read Right IR Sensor
                cmpa    #129            ; Check if path is somewhat blocked to right
                bhi     SLOW_REV_TEST   ; Test failed so do next test
                ldaa    #CSLOWF         ; Test passed and so path is somewhat blocked
                staa    COLLISION_SPEED ; Set speed to slow
                bra     RSPEED          ; Done with collision speed

SLOW_REV_TEST   ldaa    ADR4            ; Read Rear IR Sensor
                cmpa    #127            ; Check if path is clear to rear
                bls     SLOW_REV_1      ; Test failed so do next test
                ldaa    #CSTOP
                staa    COLLISION_SPEED
                bra     RSPEED
SLOW_REV_1      ldaa    #CMEDR         ; Test passed and therefore path is clear
                staa    COLLISION_SPEED ; Set speed to reverse
                clr     REVTIME
                bra     RSPEED          ; Done with collision speed

RSPEED          pulb                    ; Restore register
                pula                    ; Restore register
                rts                     ; Return from subroutine

********************************************************************************
* SUBROUTINE:   INIT_CONTROL                                                   *
* FUNCTION:     Set throttle speed to zero and center steering.                *
* INPUT:        None.                                                          *
* OUTPUT:       None.                                                          *
* DESTROYS:     None.                                                          *
* CALLS:        None.                                                          *
********************************************************************************
INIT_CONTROL    jsr     Stop             ; Stop throttle
                jsr     Straight         ; Center Steering
                rts                      ; Return from INIT_CONTROL sub

********************************************************************************
* SUBROUTINE:   Straight                                                       *
* FUNCTION:     Center steering wheels. These wheels are controlled by a servo.*
* INPUT:        None.                                                          *
* OUTPUT:       None.                                                          *
* DESTROYS:     None.                                                          *
* CALLS:        None.                                                          *
********************************************************************************
Straight        psha                    ; Save register
                pshb                    ; Save register
                ldd     #STRAIGHT_PWM
                std     PWMSERVO        ; Store pulse width high time
                pulb                    ; Restore register
                pula                    ; Restore register
                rts                     ; return from subroutine

********************************************************************************
* SUBROUTINE:   HardLeft                                                       *
```

```
* FUNCTION:     Turn wheels all the way to the left. These wheels are         *
*               are controlled by a servo.                                    *
* INPUT:        None.                                                         *
* OUTPUT:       None.                                                         *
* DESTROYS:     None.                                                         *
* CALLS:        None.                                                         *
*******************************************************************************
HardLeft        psha                    ; Save register
                pshb                    ; Save register
                ldd     #HARDLEFT_PWM
                std     PWMSERVO        ; Store pulse width high time
                pulb                    ; Restore register
                pula                    ; Restore register
                rts                     ; return from subroutine

*******************************************************************************
* SUBROUTINE:   HardRight                                                     *
* FUNCTION:     Turn wheels all the way to the right. These wheels are        *
*               are controlled by a servo.                                    *
* INPUT:        None.                                                         *
* OUTPUT:       None.                                                         *
* DESTROYS:     None.                                                         *
* CALLS:        None.                                                         *
*******************************************************************************
HardRight       psha                    ; Save register
                pshb                    ; Save register
                ldd     #HARDRIGHT_PWM
                std     PWMSERVO        ; Store pulse width high time
                pulb                    ; Restore register
                pula                    ; Restore register
                rts                     ; return from subroutine

*******************************************************************************
* SUBROUTINE:   MedLeft                                                       *
* FUNCTION:     Turn wheels half way to the left. These wheels are controlled *
*               by a servo.                                                   *
* INPUT:        None.                                                         *
* OUTPUT:       None.                                                         *
* DESTROYS:     None.                                                         *
* CALLS:        None.                                                         *
*******************************************************************************
MedLeft         psha                    ; Save register
                pshb                    ; Save register
                ldd     #MEDLEFT_PWM
                std     PWMSERVO        ; Store pulse width high time
                pulb                    ; Restore register
                pula                    ; Restore register
                rts                     ; return from subroutine

*******************************************************************************
* SUBROUTINE:   MedRight                                                      *
* FUNCTION:     Turn wheels half way to the right. These wheels are controlled *
*               by a servo.                                                   *
* INPUT:        None.                                                         *
* OUTPUT:       None.                                                         *
* DESTROYS:     None.                                                         *
* CALLS:        None.                                                         *
*******************************************************************************
MedRight        psha                    ; Save register
                pshb                    ; Save register
                ldd     #MEDRIGHT_PWM
                std     PWMSERVO        ; Store pulse width high time
                pulb                    ; Restore register
                pula                    ; Restore register
                rts                     ; return from subroutine

*******************************************************************************
```

```
* SUBROUTINE:   SoftLeft                                                   *
* FUNCTION:     Turn wheels a little to the left. These wheels are controlled  *
*               by a servo.                                                *
* INPUT:        None.                                                      *
* OUTPUT:       None.                                                      *
* DESTROYS:     None.                                                      *
* CALLS:        None.                                                      *
****************************************************************************
SoftLeft        psha                    ; Save register
                pshb                    ; Save register
                ldd     #SOFTLEFT_PWM
                std     PWMSERVO        ; Store pulse width high time
                pulb                    ; Restore register
                pula                    ; Restore register
                rts                     ; return from subroutine

****************************************************************************
* SUBROUTINE:   SoftRight                                                  *
* FUNCTION:     Turn wheels a little to the right. These wheels are controlled *
*               by a servo.                                                *
* INPUT:        None.                                                      *
* OUTPUT:       None.                                                      *
* DESTROYS:     None.                                                      *
* CALLS:        None.                                                      *
****************************************************************************
SoftRight       psha                    ; Save register
                pshb                    ; Save register
                ldd     #SOFTRIGHT_PWM
                std     PWMSERVO        ; Store pulse width high time
                pulb                    ; Restore register
                pula                    ; Restore register
                rts                     ; return from subroutine

****************************************************************************
* SUBROUTINE:   Stop                                                       *
* FUNCTION:     Set throttle speed to off.  Throttle is controlled by an   *
*               electronic speed control (ESC) which expects a pulse modulated *
*               signal.                                                    *
* INPUT:        None.                                                      *
* OUTPUT:       None.                                                      *
* DESTROYS:     None.                                                      *
* CALLS:        None.                                                      *
****************************************************************************
Stop            psha                    ; Save register
                pshb                    ; Save register
                ldd     #STOP_PWM
                std     PWMESC          ; Store pulse width high time
                pulb                    ; Restore register
                pula                    ; Restore register
                rts                     ; return from subroutine

****************************************************************************
* SUBROUTINE:   MaxFrw                                                     *
* FUNCTION:     Set throttle to maximum forward speed.  Throttle is controlled *
*               by an electronic speed control (ESC) which expects a pulse *
*               width modulated signal.                                    *
* INPUT:        None.                                                      *
* OUTPUT:       None.                                                      *
* DESTROYS:     None.                                                      *
* CALLS:        None.                                                      *
****************************************************************************
MaxFrw          psha                    ; Save register
                pshb                    ; Save register
                ldd     #MAXFRW_PWM
                std     PWMESC          ; Store pulse width high time
                pulb                    ; Restore register
                pula                    ; Restore register
```

```
                rts                     ; return from subroutine

********************************************************************************
* SUBROUTINE:   MaxRev                                                         *
* FUNCTION:     Set throttle to maximum reverse speed.  Throttle is controlled *
*               by an electronic speed control (ESC) which expects a pulse     *
*               width modulated signal.                                        *
* INPUT:        None.                                                          *
* OUTPUT:       None.                                                          *
* DESTROYS:     None.                                                          *
* CALLS:        None.                                                          *
********************************************************************************
MaxRev          psha                    ; Save register
                pshb                    ; Save register
                ldd     #MAXREV_PWM
                std     PWMESC          ; Store pulse width high time
                pulb                    ; Restore register
                pula                    ; Restore register
                rts                     ; return from subroutine

********************************************************************************
* SUBROUTINE:   MedFrw                                                         *
* FUNCTION:     Set throttle to medium forward speed.  Throttle is controlled  *
*               by an electronic speed control (ESC) which expects a pulse     *
*               width modulated signal.                                        *
* INPUT:        None.                                                          *
* OUTPUT:       None.                                                          *
* DESTROYS:     None.                                                          *
* CALLS:        None.                                                          *
********************************************************************************
MedFrw          psha                    ; Save register
                pshb                    ; Save register
                ldd     #MEDFRW_PWM
                std     PWMESC          ; Store pulse width high time
                pulb                    ; Restore register
                pula                    ; Restore register
                rts                     ; return from subroutine

********************************************************************************
* SUBROUTINE:   MedRev                                                         *
* FUNCTION:     Set throttle to medium reverse speed.  Throttle is controlled  *
*               by an electronic speed control (ESC) which expects a pulse     *
*               width modulated signal.                                        *
* INPUT:        None.                                                          *
* OUTPUT:       None.                                                          *
* DESTROYS:     None.                                                          *
* CALLS:        None.                                                          *
********************************************************************************
MedRev          psha                    ; Save register
                pshb                    ; Save register
                ldd     #MEDREV_PWM
                std     PWMESC          ; Store pulse width high time
                pulb                    ; Restore register
                pula                    ; Restore register
                rts                     ; return from subroutine

********************************************************************************
* SUBROUTINE:   SlowFrw                                                        *
* FUNCTION:     Set throttle to slow forward speed.  Throttle is controlled by *
*               an electronic speed control (ESC) which expects a pulse width  *
*               modulated signal.                                              *
* INPUT:        None.                                                          *
* OUTPUT:       None.                                                          *
* DESTROYS:     None.                                                          *
* CALLS:        None.                                                          *
********************************************************************************
SlowFrw         psha                    ; Save register
```

```
                pshb                    ; Save register
                ldd     #SLOWFRW_PWM
                std     PWMESC          ; Store pulse width high time
                pulb                    ; Restore register
                pula                    ; Restore register
                rts                     ; return from subroutine

********************************************************************************
* SUBROUTINE:   SlowRev                                                        *
* FUNCTION:     Set throttle to slow reverse speed.  Throttle is controlled by *
*               an electronic speed control (ESC) which expects a pulse width  *
*               modulated signal.                                              *
* INPUT:        None.                                                          *
* OUTPUT:       None.                                                          *
* DESTROYS:     None.                                                          *
* CALLS:        None.                                                          *
********************************************************************************
SlowRev         psha                    ; Save register
                pshb                    ; Save register
                ldd     #SLOWREV_PWM
                std     PWMESC          ; Store pulse width high time
                pulb                    ; Restore register
                pula                    ; Restore register
                rts                     ; return from subroutine

********************************************************************************
* SUBROUTINE:   OC2ISR                                                         *
* FUNCTION:     Produces pulse modulated signal on output compare 2 (pin 28).  *
*               This signal is used to control the throttle of the robot.      *
* INPUT:        None.                                                          *
* OUTPUT:       None.                                                          *
* DESTROYS:     None.                                                          *
* CALLS:        None.                                                          *
********************************************************************************
OC2ISR          psha                    ; Save register
                pshb                    ; Save register
                ldaa    #BIT6           ; Clear OC2 Interrupt Flag
                staa    TFLG1
                ldaa    TCTL1
                ANDA    #BIT6
                BEQ     LASTLOW
                BRA     LASTHI
LASTLOW         ldaa    TCTL1
                ORA     #BIT6
                staa    TCTL1
                ldd     #PERIOD
                SUBD    PWMESC
                ADDD    TOC2
                std     TOC2
                BRA     RTOC2
LASTHI          ldaa    TCTL1
                ANDA    #IBIT6
                staa    TCTL1
                ldd     TOC2
                ADDD    PWMESC
                std     TOC2
RTOC2           pulb                    ; Restore register
                pula                    ; Restore register
                RTI                     ; Return from OC2 ISR

********************************************************************************
* SUBROUTINE:   OC3ISR                                                         *
* FUNCTION:     Produces pulse modulated signal on output compare 3 (pin 29).  *
*               This signal is used to control the direction of the robot.     *
* INPUT:        None.                                                          *
* OUTPUT:       None.                                                          *
* DESTROYS:     None.                                                          *
```

```
* CALLS:        None.                                                        *
******************************************************************************
OC3ISR          psha                    ; Save register
                pshb                    ; Save register
                ldaa    #BIT5           ; Clear OC3 Interrupt Flag
                staa    TFLG1
                ldaa    TCTL1
                ANDA    #BIT4
                BEQ     LSTLOW
                BRA     LSTHI
LSTLOW          ldaa    TCTL1
                ORA     #BIT4
                staa    TCTL1
                ldd     #PERIOD
                SUBD    PWMSERVO
                ADDD    TOC3
                std     TOC3
                BRA     RTOC3
LSTHI           ldaa    TCTL1
                ANDA    #IBIT4
                staa    TCTL1
                ldd     TOC3
                ADDD    PWMSERVO
                std     TOC3
RTOC3           pulb                    ; Restore register
                pula                    ; Restore register
                RTI                     ; Return from OC3 ISR

******************************************************************************
* SUBROUTINE:   OC4ISR                                                       *
* FUNCTION:     Used as a timer to detect when the radio receiver signal goes *
*               dead.  Input capture will still be waiting for a rising edge *
*               pulse so it will not trigger.  This routine will set the radio *
*               throttle and steering settings to zero to relinquish control *
*               to the autonomous routines.                                  *
* INPUT:        None.                                                        *
* OUTPUT:       None.                                                        *
* DESTROYS:     None.                                                        *
* CALLS:        None.                                                        *
******************************************************************************
OC4ISR          psha                    ; Save register
                pshb                    ; Save register
                ldaa    #BIT4           ; Clear OC4 Interrupt Flag
                staa    TFLG1
                ldd     #0              ; Radio receiver went dead so,
                std     PWMRSERVO       ; clear radio steering and
                std     PWMRESC         ; throttle pulses.
RTOC4           pulb                    ; Restore register
                pula                    ; Restore register
                RTI                     ; Return from OC4 ISR

******************************************************************************
* SUBROUTINE:   IC2ISR                                                       *
* FUNCTION:     Read the pulse modulated signal on input capture 2 (pin 33). *
*               This is the receiver signal for the throttle of the robot.   *
* INPUT:        None.                                                        *
* OUTPUT:       None.                                                        *
* DESTROYS:     None.                                                        *
* CALLS:        None.                                                        *
******************************************************************************
IC2ISR          psha                    ; Save register
                pshb                    ; Save register
                ldaa    #BIT1           ; Clear IC2 Interrupt Flag
                staa    TFLG1
                ldaa    TCTL2
                anda    #BIT2
                beq     WASHIGH         ; Was capturing on high-to-low
```

```
                bra     WASLOW          ; Was capturing on low-to-high
WASHIGH         ldd     TIC2
                std     L_TIC2
                subd    LAST_TIC2       ; Calculate pulse width
                std     PWMRESC         ; Store throttle pulse width
                ldaa    TCTL2           ; Set IC2 to capture on low-to-high
                anda    #%11110111
                ora     #%00000100
                staa    TCTL2           ; Low-to-high capture
                ldd     TCNT            ; Store time to check if radio
                std     TOC4            ; signal went dead
                bra     RTIC2
WASLOW          ldd     TIC2
                subd    LAST_TIC2
                std     TOTAL
                ldd     TIC2
                std     LAST_TIC2

                ldaa    TCTL2           ; Set IC2 to capture on high-to-low
                anda    #%11111011
                ora     #%00001000
                staa    TCTL2           ; High-to-low capture


RTIC2           pulb                    ; Restore register
                pula                    ; Restore register
                RTI                     ; Return from IC2 ISR

********************************************************************************
* SUBROUTINE:   IC3ISR                                                         *
* FUNCTION:     Read the pulse modulated signal on input capture 3 (pin 34).   *
*               This is the receiver signal for the steering of the robot.     *
* INPUT:        None.                                                          *
* OUTPUT:       None.                                                          *
* DESTROYS:     None.                                                          *
* CALLS:        None.                                                          *
********************************************************************************
IC3ISR          psha                    ; Save register
                pshb                    ; Save register
                ldaa    #BIT0           ; Clear IC3 Interrupt Flag
                staa    TFLG1
                ldaa    TCTL2
                anda    #BIT0
                beq     WSHIGH          ; Was capturing on high-to-low
                bra     WSLOW           ; Was capturing on low-to-high
WSHIGH          ldd     TIC3
                subd    LAST_TIC3       ; Calculate pulse width
                std     PWMRSERVO       ; Store steering pulse width
                ldaa    TCTL2           ; Set IC3 to capture on low-to-high
                anda    #%11111101
                ora     #%00000001
                staa    TCTL2           ; Low-to-high capture
                bra     RTIC2
WSLOW           ldd     TIC3
                std     LAST_TIC3
                ldaa    TCTL2           ; Set IC3 to capture on high-to-low
                anda    #%11111110
                ora     #%00000010
                staa    TCTL2           ; High-to-low capture
RTIC3           pulb                    ; Restore register
                pula                    ; Restore register
                RTI                     ; Return from IC3 ISR

********************************************************************************
* SUBROUTINE:   IN_CHAR                                                        *
* FUNCTION:     Waits for a character to be received from the SCI.  When a     *
*               character is received, it is put into register A and          *
```

```
*               the subroutine exits.                                              *
* INPUT:        None.                                                              *
* OUTPUT:       Register A = input from SCI                                        *
* DESTROYS:     A register.                                                        *
* CALLS:        None.                                                              *
************************************************************************************
IN_CHAR         ldaa    SCSR            ; check status reg.
                anda    #%00100000      ; check if receive buffer full
                beq     IN_CHAR         ; wait until data present
                ldaa    SCDR            ; data -> A register
                rts                     ; return from subroutine

************************************************************************************
* Subroutine:   OUT_CHAR                                                           *
* Function:     Outputs the character in register A to the screen                  *
*               once the transmission data register is empty.                      *
* Input:        Data to be transmitted in register A.                              *
* Output:       Transmitted data.                                                  *
* Destroys:     None.                                                              *
* Calls:        None.                                                              *
************************************************************************************
OUT_CHAR        pshb                    ; Save register
L_OUT_CHAR      ldab    SCSR            ; Check status register.
                andb    #$80            ; Check if trans. buffer empty.
                beq     L_OUT_CHAR      ; Wait until empty.
                staa    SCDR            ; Output character.
                pulb                    ; restore register
                rts                     ; Return from subroutine

                end                     ; End of program
```