University of Florida
Department of Electrical and Computer Engineering
EEL 5666
Intelligent Machines Design Laboratory

Final Report

Wi-Scan 345

Date: *12/09/03*
Student Name: Jeff Putney
TA : Uriel Rodriguez
Louis Brandy
Instructor. A. A Arroyo

Table of Contents

Abstract

The Wi-Scan 345 is an autonomous robot designed to use wireless 802.11b connections and then interact with people on the internet through Email and a web server. The Wi-Scan 345 takes commands from it's website and report on it's the status of it's travels as it searches for an object or wall to explore. The data can be stored in the form of an email journal that can be sent when a internet connection is available or the data can be posted directly to the website in real time. The Wi-Scan 345 will use omnidirectional wheels so that the direction of travel of the robot can be changed without turning the robot or the robot can rotate while following a straight path.

Executive Summary

The Wi-Scan 345 project was a success in building a website interactive autonomous robot. The use of the Rabbit microcontroller allowed for easy use of ethernet and internet capabilities. It also proved to be an excellent microcontroller for programing robot behaviors and functions. The only missing feature was an A/D converter but that was easily added and connected without trouble once the right chip was selected. The omni-directional wheels proved to be one of the most exciting features of the robot. They allowed for extremely versatile movements that would not be possible with a normal steering system. They also proved to be very efficient at producing wall following motions. The DWL-810 wireless adapter worked perfectly for this application though in the future a more versatile adapter could be used. The DWL-810 only allows for connecting to a predetermined wireless SSID, so if there is no network with that ID the robot cannot connect. A Compact Flash wireless adapter would allow for the scanning of networks and there properties so that if any wireless connection is available the robot could log on. This would also be a physically smaller and lighter solution.

Introduction

The main goal of the Wi-Scan 345 is to safely navigate it's world while posting a website that gives information on it's status and takes in user input. This goal can be broken into four main tasks:

- Open an internet connection
- Take commands from the website user
- Navigate using obstacle avoidance
- Send out status data in email or webpage form

The distinguishing features of the Wi-Scan 345 are the omni-directional wheels and the internet connection. The rest of this paper will describe the separate embedded system components and physical characteristics the Wi-Scan 345.

Integrated System

The Wi-Scan 345 is based around a RabbitCore RCM3200 core module and prototyping board from Rabbit Semiconductors. The Rabbit 3000 microprocessor is a 44.2 MHz 8-bit processor specifically designed for embedded systems that wish to use ethernet connections.

The RCM3200 provodes the following features:

- 10/100Base-T Ethernet
- Rabbit 3000 microprocessor at up to 44.2 MHz
- 3.3 V operation (all pins are 5V tolerant and can run open-drain)
- Up to 512K Flash
- Up to 512K SRAM (program), 256K SRAM (data)
- 52 digital I/O
- 6 serial ports (IrDA, SDLC/HDLC, Async, SPI)
- 4 PWM's, 2 Input Capture Channels(selectable pins), 2 Quadrature Decoders

The Development kit comes with Dynamic C, which is Rabbit's own C compiler, and lots of sample code to get you started. The PWM's are used to run the three servos and the input capture is used for reading the sonar pulse length. There is no onboard A/D converter so a Linear Technology 12-bit 46.5kHz A/D converter(LTC1294) was added for reading IR sensor values. There is a DWL-810 wireless bridge adapter that plugs into rabbit through a standard CAT5 cable. The DWL-810 will automatically connect to a specified wireless connection if available. Once the unit is configured the Rabbit can connect to the network without having to worry about wireless configuration modes.

Mobile Platform

The platform consists of a triangular base with a omnidirectional wheel at each corner. This allows the Robot to travel in any direction without rotating the body or travel a straight path while rotating. With two IR sensors mounted on a face the robot can align itself easily with a wall and move along parallel to the wall while keeping a fairly constant distance. Because it does not have to rotate it's body it can move towards or away from the wall while following the wall without having to *turn* towards or away from the wall. The triangular shape makes the body very rugged and with very little material required for strength the robots wooden body makes for only a small percent of it's weight. This basic design has been used before in the Carnegie Mellon PPRK robot.

Actuation

The wheels are driven by Balsa Products BP148X 2BB High-torque Ball-Bearing servo motors. Each servo uses 2 ball bearings and put out 69oz.in. Of torque at 4.8V and 86oz.in. at 6.0V with a max transit time of 0.17 sec./60°. These are similar to the servos used in the original PPRK robot but they are cheaper and have a higher torque rating. They are run by the PWM outputs of the Rabbit. Because the Wheels are bumpy an IR

sensor could be used to track wheel speed and then adjustments to the PWM registered could help the robot steer a proper course regardless of driving conditions or power issues. This would also allow for recording distances traveled and approximate sizes of rooms or objects.

Sensors

Because the robot is symmetrical any of the three sides can at anytime be considered the front. Placing a different sensor on each face allows the robot to view the world in a different way depending on how it decided to face an obstacle. For example one side could use a long range IR sensor to find a wall it could then rotate and use Sonar to confirm what the Ir detected and then move in closer with another short range IR. All analog sensors are interfaced with an externally connected A/D converter that uses an SPI. The Sonar unit uses input capture to time the length of the return pulse. The Wireless Bridge has to be configured first by plugging it into a computer that can access it’s settings page.

Behaviors

The first behavior is to post a webpage. This begins with configuring TCPIP settings and HTTP settings. Most of the setting are set by using “#define” statements that configure library files using built in macros. The webpage uses commented sections that can be replaces by strings to update information on the page. When a button is pressed on the webpage it calls a CGI function that actually calls a C function in the Rabbit and thus you can control the robot from the webpage.

The next behavior is finding the nearest object. This uses the sonar to take readings in a 360 degree spin. It records the closest reading and when it occured and then turns back to that location with the short range sonar and heads in that direction. After it finds the object it begins it’s next behavior which is wall following. When it sees an object close to either IR it will attempt to align itself with the object by turning left or right to balance out it’s IR readings. Once balanced it will straif left and attemp to keep the reading balanced by slight rotations left or right. If the difference in IR reading becomes

to great it, due to possibly a corner, it will turn sharply to correct and thus turn a corner if necessary. If it gets to near or to far from the wall if will scoot in or out will still moving left along the wall. If the IR on the side sees something it will turn to avoid the obstacle. Once the robot loses the wall or object it will return to heading in a straight path until another wall is found or until a new command from the website is given.

Experimental Layout and Results

The experiments have focused on finding objects with sonar and accurately following walls and avoiding objects. The sonar can only fire once every 11-40 ms and the robot averages 4 reading each time it checks the sonar value. This means that while the robot is spinning and looking for the nearest object it is averaging a few degrees of vision but as long as there are large differences in object differences it can head in approximately the correct direction for the nearest object.

The wall following was accomplished by using the properties of the omni-direction wheels to rotate or move towards or away from the wall while maintaining a sideways

velocity. This process started by writing a program that allowed me to use the keypad to see and change PWM values for each servo while the robot was running. I could then find the correct relative velocities for each wheel to produce the desired direction. The robot can also correct it's motion by slightly speeding up or slowing down one or more of it's servos to produce a desired force in a certain direction. This can be done in real time.

Conclusion

The original idea for the Wi-Scan 345 was to actually find different wireless networks and log onto them. This proved to be an impossibility with the wireless bridge used as it has to be pre-configured for a specific SSID. It was however very successful in using the wireless connection it did have. The webpage and email were successful and preformed all the functions that were expected of them. In the future I would like to include an option of reading incoming email and getting instructions from it.

The omni-directional wheels worked amazingly well. They preformed exactly as expected and allowed for an amazing range of robot motion. Strafing left and right

looks flawless and subtle turning can be easily added to any motion. The small triangular shaped body bade a great platform. It was small light-weight and very strong.

The Rabbit board proved to be very well suited to the task of running the entire robot. It had all the necessary features except the A/D. It was a very easy processor to work with and came with excellent documentation including a very helpful poster that lays our all the important registers and features of the chip. The dynamic C program has lots of powerful debugging tools and is very easy to learn. It also lets you do inline assembly which is nice for when you want to make your code go as fast as possible.

If I had to start this over I would use the same platform and board and spend more time enhancing the webpage and robot motions. I would also include small IR sensors to track wheel motion, as this would allow for more indicate behaviors. I would change the placement of some sensors so that obstacle avoidance would be more accurate and the blind spots would be smaller.

Documentation

Rabbit 3000 Microprocessor User's Manual:
http://www.rabbitsemiconductor.com/documentation/docs/manuals/Rabbit3000/UsersManual/R3000UM.pdf

Dynamic C v.8 User's Manual
http://www.rabbitsemiconductor.com/documentation/docs/manuals/DC/DCUserManual/DCPUM.pdf

Dynamic C TCP/IP User's Manual
http://www.rabbitsemiconductor.com/documentation/docs/manuals/TCPIP/UsersManual/tcpipuser.pdf

SRF04 - Ultra-Sonic Ranger
http://www.robot-electronics.co.uk/htm/srf04tech.htm

DWL-810 - Wireless Bridge
http://www.dlink.com.au/products/wireless/dwl810+/

4cm omni-directional poly roller wheel
http://www.acroname.com/robotics/parts/R97-4CM-POLY-ROLLER.html

Appendices (code only included in electronic copy)

keypadtopwm.c

```
/*************************************************************
        keypadtopwm.c
  Jeff Putney, 2003

  This program is an adaptation of the keypadtolcd.c Z world program


        Description
        ===========
        This sample program demonstrates the use of the LCD/Keypad Module
  to display and change PWM values

        Here is a list of what key controls what function.

          keypad          Controller Keypad LED
          ------                  ----------------------   ---------------
          Scroll-Left         Change to previous PWM
          Scroll-Up           Increase PWM value
          Scroll-Down       Decrease PWM value
          Scroll-Right      Change to next PWM
                  Page-Down         ---
                  Page-Up          ---
                  Enter                    ---

*********************************************************************/
#class auto                     // Change default storage class for local variables: on the stack

#define PORTA_AUX_IO            //required to run LCD/Keypad for this demo

#memmap xmem
#use rcm3200.lib                //sample library used for this demo

#define DS1 6                           //led, port G bit 6
#define DS2 7                           //led, port G bit 7
#define S2  1               //switch, port G bit 1
#define S3  0               //switch, port G bit 0
#define ON  0               //state to turn on led
#define OFF 1               //state to turn off led
#define UP  0               //state to turn on led
#define DOWN 1              //state to turn off led

// Structure to hold font information
fontInfo fi6x8, fi8x10;
```

```
//////////////////////////////////////
//////////////////////////////////////
//initialize pwm port
void initPort()
{

#asm
        ld          a,0x00                      ;clear all bits for pclk/2
        ioi     ld (PGCR),a
        ld          (PGCRShadow),a

        ld          a,0x00                      ;clear all bits for normal function
        ioi     ld (PGFR),a
        ld          (PGFRShadow),a

        ld          (PGDCRShadow),a
        or          0xC0                            ;set bits 7,6 drive open drain
        res     2,a                         ;clear bit 2 drive output
        ioi     ld (PGDCR),a
        ld          (PGDCRShadow),a

        ld          (PGDRShadow),a
        or          0xC0                            ;set bits 7,6 output high
        res     2,a                         ;clear bit 2 output low
        ioi     ld (PGDR),a
        ld          (PGDRShadow),a

        ld          a,0xC4                      ;set bits 7,6,2 to output, clear bits
5,4,3,1,0 to input
        ioi     ld (PGDDR),a
        ld          (PGDDRShadow),a

  ;set up PWM's on port F

  ld            a,0x00                    ;clear all bits for pclk/2
        ioi     ld (PFCR),a
        ld          (PFCRShadow),a

        ld          a,0x70                      ;set bits 5,6,7 to PWM function
        ioi     ld (PFFR),a
        ld          (PFFRShadow),a

        ld          (PGDCRShadow),a
        or          0xF0                            ;set upper bits to drive open
drain
        ioi     ld (PGDCR),a
```

```
        ld          (PGDCRShadow),a

  ld        a,0x01                      ;set Timer A to pclk/2
        ioi     ld (TAPR),a
        ld          (TAPRShadow),a

  ld        a,0xFF                      ;set TAT9R (PWM) timer counter
        ioi     ld (TAT9R),a
        ld          (TAT9RShadow),a

        ld          a,0x21                      ;set PWM0 count upper byte 1ms = 15
        ioi     ld (PWM0R),a
        ld          (PWM0RShadow),a

  ld        a,0x00                      ;set PWM0 count lower byte single PWM mode
        ioi     ld (PWL0R),a
        ld          (PWL0RShadow),a

  ld        a,0x21                      ;set PWM1 count upper byte 1ms = 15
        ioi     ld (PWM1R),a
        ld          (PWM1RShadow),a

  ld        a,0x00                      ;set PWM1 count lower byte single PWM mode
        ioi     ld (PWL1R),a
        ld          (PWL1RShadow),a

  ld        a,0x21                      ;set PWM2 count upper byte 1ms = 15
        ioi     ld (PWM2R),a
        ld          (PWM2RShadow),a

  ld        a,0x00                      ;set PWM2 count lower byte single PWM mode
        ioi     ld (PWL2R),a
        ld          (PWL2RShadow),a

        ld          a,0xF0                      ;set bits 7,6,5,4 to output, clear bits
3,2,1,0 to input
        ioi     ld (PFDDR),a
        ld          (PFDDRShadow),a
#endasm
}

//////////////////////////////////////////////////////
//////////////////////////////////////////////////////
//servo control
void PWM_Fine(int d, int pwm)
{
```

```
switch(pwm)
                                        {
case 0x0:
  if (d == UP)
   {
     if ((PWL0RShadow) == 0xC0)
     {

     WrPortI(PWM0R,&PWM0RShadow,PWM0RShadow + 1);
     WrPortI(PWL0R,&PWL0RShadow,0x00);
           }
           else
           {
     WrPortI(PWL0R,&PWL0RShadow,PWL0RShadow + 0x40);
     }

           }
  else
  {
           if (PWL0RShadow == 0x00)
     {
     WrPortI(PWM0R,&PWM0RShadow,PWM0RShadow - 1);
     WrPortI(PWL0R,&PWL0RShadow,0xC0);
           }
           else
           {
     WrPortI(PWL0R,&PWL0RShadow,PWL0RShadow - 0x40);
     }
           }
  break;

////////////////////////////////////////
////////////////////////////////////////PWM1
 case 0x1:

 if (d == UP)
   {
     if ((PWL1RShadow) == 0xC0)
     {

     WrPortI(PWM1R,&PWM1RShadow,PWM1RShadow + 1);
     WrPortI(PWL1R,&PWL1RShadow,0x00);
           }
           else
           {
     WrPortI(PWL1R,&PWL1RShadow,PWL1RShadow + 0x40);
```

```
  }

        }
else
{
        if (PWL1RShadow == 0x00)
  {
  WrPortI(PWM1R,&PWM1RShadow,PWM1RShadow - 1);
  WrPortI(PWL1R,&PWL1RShadow,0xC0);
        }
        else
        {
  WrPortI(PWL1R,&PWL1RShadow,PWL1RShadow - 0x40);
  }
        }
break;

//////////////////////
//////////////////////PWM2

case 0x2:
 if (d == UP)
 {
  if ((PWL2RShadow) == 0xC0)
  {

  WrPortI(PWM2R,&PWM2RShadow,PWM2RShadow + 1);
  WrPortI(PWL2R,&PWL2RShadow,0x00);
        }
        else
        {
  WrPortI(PWL2R,&PWL2RShadow,PWL2RShadow + 0x40);
  }

        }
else
{
        if (PWL2RShadow == 0x00)
  {
  WrPortI(PWM2R,&PWM2RShadow,PWM2RShadow - 1);
  WrPortI(PWL2R,&PWL2RShadow,0xC0);
        }
        else
        {
  WrPortI(PWL2R,&PWL2RShadow,PWL2RShadow - 0x40);
  }
```

```
        }
  break;
  }

  }
////////////////////////////////////////////////////////////////////////

void main()
{
        static int led, channel, wKey, keypad_active, prompt_displayed;
        static int new_keypress_message, release_value, i, pwm, pwmTwo;

        //-------------------------------------------------------------------------
        // Initialize the controller
        //-------------------------------------------------------------------------
        brdInit();                                      // Initialize the controller for this demo

        // Start-up the keypad driver and
        // Initialize the graphic driver
        displnit();
  initPort();


        // Use default key values along with a key release code
        keyConfig (  3,'R', '1', 0, 0,  0, 0 );
        keyConfig (  6,'E', '2', 0, 0,  0, 0 );
        keyConfig (  2,'D', '3', 0, 0,  0, 0 );
        keyConfig (  5,'+', '4', 0, 0,  0, 0 );
        keyConfig (  1,'U', '5', 0, 0,  0, 0 );
        keyConfig (  4,'-', '6', 0, 0,  0, 0 );
        keyConfig (  0,'L', '7', 0, 0,  0, 0 );

        // Initialize 6x8 font
        glXFontInit(&fi6x8, 6, 8, 32, 127, Font6x8);        // Initialize 6x8 font
        glXFontInit(&fi8x10, 8, 10, 32, 127, Font8x10);                         //        initialize 8x10 font
        glBlankScreen();

        // Initialize control flags
  pwm = 0;
        keypad_active = FALSE;
        prompt_displayed = FALSE;
        new_keypress_message = FALSE;
        for(;;)
        {
                costate
                {
```

```
                keyProcess ();
                waitfor ( DelayMs(10) );

        }

        costate
        {
                // Wait for any key to be pressed
                waitfor((wKey = keyGet()) != 0);
                release_value = -1;
                switch(wKey)
                {
                        case 'L':       release_value = '7'; break;
                        case '-': release_value = '6'; break;
                        case 'U':   release_value = '5'; break;
                        case '+':       release_value = '4'; break;
                        case 'D':       release_value = '3'; break;
                        case 'E':release_value = '2'; break;
                        case 'R':       release_value = '1'; break;
                }
                if(release_value != -1)
                {
                        keypad_active = TRUE;
                        // Wait for the key to be released
                        waitfor(keyGet() == release_value);
                        keypad_active = FALSE;
                }
        }

        costate
        {
                if(!keypad_active)
                {
                        if(!prompt_displayed)
                        {
    pwmTwo = pwm * 2;
                                glBlankScreen();
                                glPrintf (0,  0, &fi6x8,  "PWM %X Selected", pwm);
                                glPrintf (0,  8, &fi6x8,  "PWM 1  PWM 2  PWM 3 ");
                                glPrintf (0,  16, &fi6x8, "%X %X  %X %X  %X %X ",
(PWM0RShadow), (PWL0RShadow), (PWM1RShadow), (PWL1RShadow), (PWM2RShadow),
(PWL2RShadow));
                                glFillPolygon(4, 115, 26,  121,26,   121,31,   115, 31);
                                prompt_displayed = TRUE;
                                new_keypress_message = FALSE;
```

```
                                // Turn-Off leds on the Demo board
                                BitWrPortI(PGDR, &PGDRShadow, 1, DS1);
//DS1 off
                                BitWrPortI(PGDR, &PGDRShadow, 1, DS2);
//DS2 off
                        }

                        for(channel = 0; channel <= 6; channel++)
                        {
                                for(led = 0; led <=6; led++)
                                {
                                        if(led != channel)
                                        dispLedOut(led, 0);
                                }
                                dispLedOut(channel, 1);
                                waitfor(DelayMs(100));
                                if(keypad_active)
                                {

                                        break;
                                }
                        }
                }
        }

        costate
        {
                if(keypad_active && !new_keypress_message)
                {
                        glBlankScreen();
                        glFillPolygon(4, 113, 26,  121,26,   121,31,   113, 31);
                        switch(wKey)
                        {
                                case 'L':
        if (pwm == 0)
  {
  pwm = 2;
  }
  else {
  pwm -= 1;
  }
                                        glPrintf (0,  0, &fi8x10, "Switching PWM");
                                        glPrintf (0,  16, &fi8x10, " ");
                                        dispLedOut(0, 1);
                                        channel = 0;
                                        break;
```

```
                        case '-':
  PWM_Fine(DOWN, pwm);
                                //glPrintf (0,  0, &fi8x10, "Page-Down key");
                                //glPrintf (0,  16, &fi8x10, "is Active.");
                                dispLedOut(1, 1);
                                channel = 1;
                                BitWrPortI(PGDR, &PGDRShadow, 0, DS1);
//DS1 on
                                break;

                        case 'U':
                                glPrintf (0,  0, &fi8x10, "Scroll-Up key");
                                glPrintf (0,  16, &fi8x10, "is Active.");
                                dispLedOut(2, 1);
                                channel = 2;
                                break;

                        case '+':
  PWM_Fine(UP, pwm);
                                //glPrintf (0,  0, &fi8x10, "Page-Up key");
                                //glPrintf (0,  16, &fi8x10, "is Active.");
                                dispLedOut(3, 1);
                                channel = 3;
                                BitWrPortI(PGDR, &PGDRShadow, 0, DS2);
//DS2 on
                                break;

                        case 'D':
                                glPrintf (0,  0, &fi8x10, "Scroll-Down key");
                                glPrintf (0,  16, &fi8x10, "is Active.");
                                dispLedOut(4, 1);
                                channel = 4;
                                break;

                        case 'E':
                                glPrintf (0,  0, &fi8x10, "Enter key");
                                glPrintf (0,  16, &fi8x10, "is Active.");
                                dispLedOut(5, 1);
                                channel = 5;
                                break;

                        case 'R':
  if (pwm == 2)
  {
  pwm = 0;
```

```
        }
        else {
        pwm += 1;
        }
                                        glPrintf (0,  0, &fi8x10, "Switching PWM");
                                        glPrintf (0,  16, &fi8x10, "key is Active.");
                                        dispLedOut(6, 1);
                                        channel = 6;
                                        break;
                        }
                        for(led=0; led <=6; led++)
                        {
                                if(led != channel)
                                {
                                        dispLedOut(led, 0);
                                }
                        }
                        prompt_displayed = FALSE;
                        new_keypress_message = TRUE;
                }
        }
        costate
        {
                if(keypad_active)
                {
                        for(i=0; i<(LCD_XS-8); i+=4)
                        {
                                glHScroll(0, 26, LCD_XS, 6, -4);
                                waitfor(DelayMs(5));
                                if(!keypad_active)
                                        abort;
                        }
                        for(i=0; i<(LCD_XS-8); i+=4)
                        {
                                glHScroll(0, 26, LCD_XS, 6, 4);
                                waitfor(DelayMs(5));
                                if(!keypad_active)
                                        abort;
                        }
                }
        }
    }

}
```

Final_V5.c

```
/*********************************************************************
        Final_V5.c
        Jeff Putney, 2003


        Description
        ===========
  Posts a webpage WISCAN.shtml  and then takes comands from it that control the bahavior of the robot

*********************************************************************/
#class auto                     // Change default storage class for local variables: on the stack


float ReadAD ( int Channel, int Samples );
int SwapBytes ( int i );

float ScaleFactor;

#define SPI_SER_D
#define SPI_CLK_DIVISOR                 5
#use SPI.LIB
#use WYSCAN.LIB


#define PORTA_AUX_IO            //required to run LCD/Keypad
#memmap xmem
#use rcm3200.lib

#define DS01 0x40  //led, port G bit 6 bitmask
#define DS02 0x80  //led, port G bit 7 bitmask
#define DS1 6               //port F bit 6
#define DS2 7               //port G bit 7
#define S2  1               //switch, port G bit 1
#define S3  0               //switch, port G bit 0
#define ON  0               //state to turn on led
#define OFF 1               //state to turn off led
#define UP  0               //state to turn on led
#define DOWN 1              //state to turn off led

#define TCPCONFIG 1
#define TCP_BUF_SIZE 2048
#define HTTP_MAXSERVERS 2
#define MAX_TCP_SOCKET_BUFFERS 2
#define REDIRECTHOST    _PRIMARY_STATIC_IP
```

```
#define REDIRECTTO      "http://" REDIRECTHOST ""
#use "dcrtcp.lib"
#use "http.lib"

#ximport "samples/rcm3200/tcpip/pages/WISCAN.shtml"   index_html
#ximport "samples/rcm3200/tcpip/pages/rabbit1.gif"       rabbit1_gif
#ximport "samples/rcm3200/tcpip/pages/ledon.gif"        ledon_gif
#ximport "samples/rcm3200/tcpip/pages/ledoff.gif"       ledoff_gif
#ximport "samples/rcm3200/tcpip/pages/button.gif"       button_gif
#ximport "samples/rcm3200/tcpip/pages/showsrc.shtml"    showsrc_shtml
#ximport "samples/rcm3200/tcpip/browseled.c"            browseled_c

// Structure to hold font information
fontInfo fi6x8;

const HttpType http_types[] =
{
  { ".shtml", "text/html", shtml_handler}, // ssi
  { ".html", "text/html", NULL},          // html
  { ".cgi", "", NULL},                    // cgi
  { ".gif", "image/gif", NULL}
};
char led1[15];
char led2[15];
char message1[200];
int spin,estop,d,f,IR,new;
long int sonartemp, sonar,sdistance;
float volts0, volts1, volts2, volts3,distance;

int led1toggle(HttpState* state)    //cgi function thats sets the robot to find nearest object
{
    strcpy(message1,"Scanning Perimeter");
    spin = 1;
  cgi_redirectto(state,REDIRECTTO);
  return 0;
}

int led2toggle(HttpState* state)    //cgi function thats sets the robot to head staight and follow a wall if it finds
it
{
  d=1;
          f=0;
  spin = 0;
  IR = 1;
  strcpy(message1,"I want a wall");
  cgi_redirectto(state,REDIRECTTO);
```

```
  return 0;

}

int led3toggle(HttpState* state)    //Emegency stop function
{
          if (estop == 0)
  estop = 1;
  else
  estop = 0;
  cgi_redirectto(state,REDIRECTTO);
  return 0;
}

int led4toggle(HttpState* state)     //cgi function  not yet defined
{
  new = 1;
  cgi_redirectto(state,REDIRECTTO);
  return 0;
}


const HttpSpec http_flashspec[] =
{
  { HTTPSPEC_FILE,  "/",            index_html,    NULL, 0, NULL, NULL},
  { HTTPSPEC_FILE,  "/index.shtml",   index_html,    NULL, 0, NULL, NULL},
  { HTTPSPEC_FILE,  "/showsrc.shtml", showsrc_shtml, NULL, 0, NULL, NULL},
  { HTTPSPEC_FILE,  "/rabbit1.gif",   rabbit1_gif,   NULL, 0, NULL, NULL},
  { HTTPSPEC_FILE,  "/ledon.gif",     ledon_gif,     NULL, 0, NULL, NULL},
  { HTTPSPEC_FILE,  "/ledoff.gif",    ledoff_gif,    NULL, 0, NULL, NULL},
  { HTTPSPEC_FILE,  "/button.gif",    button_gif,    NULL, 0, NULL, NULL},

  { HTTPSPEC_FILE,  "browseled.c",  browseled_c,    NULL, 0, NULL, NULL},

  { HTTPSPEC_VARIABLE, "led1",  0,    led1, PTR16,   "%s", NULL},
  { HTTPSPEC_VARIABLE, "led2",  0,    led2, PTR16,   "%s", NULL},
  { HTTPSPEC_VARIABLE, "message1",  0,    message1, PTR16,   "%s", NULL},

  { HTTPSPEC_FUNCTION, "/led1tog.cgi",   0, led1toggle, 0, NULL, NULL},
  { HTTPSPEC_FUNCTION, "/led2tog.cgi",   0, led2toggle, 0, NULL, NULL},
  { HTTPSPEC_FUNCTION, "/led3tog.cgi",   0, led3toggle, 0, NULL, NULL},
  { HTTPSPEC_FUNCTION, "/led4tog.cgi",   0, led4toggle, 0, NULL, NULL},
};
```

```
//////////////////////////////////////////////////////////
// Initializes port G and F
// See brdInit() in sample library rcm3000.lib for
// other port initializing
// servos calibrated at PWM value 21 high 0 low
//////////////////////////////////////////////////////////
void initPort()
{

#asm
        ld          a,0x00                  ;clear all bits for pclk/2
        ioi     ld (PGCR),a
        ld          (PGCRShadow),a

        ld          a,0x00                  ;clear all bits for normal function
        ioi     ld (PGFR),a
        ld          (PGFRShadow),a

        ld          (PGDCRShadow),a
        or          0xC0                        ;set bits 7,6 drive open drain
        res     2,a                     ;clear bit 2 drive output
        ioi     ld (PGDCR),a
        ld          (PGDCRShadow),a

        ld          (PGDRShadow),a
        or          0xC0                        ;set bits 7,6 output high
        res     2,a                     ;clear bit 2 output low
        ioi     ld (PGDR),a
        ld          (PGDRShadow),a

        ld          a,0xC4                  ;set bits 7,6,2 to output, clear bits
5,4,3,1,0 to input
        ioi     ld (PGDDR),a
        ld          (PGDDRShadow),a

  ;set up PWM's on port F

  ld            a,0x00                  ;clear all bits for pclk/2
        ioi     ld (PFCR),a
        ld          (PFCRShadow),a

        ld          a,0x70                  ;set bits 5,6,7 to PWM function
        ioi     ld (PFFR),a
        ld          (PFFRShadow),a

        ld          (PGDCRShadow),a
```

```
        or      0xF4                                    ;set upper bits and bit 2 to drive
open drain
        ioi     ld (PGDCR),a
        ld      (PGDCRShadow),a
  ld    a,0x01                          ;set Timer A to pclk/2
        ioi     ld (TAPR),a
        ld      (TAPRShadow),a

  ld    a,0xFF                          ;set TAT9R (PWM) timer counter
        ioi     ld (TAT9R),a
        ld      (TAT9RShadow),a

        ld      a,0x21                          ;set PWM0 count upper byte 1ms = 15
        ioi     ld (PWM0R),a
        ld      (PWM0RShadow),a

  ld    a,0x00                          ;set PWM0 count lower byte single PWM mode
        ioi     ld (PWL0R),a
        ld      (PWL0RShadow),a

  ld    a,0x21                          ;set PWM1 count upper byte 1ms = 15
        ioi     ld (PWM1R),a
        ld      (PWM1RShadow),a

  ld    a,0x00                          ;set PWM1 count lower byte single PWM mode
        ioi     ld (PWL1R),a
        ld      (PWL1RShadow),a

  ld    a,0x21                          ;set PWM2 count upper byte 1ms = 15
        ioi     ld (PWM2R),a
        ld      (PWM2RShadow),a

  ld    a,0x00                          ;set PWM2 count lower byte single PWM mode
        ioi     ld (PWL2R),a
        ld      (PWL2RShadow),a

        ld      a,0xF4                          ;set bits 7,6,5,4,2 to output, clear bits
3,1,0 to input
        ioi     ld (PFDDR),a
        ld      (PFDDRShadow),a
#endasm
}

//////////////////////////////////////////////////////////////
// output to servo controlled by port G bit 6
// turns on if state = 0
```

```
// turns off if state = 1
//////////////////////////////////////////////////////
void SPWM(int state)
{
        if (state == ON)
        {
#asm
        ld              a,(PFDRShadow)                  ;use shadow register to keep other bit
values
        res     DS1,a                                   ;clear bit 6 only
        ioi     ld (PFDR),a                     ;write data to port g
        ld              (PFDRShadow),a                  ;update shadow register
#endasm
        }
        else
        {
#asm
        ld              a,(PFDRShadow)                  ;use shadow register to keep other bit
values
        set     DS1,a                                   ;set bit 6 only
        ioi     ld (PFDR),a                     ;write data to port g
        ld              (PFDRShadow),a                  ;update shadow register
#endasm

        }
}

//////////////////////////////////////////////////////
// DS2 led on protoboard is controlled by port G bit 7
// turns on if state = 0
// turns off if state = 1
//////////////////////////////////////////////////////
void DS2led(int state)
{
        if (state == ON)
        {
#asm
        ld              a,(PGDRShadow)                  ;use shadow register to keep other bit
values
        res     DS2,a                                   ;clear bit 7 only
        ioi     ld (PGDR),a                     ;write data to port g
        ld              (PGDRShadow),a                  ;update shadow register
#endasm
        }
        else
        {
```

```
#asm
        ld              a,(PGDRShadow)                  ;use shadow register to keep other bit
values
        set     DS2,a                                   ;set bit 7 only
        ioi     ld (PGDR),a                     ;write data to port g
        ld              (PGDRShadow),a                  ;update shadow register
#endasm
        }
}
////////////////////////////////////////////////////////
// increments or decrements the PWM0R register
// decrements if state = 0
// increments if state = 1
// assumes that the 0 bit of PWL0R is set to 0
////////////////////////////////////////////////////////
void PWM_Fine(int d, int pwm)
{
switch(pwm)
                                {
case 0x0:

  if (d == UP)
   {
    if ((PWL0RShadow) == 0xC0)
    {

    WrPortI(PWM0R,&PWM0RShadow,PWM0RShadow + 1);
    WrPortI(PWL0R,&PWL0RShadow,0x00);
        }
        else
        {
    WrPortI(PWL0R,&PWL0RShadow,PWL0RShadow + 0x40);
    }

        }
  else
  {
        if (PWL0RShadow == 0x00)
    {
    WrPortI(PWM0R,&PWM0RShadow,PWM0RShadow - 1);
    WrPortI(PWL0R,&PWL0RShadow,0xC0);
        }
        else
        {
    WrPortI(PWL0R,&PWL0RShadow,PWL0RShadow - 0x40);
    }
```

```
        }
 break;

//////////////////////////////////////
//////////////////////////////////////PWM1
case 0x1:

 if (d == UP)
  {
    if ((PWL1RShadow) == 0xC0)
    {

    WrPortI(PWM1R,&PWM1RShadow,PWM1RShadow + 1);
    WrPortI(PWL1R,&PWL1RShadow,0x00);
          }
          else
          {
    WrPortI(PWL1R,&PWL1RShadow,PWL1RShadow + 0x40);
    }

          }
 else
 {
          if (PWL1RShadow == 0x00)
    {
    WrPortI(PWM1R,&PWM1RShadow,PWM1RShadow - 1);
    WrPortI(PWL1R,&PWL1RShadow,0xC0);
          }
          else
          {
    WrPortI(PWL1R,&PWL1RShadow,PWL1RShadow - 0x40);
    }
          }
 break;

 //////////////////////
 //////////////////////PWM2

 case 0x2:
  if (d == UP)
  {
    if ((PWL2RShadow) == 0xC0)
    {

    WrPortI(PWM2R,&PWM2RShadow,PWM2RShadow + 1);
    WrPortI(PWL2R,&PWL2RShadow,0x00);
```

```
            }
            else
            {
    WrPortI(PWL2R,&PWL2RShadow,PWL2RShadow + 0x40);
    }

            }
  else
  {
            if (PWL2RShadow == 0x00)
    {
    WrPortI(PWM2R,&PWM2RShadow,PWM2RShadow - 1);
    WrPortI(PWL2R,&PWL2RShadow,0xC0);
            }
            else
            {
    WrPortI(PWL2R,&PWL2RShadow,PWL2RShadow - 0x40);
    }
            }
  break;
  }

  }


//////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////

//////////////////////////////////////////////////
//////////////////////////////////////////////////




void main()
{
            int Value;

            int spincount,closest,ego;
  int sonarh,sonarl,lastsonar,newsonar,sonarte,i,Sample;
  long int closestf;

  int a;
brdInit();                                  //initialize board for this demo
```

```
dispInit();

keypadDef();                    // Use the default keypad ASCII return values

glBackLight(0);   // Turn-on the backlight
glXFontInit(&fi6x8, 6, 8, 32, 127, Font6x8);                    //          Initialize 6x8 font
glBlankScreen();

 initPort();                                        // initializes port G only
// set up chip select port for A/D
          BitWrPortI ( PFDR, &PFDRShadow, 1, 3 );  // turn off /CS (1=off)
          BitWrPortI ( PFDCR, &PDDCRShadow, 0, 3 );         // bit 0 = "normal" output
          WrPortI ( PFCR, &PFCRShadow, 0 );                                 // bits 0..3 = clocked by perclk/2
          BitWrPortI ( PFDDR, &PFDDRShadow, 1, 3 );         // bit 0 = output

          ScaleFactor = 5.0/4096.0;
          SPIinit();

// set up input capture
  WrPortI ( ICCR, &ICCRShadow, 0 );                                 // no interrupt from IC
  WrPortI ( ICT1R, &ICT1RShadow, 0x56 );    //count from start to stop, latch counter on stop, start on rising
stop on falling
  WrPortI ( ICS1R, &ICS1RShadow, 0x88 );    // use port F bit 1
  WrPortI ( TAT8R, &TAT8RShadow, 0x0F );    // prescale input capture timer  4f is good


  strcpy(led1,"ledon.gif");
  strcpy(led2,"ledoff.gif");
  strcpy(message1,"I'm bored. What shall I do.");

  sock_init();
  http_init();
  tcp_reserveport(80);


  a = 0x15;
  d = 0;
  f = 0;       //not following wall
  Sample = 4;
  spin = 0;     // not looking for object
  IR = 1;
  estop = 0;
  ego = 1;
  new = 0;      //not looking for new wall

          while (1)
```

```
      {

      costate
{

http_handler();   // his has to be called to take care of the website
}


costate{     // continualy updateding the sonar value with averaging Sample number of samples
                  sonartemp = 0;
 for (i = 1; i<=Sample; i++ )
 {
 WrPortI(ICCSR, &ICCSRShadow, 0x00);
   BitWrPortI ( PFDR, &PFDRShadow, 0, 2 );       // turn off /CS (1=off)
   waitfor(DelayMs(10));
   BitWrPortI ( PFDR, &PFDRShadow, 1, 2 );
   waitfor(DelayMs(.01));
   BitWrPortI ( PFDR, &PFDRShadow, 0, 2 );
   waitfor(BitRdPortI(ICCSR,4) || DelayMs(1000));
   lastsonar = newsonar;
   sonarl = RdPortI(ICL1R);
   sonarh = RdPortI(ICM1R);
   newsonar =  (sonarh << 8) | sonarl;
   newsonar =  newsonar>>6;
   if ((newsonar - lastsonar) < 0x5000)
   {
   sonarte = (newsonar - lastsonar);
   sonartemp += (newsonar - lastsonar);
    }
    else
    sonartemp += sonarte;
  }

  sonar = sonartemp / Sample;
  sonar = sonartemp;
  //sonar = sonar>>6;

  }

costate
{
if (IR == 1)     //if IR is on  this gets IR values
{
volts0 = ReadAD ( 0, 40 );
volts1 = ReadAD ( 1, 40 );
```

```
volts2 = ReadAD ( 2, 40 );
volts3 = ReadAD ( 3, 40 );
}

if (estop == 1)    // checking for emergency stop
{
//ego = 0;
#asm
ld              a,0x00                                  ;set bits 5,6,7 to turn off PWM function
                ioi     ld (PFFR),a
                ld              (PFFRShadow),a
#endasm
}
else
{if (ego = 0)
{
#asm
ld              a,0x70                                  ;set bits 5,6,7 to PWM function
                ioi     ld (PFFR),a
                ld              (PFFRShadow),a
#endasm
ego = 1;
}  }
}


                costate
                {
                        waitfor(DelayMs(30));

  glPrintf (0,  0, &fi6x8,  "PWM0 %X    ", (sonar));
                        glPrintf (0,  8, &fi6x8,  "PWM1 %f  %f", volts0, volts1);

                        glPrintf (0,  16, &fi6x8, "PWM2 %X  %X", (PWM2RShadow),(PWL2RShadow));
}

 costate
 {
      waitfor(DelayMs(5000));

  if(f == 1)
  strcpy(message1,"Following wall");

  else if(spin == 1)
                        strcpy(message1,"You spin me right round, baby right round");
}
```

```
costate
                {

  if(spin == 1)   //loking for closest object
    {
    d=1;
    f=0;
    IR = 0;
    closest = 0;
    sdistance = sonar;
    for (spincount = 1; spincount <= 15; spincount++)
    {
    turn_left();
    waitfor(DelayMs(555));
    if (sonar < sdistance)
    {closest = spincount;
     sdistance = sonar;
     }
     }
     stop();
     waitfor(DelayMs(2000));
     closestf = spincount*550;
     if (closestf < 2960)   //actual offset 5867
     closestf += 8880;

     closest -= 2960;
     turn_left();
     waitfor(DelayMs(closestf));

     spin = 0;
     IR = 1;
     }


  if((f == 1) && (volts3 > 1.3) && spin == 0)
  {
  pivot_left();
  waitfor(DelayMs(2740));
  waitfor(volts3 < .5);
  d=1;
  f=0;
  }
  /*else if(sonar < 4 && spin ==0 && d==1)
  {back_up();
```

```
 waitfor(DelayMs(1000));
 turn_left();
 waitfor(DelayMs(800));
 turn_right();
 waitfor(DelayMs(5000));
 d=1;
 f=0;
 }*/


if ((d == 1) && (f==0))                 // 34 to 0c
{
    go_straight();
}



if((f == 1) && (d == 1))
 {


  if ((volts1 > 2) || (volts0 > 2))  //If we get to close back off
    {
    go_straight2();
     waitfor((volts0 < 1.2) && (volts1 < 1.2));
                                              }

  if ((volts1 < .8) && (volts0 < .8))    //If to far move in
    {
    go_straight3();
     waitfor((volts0 > 1.2) || (volts1 > 1.2) || DelayMs(1000));
     if ((volts0 < 1.2) && (volts1 < 1.2))
                {
                go_straight();
                f=0;
                }

                                              }


  if ((volts1 - volts0) >.5)
    {
    pivot_right();
  waitfor((volts1 - volts0) < .1 || DelayMs(1000));
  straif_left();
  //waitfor(DelayMs(400));
```

```
  //pivot_right();
  //waitfor(DelayMs(800));
  //straif_left();
  }

  else
  {
    if((volts0 - volts1) >  .1)
    {
     straif_leftcl();
    waitfor(((volts0 - volts1) < .01) || (volts3 > 1) || ((volts1 - volts0) >  .4));
    straif_left();
    }

    if((volts0 - volts1) < -.1)
    {
     straif_leftcr();
    waitfor(((volts0 - volts1) > 0)  || (volts3 > 1) || ((volts1 - volts0) >  .4));
    straif_left();
    }
  }
  if (new == 1)
  {
    back_up();
    waitfor(DelayMs(1000));
    turn_right();
    waitfor(DelayMs(3300));
    go_straight();
    f = 0;
    d = 1;
     new = 0;
   }

 }

if((f == 0) && (d == 1) && ((volts0 > 1.2) || (volts1 > 1.2)))
  {

  if((volts0 - volts1) > 0)
  {
  turn_left();
  waitfor((volts0 - volts1) < 0);
  distance = volts0;
  }
  else
  {
```

```
          turn_right();
          waitfor((volts0 - volts1) > 0);
          distance = volts0;
          }


            straif_left();

            f=1;
         }



       if(!BitRdPortI(PGDR, S3) || estop == 1)
       {
       stop();
        waitfor(DelayMs(6200));


        }



                    }
          }
}


#define START           0x80
#define SINGLE          0x40
#define UNIPOLAR        0x04
#define MSBFIRST        0x02
#define POWERON         0x01


float ReadAD ( int Channel, int Samples )
{
          int Count, Command, j;
          unsigned long i;
          float Voltage;
          struct                                                    // structure for returned
data
```

```
        {       char    b;
                int     i;
        } Data;

        Command = START|SINGLE|UNIPOLAR|POWERON|MSBFIRST | ((Channel/2)<<3) | ((Channel&1)<<5);
        Command <<= 5;                                          // shift so result comes
back LSB justified
        Command = SwapBytes ( Command );    // adjust so high byte goes first

        i = 0L;                                                 // init accumulator

        for ( Count = 1; Count<=Samples; Count++ )
        {       BitWrPortI ( PFDR, &PFDRShadow, 0, 3 ); // enable /CS
                SPIWrRd ( &Command, &Data, 3 );                 // 3rd xmit byte is "don't
care"
                BitWrPortI ( PFDR, &PFDRShadow, 1, 3 ); // turn off /CS (1=off)
                j = Data.i;
                j = SwapBytes ( j ) & 0x0FFF;                   // put LSB first and keep
only 12 bits
                i += j;
        // update accumulator
        }

        Voltage = (float) i * ScaleFactor / (float) Samples;
}

// swap the high and low bytes of an int
// enter and exit with the int in HL
#asm
SwapBytes::
        ld              a, L            ; save the low byte
        ld              L, H            ; copy high byte to low byte
        ld              H, A            ; copy saved low byte to high byte
        ret
#endasm
```

WYSCAN.LIB

```
/*** BeginHeader */
#ifndef __WYSCAN_LIB
#define __WYSCAN_LIB
/*** EndHeader */


/* START FUNCTION DESCRIPTION ***********************************************
turn_right          <WYSCAN.LIB>

SYNTAX: void turn_right();

DESCRIPTION: Sends NAK sequence to slave.
       NAK is often sent when transfer is finished.

RETURN VALUE: void
END DESCRIPTION ***********************************************************/


//////////////////////////////////////////////
//////////////////////////////////////////////
/*** BeginHeader turn_right */

void turn_right();

/*** EndHeader */
void turn_right()
{

WrPortI(PWM0R,&PWM0RShadow,0x21);
WrPortI(PWM1R,&PWM1RShadow,0x21);
WrPortI(PWM2R,&PWM2RShadow,0x21);

//DelayMs(100);

WrPortI(PWM0R,&PWM0RShadow,0x27);
WrPortI(PWM1R,&PWM1RShadow,0x27);
WrPortI(PWM2R,&PWM2RShadow,0x27);

}
//////////////////////////////////////////////
//////////////////////////////////////////////
/*** BeginHeader turn_left */

void turn_left();
```

```
/*** EndHeader */
void turn_left()
{
WrPortI(PWM0R,&PWM0RShadow,0x21);
WrPortI(PWL0R,&PWL0RShadow,0x00);
WrPortI(PWM1R,&PWM1RShadow,0x21);
WrPortI(PWL1R,&PWL1RShadow,0x00);
WrPortI(PWM2R,&PWM2RShadow,0x21);
WrPortI(PWL2R,&PWL2RShadow,0x00);

//DelayMs(100);

WrPortI(PWM0R,&PWM0RShadow,0x17);
WrPortI(PWM1R,&PWM1RShadow,0x17);
WrPortI(PWM2R,&PWM2RShadow,0x17);
}

////////////////////////////////////////////
////////////////////////////////////////////
/*** BeginHeader pivot_left */

void pivot_left();

/*** EndHeader */
void pivot_left()
{

WrPortI(PWM0R,&PWM0RShadow,0x21);
WrPortI(PWM1R,&PWM1RShadow,0x21);
WrPortI(PWM2R,&PWM2RShadow,0x21);

//DelayMs(100);

WrPortI(PWM0R,&PWM0RShadow,0x17);
WrPortI(PWL0R,&PWL0RShadow,0x00);
WrPortI(PWM1R,&PWM1RShadow,0x21);
WrPortI(PWL1R,&PWL1RShadow,0x00);
WrPortI(PWM2R,&PWM2RShadow,0x17);
WrPortI(PWL2R,&PWL2RShadow,0x00);
}

////////////////////////////////////////////
////////////////////////////////////////////
/*** BeginHeader pivot_right */
```

```
void pivot_right();

/*** EndHeader */
void pivot_right()
{

WrPortI(PWM0R,&PWM0RShadow,0x21);
WrPortI(PWM1R,&PWM1RShadow,0x21);
WrPortI(PWM2R,&PWM2RShadow,0x21);

//DelayMs(100);

WrPortI(PWM0R,&PWM0RShadow,0x21);
WrPortI(PWL0R,&PWL0RShadow,0xC0);
WrPortI(PWM1R,&PWM1RShadow,0x23);
WrPortI(PWL1R,&PWL1RShadow,0x40);
WrPortI(PWM2R,&PWM2RShadow,0x24);
WrPortI(PWL2R,&PWL2RShadow,0x40);
}
////////////////////////////////////////////
////////////////////////////////////////////
/*** BeginHeader straif_left */

void straif_left();

/*** EndHeader */
void straif_left()
{

WrPortI(PWM0R,&PWM0RShadow,0x21);
WrPortI(PWL0R,&PWL0RShadow,0x00);
WrPortI(PWM1R,&PWM1RShadow,0x21);
WrPortI(PWL1R,&PWL1RShadow,0x00);
WrPortI(PWM2R,&PWM2RShadow,0x21);
WrPortI(PWL2R,&PWL2RShadow,0x00);

WrPortI(PWM0R,&PWM0RShadow,0x1F);
WrPortI(PWL0R,&PWL0RShadow,0x80);

WrPortI(PWM1R,&PWM1RShadow,0x1F);
WrPortI(PWL1R,&PWL1RShadow,0x80);

WrPortI(PWM2R,&PWM2RShadow,0x24);
WrPortI(PWL2R,&PWL2RShadow,0x40);
}
////////////////////////////////////////////
```

```
//////////////////////////////////////////
/*** BeginHeader straif_right */

void straif_right();

/*** EndHeader */
void straif_right()
{

WrPortI(PWM0R,&PWM0RShadow,0x21);
WrPortI(PWL0R,&PWL0RShadow,0x00);
WrPortI(PWM1R,&PWM1RShadow,0x21);
WrPortI(PWL1R,&PWL1RShadow,0x00);
WrPortI(PWM2R,&PWM2RShadow,0x21);
WrPortI(PWL2R,&PWL2RShadow,0x00);

WrPortI(PWM0R,&PWM0RShadow,0x22);
WrPortI(PWL0R,&PWL0RShadow,0x80);

WrPortI(PWM1R,&PWM1RShadow,0x22);
WrPortI(PWL1R,&PWL1RShadow,0x80);

WrPortI(PWM2R,&PWM2RShadow,0x18);
WrPortI(PWL2R,&PWL2RShadow,0x00);
}

//////////////////////////////////////////////
//////////////////////////////////////////////
/*** BeginHeader straif_leftcl */

void straif_leftcl();

/*** EndHeader */
void straif_leftcl()
{


WrPortI(PWM0R,&PWM0RShadow,0x1E);
WrPortI(PWL0R,&PWL0RShadow,0xC0);

WrPortI(PWM1R,&PWM1RShadow,0x1E);
WrPortI(PWL1R,&PWL1RShadow,0xC0);

WrPortI(PWM2R,&PWM2RShadow,0x24);
WrPortI(PWL2R,&PWL2RShadow,0x40);
}
```

```
//////////////////////////////////////////
//////////////////////////////////////////
/*** BeginHeader straif_leftcr */
void straif_leftcr();

/*** EndHeader */
void straif_leftcr()
{


WrPortI(PWM0R,&PWM0RShadow,0x20);
WrPortI(PWL0R,&PWL0RShadow,0x40);

WrPortI(PWM1R,&PWM1RShadow,0x20);
WrPortI(PWL1R,&PWL1RShadow,0x40);

WrPortI(PWM2R,&PWM2RShadow,0x24);
WrPortI(PWL2R,&PWL2RShadow,0x40);
}



//////////////////////////////////////////////
//////////////////////////////////////////////
/*** BeginHeader go_straight */

void go_straight();

/*** EndHeader */
void go_straight()
{

WrPortI(PWM0R,&PWM0RShadow,0x27);
WrPortI(PWM1R,&PWM1RShadow,0x17);
WrPortI(PWM2R,&PWM2RShadow,0x21);

WrPortI(PWL0R,&PWL0RShadow,0x00);
WrPortI(PWL1R,&PWL1RShadow,0x00);
WrPortI(PWL2R,&PWL2RShadow,0x00);
}



//////////////////////////////////////////////
```

```
//////////////////////////////////////////////
/*** BeginHeader go_straight2 */

void go_straight2();

/*** EndHeader */
void go_straight2()
{


WrPortI(PWM0R,&PWM0RShadow,0x17);
WrPortI(PWM1R,&PWM1RShadow,0x21);
WrPortI(PWM2R,&PWM2RShadow,0x27);

WrPortI(PWL0R,&PWL0RShadow,0x00);
WrPortI(PWL1R,&PWL1RShadow,0x00);
WrPortI(PWL2R,&PWL2RShadow,0x00);
}
//////////////////////////////////////////////
//////////////////////////////////////////////
/*** BeginHeader go_straight3 */

void go_straight3();

/*** EndHeader */
void go_straight3()
{

WrPortI(PWM0R,&PWM0RShadow,0x21);
WrPortI(PWM1R,&PWM1RShadow,0x17);
WrPortI(PWM2R,&PWM2RShadow,0x27);

WrPortI(PWL0R,&PWL0RShadow,0x00);
WrPortI(PWL1R,&PWL1RShadow,0x00);
WrPortI(PWL2R,&PWL2RShadow,0x00);
}
//////////////////////////////////////////////
//////////////////////////////////////////////
/*** BeginHeader stop */

void stop();

/*** EndHeader */
void stop()
{
```

```
WrPortI(PWM0R,&PWM0RShadow,0x21);
WrPortI(PWL0R,&PWL0RShadow,0x00);
WrPortI(PWM1R,&PWM1RShadow,0x21);
WrPortI(PWL1R,&PWL1RShadow,0x00);
WrPortI(PWM2R,&PWM2RShadow,0x21);
WrPortI(PWL2R,&PWL2RShadow,0x00);
}

////////////////////////////////////////////////
////////////////////////////////////////////////
/*** BeginHeader back_up */

void back_up();

/*** EndHeader */
void back_up()
{

WrPortI(PWM0R,&PWM0RShadow,0x21);
WrPortI(PWL0R,&PWL0RShadow,0x00);
WrPortI(PWM1R,&PWM1RShadow,0x21);
WrPortI(PWL1R,&PWL1RShadow,0x00);
WrPortI(PWM2R,&PWM2RShadow,0x21);
WrPortI(PWL2R,&PWL2RShadow,0x00);

//DelayMs(100);

WrPortI(PWM0R,&PWM0RShadow,0x17);
WrPortI(PWM1R,&PWM1RShadow,0x27);
WrPortI(PWM2R,&PWM2RShadow,0x21);

}

////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////

/*** BeginHeader */
#endif
/*** EndHeader */
```