

Introduction

This document covers my robot constructed for IMDL Summer 2002. The robot I constructed is called GreenJacket and is a golf-oriented robot, designed to help an individual practice his or her putting. First, an overview of the complete robot is outlined. Then each robot subsystem is discussed in more detail.

Integrated System

The overall goal of my robot is to manipulate golf balls. Specifically, it was designed so that it could be used to practice putting. The user putts balls across the floor, trying to get them into an opening on the side of the robot. The robot is able to collect the balls when they enter the opening, by lifting them up into itself. Once four balls have been collected, the robot returns to the user to unload the balls so that they can be putted again.

Also, when the ball is moving toward the robot, the robot is able to adjust for minor errors in putts by moving to its left or right. This allows the ball to enter the opening in the side of the robot so that it may be collected. This was first proposed as “lateral tracking” of the ball, but I have since adopted the term “lateral correction” to describe it. This is due to the limitations of particular solution which I implemented.

The processor I used is the Atmel AVR Mega163. It has 8 analog-to-digital inputs (each 10 bits wide) and 3 PWM channels which were used for collecting sensor data and controlling actuation. All software development was done in C using a combination of Atmel’s AVR Studio 3.54 (Windows based assembler and programming interface) and AVR-GCC (available under the

GNU license). Programming was conducted using Atmel's AVR In-System-Programmer (AVRISP). This allowed for quick and easy in-system programming, without the need to remove the chip from the board.

The board that I used was the MegaAVR-Dev board from Progressive Resources. It provided an in-system programming port, headers for I/O ports and on-board power regulation.

Mobile Platform

The mobile platform for GreenJacket was custom designed using AutoCAD. It has four wheels, and an aperture in one side where the golf balls are collected. This can be seen in Fig. 1.

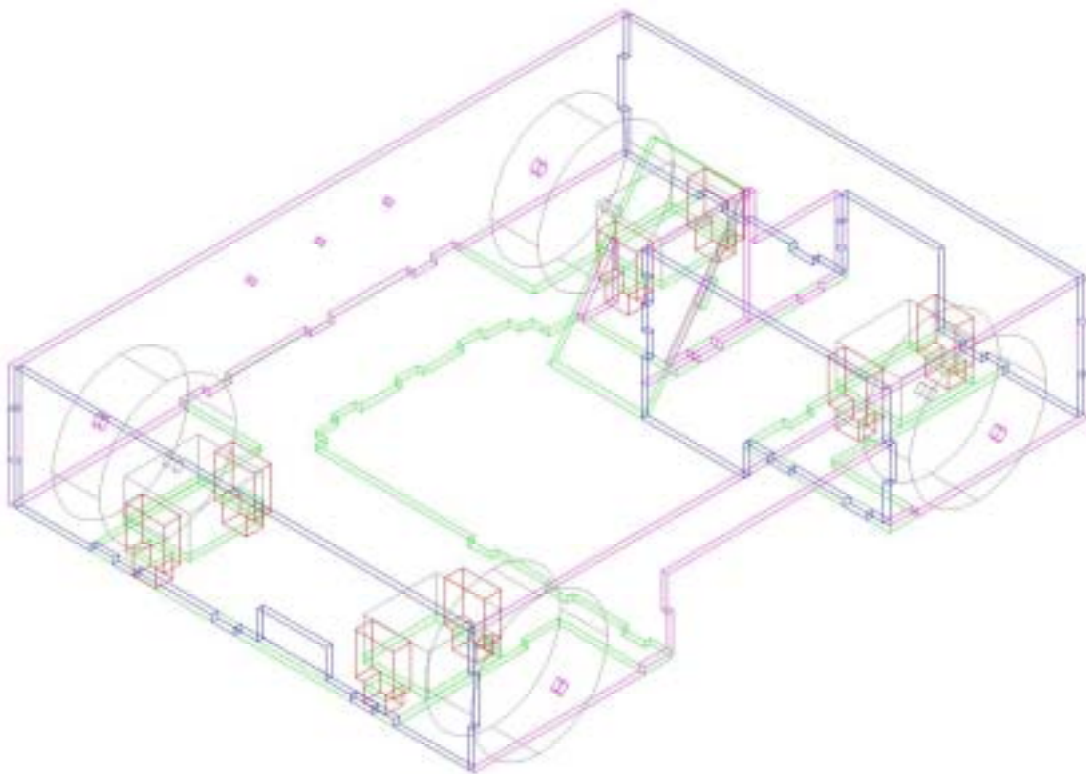


Figure 1. Isometric wireframe of GreenJacket (does not include ball collection mechanism).

Inside GreenJacket is the ball collection mechanism. Additionally, the platform has an opening where balls are released.

Actuation

Each of the four wheels is be driven by a servo that has been modified for continuous rotation. They are driven by two channels from the microcontroller. That is, the two left wheels receive the same control signal, and the two right wheels receive a different control signal. This allows GreenJacket to move similarly to a tank.

Servos also control the mechanism for picking up the balls. The ball collection mechanism was designed using two servos. One servo drives the rake, while the other servo drives the lifter. When balls enter the robot, a contact switch is triggered which generates an interrupt in the controller. The rake closes to trap the ball within the robot, and the lifter raises the ball up into the robot. The rake can be seen in yellow in Figures 2 and 3 below.

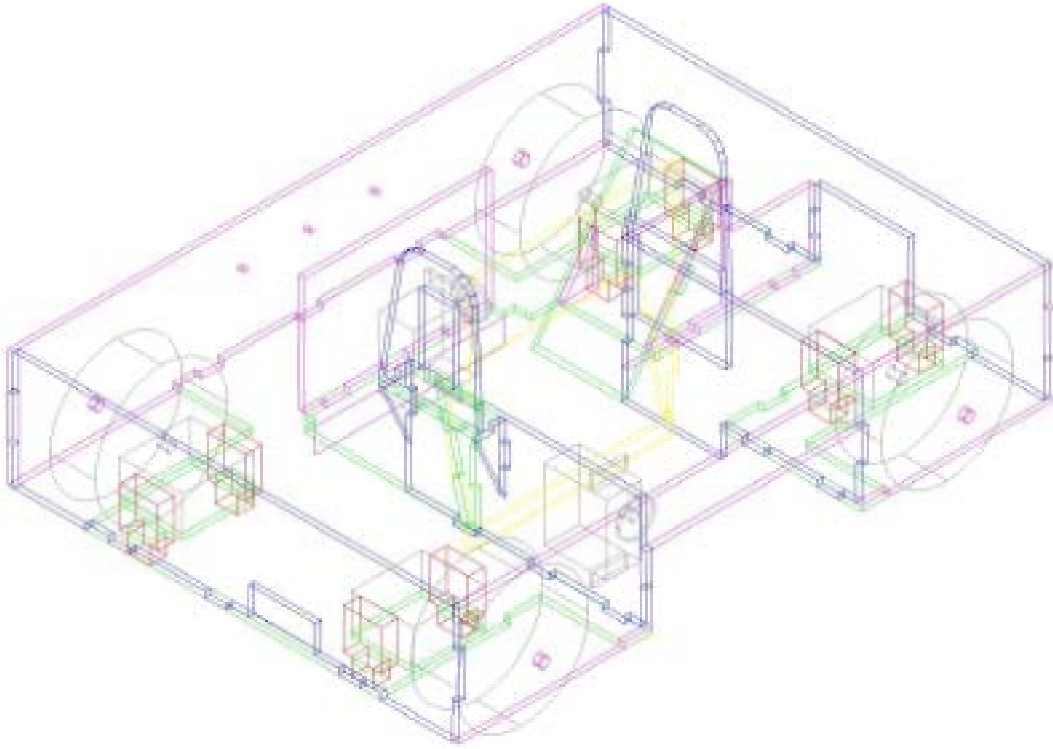


Figure 2. Isometric view of GreenJacket with ball collection mechanism.

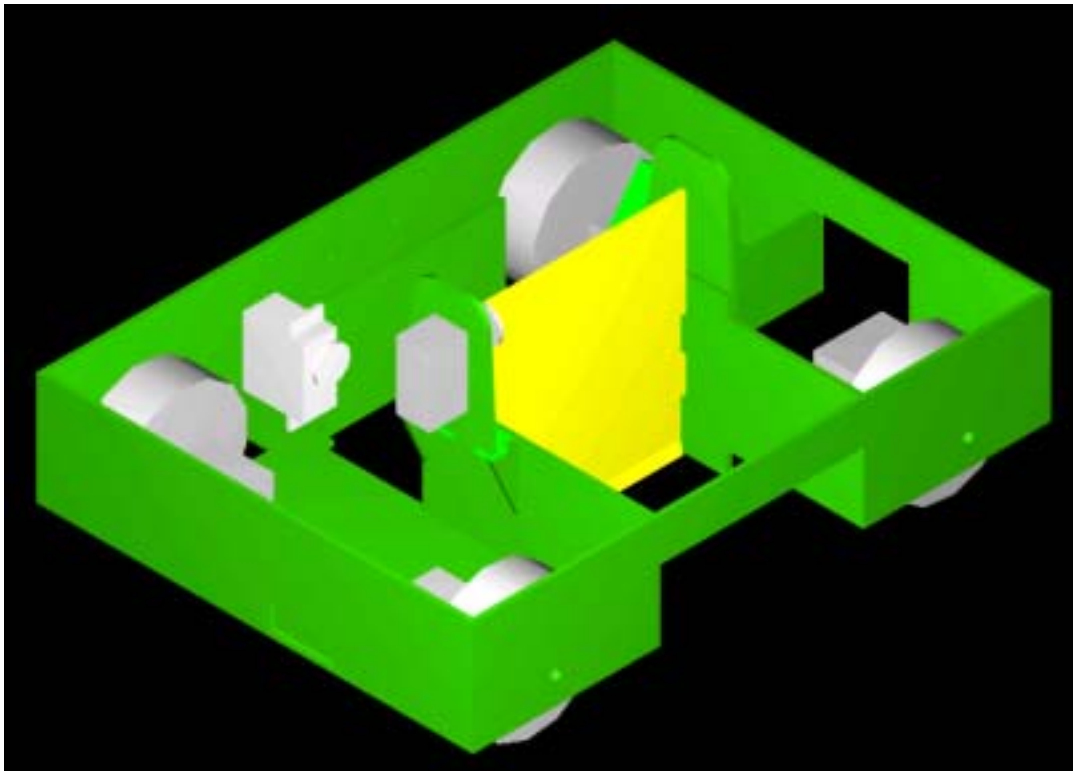


Figure 3. Rendered view of GreenJacket.

A third control servo operates the lever which controls the release of the balls. When the balls are collected, they are placed in a ramp whose end is blocked by this lever. The lever can be moved to release the balls one at a time.

Since the three PWM channels built-in to the Mega163 were used by the three servos for manipulating golf balls, I built additional hardware channels to use for the left and right motor channels. This was also done to satisfy my special sensor requirement.

Two channels alone are used for locomotion, which would leave only one channel to control a servo to manipulate golf balls. The two additional channels that I implemented left me more freedom in construction of the golf ball manipulation mechanism.

The PWM channels were implemented in an Altera MAX7032 CPLD. This a small CPLD, so the design had to be minimal. Logic for the hardware was described in VHDL and compiled using Altera's MAX+Plus II software. The overall design of the new system can be seen below in Figure 4.

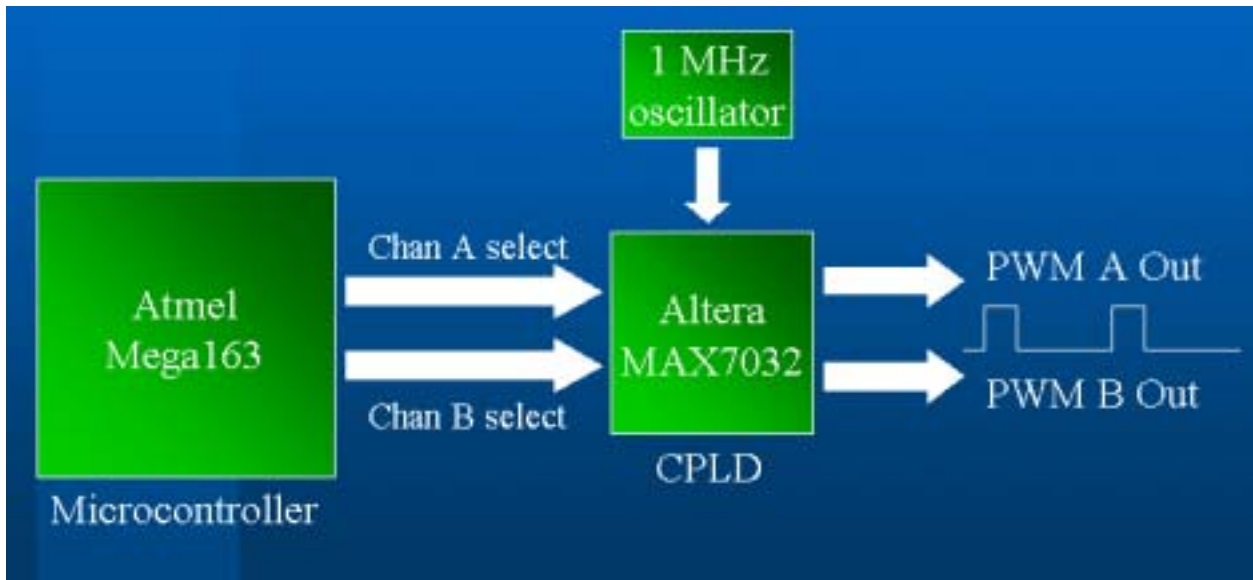


Figure 4. High-level architecture for new PWM system.

The processor communicates with the external hardware through on one of its output ports. In total, 4 bits are needed – each channel uses a 2-bit input bus to select the desired pulse width. This narrow input selection bus means that there are not that many options to select from. Due to size constraints imposed by the CPLD, three pulse width options are available: those that move the servo forward, in reverse, and stop the servo. Both servos can be set independently at any time.

Additionally, as can be seen in more detail in Figure 5, an external oscillator was needed to drive the PWM system. At the heart of the new hardware is a 15-bit up counter. This counter increments on each edge of the 1MHz signal. Internal logic resets the counter to zero when it reaches 20,000 (decimal). This provides the 20ms period necessary for the PWM output ($1e-6s$

(1 MHz) * 20,000 = 20ms).

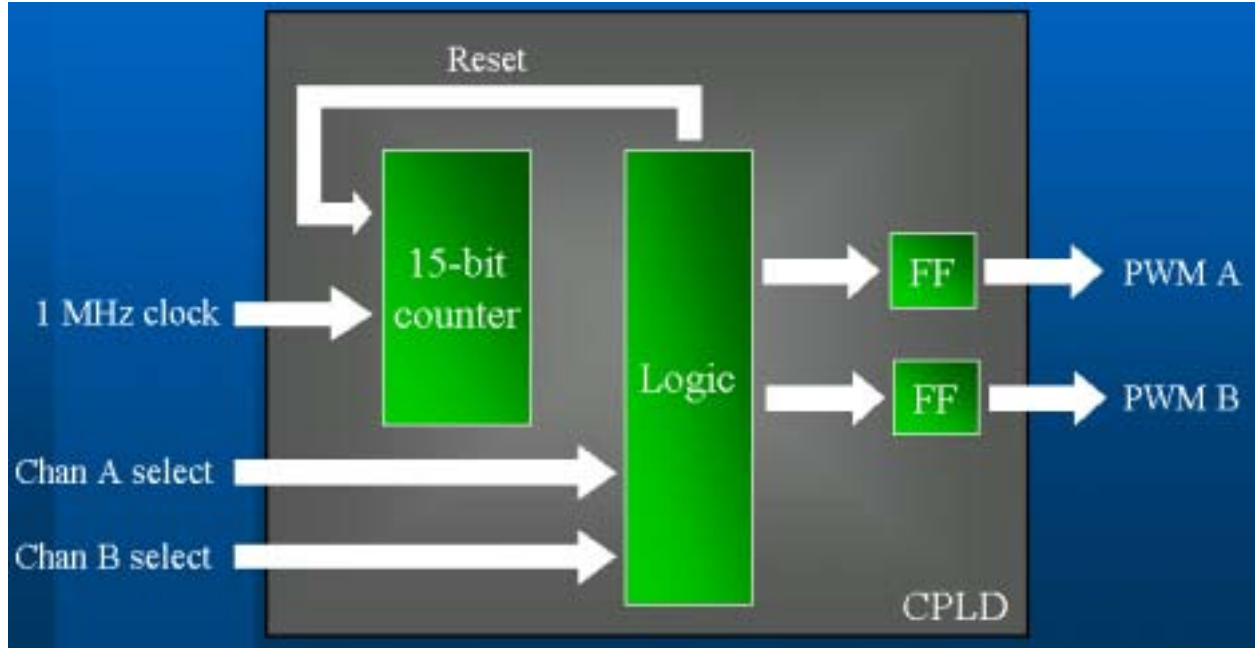


Figure 5. Internal architecture of hardware PWM system.

The high-time for the pulse output, and thus the direction of the servo, is determined by the 2-bit input selects. Each option (forward, reverse, or stop) corresponds to a particular counter value. When that input is selected, the PWM output for that channel is set high when the counter resets to zero, and then set low when the counter reaches the particular value—thus the pulse is generated for a fixed time. Finally, each output is run through a flip-flop (still in the CPLD) to synchronize it with the input clock and remove glitches (which are generated by the combinatorial logic network).

I observed the PWM system outputs using an oscilloscope. A sample reading can be seen below in Figure 6. Here, one can see the period of the signal (20 ms) and the length of the high time (1.75 ms). This is the largest value for the high-time that is used.

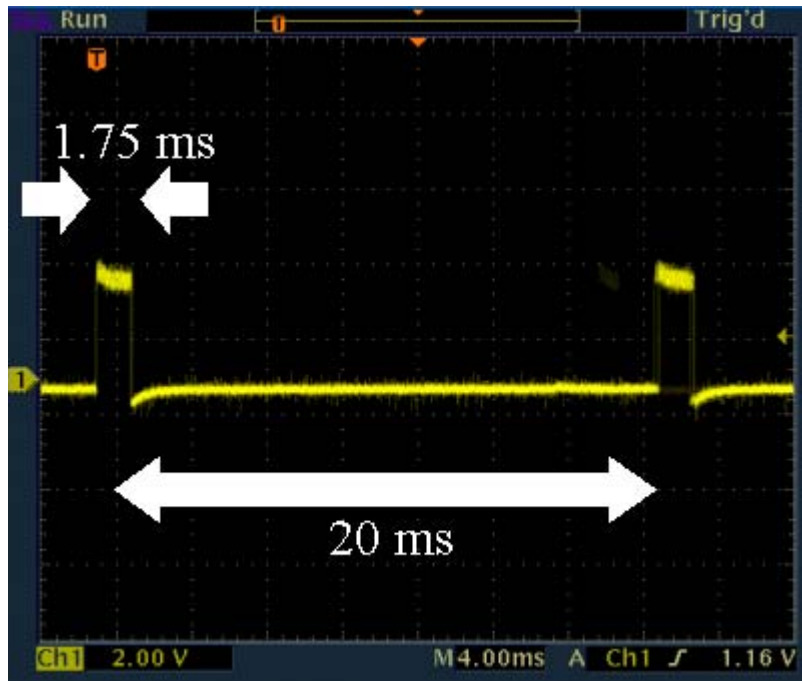


Figure 6. Oscilloscope observations made of new PWM outputs.

Most importantly, the new PWM channels work effectively with my servos. This allows me to use more servos in my robot, while abstracting into hardware the tedium of generating the correct pulse widths.

Sensors

GreenJacket uses both infrared (IR) and contact sensors. Two types of IR sensors were used: short range (Sharp GP2D120) and long range (Sharp GP2Y0A02YK). The short range sensors were used exclusively for wall detection. The long range sensors were used for both wall detection and lateral correction.

All four IR sensors are calibrated on power-up. This is done by first placing the robot in front of a wall to determine a threshold for wall avoidance. Then the long distance sensors are calibrated for lateral correction by placing balls in front of them.

Contact sensors were also used to build front and rear bump sensors. These switches are wired together in a resistor network and connected to a single analog port. A contact switch is also used inside the robot to detect when a ball is present so that it can be picked up.

Behaviors

GreenJacket's primary behavior is the collection of golf balls. Once a ball enters the robot and contacts the switch, it is lifted into the robot.

Additional behaviors complete the functions necessary to conduct a putting practice session. This includes traveling out from and back to the user, depositing golf balls, and lateral correction.

The robot starts with a 4 balls inside of it. When the user activates the putting practice program by tapping a bump sensor, the robot dispenses the balls at the feet of the user. It then drives out until it reaches a wall and then turns so that its opening is facing the user. Then it waits for the user to begin putting balls.

If a putt is coming towards the robot but is not lined up with the collection opening, the robot will move laterally to line up the opening with the path of the ball. In this way even errant shots

are collected. If an errant shot has caused the robot to move to one side, once the ball has been caught the robot will return to the position from which is started (i.e. it “re-centers” itself). Finally, once all the balls have been collected the robot returns to the user and deposits the balls so that he or she can continue practicing.

Conclusion

In conclusion, I am very pleased with the success of some parts of my robot and not as pleased with others. In general, most of the functionality was realized. While the lateral ball tracking was not particularly successful, the ball collection mechanism was a great success—it works routinely—and is due mostly to careful planning. Also, the external PWM hardware that I built worked out well. It provided for much easier software development.

If I were to continue working on GreenJacket, there are several things that I would change or do differently. In particular, I would use additional and/or different IR sensors for wall avoidance (the short range ones that I used are *very* short range) and I would attempt some other method for lateral ball tracking.

Appendix A: VHDL Code for PWM Hardware

```
-- Matthew Chernosky
-- 7/7/2002
-- motor_pwm_1M.vhd
-- Up-counter for PWM based on 1MHz clock rate.
-- Counts up to 20,000 (4E20h) and then resets. This results
-- in a period of 20ms.

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;

ENTITY motor_pwm_1M IS
PORT(
    clk : in STD_LOGIC;
    PWM_IN_A: in STD_LOGIC_VECTOR(1 downto 0);
    PWM_IN_B: in STD_LOGIC_VECTOR(1 downto 0);
    PWM_OUT_A: out STD_LOGIC;
    PWM_OUT_B: out STD_LOGIC
);
END motor_pwm_1M;

ARCHITECTURE behavior OF motor_pwm_1M IS

constant COUNT_MAX: std_logic_vector(14 downto 0)
    := "100111000100000"; --20,000
constant COUNT_RST: std_logic_vector(14 downto 0)
    := conv_std_logic_vector(0,10);

-- PWM input value constants for the three motion options
constant STOP: std_logic_vector(1 downto 0) := "00";
constant FORWARD: std_logic_vector(1 downto 0) := "01";
constant REVERSE: std_logic_vector(1 downto 0) := "10";

-- PWM counter value constants for the three motion options
constant STOP_C: std_logic_vector(14 downto 0)
    := conv_std_logic_vector(0,10);
constant FORWARD_C: std_logic_vector(14 downto 0)
    := "000010011101100"; -- 1260
constant REVERSE_C: std_logic_vector(14 downto 0)
    := "000011010111000"; -- 1720

signal count: STD_LOGIC_VECTOR (14 downto 0); -- counter

-- 00 = STOP
-- 01 = FORWARD
-- 10 = REVERSE
-- 11 = not used

-- PWM channel outputs before de-glitching
signal SPWM_OUT_A: STD_LOGIC;
signal SPWM_OUT_B: STD_LOGIC;

BEGIN
```

```

cnt: PROCESS (clk)
BEGIN
    IF (clk = '1' and clk'EVENT) THEN
        if (count < COUNT_MAX) then
            -- increment counter
            count <= count + 1;
        else
            -- reset counter when COUNT_MAX reached
            count <= COUNT_RST;
        end if;
    END IF;
END PROCESS;

```

```

update_a: PROCESS (count)
BEGIN
    if (PWM_IN_A = FORWARD) then
        if (count < FORWARD_C) then
            SPWM_OUT_A <= '1';
        else
            SPWM_OUT_A <= '0';
        end if;
    elsif (PWM_IN_A = REVERSE) then
        if (count < REVERSE_C) then
            SPWM_OUT_A <= '1';
        else
            SPWM_OUT_A <= '0';
        end if;
    else
        -- STOP
        SPWM_OUT_A <= '0';
    end if;
END PROCESS;

```

```

update_b: PROCESS (count)
BEGIN
    if (PWM_IN_B = FORWARD) then
        if (count < FORWARD_C) then
            SPWM_OUT_B <= '1';
        else
            SPWM_OUT_B <= '0';
        end if;
    elsif (PWM_IN_B = REVERSE) then
        if (count < REVERSE_C) then
            SPWM_OUT_B <= '1';
        else
            SPWM_OUT_B <= '0';
        end if;
    else
        -- STOP
        SPWM_OUT_B <= '0';
    end if;
END PROCESS;

```

```

clean_a: PROCESS (clk)
BEGIN
    if (clk = '1' and clk'EVENT) then

```

```
        PWM_OUT_A <= SPWM_OUT_A;
    end if;
END PROCESS;

clean_b: PROCESS (clk)
BEGIN
    if (clk = '1' and clk'EVENT) then
        PWM_OUT_B <= SPWM_OUT_B;
    end if;
END PROCESS;

END;
```

Appendix B: Firmware Source

```

/*****
File:          analog.h
Author:       Matthew Chernosky
Date:        7/4/2002
Description:  Includes functions read analog ports.
*****/

#include <io.h>

//channel definitions
#define CHANNEL0 0x00
#define CHANNEL1 0x01
#define CHANNEL2 0x02
#define CHANNEL3 0x03
#define CHANNEL4 0x04
#define CHANNEL5 0x05
#define CHANNEL6 0x06
#define CHANNEL7 0x07

#define MUX_CMASK 0xe0

void initAnalog(void);
uint8_t readAnalog(uint8_t channel);

/*****
Function:     initAnalog()
*****/
Parameters:  none
Returns:     void
Description:  Sets up ADC for single conversion, left
              adjusted mode.
*****/

void initAnalog(void)
{
    outp(0x00,DDRA);    //Set data direction for PORTA0 bits 5:0 to input

    outp(BV(ADLAR),ADMUX); //left align result
    outb(BV(ADEN) | BV(ADSC), ADCSR);
    while (inp(ADCSR) & BV(ADSC));
}

/*****
Function:     readAnalog(uint8_t channel)
*****/
Parameters:  The hex code for the ADC channel to read.
Returns:     The single byte representing the analog value
              read from the port.
Description:  Requests one conversion, waits for it to
              complete, and then returns the result.
*****/

uint8_t readAnalog(uint8_t channel)

```

```

{
    uint8_t old_ADMUX;

    old_ADMUX = inp(ADMUX);          /* save old ADMUX value */
    old_ADMUX = old_ADMUX & MUX_CMASK; /* clear MUX bits */
    outp(channel | old_ADMUX, ADMUX);
    outp(inp(ADCSR) | BV(ADSC), ADCSR);
    while (inp(ADCSR) & BV(ADSC));   /* wait until conversion done */
    return (inp(ADCH));
}

```

```

/*****
File:          bump.h
Author:       Matthew Chernosky
Date:        7/4/2002
Description:  Includes functions to test bumpers.
              Requires that initAnalog function in analog.h
              has been called.
              Front and rear bumpers attached on analog
              channel 4.
*****/

```

```
#include <io.h>
```

```

//bumper definitions
#define NO_BUMPER          0
#define FRONT_BUMPER      1
#define REAR_BUMPER       2

```

```

//define bumper analog channel
#define BUMP_CHANNEL      CHANNEL5

```

```
int readBumper(void);
```

```

/*****
Function:     readBumper()
*****/
Parameters:  none
Returns:     the bumper that is being pressed (integer)
Description:
*****/

```

```

int readBumper(void)
{
    uint8_t bump_value;

    bump_value = readAnalog(BUMP_CHANNEL);

    if (bump_value > 0x80)          //rear bumper pressed
    {
        return REAR_BUMPER;
    }

    else if (bump_value > 0x10)     //front bumper pressed
    {
        return FRONT_BUMPER;
    }
}

```

```

    }

    else                                     //no bumper pressed
        return NO_BUMPER;
}

/*****
File:          motors.h
Author:       Matthew Chernosky
Date:        7/14/2002
Description:   Includes functions to move motors. Drives CPLD
              PWM board connected to PORTB3:0.
              Left side = channel A = PORTB3:2
              Right side = channel B = PORTB1:0
*****/

#include <io.h>

#define ZERO_8 0x00
#define BIT45 0x30
#define ROTL 0x7a
#define ROTR 0x10
#define BIT0 0x01

/* Direction types */
#define FORWARD 0
#define REVERSE 1

/* Codes for CPLD board */
#define STOPPED 0x0 //servo stopped
#define CCLOCKW 0x1 //counterclockwise servo rotation
#define CLOCKW 0x2 //clockwise servo rotation

void initMotors(void);
void move(int direction);
void moveLeft(int direction);
void moveRight(int direction);
void turnLeft(void);
void turnRight(void);
void moveStop(void);

/*****
Function:     initMotors()
*****/
Parameters:  none
Returns:     void
Description:  Sets up low nibble of PORTB for output.
*****/

void initMotors(void)
{
    /* set low nibble of PORTB as output */
    outp(0x0f, DDRB);
}

```

```

void move(int direction)
{
    if (direction == FORWARD)
    {
        outp(((CCLOCKW << 2)|(CLOCKW)),PORTB);
    }

    else //reverse
    {
        outp(((CLOCKW << 2)|(CCLOCKW)),PORTB);
    }
}

void moveLeft(int direction)
{
    if (direction == FORWARD)
    {
        outp(((CCLOCKW << 2)|(STOPPED)),PORTB);
    }

    else //reverse
    {
        outp(((CLOCKW << 2)|(STOPPED)),PORTB);
    }
}

void moveRight(int direction)
{
    if (direction == FORWARD)
    {
        outp(((STOPPED << 2)|(CLOCKW)),PORTB);
    }

    else //reverse
    {
        outp(((STOPPED << 2)|(CCLOCKW)),PORTB);
    }
}

void turnLeft(void)
{
    outp(((CCLOCKW << 2)|(CCLOCKW)),PORTB);
}

void turnRight(void)
{
    outp(((CLOCKW << 2)|(CLOCKW)),PORTB);
}

void moveStop(void)
{
    /* write 00 for both channels */
    outp(((STOPPED << 2)|(STOPPED)),PORTB);
}

```

```

/*****

```

```

File:          servos.h
Author:       Matthew Chernosky
Date:        6/30/2002
Description:  Includes functions to move servos connected to
              PORTD4,5,6.

```

```

*****/

```

```

#include <io.h>

```

```

#define ZERO_8 0x00

```

```

void initMotors(void);
void moveServo(int servo_index, uint8_t position);

```

```

/*****
Function:    initServos
*****
Parameters:  none
Returns:     void
Description: Sets up PWM for Timer/Counter1 on PD4 and PD5.
*****/

```

```

void initServos(void)
{
    //initialize PWM on Timer/Counter1
    outp(BV(CS11),TCCR1B);
    outp(BV(PWM11)|BV(PWM10)|BV(COM1A1)|BV(COM1B1),TCCR1A);

    //initialize PWM on Timer/Counter2
    outp(BV(PWM2)|BV(COM21)|BV(CS21)|BV(CS20),TCCR2); //PCLK2/32

    //set PD7/OC2, PD5/OC1A, PD4/OC1B as output
    outp(0xB0,DDRD);
}

```

```

/*****
Function:    moveServos
*****
Parameters:  servo_index: 0, 1, 2
              position: hex value between 0x10 and 0x7a
Returns:     void
Description: Moves servo to position specified
*****/

```

```

void moveServo(int servo_index, uint8_t position)
{
    if (servo_index == 0)
    {
        //set servo 0
        outp(ZERO_8,OCR1AH);
        outp(position,OCR1AL);
    }

    else if(servo_index == 1)
    {
        //set servo 1
        outp(ZERO_8,OCR1BH);
    }
}

```

```

        outp(position,OCR1BL);
    }

    else //servo_index == 2
    {
        //set servo 2
        outp(position,OCR2);
    }
}

/*****
File:          4thewin.c
Author:        Matthew Chernosky
Date:         6/30/2002
Description:   Main robot program.
*****/

#include <io.h>
#include "motors.h"
#include "analog.h"
#include "bump.h"
#include "servos.h"
#include <interrupt.h>
#include <sig-avr.h>

#define B_THRES    0x0025           //threshold for detecting ball
#define B_THRES2  0x004f
#define O_THRES    0x37           //threshold for obstacle avoidance
                                   //lower thres = more sensitive

#define OR_TOL     0x01           //orthagonal tolerance

#define LEFT_IR    CHANNEL0
#define RIGHT_IR   CHANNEL1
#define RIGHT_BALL_IR CHANNEL2
#define LEFT_BALL_IR CHANNEL3

#define TRUE       1
#define FALSE      0

// function prototypes
void resetCatch(void);
void catchBall(void);
void obstacleAvoid(void);
int isBallPresent(void);
void turnLeft90(void);
void turnLeft90_2(void);
void turnRight90_2(void);

void wait(void);
void wait2(void);
void wait3(void);
void wait4(void);

void resetDrop(void);
void dropBall(void);

```

```

uint8_t averageLeft(void);
uint8_t averageRight(void);

void calibrateLeft(void);
void calibrateRight(void);
void getBComp(void);

void calibrateLeft0(void);
void calibrateRight0(void);

void krLED(void);
void slowFlashLED(void);
void quickFlashLED(void);
void inLED(void);

uint8_t left_value, right_value;
uint8_t left_ball_value, right_ball_value;

/* ball tracking thresholds */
uint8_t left_thres, right_thres, left_thres2, right_thres2;

/* object thresholds */
uint8_t left_othres, right_othres;
uint8_t left_ovalue, right_ovalue;

/* differences between matched sensors */
uint8_t o_comp, b_comp;

double distance;
int direction;

int bumper = NO_BUMPER;
int counter, c2;
int ballsCaught = 0;
int ballJustCaught = FALSE;
int ind;
int p;

//keep the last 32 samples for the ball IRs
uint8_t left[32];
uint8_t right[32];

int i,j,k;

int main(void)
{
    initMotors();
    initAnalog();
    initServos();

    outp(0xff,DDRC); //Set data direction for PORTC to output
    //outp(0x00,PORTC);

    outp(0x00,PORTC); //clear LEDs

```

```

/* PORTA setup */
outp(0x80,PORTA); //Enable internal pullup resistor for bit 7

/* PORTD setup */
outp(0x04,PORTD); //Enable internal pullup resistor for bit 2

/* Enable external interrupt 0 for ball catching */
outp(0x40,GIMSK);

/* Reset mechanisms */
resetCatch();
resetDrop();

/* Wait for bump to start */
while(readBumper() == NO_BUMPER);

wait();
wait();

/* Calibrate object sensors */
calibrateLeft0();
calibrateRight0();
getBComp();

/* Get difference comp for ball sensors */
calibrateLeft();
calibrateRight();
if (left_thres > right_thres)
    o_comp = left_thres - right_thres;
else
    o_comp = right_thres - left_thres;

krLED();

/* Wait for bump to calibrate */
while(readBumper() == NO_BUMPER);

wait();
wait();

/* Calibrate ball tracking sensors */
calibrateLeft();
calibrateRight();
left_thres2 = left_thres + 0x15;
right_thres2 = right_thres + 0x15;

/* enable interrupts */
sei();

krLED();

/* Wait for bump to start */
while(readBumper() == NO_BUMPER);

wait();

ind = 0;

```

```

wait();
wait();

while(1)
{
    /* unload all of the balls */
    for (counter = 0; counter < 4; counter++)
    {
        dropBall();
        wait();
        move(FORWARD);
        wait2();
        wait2();
        wait2();
        wait2();
        moveStop();
    }

    move(FORWARD);

    do{

        left_ovalue = readAnalog(LEFT_IR);
        right_ovalue = readAnalog(RIGHT_IR);
        wait4();

    }while((left_ovalue < left_othres) && (right_ovalue <
right_othres));

    moveStop();

    /* found wall */
    slowFlashLED();

    /* back up a bit */
    move(REVERSE);
    wait();
    wait();
    wait();
    wait();
    moveStop();

    turnLeft90();

    /* LEDs on, ready for putting */
    outp(0x7f,PORTC);

    /* initialize distance */
    distance = 0;

    /* wait for balls to be caught */
    while(ballsCaught < 4)
    {
        left_ball_value = readAnalog(LEFT_BALL_IR);

```

```

right_ball_value = readAnalog(RIGHT_BALL_IR);

if (left_ball_value >= left_thres)
{
    outp(0x03,PORTC); //LEDs
    move(REVERSE);
    distance--;
}
else if (right_ball_value >= right_thres)
{
    outp(0x60,PORTC); //LEDs
    move(FORWARD);
    distance++;
}
else
{
    /* LEDs on, ready for putting */
    outp(0x7f,PORTC);
    moveStop();
}

wait2();

if (ballJustCaught)
{
    if (distance < 0)
    {
        distance = 0 - distance;
        move(FORWARD);
    }

    else if (distance > 0)
        move(REVERSE);

    if (distance != 0)
    {
        p = 0;
        while (p < distance)
        {
            wait2();
            p++;
        }
    }
    distance = 0;
    ballJustCaught = FALSE;
}
}
ballsCaught = 0;

turnRight90_2();

move(REVERSE);

while(readBumper() == NO_BUMPER)
{
    wait4();
}

```

```

        move(FORWARD);
        wait();
        moveStop();
        wait();

        while(readBumper() == NO_BUMPER);
    }
}

void resetCatch(void)
{
    /* open rake */
    moveServo(0,0x32);

    /* lower lifter */
    moveServo(1,0x2b);

    wait();

    /* turn off servos */
    moveServo(0,0x00);
    moveServo(1,0x00);
}

void catchBall(void)
{
    /* close rake */
    moveServo(0,0x5a);

    wait();

    /* raise lifter */
    moveServo(1,0x69);
    wait();
    wait();
}

void resetDrop(void)
{
    moveServo(2,0x09);
    wait();
    moveServo(2,0x00);
}

void dropBall(void)
{
    moveServo(2,0x02);
    wait();
    moveServo(2,0x09);
}

void obstacleAvoid(void)
{
    left_value = readAnalog(CHANNEL0);
    right_value = readAnalog(CHANNEL1);
}

```

```

outp(left_value,PORTC);

/* object to left */
if (left_value > O_THRES)
{
    /* back up a bit */
    moveLeft(REVERSE);

    /* wait a bit */
    for (i=0; i<255; i++)
        for(j=0; j<255;j++)
            k++;
    for (i=0; i<255; i++)
        for(j=0; j<255;j++)
            k++;

    /* go forward again */
    move(FORWARD);
}

else if (right_value > O_THRES)
{
    /* back up a bit */
    moveRight(REVERSE);

    /* wait a bit */
    for (i=0; i<255; i++)
        for(j=0; j<255;j++)
            k++;
    for (i=0; i<255; i++)
        for(j=0; j<255;j++)
            k++;

    /* go forward again */
    move(FORWARD);
}
}

void turnLeft90(void)
{
    uint8_t comp;
    int s;

    /* turn a certain number of times to get close */
    for (s = 0; s < 6; s++)
    {
        moveLeft(FORWARD);
        wait();
        wait();
        moveStop();
        wait2();
        moveRight(REVERSE);
        wait();
        wait();
        moveStop();
        wait2();
    }
}

```

```

}

/* then start checking for the wall */
do
{
    moveLeft(FORWARD);
    wait2();
    wait2();
    wait2();
    wait2();
    moveStop();
    wait2();
    moveRight(REVERSE);
    wait2();
    wait2();
    wait2();
    wait2();
    moveStop();
    wait2();

    left_value = readAnalog(LEFT_IR);
    right_value = readAnalog(RIGHT_IR);

    /* get abs of the difference */
    if (left_value > right_value)
        comp = left_value - right_value;
    else
        comp = right_value - left_value;

}while(comp > o_comp);

moveLeft(FORWARD);
    wait2();
    wait2();
    wait2();
    wait2();
    moveStop();
    wait2();
    moveRight(REVERSE);
    wait2();
    wait2();
    wait2();
    wait2();
    moveStop();
    wait2();
}

void turnLeft90_2(void)
{
    uint8_t comp;
    int s;

    /* turn a certain number of times to get close */
    for (s = 0; s < 6; s++)
    {
        moveLeft(FORWARD);
        wait();
    }
}

```

```

        wait();
        moveStop();
        wait2();
        moveRight(REVERSE);
        wait();
        wait();
        moveStop();
        wait2();
    }

    /* then start checking for the wall */
do
{
    moveLeft(FORWARD);
    wait2();
    wait2();
    wait2();
    wait2();
    moveStop();
    wait2();
    moveRight(REVERSE);
    wait2();
    wait2();
    wait2();
    wait2();
    moveStop();
    wait2();

    left_value = readAnalog(LEFT_BALL_IR);
    right_value = readAnalog(RIGHT_BALL_IR);

    /* get abs of the difference */
    if (left_value > right_value)
        comp = left_value - right_value;
    else
        comp = right_value - left_value;

}while(comp > b_comp);

moveLeft(FORWARD);
    wait2();
    wait2();
    wait2();
    wait2();
    moveStop();
    wait2();
    moveRight(REVERSE);
    wait2();
    wait2();
    wait2();
    wait2();
    moveStop();
    wait2();
}

void turnRight90_2(void)
{

```

```

uint8_t comp;
int s;

/* turn a certain number of times to get close */
for (s = 0; s < 6; s++)
{
    moveRight(FORWARD);
    wait();
    wait();
    moveStop();
    wait2();
    moveLeft(REVERSE);
    wait();
    wait();
    moveStop();
    wait2();
}

/* then start checking for the wall */
do
{
    moveRight(FORWARD);
    wait2();
    wait2();
    wait2();
    wait2();
    moveStop();
    wait2();
    moveLeft(REVERSE);
    wait2();
    wait2();
    wait2();
    wait2();
    moveStop();
    wait2();

    left_value = readAnalog(LEFT_BALL_IR);
    right_value = readAnalog(RIGHT_BALL_IR);

    /* get abs of the difference */
    if (left_value > right_value)
        comp = left_value - right_value;
    else
        comp = right_value - left_value;
}while(comp > b_comp);

moveRight(FORWARD);
    wait2();
    wait2();
    wait2();
    wait2();
    moveStop();
    wait2();
    moveLeft(REVERSE);
    wait2();
    wait2();

```

```

        wait2();
        wait2();
        moveStop();
        wait2();
    }

int isBallPresent(void)
{
    uint8_t temp;

    /* read switch on PORTA bit 7 */
    temp = inp(PINA);
    temp = temp & 0x80;

    if (temp == 0x00)
        return TRUE;
    else
        return FALSE;
}

uint8_t averageLeft(void)
{
    uint16_t ret = 0;
    int c;

    for (c = 0; c < 32; c++)
        ret = ret + left[c];

    ret = ret >> 5;    // divide by 32 for average
    return (uint8_t)ret;
}

uint8_t averageRight(void)
{
    uint16_t ret = 0;
    int c;

    for (c = 0; c < 32; c++)
        ret = ret + right[c];

    ret = ret >> 5;    // divide by 32 for average
    return (uint8_t)ret;
}

void calibrateLeft(void)
{
    int c;
    for (c = 0; c < 32; c++)
    {
        left[c] = readAnalog(LEFT_BALL_IR);
        wait4();
    }
    left_thres = averageLeft();
}

void calibrateRight(void)
{

```

```

    int c;
    for (c = 0; c < 32; c++)
    {
        right[c] = readAnalog(RIGHT_BALL_IR);
        wait4();
    }
    right_thres = averageRight();
}

void getBComp(void)
{
    uint8_t left_temp, right_temp;
    int c;

    for (c = 0; c < 32; c++)
    {
        right[c] = readAnalog(RIGHT_BALL_IR);
        left[c] = readAnalog(LEFT_BALL_IR);
        wait4();
    }
    left_temp = averageLeft();
    right_temp = averageRight();

    if (left_temp > right_temp)
        b_comp = left_temp - right_temp;
    else
        b_comp = right_temp - left_temp;
}

void calibrateLeft0(void)
{
    int c;
    for (c = 0; c < 32; c++)
    {
        left[c] = readAnalog(LEFT_IR);
        wait4();
    }
    left_othres = averageLeft();
}

void calibrateRight0(void)
{
    int c;
    for (c = 0; c < 32; c++)
    {
        right[c] = readAnalog(RIGHT_IR);
        wait4();
    }
    right_othres = averageRight();
}

void krLED(void)
{
    uint8_t p;

    p = 0x40;
}

```

```

    for (counter = 0; counter < 6; counter++)
    {
        outp(p,PORTC);    //LEDs
        p = p >> 1;
        wait3();
    }
    for (counter = 0; counter < 6; counter++)
    {
        outp(p,PORTC);    //LEDs
        p = p << 1;
        wait3();
    }
    for (counter = 0; counter < 8; counter++)
    {
        outp(p,PORTC);    //LEDs
        p = p >> 1;
        wait3();
    }
}

void quickFlashLED(void)
{
    /* flash LEDs */
    outp(0x7f,PORTC);
    wait2();
    wait2();
    outp(0x00,PORTC);
    wait2();
    wait2();
    outp(0x7f,PORTC);
    wait2();
    wait2();
    outp(0x00,PORTC);
}

void slowFlashLED(void)
{
    /* flash LEDs */
    outp(0x7f,PORTC);
    wait();
    outp(0x00,PORTC);
    wait();
    outp(0x7f,PORTC);
    wait();
    outp(0x00,PORTC);
}

void inLED(void)
{
    int l;
    for (l = 0; l < 2; l++)
    {
        outp(0x41,PORTC);
        wait2();
        outp(0x22,PORTC);
        wait2();
    }
}

```

```

        outp(0x14,PORTC);
        wait2();
        outp(0x08,PORTC);
        wait2();
    }
    outp(0x00,PORTC);
}

void wait(void)
{
    for (i=0; i<255; i++)
        for(j=0; j<255;j++)
            k++;
}

void wait2(void)
{
    for (i=0; i<255; i++)
        for(j=0; j<50;j++)
            k++;
}

void wait3(void)
{
    for (i=0; i<255; i++)
        for(j=0; j<20;j++)
            k++;
}

void wait4(void)
{
    for (i=0; i<255; i++)
        for(j=0; j<5;j++)
            k++;
}

/*
    INT0: INTERRUPT SERVICE ROUTINE

    Ball sensor connected external interrupt
    pin INT0 (PORTD2). Collect ball when sensor
    generates and interrupt.
*/

SIGNAL(SIG_INTERRUPT0)
{
    catchBall();
    inLED();
    resetCatch();
    ballsCaught++;
    ballJustCaught = TRUE;

    /* LEDs on, ready for putting */
    outp(0x7f,PORTC);
}

```