```
#include <avr/io.h>
#include <math.h>
//#include <stdio.h>
//#include <stdlib.h>

typedef unsigned long u32;

/// Defines ///////////
double Lc = 5.25;  // suction cup asseembly length.
double Lr = 6;  // bump rod length.
double Drs = 9;  // distance from robot side to arm center.
double Dwtc = 2;  // distance from table wall to pocket center for sides
double Drtp = 1.75;  // distance from robot bottom left corner to pocket center
double Dls = 10.5;  // distance from robot arm center to light sensors.
double Db = .19;  // distance from robot side to end of derepressed bump sensor.
double Dirm = 2.4375; // distance from the ir to the side of the robot.
double thetaLS = 48.6;  // angle of light sensor box from the back surface opening towards
 the left.
double Hls = 1.25;  //height from platform to vacuum cup to release ball at the light sens
or.
double Hrack = .25;  //height from platform to vacuum cup to release ball at the rack.
double Hbase = 2.75;  // height from the platform to the pivot point of the shoulder.
double Hp = -.535;  // height from platform to the top of the pocket;
double A = 6.0625;  // upper arm length.
double B = 6.09375;  // fore arm length.
double C = 1.09375;  // wrist length.
double D = 1.8;  // ball finder/retriver length.
double Pdepth = -9.1; // distance from top of pocket to the bottom;
int normalL = 1470;  // tire speeds
int normalR = 1570;
int slowL   = 1510;
int slowR   = 1535;
int halfL   = 1490;
int halfR   = 1550;
int halt    = 1525;

double b1,b2,b3,b4;
int Sbump1,Sbump2,Sbump3,Sbump4, done;
double alpha, beta, theta, change, Lp, Lb, alphaSave, betaSave, hSave, Bchange;
int space, LSone[7], LStwo[7], LSthree[7], LSfour[7], LSfive[7], color[7], solids, balls;
int blueTwo = 0, blueThree = 0,orangeTwo = 0,orangeThree = 0,burgundyTwo = 0,burgundyThree
 = 0;
int greenTwo = 0,greenThree = 0,redTwo = 0,redThree = 0,purpleTwo = 0,purpleThree = 0;
int yellowTwo = 0, yellowThree = 0;
int percentL, percentR;
int IRtoTurn, IRtoSide, IRtoBack, IRtoFront, IRsave;

void Delay (u32 count);
void MuxOn(void);  // turns on the multiplexor for more A2D
void MuxOff(void);  // turns off the multiplexor
void ArmOn(void);  // turns the arm on
void initADC(void);  // initiates a to d conversions
void AtoDone(void);  // reads a2d from the multiplexor
void AtoDtwo(void);  // reads a 2 d from the front side ir
void AtoDthree(void); // reads a 2 d from the back side ir
void startADC(void); // starts free running conversions
void stopADC(void);  // stops conversions
void BumpHit(void);  // sees if the arm hits a ball
void BallDist(void);  //defines Lb witch is the length from the arm center to the ball
void BallRotation(void); // determines the change in theta needed to get teh ball
void ArmFindBall(void);  //The arm moves down on the pocket to find any balls
void ArmToBall(void);   // the arm once having found a ball moves to it and grabs it
void ArmFromPocket(void); // The arm moves up from the pocket
void ArmToPocket(void);  // The arm goes to the pocket
```

```c
void ArmNeutral(void); // the arm goes to a neutral postion
void ArmOff(void);  // The arm turns off
void ArmToLS(void);  // The arm goes to the Light Sensor box
void DetermineBallType(void);  // The cds cells take measuremnts to determine ball type/co
lor
void ArmToRack(void); // the arm moves the ball to its postion in the rack
double posA(double p);  // generates the pwm for the shoulder servo (angle alpha)
double posB(double p);  // generates the pwm for the elbow servo (angle beta)
double posG(double p);  // generates the pwm for the wrist servo (angle gamma)
double posBase(double p);  // generates the pwm for the wrist servo (angle theta)
double sine(double x);
double cosine(double x);
double asine(double x);
double acosine(double x);
double tangent(double x);
double atangent(double x);
void DriveOn(void);  // turns on the pwm for the wheels
void DriveOff(void);  // turns of the pwm for the wheels
void DriveFirstLeg(void);  // drives the front and back walls of the table
void Turn(void);  // turns the robot to the right, and backs it up to the pocket
void DriveSecondLeg(void); // drives the first part of the side walls
void DriveThirdLeg(void);  // drives the second part of the side walls
void calibrate(void);  // calibrates the ir sensors at he start
void Superturn(void);  // moves the robot to the rack postion when finished
void checkSpacing(void);  // determines if the robot is close enough to the wall after a t
urn
double dist(int d);  // gives the robots distance  from the wall using an average of the s
ide IR.
void lcd_int();
void lcd_string();
void lcd_delay();          // short delay (50000 clocks)
void lcd_init();          // sets lcd in 4 bit mode, 2-line mode, with cursor on and set to
 blink
void lcd_cmd();             // use to send commands to lcd
void lcd_disp();          // use to display text on lcd
void lcd_clear();          // use to clear LCD and return cursor to home position
void lcd_row(int row);       // use to put the LCD at the desired row


int main(void)
{
    // ORR1A = B5 = elbow, left , mid , right
    // ORR1B = B6 = base, left , mid , right
    // ORR1C = B7 = wrist, left , mid , right
    // ORR3A = E3 = shoulder, left , mid , right

    double x;
    double y;
    double PanlgeC;  //pocket angle corner
    double PanlgeS = 0;  //pocket angle side
    double Dir;
    Delay(1000000);
    ArmOn();  // get balls from pocket one
    MuxOn();
    initADC();
    startADC();
    lcd_init();      // set lcd in 4 bit mode, 2-line mode, with cursor on and set to blin
k
    lcd_string("RackAttack Ready to Roll");
    lcd_row(1);
    lcd_string("Begining IR Calibration");
    calibrate();
    Dir = dist(IRsave);
    x = Drs + Dir + 1.23744;
    y = 2.2187 + Db + 1.23744;
    PanlgeC = atangent(y/x);
    Lp = sqrt(y*y + x*x);
    done = 0;
    while (done == 0)
    {
        ArmNeutral();
        theta = PanlgeC;
        ArmToPocket();
        ArmFindBall();
        if (done == 0)
        {
            BallDist();  // defines Lb
            BallRotation();  // defines theta
            ArmToBall();
            ArmFromPocket();
            ArmNeutral();
            ArmToLS();
            DetermineBallType();
            ArmToRack();
        }
    }

    // initiates when all balls have been recovered from the pocket 1
    DriveOn();

    DriveFirstLeg();
    Turn();
    checkSpacing();
    Dir = dist(IRsave);
    DriveOff();

    x = Drs + Dir + 1.23744;
    y = 2.2187 + Db + 1.23744;
    PanlgeC = atangent(y/x);
    Lp = sqrt(y*y + x*x);
    ArmOn();    // get balls from pocket two
    done = 0;
    while (done == 0)
    {
        ArmNeutral();
        theta = PanlgeC;
        ArmToPocket();
        ArmFindBall();
        if (done == 0)
        {
            BallDist();  // defines Lb
            BallRotation();  // defines theta
            ArmToBall();
            ArmFromPocket();
            ArmNeutral();
            ArmToLS();
            DetermineBallType();
            ArmToRack();
        }
    }

    // initiates when all balls have been recovered from the pocket 2
    DriveSecondLeg();
    DriveOff();
    checkSpacing();
    Dir = dist(IRsave);
    Lp = Drs + Dwtc + Dir;
    ArmOn();    // get balls from pocket three
    done = 0;
    while (done == 0)
    {
        ArmNeutral();
        theta = PanlgeS;
        ArmToPocket();
        ArmFindBall();
        if (done == 0)
        {
            BallDist();  // defines Lb
            BallRotation();  // defines theta
            ArmToBall();
            ArmFromPocket();
            ArmNeutral();
            ArmToLS();
            DetermineBallType();
            ArmToRack();
        }
    }

    // initiates when all balls have been recovered from the pocket 3
    DriveThirdLeg();
    Turn();
    checkSpacing();
    Dir = dist(IRsave);;
    DriveOff();

    x = Drs + Dir + 1.23744;
    y = 2.2187 + Db + 1.23744;
    PanlgeC = atangent(y/x);
    Lp = sqrt(y*y + x*x);
    ArmOn();    // get balls from pocket four
    done = 0;
    while (done == 0)
    {
        ArmNeutral();
        theta = PanlgeC;
        ArmToPocket();
        ArmFindBall();
        if (done == 0)
        {
            BallDist();  // defines Lb
            BallRotation();  // defines theta
            ArmToBall();
            ArmFromPocket();
            ArmNeutral();
            ArmToLS();
            DetermineBallType();
            ArmToRack();
        }
    }

    // initiates when all balls have been recovered from the pocket 4
    DriveFirstLeg();
    Turn();
    checkSpacing();
    Dir = dist(IRsave);;
    DriveOff();

    x = Drs + Dir + 1.23744;
    y = 2.2187 + Db + 1.23744;
    PanlgeC = atangent(y/x);
    Lp = sqrt(y*y + x*x);
    ArmOn();    // get balls from pocket five
    done = 0;
    while (done == 0)
    {
        ArmNeutral();
        theta = PanlgeC;
        ArmToPocket();
        ArmFindBall();
        if (done == 0)
        {
            BallDist();  // defines Lb
            BallRotation();  // defines theta
            ArmToBall();
            ArmFromPocket();
            ArmNeutral();
            ArmToLS();
            DetermineBallType();
            ArmToRack();
        }
    }

    // initiates when all balls have been recovered from the pocket 5
    DriveSecondLeg();
    DriveOff();
    checkSpacing();
    Dir = dist(IRsave);

    Lp = Drs + Dwtc + Dir;
    ArmOn();    // get balls from pocket six
    done = 0;
    while (done == 0)
    {
        ArmNeutral();
        theta = PanlgeS;
        ArmToPocket();
        ArmFindBall();
        if (done == 0)
        {
            BallDist();  // defines Lb
            BallRotation();  // defines theta
            ArmToBall();
            ArmFromPocket();
            ArmNeutral();
            ArmToLS();
            DetermineBallType();
            ArmToRack();
        }
    }

    // initiates when all balls have been recovered from the pocket 6
    DriveThirdLeg();
    Superturn();
    DriveOff();
    lcd_clear();
    lcd_string("Rack Attack Victory!!!");
    lcd_row(1);
    lcd_string("Remove me and Play Pool");
    MuxOff();
    // Allert();

 return 0;
}
```

```
void Delay(u32 count)
{
    while(count--);
}

void ArmOn(void)
{
    PORTD=0xFF;  //Enable internal pull ups

    DDRB |= 0b11100000; //sets pins B5:7 to output
    DDRE |= 0b00001000; //sets pin E3 to output

    TCCR1A |= 0b10101000; // turn on non inverting pwm for channel A/B/C, on port B pin5-7

    TCCR1B |= 0b00010010; // bits 4&3 combined with bits 1&0 on TCC1A set a phase and freq
 correct PMWM with timer 3
    TCCR3A |= 0b10000000; // turn on non inverting pwm for channel A, on port E pin3
    TCCR3B |= 0b00010010; // bits 4&3 combined with bits 1&0 on TCC1A set a phase and freq
 correct PMWM with timer 3

    ICR1=20000;
    ICR3=20000;

    DDRG |= 0b00000010; // pin g1 set to output
    PORTG |= 0b00000010; // power servo relay switch
}

void MuxOn(void)
{
    DDRA |= 0b11110000;  // set a7-4 to output (for mux inputs)

}

void MuxOff(void)
{
    PORTA &= 0b00001111; // stop power to mux inputs
}


void initADC(void)
{
   DDRF = 0b00000000; // set port F to all input
                  // Note: when JTAGEN fuse is set, F4 - F7 don't work
   PORTF = 0x00;      // make sure pull up resistor is not enabled

   ADMUX = 0b01100000; // 5V reference, select
   ADCSRA |= 0b10100111; // turn on ADC, don't start conversions
                    // free funning
                    // divide clock by 128

}
void AtoDone(void)
{
   ADMUX = 0b0110000; // 5V reference, select channel0 (pin F0)
}

void AtoDtwo(void)
{
   ADMUX = 0b01100001; // 5V reference, select channel0 (pin F0)
}

void AtoDthree(void)
{
   ADMUX = 0b01100010; // 5V reference, select channel0 (pin F0)
}

void startADC(void)
{
   ADCSRA |= 0b01000000;   // start free running conversions

}

void stopADC(void)
{
   ADCSRA &= 0b10111111;   // stop free running conversions

}


void BumpHit(void)  // Analyze the bump sensors to determine if a ball was found. If one
                    // was found, save the location of that ball relative to the suction c
up.
{
    int value = 0;

    if (Sbump1 > 30 || Sbump2 > 30 || Sbump3 > 30 || Sbump4 > 30)
    {
        b3= 1;  // this tells me there is a ball
        value = Sbump1;
        b4 = 1;                      // this tells me what quadrent the ball is in
        if (Sbump2 > Sbump1 && Sbump2 > Sbump3 && Sbump2 > Sbump4) {
            value = Sbump2;
            b4 = 2;
            }
        if (Sbump3 > Sbump1 && Sbump3 > Sbump2 && Sbump3 > Sbump4) {
            value = Sbump3;
            b4 = 3;
            }
        if (Sbump4 > Sbump1 && Sbump4 > Sbump2 && Sbump4 > Sbump3) {
            value = Sbump4;
            b4 = 4;
            }
    }

    if (value < 34 || value > 31)
    {
        b1= 1.375;                     // this is the x direction corection
        b2= 0;                         // this is the y direction corection
    }

    if (value < 41 || value > 36)
    {
        b1= 1.191;
        b2= -.6875;
    }
    if (value < 53 || value > 42)
    {
        b1= 1.328;
        b2= -.3559;
    }
    if (value < 69 || value > 54)
    {
        b1= 1.191;
        b2= .6875;
    }
    if (value < 90 || value > 70)
    {
        b1= 1.191;
        b2= .3559;
    }

    if (Sbump1 > 30 && Sbump2 > 30)
    {
        b1= .9723;
        b2= .9723;
        b4 = 5;
    }
    if (Sbump1 > 30 && Sbump4 > 30)
    {
        b1= .9723;
        b2= .9723;
        b4 = 8;
    }
    if (Sbump2 > 30 && Sbump3 > 30)
    {
        b1= .9723;
        b2= .9723;
        b4 = 6;
    }
    if (Sbump3 > 30 && Sbump4 > 30)
    {
        b1= .9723;
        b2= .9723;
        b4 = 7;
    }

}

void BallDist(void)  // Determine the length of the arm needed to
                       // position the suction cup over the found ball.
{
    lcd_clear();
    lcd_string("Analyzing ball position");
    lcd_row(1);
    lcd_string("and Retrieving Ball");
    if (b4 == 1 || b4 == 5)
    {
        Lb = sqrt((b1+Lp)*(b1+Lp) + b2*b2);
    }

    if (b4 == 2 || b4 == 6)
    {
        if (b2>=0) Lb = sqrt((Lp-b2)*(Lp-b2) + b1*b1);
        else Lb = sqrt((Lp-b2)*(Lp-b2) + b2*b2);
    }

    if (b4 == 3 || b4 == 7)
    {
        Lb= sqrt((Lp-b1)*(Lp-b1) + b2*b2);
    }

    if (b4 == 4 || b4 == 8)
    {
        if (b2>=0) Lb = sqrt((Lp+b2)*(Lp+b2) + b1*b1);
        else Lb = sqrt((Lp+b2)*(Lp+b2) + b2*b2);
    }

}

void BallRotation(void)  // Determine the angle of rotation to position the suction cup ov
er the found ball
```

```
{
    double L = (A * cosine(alpha)) + (B * sine(beta - (90 - alpha))) + C + D;
    if (b4 == 1 || b4 == 5)
    {
        if (b2>=0)  theta = theta - atangent(b2/(L+b1));
        else        theta = theta + atangent(b2/(L-b1));
    }

    if (b4 == 2 || b4 == 6)
    {
        if (b2>=0)  theta = theta - atangent(b1/(L-b2));
        else        theta = theta - atangent(b1/(L-b2));
    }

    if (b4 == 3 || b4 == 7)
    {
        if (b2>=0)  theta = theta - atangent(b2/(L-b1));
        else        theta = theta + atangent(b2/(L+b1));
    }

    if (b4 == 4 || b4 == 8)
    {
        if (b2>=0)  theta = theta - atangent(b1/(L+b2));
        else        theta = theta - atangent(b1/(L+b2));
    }

}

void ArmFindBall(void)  // Lower the hand slowly into the pocket while serching for balls.
{
    Delay(33000);
        int i = 1;
        int j = 0;
        int k = 0;
    while (i == 1)
    {

        change = 2;
        b3 = 0;
        double L;
        if (k == 0) L = (A * cosine(alpha)) + (B * sine(beta - (90 - alpha))) + C + D;
        k = 1;
        alpha = alpha - change;
        double Bchange = asine((L - A*cosine(alpha))/B);
        beta = beta - (beta -Bchange);
        int H = (A * sine(alpha)) - (B * cosine(beta -(90-alpha))) + Hbase - Lr;
        hSave = H;
        if (H < (Hp - Pdepth + 2.25))
            {
                ArmFromPocket();
                ArmOff();
                done = 1;
                return;
            }
        OCR3A = posA(alpha);
        OCR1A = posB(beta);
        OCR1C = posG(180-alpha-beta);
        // set mux to each of the 4 bumps sequentually, and save the 4 a2d values
        AtoDone();
        PORTA |= 0x00;
        Delay(100);
        Sbump1 = ADCH;
        PORTA |= 0b10000000;
        Delay(100);
        Sbump2 = ADCH;
        PORTA |= 0b01000000;
        Delay(100);
        Sbump3 = ADCH;
        PORTA |= 0b11000000;
        Delay(100);
        Sbump4 = ADCH;
        BumpHit();
        if (b3 == 1)
        {
            change = change/2;
            j = j +1;
            b3 = 0;
            if (j == 1)
            i = 0;
            {
                alphaSave = alpha;
                betaSave = beta;
            }
        }

    }
}

void ArmToBall(void)  // once the balls position is found move the vacuum over it, onto it
, and engage.
{
    int H;
    int L;
    L = (A * cosine(alpha)) + (B * sine(beta -(90-alpha))) + C + D;
    H = (A * sine(alpha)) - (B * cosine(beta -(90-alpha))) + Hbase - Lc;  // move the hand
 up above the pocket
    while (H < Hp)
    {
        Delay (10000);
        change = 1;
        alpha = alpha + change;
        double Bchange = asine((L - A*cosine(alpha))/B);
        beta = beta - (beta -Bchange);
        OCR3A = posA(alpha);
        OCR1A = posB(beta);
        OCR1C = posG(180-alpha-beta);
        H = (A * sine(alpha)) - (B * cosine(beta -(90-alpha))) + Hbase - Lc;
    }

    Delay (10000);   //  move the hand over the ball postion
    alpha = 90;
    beta = 90;
    change = 5;
    OCR1B = posBase(theta);
    L = (A * cosine(alpha)) + (B * sine(beta -(90-alpha))) + C + D;
    while (L < Lb)
    {
        alpha = alpha - change;
        beta = beta + change;
        OCR1A = posB(beta);
        OCR3A = posA(alpha);
        L = (A * cosine(alpha)) + (B * sine(beta -(90-alpha))) + C + D;
        OCR1C = posG(180-alpha-beta);
        Delay(33000);
    }

    Delay(33000);
    double Hb = (hSave - (Lr - Lc) - .2);  // distance to ball is height bump rod hit at -
 difference from rod to cup - a correction factor.
    while (H > Hb)  // move the hand down onto the ball and engage suction
    {
        change = 2;
        alpha = alpha - change;
        Bchange = asine((L - A*cosine(alpha))/B);
        beta = beta - (beta -Bchange);
        H = (A * sine(alpha)) - (B * cosine(beta -(90-alpha))) + Hbase - Lc;
        OCR3A = posA(alpha);
        OCR1A = posB(beta);
        OCR1C = posG(180-alpha-beta);
    }
    DDRG |= 0b00000001; // pin g1 set to output
    PORTG |= 0b00000001; // power vacuum relay switch

}

void ArmFromPocket(void)
{
    if (b3 == 1)  // Since the pocket contained a ball move the
                  // arm slowly to a neutral position over the pocket.
    {
        while (alpha < 90 && beta > 90)
        {
            Delay (10000);
            change = 1;
            alpha = alpha + change;
            beta = beta - change;
            OCR3A = posA(alpha);
            OCR1A = posB(beta);
            OCR1C = posG(180-alpha-beta);
        }
        while (beta < 90)
        {
            Delay (10000);
            beta = beta - change;
            OCR1A = posB(beta);
            OCR1C = posG(180-alpha-beta);
        }
        while (alpha < 90)
        {
            Delay (10000);
            alpha = alpha + change;
            OCR3A = posA(alpha);
            OCR1C = posG(180-alpha-beta);
        }
    }
    else   // since the pocket is empty move the arm away slow, then fast.
    {
        int H = (A * sine(alpha)) - (B * cosine(beta -(90-alpha))) + Hbase - Lc;
        while (H < Hp)
        {
            Delay (10000);
            change = 1;
            alpha = alpha + change;
            beta = beta - change;
            OCR3A = posA(alpha);
            OCR1A = posB(beta);
            OCR1C = posG(180-alpha-beta);
            H = (A * sine(alpha)) - (B * cosine(beta -(90-alpha))) + Hbase - Lc;
        }
        OCR3A = posA(90);
        OCR1A = posB(90);
        OCR1C = posG(180-alpha-beta);
    }
}
void ArmToPocket(void)  // position the ball above the pocket
{
    lcd_clear();
    lcd_string("Move arm to Pocket");
    lcd_row(1);
    lcd_string("and begin ball search");
    Delay (10000);
    int alpha = 90;
    int beta = 90;
    int change = 5;
    OCR1B = posBase(theta);
    int L = (A * cosine(alpha)) + (B * sine(beta -(90-alpha))) + C + D;
    while (L < Lp)
    {
        alpha = alpha - change;
        beta = beta + change;
        OCR1A = posB(beta);
        OCR3A = posA(alpha);
        L = (A * cosine(alpha)) + (B * sine(beta -(90-alpha))) + C + D;
        Delay(33000);
    }
    OCR1C = posG(180-alpha-beta);
}
void ArmNeutral(void)
{
    lcd_clear();
    lcd_string("Move arm to Netral position");
    Delay (10000);
    OCR3A = posA(90); //verticle
    OCR1A = posB(90); //horizontal
    OCR1C = posG(0);  //180 - 90 -90, always horizontal
    OCR1B = posBase(0); // Parallel to the front and back, pointing left.
}

void ArmOff(void)
{
    Delay (10000);

    OCR1B = posBase(0);
    OCR3A = posA(90);
    OCR1C = posG(180-90-30);
    OCR1A = posB(30);

    DDRB &= 0b00000000;
    DDRE &= 0b00000000;

    PORTG &= 0b00000000;
    DDRG &= 0b00000000;

}
void ArmToLS(void)
{
    lcd_clear();
    lcd_string("Moving ball to determine");
    lcd_row(1);
    lcd_string("ball type");
    int L = (A * cosine(alpha)) + (B * sine(beta -(90-alpha))) + C + D;
    int H = (A * sine(alpha)) - (B * cosine(beta -(90-alpha))) + Hbase - Lc;
    OCR1B = posBase(thetaLS);
    change = 1;
    int i;
    while (L < Dls)  // moving the ball above the light sensor.
    {
        Delay(33000);
        alpha = alpha - change;
        beta = beta + change;
        OCR1A = posB(beta);
        OCR3A = posA(alpha);
        OCR1C = posG(180-alpha-beta);
        L = (A * cosine(alpha)) + (B * sine(beta -(90-alpha))) + C + D;
        alphaSave = alpha;
        betaSave = beta;
    }
    while (H > Hls -.2) // moving down to the light sensor.
    {
        Delay(33000);
        alpha = alpha - change;
        Bchange = asine((L - A*cosine(alpha))/B);
        beta = beta -  (beta -Bchange);
        OCR1A = posB(beta);
        OCR3A = posA(alpha);
        OCR1C = posG(180-alpha-beta);
        H = (A * sine(alpha)) - (B * cosine(beta -(90-alpha))) + Hbase - Lc;
    }

    PORTG &= 0b11111110; // power off vacuum relay switch

    AtoDone();
    i = 0;
    DDRG |= 0b00000100;
    PORTG |= 0b00000100;      // Turns LED from off to red

    while (i <7)
    {
        PORTA |= 0b11010000;  // Light sensor one output to mux 11
        Delay(33000);
        LSone[i] = ADCH;
        PORTA |= 0b00110000;  // mux 12
        Delay(33000);
        LStwo[i] = ADCH;
        PORTA |= 0b00010000;  // mux 8
        Delay(33000);
        LSthree[i] = ADCH;
        PORTA |= 0b10010000;  // mux 9
        Delay(33000);
        LSfour[i] = ADCH;
        PORTA |= 0b01010000;  // mux 10
        Delay(33000);
        LSfive[i] = ADCH;
        Delay(400000);
        i++;
        PORTG &= 0b11111011;
        Delay(100000);
        PORTG |= 0b00000100;  // turns LED off
        Delay(400000);
        PORTG &= 0b11111011;
        Delay(100000);
        PORTG |= 0b00000100;  // changes LED to next color
    }
    PORTG &= 0b11111011;
    Delay(100000);
    PORTG |= 0b00000100;      // Turns LED off, and ready to be red.

}

void DetermineBallType(void) // Finds the AtoD of the balls color, then determines if the
```

```
ball
                              // is a stripe or a solid based on the number of readings of
 that
                              // color, then decides what color that is and assigns it to
a spot.
{
    lcd_clear();
    lcd_string("Analyzing ball color");
    lcd_row(1);
    lcd_string("Analyzing Stripe-y-ness");
    int matches = 0;
    space = 1;

    while (space == 0)
    {
        int matchesOne = 1;
        int matchesTwo = 1;
        int matchesThree = 1;
        int matchesFour = 1;
        int matchesFive = 1;
        int solid = 0;  // logic, 0=stripe, 1=solid

        if (LSone[2] > LStwo[2]-8 && LSone[2] < LStwo[2]+8) matchesTwo = matchesTwo + 1;
        if (LSone[2] > LSthree[2]-8 && LSone[2] < LSthree[2]+8) matchesTwo = matchesTwo +
1;
        if (LSone[2] > LSfour[2]-8 && LSone[2] < LSfour[2]+8) matchesTwo = matchesTwo + 1;
        if (LSone[2] > LSfive[2]-8 && LSone[2] < LSfive[2]+8) matchesTwo = matchesTwo + 1;

        if (LStwo[2] > LSone[2]-8 && LStwo[2] < LSone[2]+8) matchesTwo = matchesTwo + 1;
        if (LStwo[2] > LSthree[2]-8 && LStwo[2] < LSthree[2]+8) matchesTwo = matchesTwo +
1;
        if (LStwo[2] > LSfour[2]-8 && LStwo[2] < LSfour[2]+8) matchesTwo = matchesTwo + 1;
        if (LStwo[2] > LSfive[2]-8 && LStwo[2] < LSfive[2]+8) matchesTwo = matchesTwo + 1;

        if (LSthree[2] > LStwo[2]-8 && LSthree[2] < LStwo[2]+8) matchesThree = matchesThre
e + 1;
        if (LSthree[2] > LSone[2]-8 && LSthree[2] < LSone[2]+8) matchesThree = matchesThre
e + 1;
        if (LSthree[2] > LSfour[2]-8 && LSthree[2] < LSfour[2]+8) matchesThree = matchesTh
ree + 1;
        if (LSthree[2] > LSfive[2]-8 && LSthree[2] < LSfive[2]+8) matchesThree = matchesTh
ree + 1;

        if (LSfour[2] > LStwo[2]-8 && LSfour[2] < LStwo[2]+8) matchesFour = matchesFour +
1;
        if (LSfour[2] > LSthree[2]-8 && LSfour[2] < LSthree[2]+8) matchesFour = matchesFou
r + 1;
        if (LSfour[2] > LSone[2]-8 && LSfour[2] < LSone[2]+8) matchesFour = matchesFour +
1;
        if (LSfour[2] > LSfive[2]-8 && LSfour[2] < LSfive[2]+8) matchesFour = matchesFour
+ 1;

        if (LSfive[2] > LStwo[2]-8 && LSfive[2] < LStwo[2]+8) matchesFive = matchesFive +
1;
        if (LSfive[2] > LSthree[2]-8 && LSfive[2] < LSthree[2]+8) matchesFive = matchesFiv
e + 1;
        if (LSfive[2] > LSfour[2]-8 && LSfive[2] < LSfour[2]+8) matchesFive = matchesFive
+ 1;
        if (LSfive[2] > LSone[2]-8 && LSfive[2] < LSone[2]+8) matchesFive = matchesFive +
1;

        matches = matchesOne;
        if (matchesTwo > matchesOne) matches = matchesTwo;
        if (matchesThree > matchesOne && matchesThree > matchesTwo) matches = matchesThree
```

```
;
        if (matchesFour > matchesOne && matchesFour > matchesTwo && matchesFour > matchesT
hree) matches = matchesFour;
        if (matchesFive > matchesOne && matchesFive > matchesTwo && matchesFive > matchesT
hree && matchesFive > matchesFour) matches = matchesFive;

        color[0] = LSone[0];
        color[1] = LSone[1];
        color[2] = LSone[2];
        color[3] = LSone[3];
        color[4] = LSone[4];
        color[5] = LSone[5];
        color[6] = LSone[6];
        if (LStwo[2] < LSone[2]  && (LStwo[2] > 235 || LStwo[2] < 200))
        {
            color[0] = LStwo[0];
            color[1] = LStwo[1];
            color[2] = LStwo[2];
            color[3] = LStwo[3];
            color[4] = LStwo[4];
            color[5] = LStwo[5];
            color[6] = LStwo[6];
        }
        if (LSthree[2] < LSone[2] &&  LSthree[2] < LStwo[2] && (LSthree[2] > 235 || LSthre
e[2] < 200))
        {
            color[0] = LSthree[0];
            color[1] = LSthree[1];
            color[2] = LSthree[2];
            color[3] = LSthree[3];
            color[4] = LSthree[4];
            color[5] = LSthree[5];
            color[6] = LSthree[6];
        }
        if (LSfour[2] < LSone[2] &&  LSfour[2] < LStwo[2] &&  LSfour[2] < LSthree[2] && (L
Sfour[2] > 235 || LSfour[2] < 200))
        {
            color[0] = LSfour[0];
            color[1] = LSfour[1];
            color[2] = LSfour[2];
            color[3] = LSfour[3];
            color[4] = LSfour[4];
            color[5] = LSfour[5];
            color[6] = LSfour[6];
        }
        if (LSfive[2] < LSone[2] && LSfive[2] < LStwo[2] && LSfive[2] < LSthree[2] && LSfi
ve[2] < LSfour[2] && (LSfive[2] > 235 || LSfive[2] < 200))
        {
            color[0] = LSfive[0];
            color[1] = LSfive[1];
            color[2] = LSfive[2];
            color[3] = LSfive[3];
            color[4] = LSfive[4];
            color[5] = LSfive[5];
            color[6] = LSfive[6];
        }

        if (matches == 4 || matches == 3) solid = 1;

        if (color[0] > 220 && color[2] < 240 && color[3] > 220)  // blue
        {
            lcd_clear();
            lcd_string("The ball is BLUE");
             if (blueTwo == 0) // first blue ball found
             {
                 space = 2;
                 if (solid == 1)
                    {
                        space = 3;
                        blueThree = 1;  // the solid space is now taken
                    }
                 blueTwo = 1;
             }
            if (blueTwo == 1) // second blue ball found
            {
                space = 3;
                if (blueThree == 1) space = 2;
            }
        }
        if ( color[4] < 180)  // orange
        {
            lcd_clear();
            lcd_string("The ball is ORANGE");
             if (orangeTwo == 0) // first orange ball found
             {
                 space = 6;
                 if (solid == 1)
                    {
                        space = 4;
                        orangeThree = 1;  // the solid space is now taken
                    }
                 orangeTwo = 1;
             }
            if (orangeTwo == 1) // second orange ball found
            {
                space = 4;
                if (orangeThree == 1) space = 6;
            }
        }
        if (color[0] < 180 && color[1] > 230)  // red
        {
            lcd_clear();
            lcd_string("The ball is RED");
             if (redTwo == 0) // first red ball found
             {
                 space = 15;
                 if (solid == 1)
                    {
                        space = 11;
                        redThree = 1;  // the solid space is now taken
                    }
                 redTwo = 1;
             }
            if (redTwo == 1) // second red ball found
            {
                space = 11;
                if (redThree == 1) space = 15;
            }
        }
        if (color[0] < 210 && color[1] > 240 && color[3] > 200)  // burgundy
        {
            lcd_clear();
            lcd_string("The ball is BURGUNDY");
             if (burgundyTwo == 0) // first burgundy ball found
             {
                 space = 7;
                 if (solid == 1)
                    {
                        space = 10;
                        burgundyThree = 1;  // the solid space is now taken
                    }
                 burgundyTwo = 1;
             }
            if (burgundyTwo == 1) // second burgundy ball found
            {
                space = 10;
                if (burgundyThree == 1) space = 7;
            }
        }
        if (color[0] < 225 && color[2] > 243 && color[3] < 221)  // green
        {
            lcd_clear();
            lcd_string("The ball is GREEN");
             if (greenTwo == 0) // first green ball found
             {
                 space = 9;
                 if (solid == 1)
                    {
                        space = 8;
                        greenThree = 1;  // the solid space is now taken
                    }
                 greenTwo = 1;
             }
            if (greenTwo == 1) // second green ball found
            {
                space = 8;
                if (greenThree == 1) space = 9;
            }
        }if (color[0] > 210 && color[3] < 225 && color[1] > 247 && color[5] < 217)  // pur
ple
        {
            lcd_clear();
            lcd_string("The ball is PURPLE");
             if (purpleTwo == 0) // first purple ball found
             {
                 space = 12;
                 if (solid == 1)
                    {
                        space = 14;
                        purpleThree = 1;  // the solid space is now taken
                    }
                 purpleTwo = 1;
             }
            if (purpleTwo == 1) // second purple ball found
            {
                space = 14;
                if (purpleThree == 1) space = 12;
            }
        }
        if (color[0] < 180  && color[2] > 226 && color[1] < 225)  // yellow
        {
            lcd_clear();
            lcd_string("The ball is YELLOW");
             if (yellowTwo == 0) // first yellow ball found
             {
                 space = 13;
                 if (solid == 1)
                    {
                        space = 1;
                        yellowThree = 1;  // the solid space is now taken
                    }
                 yellowTwo = 1;
             }
            if (yellowTwo == 1) // second yellow ball found
            {
                space = 1;
                if (yellowThree == 1) space = 13;
            }
        }
        if (color[1] > 248 && color[0] > 220 && color[6] > 228)
        {
            lcd_clear();
            lcd_string("The ball is BLACK");
            space = 5;  //black
        }
        if (color[2] < 235 && color[1] > 200 && color[4] > 224 && color[3] < 180)
        {
            lcd_clear();
            lcd_string("The is the CUE ball");
            space = 16;  // white
        }
    }

}


void ArmToRack(void)
{
    lcd_clear();
    lcd_string("Moving ball to rack");

    PORTG |= 0b00000001; // power vacuum relay switch
    int L = (A * cosine(alpha)) + (B * sine(beta -(90-alpha))) + C + D;
    int H;
    double x;
    double y;

    while (alpha < alphaSave || beta < betaSave)  // moving up from the light sensor.
    {
        Delay(33000);
        alpha = alpha + change;
        Bchange = asine((L - A*cosine(alpha))/B);
        beta = beta +  (Bchange - beta);
        OCR1A = posB(beta);
        OCR3A = posA(alpha);
        OCR1C = posG(180-alpha-beta);
        H = (A * sine(alpha)) - (B * cosine(beta -(90-alpha))) + Hbase - Lc;
    }
    while (alpha < 90 && beta > 90)   // moving to a netral psotion above the light sensor
.
        {
            Delay (10000);
            change = 1;
            alpha = alpha + change;
            beta = beta - change;
            OCR3A = posA(alpha);
            OCR1A = posB(beta);
            OCR1C = posG(180-alpha-beta);
        }
    while (beta < 90)
    {
        Delay (10000);
        beta = beta - change;
        OCR1A = posB(beta);
        OCR1C = posG(180-alpha-beta);
    }
    while (alpha < 90)
    {
        Delay (10000);
        alpha = alpha + change;
        OCR3A = posA(alpha);
        OCR1C = posG(180-alpha-beta);
    }
    if (space == 1)  // yellow solid
    {
        lcd_row(1);
        lcd_string("Position 1");
        x = -.1;
        y = 9.75;
    }
    if (space == 2)  // blue stripe
    {
        lcd_row(1);
        lcd_string("Position 2");
        x = -1.36;
        y = 7.36;
    }
    if (space == 3)  // blue solid
    {
        lcd_row(1);
        lcd_string("Position 3");
        x = 1.18;
        y = 7.36;
    }
    if (space == 4)  // orange solid
    {
        lcd_row(1);
        lcd_string("Position 4");
        x = -2.54;
        y = 5.24;
    }
    if (space == 5)  // black
    {
        lcd_row(1);
        lcd_string("Position 5");
        x = -.09;
        y = 5.24;
    }
    if (space == 6)  // orange sripe
    {
        lcd_row(1);
        lcd_string("Position 6");
        x = 2.36;
        y = 5.24;
    }
    if (space == 7)  // burgundy stripe
    {
        lcd_row(1);
        lcd_string("Position 7");
        x = -3.88;
        y = 3.21;
    }
    if (space == 8)  // green solid
    {
        lcd_row(1);
        lcd_string("Position 8");
        x = -1.43;
        y = 3.21;
    }
    if (space == 9)  //green stripe
    {
        lcd_row(1);
        lcd_string("Position 9");
        x = 1.12;
        y = 3.21;
    }
    if (space == 10)  // burgundy solid
    {
        lcd_row(1);
        lcd_string("Position 10");
        x = 3.5;
        y = 3.21;
    }
    if (space == 11)  // red solid
    {
        lcd_row(1);
        lcd_string("Position 11");
        x = -5;
        y = 1.1;
    }
    if (space == 12)  // puple stripe
    {
        lcd_row(1);
        lcd_string("Position 12");
        x = -2.56;
        y = 1.1;
    }
    if (space == 13)  // yellow stripe
    {
        lcd_row(1);
        lcd_string("Position 13");
        x = -.157;
        y = 1.1;
    }
    if (space == 14)  // purple solid
    {
        lcd_row(1);
        lcd_string("Position 14");
        x = 2.4;
        y = 1.1;
    }
    if (space == 15)  // red stripe
    {
        lcd_row(1);
        lcd_string("Position 15");
        x = 4.76;
        y = 1.1;
    }
    if (space == 16)  // white
    {
        lcd_row(1);
        lcd_string("Position 16");
        x = 12;
        y = 0;
    }

    Delay(33000);
    theta = 90 + atangent(x/(y+2.34375));
    double Lrack = sqrt(x*x +(y+2.34375)*(y+2.34375));

    //  moving to the rack position for the ball in hand.

    L = (A * cosine(alpha)) + (B * sine(beta -(90-alpha))) + C + D;
    H = (A * sine(alpha)) - (B * cosine(beta -(90-alpha))) + Hbase - Lc;
    Delay(33000);
    OCR1B = posBase(theta);
    change = 1;
    if (space == 1 || space == 2 || space == 3 || space == 16)
    {
        while (L < Lrack)
        {
            Delay(33000);
            alpha = alpha - change;
            beta = beta + change;
            OCR1A = posB(beta);
            OCR3A = posA(alpha);
            OCR1C = posG(180-alpha-beta);
            L = (A * cosine(alpha)) + (B * sine(beta -(90-alpha))) + C + D;
        }
        while (H > Hrack)
        {
            Delay(33000);
            alpha = alpha - change;
            Bchange = asine((L - A*cosine(alpha))/B);
            beta = beta -  (beta -Bchange);
            OCR1A = posB(beta);
            OCR3A = posA(alpha);
            OCR1C = posG(180-alpha-beta);
            H = (A * sine(alpha)) - (B * cosine(beta -(90-alpha))) + Hbase - Lc;
        }
    }
    if (space == 4 || space == 5 || space == 6 || space == 7 || space == 8 || space == 9 |
| space == 10 || space == 11 || space == 15)
    {
        while (L > Lrack)
        {
            Delay(33000);
            alpha = alpha + change;
            beta = beta - change*1.25;
            OCR1A = posB(beta);
            OCR1C = posG(180-alpha-beta);
            L = (A * cosine(alpha)) + (B * sine(beta -(90-alpha))) + C + D;
        }
        while (H > Hrack)
        {
            Delay(33000);
            alpha = alpha - change;
            Bchange = asine((L - A*cosine(alpha))/B);
            beta = beta -  (beta -Bchange);
            OCR1A = posB(beta);
            OCR3A = posA(alpha);
            OCR1C = posG(180-alpha-beta);
            H = (A * sine(alpha)) - (B * cosine(beta -(90-alpha))) + Hbase - Lc;
        }
    }
    if (space == 12 || space == 13 || space == 14)
    {
        while (alpha < 159)  alpha = alpha + change;
        while (L < Lrack)
        {
            Delay(33000);
            beta = beta - change;
            OCR1A = posB(beta);
            OCR1C = posG(180-alpha-beta);
            L = (A * cosine(alpha)) + (B * sine(beta -(90-alpha))) + C + D;
        }

    }
    PORTG &= 0b11111110; // power off vacuum relay switch

}

double posA(double p)  // can handle angles from 10 to 155
                       // arm is attached to servo at alpha 10 degrees.
{
    double pwm;

    if (p < 155  && p > 10) pwm = 10*(p-10) + 750;

    if (p >= 155) pwm = 2200;

    if (p <= 10) pwm = 750;

    return pwm;
}

double posB(double p)  // can handle angles from 23 to 168
                       // arm is attached to servo at beta 23 degrees.
{
    double pwm;

    if (p < 168  && p > 23) pwm = 10*(p-23) + 750;

    if (p >= 168)pwm = 2200;

    if (p <= 23) pwm = 750;

    return pwm;
}
double posG(double p)  // can handle angles from -65 to 85
                       // arm is attached to servo at gamma -65 degrees.
{
    double pwm;

    if (p < 80  && p > -65) pwm = 10*(p+65) + 750;

    if (p >= 80) pwm = 2200;

    if (p <= -65) pwm = 750;

    return pwm;
}
double posBase(double p)
{
    double pwm;
    if (p < 165  && p > -10) pwm = 10*(p+10) + 700;

    if (p >= 80) pwm = 2500;

    if (p <= -65) pwm = 700;
}

double sine(double x)
{
    double y = sin(x*3.14159265/180);
    return y;
}

double cosine(double x)
{
    double y = cos(x*3.14159265/180);
    return y;
}

double asine(double x)
{
    double y = asin(x)*180/3.14159265;
    return y;
}

double acosine(double x)
{
    double y = acos(x)*180/3.14159265;
    return y;
}

double tangent(double x)
{
    double y = tan(x)*180/3.14159265;
    return y;
}

double atangent(double x)
{
    double y = atan(x)*180/3.14159265;
    return y;
}

void DriveOn(void)
{
    PORTD=0xFF;  //Enable internal pull ups
    DDRE |= 0b00110000; //sets pin E4-5 to output
    TCCR3A |= 0b00101000; // turn on non inverting pwm for channel A, on port E pins 4&5
    TCCR3B |= 0b00010010; // bits 4&3 combined with bits 1&0 on TCC1A set a phase and freq
 correct PMWM with timer 3

    ICR3=20000;

    DDRG |= 0b00000010; // pin g1 set to output
    PORTG |= 0b00000010; // power servo relay switch

}

void DriveOff(void)
{
    TCCR3A |= 0b00000000;
}

void DriveFirstLeg(void)   // front or back wall driving
{
    lcd_clear();
    lcd_string("Driving to next hole");
    lcd_row(1);
    lcd_string("4 IR, 4 Bump running");
    change = 1;
    int speedL = normalL;
    int speedR = normalR;
    int increase = 0;
    int turn = 0;
    OCR3B = speedL;  // left wheel
    OCR3C = speedR;  // right wheel
    int one, two, three, four, five, SfrontIR, SbackIR;
    int i = 0;
    int oldIR;
    while (turn == 0)  // wall folowing
    {
        AtoDtwo();
        Delay(1000);
        one = ADCH;
        two = ADCH;
        three = ADCH;
        four = ADCH;
        SfrontIR = (one+two+three+four)/4;
        AtoDthree();
        Delay(1000);
        one = ADCH;
        two = ADCH;
        three = ADCH;
        four = ADCH;
        SbackIR = (one+two+three+four)/4;
        if (SfrontIR > SbackIR)
        {
            speedL = speedL - 4;
            OCR3B = speedL;
            i = 1;
            oldIR = SfrontIR/SbackIR;
        }
        if (SfrontIR < SbackIR)
        {
            speedL = speedL + 4;
            OCR3B = speedL;
            i = 2;
            oldIR = SbackIR/SfrontIR;
        }

        if (speedL <= normalL - 20 || speedL >= normalL + 20) speedL = normalL;

        AtoDone();
        PORTA |= 0b00100000;  // Front IR
        Delay(10000);
        int frontIR = ADCH;
        PORTA |= 0b10100000;  // Back IR
        Delay(10000);
        int backIR = ADCH;
        if (backIR/frontIR < 1.08) turn = 1;
        IRtoTurn = frontIR;
    }
}

void Turn(void)
{
    lcd_clear();
    lcd_string("Turning short");
    lcd_row(1);
    lcd_string("Then backing up to Pocket");
    OCR3C = halfR;  // right wheel to half speed of left wheel
    OCR3B = normalL;            // left wheel to normal speed regardless of what it had be
en.
    int done = 0;
    Delay(320000);
    int SfrontIR;
    int SbackIR;
    while (done == 0)
    {
        AtoDtwo();
        SfrontIR = ADCH;
        AtoDthree();
        SbackIR = ADCH;
        if (SfrontIR/SbackIR < 1.01  &&  SfrontIR/SbackIR > 1 ) done = 1;
    }
    Delay(32000);
    OCR3B = halt;  // left wheel
    OCR3C = halt;  // right wheel
    Delay(32000);
    int speedL = normalR;
    int speedR = normalL;
    OCR3B = speedL;  // left wheel
    OCR3C = speedR;  // right wheel
    int increases = 0;
    int reversed = 0;
    change = 1;
    int slowed = 0;
    int lrbs;
    int rrbs;
    int one, two, three, four, five;
    int i = 0;
    int oldIR;

    while (reversed == 0)  // wall folowing backwards
    {
        AtoDtwo();
        Delay(1000);
        one = ADCH;
        two = ADCH;
        three = ADCH;
        four = ADCH;
        SfrontIR = (one+two+three+four)/4;
        AtoDthree();
        Delay(1000);
        one = ADCH;
        two = ADCH;
        three = ADCH;
        four = ADCH;
        SbackIR = (one+two+three+four)/4;
        if (SfrontIR > SbackIR)
        {
            speedL = speedL + 4;
            OCR3B = speedL;
            i = 1;
            oldIR = SfrontIR/SbackIR;
        }
        if (SfrontIR < SbackIR)
        {
            speedL = speedL - 4;
            OCR3B = speedL;
            i = 2;
            oldIR = SbackIR/SfrontIR;
        }

        if (speedL <= normalL - 20 || speedL >= normalL + 20) speedL = normalL;
        AtoDone();
        PORTA |= 0b10100000;  // Back IR
        int backIR = ADCH;
        if (backIR/105 < 1.01)
        {
            if (slowed == 0)
            {
                OCR3B = slowL;
                OCR3C = slowR;
                slowed = 1;                 // only slow down one time
            }
        }
        AtoDone();
        PORTA |= 0b01100000;  // left rear bump sensor
        Delay(1000);
        lrbs = ADCH;
        PORTA |= 0b11100000;  // right rear bump sensor
        Delay(1000);
        rrbs = ADCH;
        if (lrbs > 230)
        {
            OCR3B = halt;
            PORTA |= 0b11100000;
            Delay(1000);
            while (rrbs < 230) rrbs = ADCH;
            OCR3C = halt;
            reversed = 1;
        }
        if (rrbs > 230)
        {
            OCR3C = halt;
            PORTA |= 0b01100000;
            Delay(1000);
            while (lrbs < 230) lrbs = ADCH;
            OCR3B = halt;
            reversed = 1;
        }

    }
    IRsave = (SfrontIR + SbackIR)/2;  // used to see if the bot is close enough to get the
 balls
}
    // retrieve all the balls from the pocket.

void DriveSecondLeg(void)   // side wall drivng first part
{
    lcd_clear();
    lcd_string("Driving to next hole");
    lcd_row(1);
    lcd_string("4 IR, 4 Bump running");
    change = 1;
    int increases = 0;
    int stop = 0;
    int nearPocket = 0;
    int oldReading;
    int speedL = normalL;
    int speedR = normalR;
    int increase = 0;
    int turn = 0;
    OCR3B = speedL;  // left wheel
    OCR3C = speedR;  // right wheel
    int one, two, three, four, five, SfrontIR, SbackIR;
    Delay(400000);
    int i = 0;
    int oldIR;

    while (stop == 0)  // wall folowing
    {
        AtoDtwo();
        Delay(1000);
        one = ADCH;
        two = ADCH;
        three = ADCH;
        four = ADCH;
        SfrontIR = (one+two+three+four)/4;
        AtoDthree();
        Delay(1000);
        one = ADCH;
        two = ADCH;
        three = ADCH;
        four = ADCH;
        SbackIR = (one+two+three+four)/4;
        if (SfrontIR > SbackIR)
        {
            speedL = speedL - 4;
            OCR3B = speedL;
            i = 1;
            oldIR = SfrontIR/SbackIR;
        }
        if (SfrontIR < SbackIR)
        {
            speedL = speedL + 4;
            OCR3B = speedL;
            i = 2;
            oldIR = SbackIR/SfrontIR;
        }

        if (speedL <= normalL - 20 || speedL >= normalL + 20) speedL = normalL;


        if (SbackIR/SfrontIR > 1.08)
        {
            OCR3B = slowL;  // left wheel slow down
            OCR3C = slowR;  // right wheel slow down
            nearPocket = 1;
        }
        if (nearPocket == 1)
        {
            AtoDthree();
            Delay(1000);
            SbackIR = ADCH;
            if (SfrontIR/SbackIR > 1.08) nearPocket = 2;
            oldReading = 140;
        }
        if (nearPocket == 2)
        {
            SbackIR = ADCH;
            if (oldReading < SbackIR) stop = 1;
            oldReading = ADCH;
        }

    }
    OCR3B = halt;  // left wheel
    OCR3C = halt;  // right wheel
    DriveOff();
}

    // get the balls from this pocket
void DriveThirdLeg(void)    // side wall drivng second part
{
    lcd_clear();
    lcd_string("Driving to next hole");
    lcd_row(1);
    lcd_string("4 IR, 4 Bump running");
    change = 1;
    int speedL = normalL;
    int speedR = normalR;
    int increase = 0;
    int turn = 0;
    OCR3B = speedL;  // left wheel
    OCR3C = speedR;  // right wheel
    int one, two, three, four, five, SfrontIR, SbackIR, frontIR, backIR;
    Delay(400000);
    int i = 0;
    int oldIR;

    while (turn == 0)  // wall folowing
    {
        AtoDtwo();
        Delay(1000);
        one = ADCH;
        two = ADCH;
        three = ADCH;
        four = ADCH;
        SfrontIR = (one+two+three+four)/4;
        AtoDthree();
        Delay(1000);
        one = ADCH;
        two = ADCH;
        three = ADCH;
        four = ADCH;
        SbackIR = (one+two+three+four)/4;
        if (SfrontIR > SbackIR)
        {
            speedL = speedL - 4;
            OCR3B = speedL;
            i = 1;
            oldIR = SfrontIR/SbackIR;
        }
        if (SfrontIR < SbackIR)
        {
            speedL = speedL + 4;
            OCR3B = speedL;
            i = 2;
            oldIR = SbackIR/SfrontIR;
        }

        if (speedL <= normalL - 20 || speedL >= normalL + 20) speedL = normalL;

        AtoDone();
        PORTA |= 0b00100000;  // Front IR
        frontIR = ADCH;
        PORTA |= 0b10100000;  // Back IR
        backIR = ADCH;
        if (frontIR == IRtoTurn) turn = 1;
    }
}

void calibrate(void)
{
        AtoDone();
        PORTA |= 0b00100000;  // Front IR
        int frontIR = ADCH;
        PORTA |= 0b10100000;  // Back IR
        int backIR = ADCH;
        IRtoBack = frontIR;
        IRtoFront = backIR;
        AtoDtwo();
        int SfrontIR = ADCH;
        AtoDthree();
        int SbackIR = ADCH;
        IRtoSide = (SfrontIR+SbackIR)/2;
        IRsave = IRtoSide;
}
```

```c
void Superturn(void)
{
    lcd_clear();
    lcd_string("Retrieved all the Balls");
    lcd_row(1);
    lcd_string("Moving to Rack Position");
    OCR3C = slowR;  // right wheel to very slow speed
    OCR3B = normalL; // left wheel to normal speed regardless of what it had been.
    int done = 0;
    int SfrontIR;
    int SbackIR;
    Delay(400000);
    while (done == 0)
    {
        AtoDtwo();
        SfrontIR = ADCH;
        AtoDthree();
        SbackIR = ADCH;
        if (SfrontIR/SbackIR < 1.04  &&  SfrontIR/SbackIR > 1 ) done = 1;
    }
    Delay(32000);
    OCR3B = halt;  // left wheel
    OCR3C = halt;  // right wheel
    Delay(32000);
    int speedL = normalR;
    int speedR = normalL;
    OCR3B = speedL;  // left wheel
    OCR3C = speedR;  // right wheel
    int increases = 0;
    int racked = 0;
    change = 1;

    while (racked == 0)  // wall folowing backwards
    {
        AtoDtwo();
        int SfrontIR = ADCH;
        AtoDthree();
        int SbackIR = ADCH;
        if (increases < 10)
        {
            if (SfrontIR > SbackIR) OCR3B = speedL - change;
            if (SfrontIR < SbackIR) OCR3C = speedR - change;
            increases = increases + 1;
        }
        if (increases >= 10)
        {
            if (SfrontIR < SbackIR) OCR3B = speedL + change;
            if (SfrontIR > SbackIR) OCR3C = speedR + change;
            increases = increases + 1;
            if (increases == 19) increases = 0;
        }
        AtoDone();
        PORTA |= 0b10100000;  // Back IR
        int backIR = ADCH;
        if (backIR >=  IRtoFront + 5)
        {
            OCR3B = halt;  // left wheel
            OCR3C = halt;  // right wheel
            DriveOff();
            racked = 1;
        }
    }
}
```

```
void checkSpacing(void)
{
    lcd_clear();
    lcd_string("Deciding if I'm close");
    lcd_row(1);
    lcd_string("Moving if I'm not.");
    int SfrontIR;
    int SbackIR;
    if (IRsave < IRtoSide - 20) // IRsave taken after Turn.
    {
        change = 1;
        int speedL = halfL;
        int speedR = halfR;
        int increases = 0;
        OCR3B = speedL;  // left wheel - 2515
        OCR3C = speedR + 15;  // right wheel +15
        Delay (330000);
        OCR3B = speedL;  // left wheel
        OCR3C = slowR;  // right wheel
        Delay (150000);
        OCR3B = slowL;
        OCR3C = slowR;
        int reversed = 0;

        AtoDtwo();
        Delay(1000);
        SfrontIR = ADCH;
        AtoDthree();
        Delay(1000);
        SbackIR = ADCH;

        while (SbackIR/SfrontIR > 1.08 || SbackIR/SfrontIR < .92)  // wall folowing
        {
            AtoDtwo();
            Delay(1000);
            SfrontIR = ADCH;
            AtoDthree();
            Delay(1000);
            SbackIR = ADCH;
            if (increases < 10)
            {
                if (SfrontIR > SbackIR) OCR3B = speedL + change;
                if (SfrontIR < SbackIR) OCR3C = speedR + change;
                increases = increases + 1;
            }
            if (increases >= 10)
            {
                if (SfrontIR < SbackIR) OCR3B = speedL - change;
                if (SfrontIR > SbackIR) OCR3C = speedR - change;
                increases = increases + 1;
                if (increases == 19) increases = 0;
            }
        }
        OCR3B = halt;
        OCR3C = halt;

        int slowed = 0;

        while (reversed == 0)  // wall folowing backwards
        {
            AtoDtwo();
            Delay(10000);
            SfrontIR = ADCH;
            AtoDthree();
            Delay(10000);
            SbackIR = ADCH;
            speedL = halfL;
            speedR = halfR;

            if (increases < 10)
            {
                if (SfrontIR > SbackIR) OCR3B = speedL - change;
                if (SfrontIR < SbackIR) OCR3C = speedR - change;
                increases = increases + 1;
            }
            if (increases >= 10)
            {
                if (SfrontIR < SbackIR) OCR3B = speedL + change;
                if (SfrontIR > SbackIR) OCR3C = speedR + change;
                increases = increases + 1;
                if (increases == 19) increases = 0;
            }
            AtoDone();
            PORTA |= 0b10100000;  // Back IR
            int backIR = ADCH;
            if (backIR/105 < 1.01)
            {
                if (slowed == 0)
                {
                    OCR3B = slowL;
                    OCR3C = slowR;
                    slowed = 1;                // only slow down one time
                }
            }
            AtoDone();
            PORTA |= 0b01100000;  // left rear bump sensor
            Delay(1000);
            int lrbs = ADCH;
            PORTA |= 0b11100000;  // right rear bump sensor
            Delay(1000);
            int rrbs = ADCH;
            if (lrbs < 250)
            {
                OCR3B = halt;
                PORTA |= 0b11100000;
                Delay(1000);
                while (rrbs >250) rrbs = ADCH;
                OCR3C = halt;
                reversed = 1;
            }
            if (rrbs < 250)
            {
                OCR3C = halt;
                PORTA |= 0b01100000;
                Delay(1000);
                while (lrbs >250) lrbs = ADCH;
                OCR3B = halt;
                reversed = 1;
            }

        }
        IRsave = (SfrontIR + SbackIR)/2;  // used to see if the bot is close enough to get
 the balls
    }
    IRsave = (SfrontIR + SbackIR)/2;
}
```

```
double dist(int d)   // For short IR on the side, distance from robot side to wall
{
    double Dir = -.0541*d*d*d + 2.2844*d*d - 34.44*d + 222.75 - Dirm;
    return Dir;

    //For LONG IR
    // double Dir = 17.74-.1319*d;
    // return Dir;
}

/* IMPORTANT!

Before using this code make sure your LCD is wired up the same way mine is, or change the
code to
match the wiring of your own LCD.  My LCD is connected to PortC of the At-Mega128 in the f
ollowing manner:

PortC bit 7   :   LCD data bit 7 (MSB)
PortC bit 6   :   LCD data bit 6
PortC bit 5   :   LCD data bit 5
PortC bit 4   :   LCD data bit 4 (LSB)

PortC bit 3   :   (not connected)
PortC bit 2   :   LCD enable pin (clock)
PortC bit 1   :   LCD R/W (read / write) signal
PortC bit 0   :   LCD RS (register select) pin

Also remember you must connect a potentiometer (variable resistor) to the vcc, gnd, and co
ntrast pins on the LCD.
The output of the pot (middle pin) should be connected to the contrast pin.  The other two
 can be on either pin.

*/


void lcd_delay()      // delay for 10000 clock cycles
{
   long int ms_count = 0;
   while (ms_count < 10000)
   {
      ms_count = ms_count + 1;
   }
}

void lcd_cmd( unsigned int myData )
{

   /* READ THIS!!!

   The & and | functions are the BITWISE AND and BITWISE OR functions respectively.  DO NO
T
   confuse these with the && and || functions (which are the LOGICAL AND and LOGICAL OR fu
nctions).

   The logical functions will only return a single 1 or 0 value, thus they do not work in
this scenario
   since we need the 8-bit value passed to this function to be preserved as 8-bits
   */


   unsigned int temp_data = 0;

   temp_data = ( myData | 0b00000100 );   // these two lines leave the upper nibble as-is,
 and set
```

```
   temp_data = ( temp_data & 0b11110100 );   // the appropriate control bits in the lower
nibble
   PORTC = temp_data;
   lcd_delay();
   PORTC = (temp_data & 0b11110000);      // we have written upper nibble to the LCD

   temp_data = ( myData << 4 );         // here, we reload myData into our temp. variable
and shift the bits
                                 // to the left 4 times.  This puts the lower nibble into
the upper 4 bits

   temp_data = (temp_data & 0b11110100);   // temp_data now contains the original
   temp_data = (temp_data | 0b00000100);   // lower nibble plus high clock signal

   PORTC = temp_data;                  // write the data to PortC
   lcd_delay();
   PORTC = (temp_data & 0b11110000);      // re-write the data to PortC with the clock sig
nal low (thus creating the falling edge)
   lcd_delay();

}

void lcd_disp(unsigned int disp)
{

   /*

   This function is identical to the lcd_cmd function with only one exception.  This least
 significant bit of
   PortC is forced high so the LCD interprets the values written to is as data instead of
a command.

   */

   unsigned int temp_data = 0;

   temp_data = ( disp & 0b11110000 );
   temp_data = ( temp_data | 0b00000101 );
   PORTC = temp_data;
   lcd_delay();
   PORTC = (temp_data & 0b11110001);
   lcd_delay();                  // upper nibble

   temp_data = (disp << 4 );
   temp_data = ( temp_data & 0b11110000 );
   temp_data = ( temp_data | 0b00000101 );
   PORTC = temp_data;
   lcd_delay();
   PORTC = (temp_data & 0b11110001);
   lcd_delay();                  // lower nibble

}


void lcd_init()
{
   lcd_cmd(0x33);      // writing 0x33 followed by
   lcd_cmd(0x32);      // 0x32 puts the LCD in 4-bit mode

   lcd_cmd(0x28);      // writing 0x28 puts the LCD in 2-line mode

   lcd_cmd(0x0F);      // writing 0x0F turns the display on, curson on, and puts the curso
r in blink mode
```

```
   lcd_cmd(0x01);      // writing 0x01 clears the LCD and sets the cursor to the home (top left) position

   //LCD is on... ready to write

}

void lcd_string(char *a)
{

   /*

   This function writes a string to the LCD.  LCDs can only print one character at a time so we need to
   print each letter or number in the string one at a time.  This is accomplished by creating a pointer to
   the beginning of the string (which logically points to the first character).  It is important to understand
   that all strings in C end with the "null" character which is interpreted by the language as a 0.  So to print
   an entire string to the LCD we point to the beginning of the string, print the first letter, then we increment
   the pointer (thus making it point to the second letter), print that letter, and keep incrementing until we reach
   the "null" character".  This can all be easily done by using a while loop that continuously prints a letter and
   increments the pointer as long as a 0 is not what the pointer points to.

   */

   while (*a != 0)
   {
      lcd_disp((unsigned int) *a);   // display the character that our pointer (a) is pointing to
      a++;                     // increment a
   }
   return;

}


void lcd_int(int value)
{

   /*
   This routine will take an integer and display it in the proper order on
   your LCD.  Thanks to Josh Hartman (IMDL Spring 2007) for writing this in lab
   */

   int temp_val;
   int x = 10000;         // since integers only go up to 32768, we only need to worry about
                     // numbers containing at most a ten-thousands place

   while (value / x == 0)   // the purpose of this loop is to find out the largest position (in decimal)
   {                  // that our integer contains.  As soon as we get a non-zero value, we know
      x/=10;            // how many positions there are int the int and x will be properly initialized to the largest
   }                  // power of 10 that will return a non-zero value when our integer is divided by x.
   while (value > 0)            // this loop is where the printing to the LCD takes place.
  First, we divide
   {                        // our integer by x (properly initialized by the last loop) an
d store it in
      temp_val = value / x;        // a temporary variable so our original value is preserve
d.  Next we subtract the
      value -= temp_val * x;      // temp. variable times x from our original value.  This
 will "pull" off the most
      lcd_disp(temp_val+ 0x30);   // significant digit from our original integer but leave
 all the remaining digits alone.
                           // After this, we add a hex 30 to our temp. variable because AS
CII values for integers
      x /= 10;               // 0 through 9 correspond to hex numbers 30 through 39.  We t
hen send this value to the
   }                        // LCD (which understands ASCII).  Finally, we divide x by 10
and repeat the process
                           // until we get a zero value (note: since our value is an integ
er, any decimal value
   return;                     // less than 1 will be truncated to a 0)

}

void lcd_clear()      // this function clears the LCD and sets the cursor to the home (upp
er left) position
{
   lcd_cmd(0x01);

   return;
}

void lcd_row(int row)   // this function moves the cursor to the beginning of the specifie
d row without changing
{                  // any of the current text on the LCD.

   switch(row)
   {

      case 0: lcd_cmd(0x02);
      case 1:   lcd_cmd(0xC0);

   }

   return;
}
```