Mop 'n Bot

Michael Hall

April 23rd, 2003

University of Florida
Department of Electrical and Computer Engineering

EEL5666
Intelligent Machines Design Laboratory

Table of Contents

Abstract

The purpose of my robot is to clean a tile or linoleum floor while staying off any adjacent carpet as well as avoiding objects. The robot starts by finding a wall or the edge of the carpet; it then follows the wall and eventually traverses the entire surface in a spiral pattern. A cleaning solution is deposited on the surface and an absorbent cloth on the rear of the robot cleans the floor.

Executive Summary

The Mop 'n Bot went through a number of steps of development. First a platform had to be conceived that would house a mop. The actual attaching of the cleaning units would be the final step however.

Once I mounted the servos and actuation was on the horizon, IR transmitters and detectors were mounted. After extensive testing of the IR sensors, and a quick modification to the servos, the development of behaviors could begin. Obstacle avoidance was soon realized in a simplistic manner.

The next step was to add bumpers which would handle any physical encounters with the Mop 'n Bot, like walls and the like. Using bump switches, the robots avoidance of obstacles was greatly improved. Actually coding this behavior turned out to be the most challenging part of the entire project.

I next tackled the problem of detecting the carpet so as to avoid trying to clean carpet with the mop. I found vertical whisker on the front of the robot were very effective. Using this system, the robot could now avoid obstacles and stay within the confines of what would be its world.

I next had to develop a pump system for depositing cleaning solution on the surface to be cleaned. I found a small, low voltage piston pump to be the most appropriate. After integrating this into my software, the robot could deposit solution only when moving forward.

The final stage in the project was to attach the cleaning cloth, or mop. This was positioned in the rear. Thus by depositing cleaning solution and going over it with a cleaning cloth, clean floors can obtained using the Mop 'n Bot.

Introduction

Household chores have long plagued the home owner. With the growth of technology, automating these tasks has become a possibility. Cleaning floors is a bothersome chore. The Mop ‘n Bot accomplishes the job of cleaning tile or linoleum floors with no effort required of the user. This paper describes the development of the Mop ‘n Bot.

Integrated System

The Mop ‘n Bot consists of six independent systems working in conjunction; actuation, bumpers, proximity detectors, carpet detectors, cleaning solution dispenser, and cleaning cloth. They are all controlled by an Atmel AtMega323 microcontroller development board.

Actuation is the primary component in all the behaviors implemented, it is accomplished using two standard servos. The proximity of objects is determined using IR transmitter/detector pairs, GP2D120. The servos, IRs, and bumpers are used in conjunction to achieve obstacle avoidance. I used three IR sensors, aimed forward, left, and right. The bumpers are on the front with some side impact capabilities. Figure 1 shows the exact placement of the sensors.

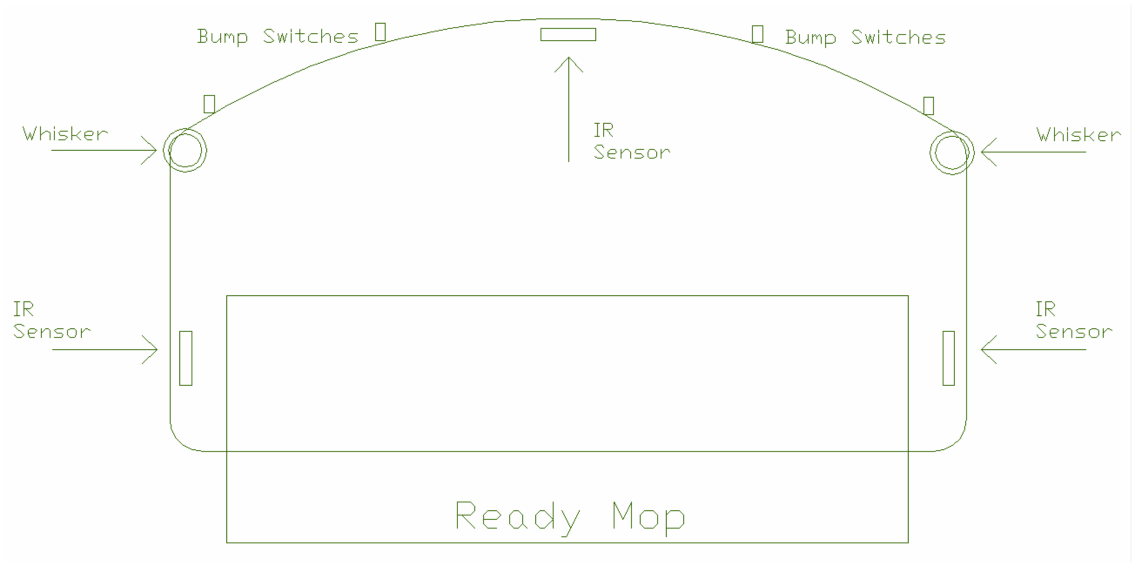


Figure 1: Layout of platform

The whiskers are forward positioned, as shown in Figure 1. Once I integrating these into the software, the robot was capable of treating the division between the tile and the carpet as another wall. This enabled Mop ‘n Bot to stay in the kitchen or bathroom, which is vital for a tile cleaning system.

The final subsystem in the project was that of the actual cleaning system. The cleaning cloth to be pulled over the surface was quite simple. The pump system also proved to be simpler than I had anticipated. The cloth was taken from a Clorox ReadyMop® and attached to the rear of the unit. I experimented with a number of pumps that required far more complex controllers than the pump I settled on. I used a small, low voltage piston pump to achieve the desired pulsating dispensing of cleaning solution.

I designed the structure of the Mop ‘n Bot to house the Clorox ReadyMop® as well as ease the cleaning of floors. Thus, I made a platform slightly wider that the mop and as short and thin as possible, this allows for less encumbered spinning and maneuvering.

Mobile Platform

The platform is roughly rectangular with two forward mounted wheels. The rear of the unit rests on the cleaning cloth. The whiskers are mounted such that the distance between them is greater than the width of the cleaning cloth, and they are positioned in front of the wheel. This will ensure that the robot will not go over any carpet. The bottle containing the solution is positioned above the cleaning cloth; this increases the overall stability of the robot and increasing pressure on the cloth increases cleaning ability. The structure of the robot can be seen in Figure 1.

In a second prototype of the Mop 'n Bot I would build an encasement to house the circuitry and sensors of the system. The current Mop 'n Bot is prone to the environment, also it is not as ascetically pleasing as it could be.

Actuation

The primary actuation of the robot is accomplished using two servos, operating the two forward mounted wheels. These are controlled using two of the pulse width modulation channels on the AtMega323 microcontroller.

A piston pump is used to control the output of the bottle containing the cleaning solution. I chose this type of pump because I wanted to pulsate the output; the piston pump automatically does this. Thus the pump needs only be turned on and off to start and stop a pulsating stream of cleaning solution. The circuit in figure 2 shows the electronic switch I used to turn the pump on and off from the AtMega323.

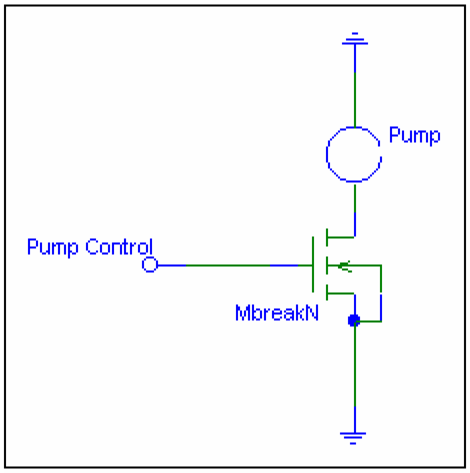


Figure 2: Pump control circuit

Sensors

Bump Sensors:

Four bump switches are positioned on the lower front of the robot. The bumper wraps around the sides slightly, to detect side impacts. These will stop the robot from trying to go through walls. One end of each switch is fed into an output port, port C. The other end is sent to an input port, port B. The internal pull-up resisters of port B are enabled and 0x00 is written to port C. Thus port B remains pulled-up, to a logic one until the switch is triggered, when the signal becomes that of port C, i.e. zero volts.

Whisker Sensors:

Two whisker sensors are placed on the front of the robot to detect carpet collisions. They are positioned in front of the wheels so as to keep the robot from going over the carpet. Each whisker was realized using a spring with a straight wire through its middle. When the spring collides with an object it makes contact with the center wire. These sensors were fed into the microcontroller in the same fashion as the bump sensors.

The center wire is puller-up using the internal pull-up resisters on port B, and the spring held at a logic one from port C. If port B goes to a logic zero, there has been a collision.

IR Sensors:

Three IR sensors alert the "Mop 'n Bot" of obstacles to its sides and front. The sensors are comprised of an emitter and a detector, powered by a 40 KHz signal generated by the built in circuitry of the GP2D120. Each sensor has three connectors, power, approximately 6.5V, ground, and an analogue output. The output is fed directly into the analogue to digital port of the microcontroller, port A. Through experimentation analogue values equating to distances from the sensors were found. Using this system, obstacles within certain proximity can be detected.

Behaviors

The robot is capable of four behaviors; random wandering, left wall-following, depositing cleaning solution, and cleaning. Avoiding obstacles is an inherent behavior in this design.

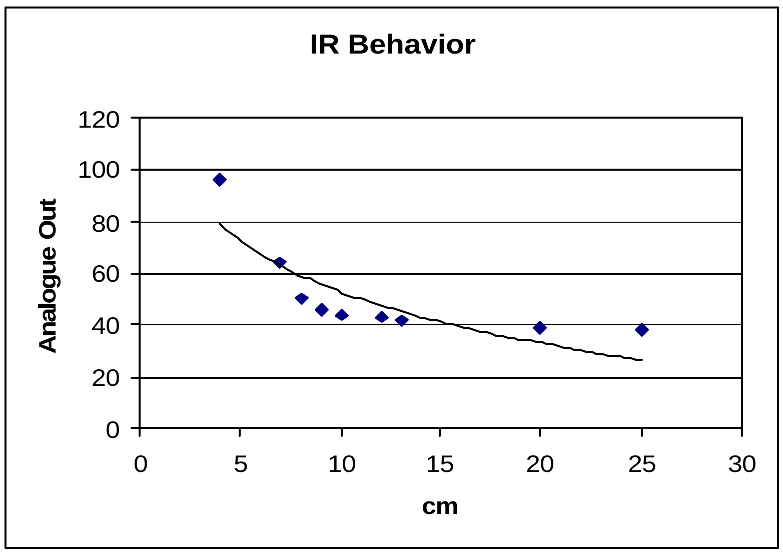
When wandering randomly, the Mop 'n Bot will move forward until it encounters an obstacle, then it will turn appropriately. Moving across the floor in lines allows the robot to cover a large percentage of the surface. Encounters with carpet are treated much the same as encounters with other objects that are not sensed by IR.

The wall-following behavior operates in conjunction with the random wandering behavior. If the robot finds that there has been a wall on it's left side for an amount of time, it will enter the appropriate wall-following behavior. When it encounters something other than that wall it will go back to random wandering. By combining these behaviors, the robot can clean a surface without missing the edges.

The cleaning behavior is a continuous process. The pump is turned on whenever the robot is progressing forward. The rear mounted cleaning cloth is always in operation, so even when it is moving backwards, it is still cleaning.

Experimental Layout and Results

The IR sensors were tested by viewing a digital representation of the analogue output at many distances. The following graph represents its functionality.



The distances of 7cm and 15cm were set as ‘near’ and ‘far’ respectively. As can be seen in the graph, these correspond to analogue values of 0x30 and 0x47.

The bump sensors and the whiskers were tested by connecting them as described above and viewing the value of port B. It could easily be determined when they were working.

Conclusion

The Mop ‘n Bot displays the simplicity of designing an automated tile cleaning agent. This implementation is an effective solution to the problem addressed, but many improvements could be made. I had to overcome many limitations during the course of the development of the Mop ‘n Bot.

Money was my largest limitation and it hindered the quality of my robot more than anything else. With more funding, better quality IR sensors would improve obstacle avoidance. The whiskers could work much better if I had not bought the cheapest springs at the hardware store. The mounting of the cleaning cloth would also be much more effective if it could swivel slightly.

These improvements required money, tools and time that I did not have at my disposal. The result was a cheaply implemented prototype that provides a viable solution for dirty tile floors. However, the Mop ‘n Bot will not clean a thousand floors. It is a prototype and it is not built for extensive use.

An encasement would greatly increase the robot’s life. In the next phase of the Mop ‘n Bot, enclosing the entire system in a streamlined case would be crucial. Adding more bumpers would also improve the obstacle avoidance behavior. By adding an encasement that had full coverage of bumper, obstacle avoidance would be flawless.

Documentation

Parts/Part Price:

GP2D120	www.arrow.com	3 x $7.92 =	$23.76
		shipping	$13.66
NiCads	www.digikey.com	17 x $1.49 =	$25.33
		shipping	$5.81
Servos	www.junun.org/MarkIII/	2 x $10.25 =	$20.50
		shipping	$2.00
Wheels	www.junun.org/MarkIII/	2 x $6.00 =	$12.00
		shipping	$1.85
AtMega232	www.prllc.com		$56.00
		Shipping	$6.00
Piston Pump	www.herbach.com		$3.75
Electrical Tape	Lowes	2 x $0.85 =	$1.70
Springs	Lowes		$3.50
Epoxy	Lowes		$3.85
		Total:	**$166.91**

Advice/Suggestions:

-Spring 2003 IMDL class
-Prof. A. Antonio Arroyo
-Prof. Eric Schwartz
-Uriel Rodriguez
-Jason Plew

Appendix

```
/****************************

Auther: Michael Hall
Date: 4-23-03
Course: EEL6841
Robot: Mop 'n Bot

   This file implements behaviors
for a stimulus-response unit
equiped with IR, bump sensors,
servos for actuation, and a pump.

****************************/


#include <io.h>

/******** CONSTANTS ********/

//#define      IR_far_F       0x23
#define IR_far_F       0x21
#define IR_far_L       0x33
#define IR_far_R       0x33

#define IR_close_L     0x49
#define IR_close_R     0x49
#define IR_close_F     0x49

#define IR_RIGHT       2
#define IR_FRONT       4
#define IR_LEFT        1

#define SURFACE_LEFT   5
#define SURFACE_RIGHT  6
#define CARPET         0xF7

#define Bumped_LL      0xFE
#define Bumped_LC      0xFB
#define Bumped_Left    0xFA

#define Bumped_RR      0xEF
#define Bumped_RC      0xFD
#define Bumped_Right   0xED

#define R_FOR          -1000
#define L_FOR          1000
#define R_BACK         1000
#define L_BACK         -1000

#define L_BACKh        -610
#define R_BACKh        610

#define L_FORh         600
#define R_FORh         -600

#define LEFT_SERVO     0
#define RIGHT_SERVO    1

#define K              0
#define WALL_COUNT     3


/******** GLOBAL VARS ********/

typedef unsigned short u16;
typedef volatile unsigned char u08;
```

```
int rv, lv, prev_right_speed, prev_left_speed, LorR, wall_follow_R, wall_follow_L;

u08 bumpers;
u08 rval;
u08 lval;
u08 fval;
int i, countR, countL, countWFL, countWFR, countP;



/******** BEGIN ROUTINES ********/

// DELAY \\
// does nothing for given number of miliseconds
void delay(u16 delay_time)
{
	do
	{
		u08 i=0;
		do
		{
			asm volatile("nop\n\t"
			"nop\n\t"
			"nop\n\t"
			"nop\n\t"
			::);
		} while(--i);
	} while(--delay_time);
}


// ADC_GETREADING \\
// retuns analogue value on A/D channel
u08 ADC_getreading(u08 channel)
{
	u08 temp_valueH;
	outp((1<<REFS0)|(1<<REFS1)|(1<<ADLAR),ADMUX); //use 2.56V as reference voltage

	if (channel == SURFACE_LEFT || channel == SURFACE_RIGHT)
		outp((1<<REFS0)|(0<<REFS1)|(1<<ADLAR),ADMUX);
							//use 5V as reference voltage

	ADMUX=ADMUX | channel;

	sbi(ADMUX,ADLAR); /* result is left adjusted */

	sbi(ADCSR, ADSC);
	loop_until_bit_is_set(ADCSR, ADIF);
		//wait till conversion is complete
	temp_valueH = inp(ADCH);
	sbi(ADCSR, ADIF);
	ADMUX=0;
	return temp_valueH;
}

// ADC_INIT \\
// initializes A/D system
void ADC_init(void)
{
	int i;
	u08 f, r, l;

	DDRA=0;
	outp((1<<ADEN) | (1<<ADPS2) | (ADPS1),ADCSR);
		//Initialize to use 8bit resolution for all channels
}

// MOTOR_INIT \\
// initializes PWM channels
void motor_init(void)
{
        prev_right_speed = R_FOR;
        prev_left_speed = L_FOR;

        outp((1<<COM1A1) | (1<<COM1B1) | (1<<PWM10) | (1<<PWM11), TCCR1A);
        outp( (1<<CS11), TCCR1B);
        sbi(DDRD, PD4); // port D pin 4 is servo1
        sbi(DDRD, PD5); // port D pin 5 is servo2
}

// SERVO \\
// new speed is averaged into the previous speed of servo s
void servo(int s, int new_speed)
{
        int speed;

        if (s == RIGHT_SERVO) {
                speed = ( (K * prev_right_speed) + new_speed ) / (K + 1);
                outw(OCR1AL, speed);
                prev_right_speed = speed;
        }
        else {
                speed = ( (K * prev_left_speed) + new_speed ) / (K + 1);
                outw(OCR1BL, speed);
                prev_left_speed = speed;
        }
}

// WALL_FOLLOW \\
void wall_follow_left(void)
{
        if (countL >= WALL_COUNT || countR >= WALL_COUNT) {

                countWFL = countWFL + 1; // how long have we been wall following?

                // back left obstacle
                if (fval < IR_far_F && rval < IR_far_R && lval > IR_far_L ) {
                        pump_on();
                        lv = 610;
                        rv = R_FOR;
                        servo(RIGHT_SERVO, rv);
                        servo(LEFT_SERVO, lv);
                        delay(50);
                }
                // forward left obstacle
                if (fval < IR_far_F && rval > IR_far_R && lval < IR_far_L ) {
                        pump_on();
                        lv = L_FOR;
                        rv = -600;
                        servo(RIGHT_SERVO, rv);
                        servo(LEFT_SERVO, lv);
                        delay(50);
                }
                // forward and back left obstacle
                if (fval < IR_far_F && rval > IR_far_R && lval > IR_far_L ) {
                        pump_on();
                        lv = L_FOR;
                        rv = -600;
                        servo(RIGHT_SERVO, rv);
                        servo(LEFT_SERVO, lv);
                        delay(50);
                }
                // nothing there
                if (fval < IR_far_F && rval < IR_far_R && lval < IR_far_L) {
                        pump_on();
                        lv = 600;
                        rv = R_FOR;
                        servo(RIGHT_SERVO, rv);
                        servo(LEFT_SERVO, lv);
                        delay(50);
                }
                // front and left obstacle
                if (fval > IR_far_F && rval > IR_far_R && lval > IR_far_L) {
                        pump_off();
                        lv = L_FOR;
                        rv = R_BACK;
                        servo(RIGHT_SERVO, rv);
                        servo(LEFT_SERVO, lv);
                        delay(150);
                        countL = 0;    // stop wall following
                        countR = 0;
                        countWFL = 0;
                }
                // front obstacle
                if (fval > IR_far_F && rval < IR_far_R && lval < IR_far_L) {
                        pump_off();
                        lv = L_FOR;
                        rv = R_BACK;
                        servo(RIGHT_SERVO, rv);
                        servo(LEFT_SERVO, lv);
                        delay(150);
                        countL = 0;    // stop wall following
                        countR = 0;
                        countWFL = 0;
                }
        }

}

// COLLISION \\
// handles bumpers and carpet collision
void collision(void)
{
        // carpet!
        if (bumpers == CARPET)
        {
                countR = 0;    // stop all wall following
                countWFR = 0;
                countL = 0;
                countWFL = 0;

                pump_off();
                lv = L_BACK;
                rv = R_BACK;
                for (i=0; i<30;       i++) {
                        servo(RIGHT_SERVO, rv);
                        servo(LEFT_SERVO, lv);
                        delay(110);
                }
                lv = L_BACK;
                rv = R_FOR;
                if (LorR == 1) {
                        lv = L_BACK;
                        rv = R_FOR;
                }
                for (i=0; i<40;       i++) {
                        servo(RIGHT_SERVO, rv);
                        servo(LEFT_SERVO, lv);
                        delay(50);
                }
        }
        // bump on the left side
        if (bumpers == Bumped_LL || bumpers == Bumped_LC) {
                countR = 0;    // stop all wall following
                countWFR = 0;
                countL = 0;
                countWFL = 0;

                pump_off();
                lv = L_BACKh;
                rv = R_BACK;
                servo(RIGHT_SERVO, rv);
                servo(LEFT_SERVO, lv);
                delay(3000);
                lv = L_FOR;
                rv = R_BACK;
                for (i=0; i<30;       i++) {
                        servo(RIGHT_SERVO, rv);
                        servo(LEFT_SERVO, lv);
                        delay(50);
                }

        }
        // bump on the right side
        if (bumpers == Bumped_RR || bumpers == Bumped_RC) {
                countR = 0;    // stop all wall following
                countWFR = 0;
                countL = 0;
                countWFL = 0;

                pump_off();
                lv = L_BACK;
                rv = R_BACKh;
                servo(RIGHT_SERVO, rv);
                servo(LEFT_SERVO, lv);
                delay(3000);
                lv = L_BACK;
                rv = R_FOR;
                for (i=0; i<40;       i++) {
                        servo(RIGHT_SERVO, rv);
                        servo(LEFT_SERVO, lv);
                        delay(50);
                }

        }
}

// PUMP_OFF \\
// turns off the pump
void pump_off(void)
{
        outp(0x00, PORTC);
}

// PUMP_ON \\
//turns on the pump
void pump_on(void)
{
        outp(0x80, PORTC);
        countP = countP + 1;
        if (countP >= 4) {
                outp(0x00, PORTC);
                if (countP == 8) countP = 0;
        }
}

// WANDER \\
// random wandering subroutine
void wander(void)
{
        // nothing there
        if (fval < IR_far_F && rval < IR_far_R && lval < IR_far_L) {
                pump_on();
                lv = L_FOR;
                rv = R_FOR;
                servo(RIGHT_SERVO, rv);
                servo(LEFT_SERVO, lv);
                delay(150);
        }
        // front obstacle
        if (fval > IR_far_F && rval < IR_close_R && lval < IR_close_L) {
                pump_off();
                lv = L_FOR;
                rv = R_BACK;
                servo(RIGHT_SERVO, rv);
                servo(LEFT_SERVO, lv);
                delay(300);
                countL = 0;    // stop all wall following
                countWFL = 0;
                countR = 0;
                countWFR = 0;
        }
        // right obstacle
        if (fval < IR_far_F && rval > IR_far_R && lval < IR_close_L) {
                pump_on();
                lv = L_FOR;
                rv = 0;
                servo(RIGHT_SERVO, rv);
                servo(LEFT_SERVO, lv);
                delay(50);
//              countL = 0;    // stop left wall following
//              countWFL = 0;
                countR = countR + 1;
        }
        // left obstacle
        if (fval < IR_far_F && rval < IR_far_R && lval > IR_far_L) {
                pump_on();
                lv = L_FOR;
                rv = -550;
                servo(RIGHT_SERVO, rv);
                servo(LEFT_SERVO, lv);
                delay(50);
//              countR = 0;    // stop right wall following
//              countWFR = 0;
                countL = countL + 1;
        }
        // right and left obstacle
        if (fval < IR_far_F && rval > IR_far_R && lval > IR_far_L ) {
                pump_on();
                lv = L_FOR;
                rv = -550;
                servo(RIGHT_SERVO, rv);
                servo(LEFT_SERVO, lv);
                delay(50);
        }
        // front and right obstacle
        if (fval > IR_far_F && rval > IR_far_R && lval < IR_far_L) {
                pump_off();
                lv = L_BACKh;
                rv = R_BACK;
                servo(RIGHT_SERVO, rv);
                servo(LEFT_SERVO, lv);
                delay(800);
        }
        // front and left obstacle
        if (fval > IR_far_F && rval < IR_far_R && lval > IR_far_L) {
                outp(0x00, PORTC);
                lv = L_BACK;
                rv = R_BACK;
                servo(RIGHT_SERVO, rv);
                servo(LEFT_SERVO, lv);
                delay(1000);
                lv = L_FOR;
                rv = R_BACK;
                servo(RIGHT_SERVO, rv);
                servo(LEFT_SERVO, lv);
                delay(500);
                countR = 0;    // stop all wall following
                countWFR = 0;
                countL = 0;
                countWFL = 0;
        }
        // i'm surrounded!
        if (fval > IR_far_F && rval > IR_far_R && lval > IR_far_L) {
                pump_off();
                lv = L_BACK;
                rv = R_BACK;
                servo(RIGHT_SERVO, rv);
                servo(LEFT_SERVO, lv);
                delay(1000);
                lv = L_FOR;
                rv = R_BACK;
                servo(RIGHT_SERVO, rv);
                servo(LEFT_SERVO, lv);
                delay(500);
                countR = 0;    // stop all wall following
                countWFR = 0;
                countL = 0;
                countWFL = 0;
        }
}

// MAIN \\
// loops forever... achieves behaviors based on environment
int main(void)
{
        countR = 0;
        countL = 0;
        countWFR = 0;
        countWFL = 0;
        countP = 0;
        wall_follow_L = 0;
        wall_follow_R = 0;

        DDRC=0xff;     // set C as output
        DDRB=0x00;     // set B as input
        DDRD=0xff;     // set D as output
        DDRA=0x00;     // set A as input
        PORTB=0xFF;    // enable portB pull-ups

        // init timer0
        TCCR0=0x05;
        TCNT0=0x00;
        OCR0=0x00;

        LorR = 0;
        rval = 0x00;
        fval = 0x00;
        lval = 0x00;

        motor_init();
        ADC_init();

        outp(0x80, PORTC);

        while(1)
        {
                fval = ADC_getreading(IR_FRONT);
                rval = ADC_getreading(IR_RIGHT);
                lval = ADC_getreading(IR_LEFT);
                bumpers = inp(PINB);

                delay(250);

                if(rand() > 0.5) //create random number to decide if left or right...
                        LorR = 1;

                collision();
                if (countR < WALL_COUNT && countL < WALL_COUNT) wander();
                wall_follow_left();
        }


}

/******** END ROUTINES ********/
```

Pictures