Andrew Kobyljanec

Intelligent Machine Design Lab EEL 5666C
April 22, 2008

GraffitiBot

Formal Report

Instructors:
Dr. Arroyo, Dr. Swartz

TAs:
Adam Barnett
Mike Pridgen
Sara Keen

Table of Contents

Opening

Abstract

GraffitiBot is a robot which roams around an environment 'tagging' its territory with its special gang's colors. If the GraffitiBot encounters colors of a rival gang member, it will paint over it. Multiple GraffitiBots could potentially roam a region, competing to gain territory from each other.

GraffitiBot uses multiple IR sensors to avoid obstacles around it. It will stop in mid-paint to turn and dodge things it sees, and then resume painting when it can travel straight. It is also possible to paint pre-programmed shapes such as circles and U's. An LCD screen provides real-time data that the robot is picking up as well.

Executive Summary

GraffitiBot is an autonomous robot capable of operating a can of spray paint. In its normal mode of operation, GraffitiBot will determine the color of its environment and find its own spray paint color as well. The robot will then begin to roam the environment, painting areas as it goes. When it encounters its own color, it will not re-paint over it. GraffitiBot will, though, paint over other colors it comes across that vary from the environment. The robot will also avoid obstacles during this process.

GraffitiBot will al so be capable of running pre-programmed routines to operate like a 'turtle graphics' turtle, drawing shapes that a program tells it to. In this mode GraffitiBot will only stop if it encounters an obstacle, and then resume when it is removed.

GraffitiBot collects information from the environment with CdS cells and IR sensors, and uses servo motors to move around as well as actuate the spray paint can.

Introduction

Graffiti has a long and illustrious history in Gainesville. Spray painting over large public objects is rarely seen as a societal nuisance, but rather a distinguished form of artistic expression. However, it has long been understood that only humans were welcome in the realm of Graffiti, and that spray-paint-can wielding robots were strictly forbidden. GraffitiBot is a robot made to break down those walls, and allow robots everywhere to finally express whatever random electrons moving around their microchips have been dreaming up in silence.

GraffitiBot is an experiment in art as well as robotics. One goal of the project was to find what kind of result would come out given some basic conditions and commands? Basically, if we set GraffitiBot loose on some prime sidewalk, what kind of art will we end up with?

Another goal was to create a robot that can spray patterns and avoid obstacles simultaneously. 'Graphitizing' is dangerous business, and one cannot assume that no obstacles will be around when you are busy creating art.

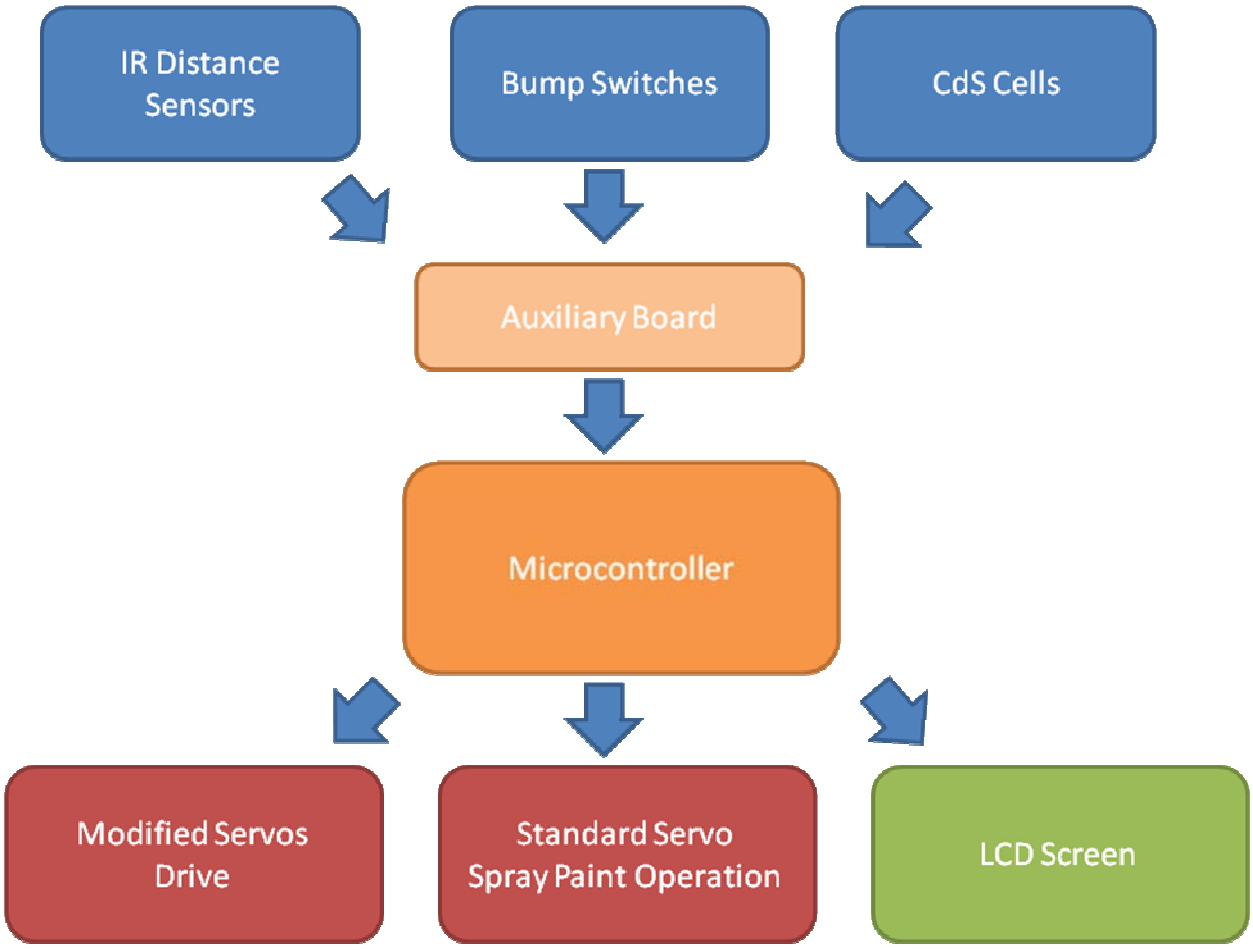
This paper will outline all of the components that went into GraffitiBot, as well as some explanations on how it all comes together as a system.

Main Body

Integrated System

Many components need to come together for GraffitiBot to function well. The robot moves around with servo motors, and controls the can of spray paint with another servo mounted on a lever. The robot also receives information from the sensors, which are connected via an auxiliary board designed specifically for GraffitiBot.

In the center of the diagram is the microcontroller, an ATMega 128 contained in the popular Mavric II-B board.



1: Flowchart of physical components

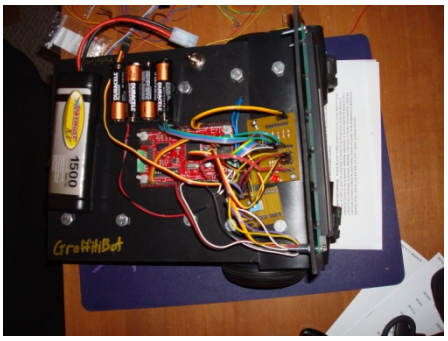
Functionally, GraffitiBot follows a similar routine every time it starts up. The microcontroller will read a value from the CdS cells, which it stores as the environment color. It will also spray a bit, and find out what color that is. It then uses its servos to move around and spray paint.

Mobile Platform

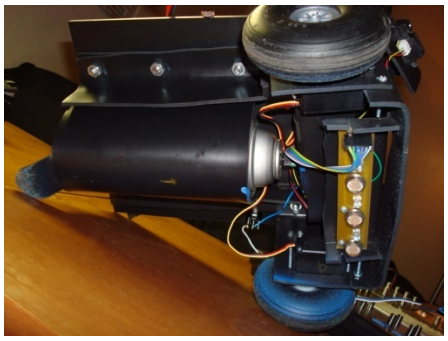
The main design consideration for GraffitiBot was the ability to carry and activate a single can of spray paint. The following list contains the other design specifications.

- Mounting for:
  - Mavric Board, with easy access for programming
  - Auxiliary Board (including LCD interfacing board)
  - 2 IR sensors on the front, with variable angle (slightly out works best)
  - LCD Screen
  - 2 Servos with wheels
  - 1 servo with lever to spray paint
  - CdS cell board, isolated from ambient light, mounted on front / bottom
  - Spray paint can, with clear path to spray
  - Large servo battery (Velcro)
  - Electronics battery (Velcro)
- Strong enough to depress spray paint can button
- Replace spray paint without complete disassembly

2: Front View



3: Top View



4: Bottom View

The lever mechanism to activate the spray paint must also be reliable and consistent. Sintra, an expanded PVC plastic that can be easily formed and cut by heating, was used to create the assembly. The design features over a dozen pieces of Sintra bent in order to mount the various components and meet the specifications. In the front view (left), the IR sensors, LCD screen, wheels with servos, and the cover for the CdS cells can be seen. Most of the electronics are mounted on the top (center). The batteries, Mavric board, and auxiliary board can be seen. The bottom view (right) shows the spray paint can be held by a bent piece of Sintra. Support brackets spanning the length of the robot can also be seen. These were found to be necessary after the robot was bending too much during the spray paint can operation. The CdS cell array is on the right, which is the front of the robot.

Many lessons were learned from creating this robot. The best lesson was discovering Sintra later in the semester. I wish I knew of this before, it made it very easy to modify my designs since Sintra can be bent, flattened, and re-bent as many times as you want. However, I should have started making the

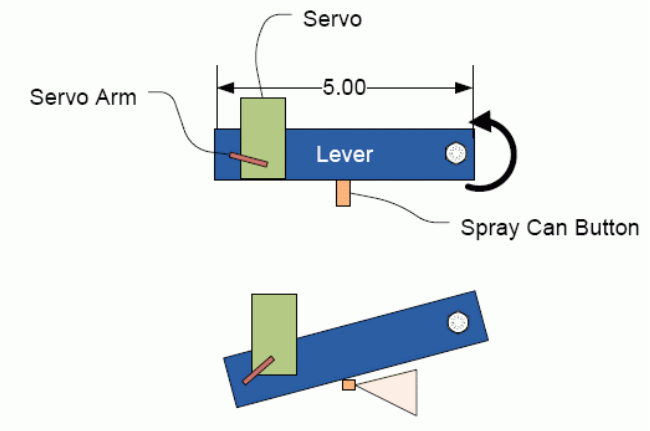
frame earlier since I ran into much unexpected trouble getting the spray paint can to spray. Unexpected trouble somewhere should always be expected. By using bolts and nuts to hold together the robot, I was able to create a modular design that was easily modified to match my changing ideas.

Actuation

Servos are used as the main actuation devices on GraffitiBot. Servos are ideal since they require no additional circuitry to function with the ATMega 128 chip. The Mavric board includes headers which servos can plug directly into, which makes interfacing with servos even easier. Servos interface with three wires: power (5-6 V), ground, and signal. GraffitiBot has a dedicated servo power supply of 4 AA batteries.

Two of the servos have been modified for continuous rotation. These servos are directly connected to the wheels and control the speed and direction the robot goes. The third servo was not modified, and operates the spray paint can mechanism. The speeds of the servos were never measured, because precise control is not necessary. The servos did run noticeably faster when I connected them to a 7.2V source, although this is probably not a great long-term solution. The listed speed and torque at 6V is .15s / 60° and 3.7 kg/cm.

The spray paint can mechanism is a simple lever which is actuated by the servo. When the servo was directly pushing against the spray paint can button, there was simply not enough torque to push it down.



5: The Lever Mechanism for Spraying Paint

The servos are all controlled by PWM signals generated from a 16bit timer on the ATMega 128. The timer generates a pulse between 1ms and 2ms long, and the period of the entire wave is 20ms. By changing the pulse length, the servo either moves full forward or reverse (for modified servos) or -90° to 90° (non-modified servos). Servo code for GraffitiBot is included in the Appendix.

The spray paint servo is also controlled by another much slower timer, which tells it whether to turn on or off the spray paint. This timer runs independent of the obstacle avoidance code, so both can be run simultaneously.

The best lesson I learned was that servos are readily available in Gainesville at the hobby shop. This saved me quite a bit of time and money, since I did not have to pay for shipping. I did have quite a bit of trouble learning how to program the timers on my own, however.

Sensors

Sensors a critical component of the robot and care needs to be taken to make sure that they are properly assembled and programmed. The sensors used fall into two main categories: navigation and task oriented.

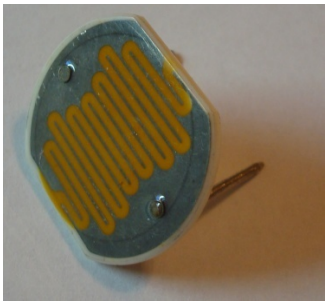
The navigation sensors will tell the robot its relative location to obstacles in the environment so it can avoid them. The navigation sensors will also allow the robot know if it has physically bumped into something. The navigation sensors are critical to the operation of the GraffitiBot, since it needs to roam around the environment to complete its task.

Task sensors will help the robot complete its designated tasks. A CdS cell will help the robots determine colors on the ground. The CdS cell will be contained in a black box with its own white LED's to be consistent under any ambient lighting conditions. Multiple CdS cells will be used to expand the robot's ability to find colors. The CdS cells will be the special sensor.

CdS Cells

Overview

CdS cells are commonly available light sensors, which allows a robot to determine how light or dark its environment is. The CdS cell is essentially a resistor that changes its resistance based on the amount of light. When there is no light, the resistance is very high, and conversely when there is light the resistance lowers.



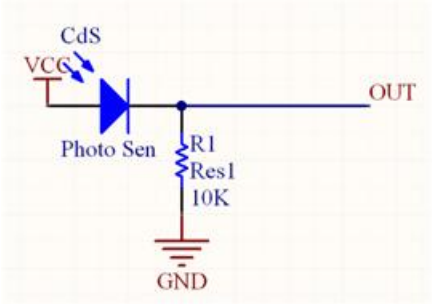
6: A large CdS Cell

Usage

CdS cells can be used in a voltage divider circuit. This allows for simple interfacing with the microcontroller. The electrical orientation of the CdS cell is irrelevant.

On the GraffitiBot, the CdS cells are isolated in a dark box with their own lighting, which will reduce the effects of ambient light. The CdS cells will be pointed towards the ground, and will look for paint colors. Since different colors of paint would reflect a different amount of light (if they are at different brightness

levels) the robot will be able to tell the difference between paint colors. In the code, the value from 4 CdS cells will be taken and averaged, with the top and bottom value thrown out. This prevents one bad measurement from throwing off the average, and works pretty well at smoothing the data.



7: CdS circuit

Data

To test the CdS cells, a light source and a colored surface were held near the cell while blocking ambient light. Data is tabulated below for various colors:

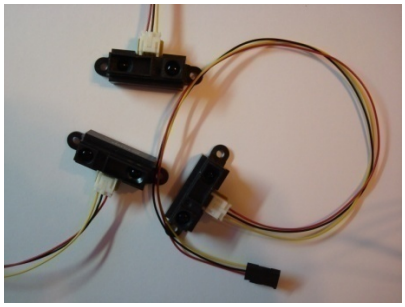
Table 1: CdS Cell Test Results

Black	0-150
Yellow	660
Red	450
Blue	185
White	685

Sharp IR Distance Sensor

Overview

The Sharp IR sensor is a popular distance sensor which comes in a variety of permutations. It uses a unique way of finding ranges with IR light which is less susceptible to ambient light and the reflectivity of objects it detects. The sensor sends out a pulse of IR light into the environment. If it hits an object, the light reflects back to the sensor. When the light returns to the sensor, it arrives at an angle dependant of the distance it reflected back from. The sensor has an included integrated circuit which provides an analog value corresponding to the range it finds. I chose the GP2D120 model, which has a range of 1.5” to 12”, although objects almost 20” away were detected in testing.



8: Three Sharp IR Distance Sensors

Usage

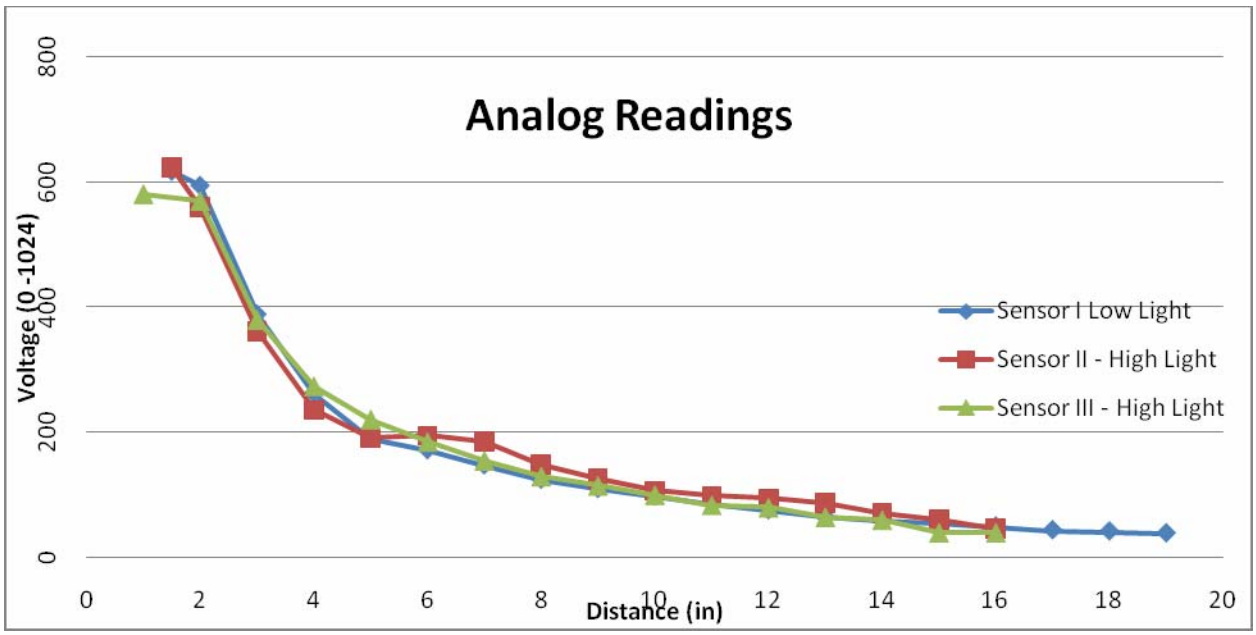
As far as the microcontroller is concerned, the Sharp IR sensor acts like any other voltage divider circuit. 5V is inputted, and an analog value corresponding to the distance of an object is returned. However, the relation of voltage vs. range is not linear and a calculation is needed to convert it to range.

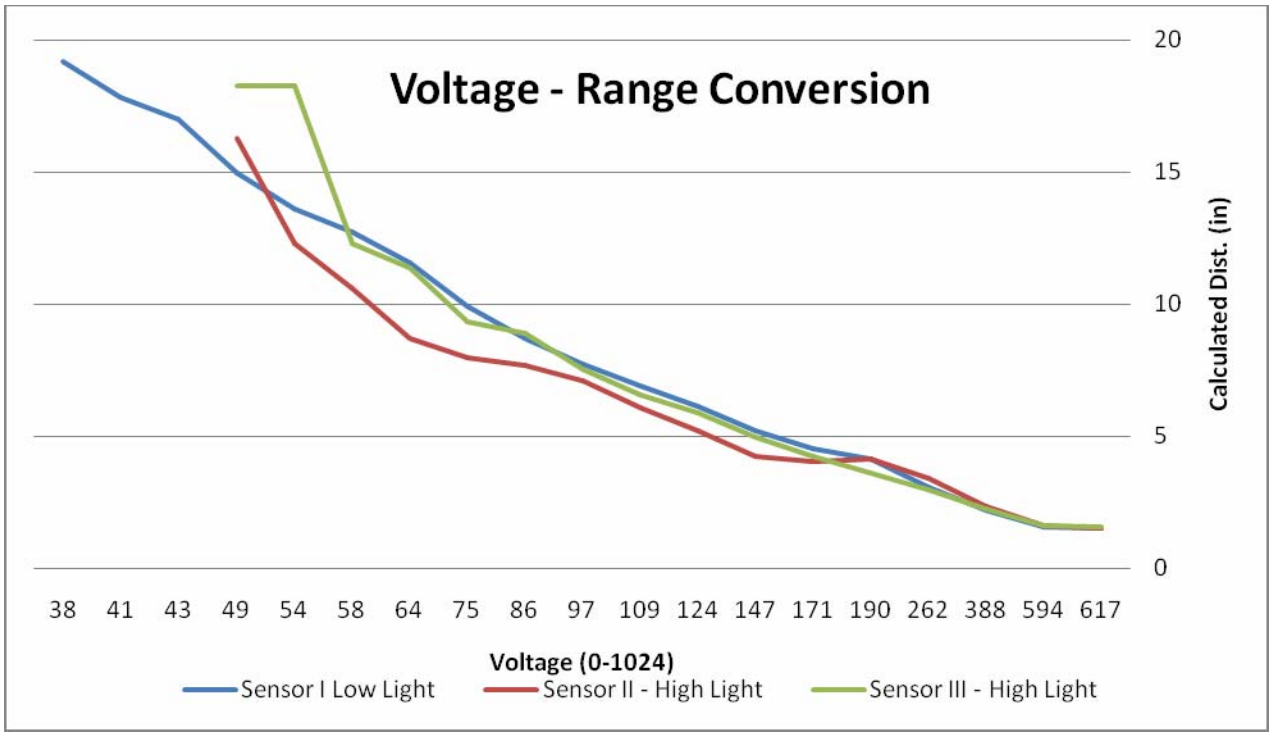
Data

Three of the Sharp IR sensors were tested by mounting them on a box and moving an object down a ruler, noting the voltage returned at every inch. This level of precision is not necessary for the robot, but this information was helpful in creating a conversion function to help convert the voltage returned into an intuitive range value. The first chart below shows the voltage returned for each sensor, based on the range the object was placed. The second chart displays what range would be calculated from the provided from the same voltage from the first chart. Although the calculated value does not perfectly match the actual object distance, it is fairly close and will work well in this application where the robot must simply avoid objects.

The function calculated for converting voltage *V* (0-1024 value) to the range *R* is:

$$R = \left(\frac{1}{V \cdot \frac{5}{1024}} - .33\right) \cdot \frac{17.5}{5} + 1.5$$

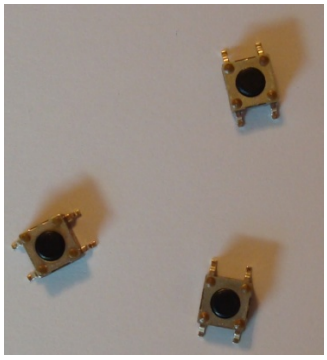




Bump Switches

Overview

Bump switches are simple momentary switches which allow the robot to detect when it hits something. When the switch is up there is an open circuit. When the switch is depressed by hitting an object, the circuit is closed and the microcontroller detects a different voltage.



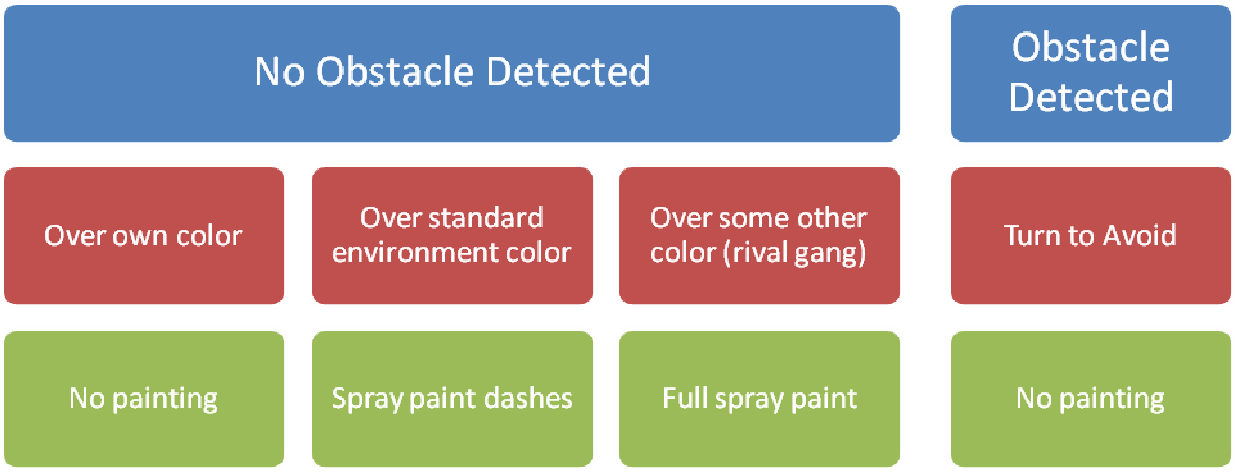
9: A collection of bump switches

Usage

The switches should be mounted in such a way that if the robot collides with an object, the switch will depress and the robot can react. Multiple bump switches are placed around the robot to keep it from colliding with objects at any angle. The switches are constructed so that they return to the up position when no longer pressed, which is ideal for the bump sensor.

Behaviors

GraffitiBot displays several behaviors while functioning. After initializing, GraffitiBot roams about its environment in one of two modes: standard and pre-programmed. In the standard mode, it roams around randomly. Generally, if there are no obstacles around it will go straight, but it may turn randomly. Obstacles will be avoided as they are sensed. In this mode, GraffitiBot will also note what its own spray paint color is as well as the general environment color. Based on this information, it will perform various actions:



In pre-programmed mode, a user can create a pattern using simple code in the AVR Studio environment, not unlike 'turtle graphics'. The programmer can tell the robot to move forward, turn a certain amount, and turn on/turn off the spray paint. GraffitiBot will just halt if it encounters an obstacle in this mode.

Closing

Conclusion

GraffitiBot was a large undertaking, and much was accomplished in one semester. Physically, the entire robot was made from scratch using nothing but Sintra, an X-acto knife, an oven, nuts and bolts. Just as difficult, though, was the software which was programmed into GraffitiBot. It took me a fair amount of thinking before I figured out a way to avoid obstacles and spray paint at the same time, since delays caused many problems in object avoidance.

There are several limitations to GraffitiBot. First of all, GraffitiBot cannot truly tell differences between colors, like a camera-equipped robot could. The CdS cells only return a value that corresponds to the average brightness. Also, GraffitiBot is not terribly accurate, and only has a few behaviors it can do. Perhaps down the road more behaviors can be created to compliment the ones already in place. Graffitibot did exceed my expectations for amusement, though. I have had a great time watching an autonomous robot spray paint everywhere.

I would make several recommendations for future students. First of all, it is imperative to begin early, and familiarizing oneself with the sometimes odd programming environment of AVR, if that is the chosen microcontroller. Even programming bits and pieces helped me down the road when I was creating the main program. Also, it is very important to start small with the overall goals, and build up from there, rather than the other way around. Getting an idea early helps you start early as well. If I did the robot again, I would try to integrate the pieces together sooner. Although I did have things working separately early, I incorrectly assumed that incorporating them together would be simple, and that caused me trouble.

If I were to make GraffitiBot again, I would try to make it a bit smaller, without the larger Mavric board, and make multiple of them so they can interact with each other. That was the original goal I had, but one GraffitiBot turned out to be too complex for me to have time to make more than one.

Appendices

Program Code

LCD Functions

```
//  LCD pins    -   ATMega Pins
//      D4                      -       0
//      D5                      -       1
//      D6                      -       2
//      D7                      -       3
//      EN                      -       6
```

```
//      RS                      -       5


// LCD Port
#define lcd_port    PORTC

//LCD Registers addresses
#define LCD_EN      0x40        // 0b0100 0000 pin 6
#define LCD_RS      0x20        // 0b0010 0000 pin 5

void lcd_reset()
{
        lcd_port = 0xFF;
        _delay_ms(20);
        lcd_port = 0x03+LCD_EN;
        lcd_port = 0x03;
        _delay_ms(10);
        lcd_port = 0x03+LCD_EN;
        lcd_port = 0x03;
        _delay_ms(1);
        lcd_port = 0x03+LCD_EN;
        lcd_port = 0x03;
        _delay_ms(1);
        lcd_port = 0x02+LCD_EN;
        lcd_port = 0x02;
}

void lcd_init ()
{
        lcd_reset();            // Call LCD reset
        lcd_cmd(0x28);          // 4-bit mode - 2 line - 5x8 font.
        //lcd_cmd(0x0E);        // Display  cursor - no blink.
                lcd_cmd(0x0C);                  //  Turn off cursor and blink
        //lcd_cmd(0x06);        // Automatic Increment - No Display shift.
        lcd_cmd(0x80);          // Address DDRAM with 0 offset 80h.
                _delay_us(1);
}

void lcd_cmd (char cmd)
{
        lcd_port = ((cmd >> 4) & 0x0F)|LCD_EN;
        lcd_port = ((cmd >> 4) & 0x0F);

                //_delay_us(20);

        lcd_port = (cmd & 0x0F)|LCD_EN;
        lcd_port = (cmd & 0x0F);

        _delay_us(400);
}

void lcd_data (unsigned char dat)
{
        lcd_port = (((dat >> 4) & 0x0F)|LCD_EN|LCD_RS);
        lcd_port = (((dat >> 4) & 0x0F)|LCD_RS);

           //_delay_us(20);

            lcd_port = ((dat & 0x0F)|LCD_EN|LCD_RS);
        lcd_port = ((dat & 0x0F)|LCD_RS);

        _delay_us(400);
}

void lcd_string(char *a)
{
   while (*a != 0)
   {
      lcd_data((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
   if (value < 0)
   {
               lcd_data('-');          // display negative sign, then mult. by -1 to return positive
integer
               value *= -1;            // to display the rest of it normally
   }

   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.

   if (value==0) lcd_data(0x30);

   else while (x >= 1)            // this loop is where the printing to the LCD takes place.  First, we
divide
   {                           // our integer by x (properly initialized by the last loop) and store it
in
      temp_val = value / x;     // a temporary variable so our original value is preserved.Next we
subtract the
      value -= temp_val * x;    // temp. variable times x from our original value.  This will "pull" off
the most
      lcd_data(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 ASCII
values for integers
      x /= 10;                  // 0 through 9 correspond to hex numbers 30 through 39.  We then 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
integer, any decimal value
   return;                     // less than 1 will be truncated to a 0)
}


void lcd_goto(int row, int col)
// Row is 1 or 2,
// Col is 0 through 39
{
        int address, command;
        if (row == 1)
        {
                address = col;          // for row 1, address is simply binary form of the col location
        }
        if (row == 2)
        {
                address = 64+col;       // the first col on row 2 is 0x40 -> 64
        }
        command = 0b10000000|address;   // add a 1 to highest bit
        lcd_cmd(command);
}
```

```
void lcd_clearLine(int row)
// row is 1 or 2
{
        lcd_goto(row,0);
        lcd_string("                                        ");  // print 40 blank chars
        lcd_cmd(0x02);                                      // return home
}
```

Servo Code

```
#define  servo_l OCR1A
#define  servo_r OCR1B
#define  servo_can OCR1C
// OCR1 pins are B5:7

// Servo info
// Servo A - Right side (modified)
// Connected to Output compare 1B
// Halt count = 1384
#define RStop 1384
// Full Forward = 2200
#define RForward 2200
// Full Reverse = 1200
#define RReverse 1200

// Servo B - Left side (modified)
// Connected to Output compare 1A
// Halt count = 1383
#define LStop 1383
// Full Forward = 1200
#define LForward 1200
// Full Reverse = 2200
#define LReverse 2200

// Servo A - Spray can (non-modified)
// Connected to Output compare 1C
// Center = 1383
// Full Open = 2200
// Full Closed (depress nozzle) = 800


void servo_init(void)
{
        TCCR1A = 0b10101000;     // Clear OCnA/OCnB/OCnC on compare match (set output to low level).
                                                        // Clear OCnA/OCnB/OCnC on compare match, set
OCnA/OCnB/OCnC at BOTTOM,(non-inverting mode)
                                                        // basically all this crap means fast pwm mode
(non-inverting, only counts up)
        TCCR1B = 0b00010010;     // wgm 13 and wgm 12 (10), combined with wgm 11 and wgm 10 (00) from
TCCR1A
                                                        // means the waveform is (1000) PWM, Phase and
Frequency Correct
                                                        // top = ICR1
                                                        // clock/8 prescaler (last 3 digits 010)

        ICR1 = 0x480B;                  // for 14.754 MHz, /8 clock divider, to get to 50Hz
                                                        // 0x480B is 18443 in decimal.  This is TOP

        DDRB = 0xFF;                    // OC1A, OC1B outputs.  Set all port b to output

        servo_l = LStop;                // Center output for A (pin B 5)  (1.5ms)
        servo_r = RStop;
        OCR1C = 1383;
}

void go_forward()
{
```

```
        servo_l = LForward;
        servo_r = RForward;
}

void go_reverse()
{
        servo_l = LReverse;
        servo_r = RReverse;
}


void go_turn(int turn)
// 0 is full left, 50 straight, 100 is full right
{
                if(turn<51)
                {
                        servo_l = LForward;
                        servo_r = RStop + (RForward - RStop)*(turn/50);
                }
                if(turn>50)
                {
                        servo_r = RForward;
                        servo_l = LStop + (LForward - LStop)*((turn-50)/50);
                }
}

void stop()
{
        servo_l = LStop;
        servo_r = RStop;
}

void turn_left()
{
        servo_l = LStop;
        servo_r = RForward;             // go with right servo only
}

void turn_right()
{
        servo_l = LForward;
        servo_r = RStop;
}

void spray_on()
{
        servo_can = 1850;               // depress lever
}

void spray_off()
{
        servo_can = 1100;               // pull back lever
}

void shake_can()
{
        int delay = 250;         // 1/2 shake period
        int i = 0;
        int shakes = 4;                 // number of shakes
        for(i=0;i<shakes;i++)
        {
                go_forward();
                _delay_ms(delay);
                go_reverse();
                _delay_ms(delay);
        }
        stop();
}
```

```
// ******************** Spray functions ******************** \\

void spray_init()
// set up spray paint clock
{
        //TCCR3B |= (1 << WGM12); // Configure timer 1 for CTC mode
    //OCR3A   = 10000; // CHANGE THIS VALUE

    //TCCR3B |= ((1 << CS10) | (1 << CS12)); // Start timer at Fcpu/64

        TCCR3A = 0b00000000;     // Clear OCnA/OCnB/OCnC on compare match (set output to low level).
                                                        // Clear OCnA/OCnB/OCnC on compare match, set
OCnA/OCnB/OCnC at BOTTOM,(non-inverting mode)
                                                        // basically all this crap means fast pwm mode
(non-inverting, only counts up)
        TCCR3B = 0b00001101;     // prescale of 1024, otherwise same as servos
        //ICR3 = 14258;                 // .5 Hz cycle time
        ICR3 = 30000;                   // ~2 Hz
        OCR3A = 15000;                  // ~2/3 duty cycle
}
```

Analog Functions

```
// Initialize A/D converter
void analog_init(void)
{
        DDRF = 0x00;                    // Port F is input, it has all the ADC pins
                                                        // JTAG uses F4:7
        PORTF = 0x00;                   // Disable pull-up resistors
        ADMUX = 0b01000000;             // Use 5v reference
        ADCSRA = 0b10000111;     // Turn on A/D converter, dont run it
                                                        // no free-running, enable complete flag
                                                        // divide clock by 128
}

// Return an analog value from a given pin
int analog(int channel)
{
        int value;
        ADMUX = 0b01000000|channel;             // channel binary for what pin in F we're looking at
        ADCSRA |= (1 << ADSC);                  // start Running the A/D converter!!

        // check this code later:
        while ( ADCSRA & (1 << ADSC) );
      value = ADCL | (ADCH << 8); //place ACD value into one variable
    return value;
        _delay_us(20);
}


// Returns an average of the last m analog values, weighted average
int analog_avg(int channel)
{
        int m = 10;                                     // average 10 analog values
        int result = 0;
        int values[m];
        int i, n;
        for(i=0 ; i<m ; i++)
        {
                values[i] = 0;                          // set array values to 0
        }
        for(i=0 ; i<m ; i++)
        {
        values[i] = analog(channel);
                _delay_us(20);                          // delay for 50 kHz reading frequency
        }
```

```
        for(i=0 ; i<m ; i++)
        {
                n = m - i;
                result += (n*values[i]) / (m*(m+1)*.5);          // running weighted average formula
        }
        return result;
}


int cds_average(int cds1, int cds2, int cds3, int cds4)
// takes all four samples from the CdS cells, throws away the highest and lowest
// and averages the other 2
{
        int cds[4];
        int i=0;                                                 // counter
        int high, low;                                           // high and low values in array
        int highAddress, lowAddress;             // locations of high and low values
        int sum = 0;
        /*for(i=0;i<4;i++)                    // load in cds values into cds array
        {
        cds[i] = analog_avg(i);
        }*/

        cds[0] = analog_avg(cds1);
        cds[1] = analog_avg(cds2);
        cds[2] = analog_avg(cds3);
        cds[3] = analog_avg(cds4);

        high = low= cds[0];         // just put one of the values in for high/low to start
    highAddress = lowAddress = 0;

        for(i=0;i<4;i++)
        {
        if(cds[i] > high)
        {
            high = cds[i];              // Find the highest value in the array
            highAddress = i;        // store address
        }
        if(cds[i] < low)
        {
            low = cds[i];              // find lowest value
            lowAddress = i;         // store address
        }
        }
        for(i=0;i<4;i++)
        {
        if(i == highAddress || i == lowAddress)
        {
            cds[i] = -1;            // plug in -1 so we can pull them out form the average
        }
    }
    for(i=0;i<4;i++)
    {
        if(cds[i]>0)
        {
            sum += cds[i];
        }
        }
        return sum / 2;
}
```

Main Loop

```
int main(void)
{

        //  ***** Initialize GraffitiBot ***** \\
        // Initialize variables
```

```
int i,j = 0;            // counters
int temp;
int IR_left, IR_right;                                          // IR variables
int cds_avg=0;          // CdS variables
int tolerance = 15;             // tolerance of cds cell values

// Stored color information
int color_env = 5;                      // environment color
int color_own = 0;                      // own paint color

DDRC = 0xFF;            // LCD is connected to C
DDRB = 0xFF;
//DDRE = 0xFF;
PORTB = 0x01;           // turn on LED

lcd_init();                     // initialize LCD, servos, analog
lcd_reset();
servo_init();
analog_init();
spray_init();
_delay_ms(1000);


// Test LCD Coding...
lcd_cmd(0x01);          // clear display
lcd_cmd(0x02);          // Return to home position
_delay_ms(10);
lcd_goto(1,10);
lcd_string("GoGoGo GraffitiBot!");
lcd_goto(2,0);
lcd_string("Connect Servos... 3 s delay ");                     // delay to plug in servos
int delay = 3000;               // delay in ms
delay /= 10;                    // divide by 10 for loop
for(i=0;i<10;i++)
{
        lcd_string(".");
        _delay_ms(delay);
}



// spray can for a bit, back up to find its color
lcd_cmd(0x01);          // clear display
lcd_cmd(0x02);          // Return to home position
_delay_ms(10);
lcd_goto(1,0);
lcd_string("Test Spray");

spray_on();
_delay_ms(300);
lcd_string("          Backing up");
go_reverse();
_delay_ms(300);
spray_off();
stop();                 // stop, take reading

color_own = cds_average(2,3,4,5);
lcd_goto(2,0);
lcd_string("Shaking the can");
shake_can();

lcd_string("     Taking measurements");
go_forward();
_delay_ms(1000);
stop();
color_env = cds_average(2,3,4,5);

_delay_ms(500);
```

```
lcd_cmd(0x01);          // clear display
_delay_ms(10);
lcd_goto(1,0);
lcd_string("IR:   Left:      Right:");
lcd_goto(2,0);
lcd_string("CdS:  ");
lcd_string("Environ:  ");
lcd_int(color_env);
lcd_string("  Own: ");
lcd_int(color_own);


// Beginning of run loop...
while(1)
{
        IR_left = analog_avg(0);                // update IR values
        IR_right = analog_avg(1);

        cds_avg = cds_average(2, 3, 4, 5);      // input channels cds cells hooked up to
        lcd_goto(1,12);                                         // print IR values
        lcd_string("   ");
        lcd_goto(1,12);
        lcd_int(IR_left);
        lcd_goto(1,24);
        lcd_string("   ");
        lcd_goto(1,24);
        lcd_int(IR_right);

        lcd_goto(1,36);
        lcd_string("    ");
        lcd_goto(1,36);
        lcd_int(cds_avg);


        // If loops to avoid crap:
        if(IR_left > 130 && IR_right > 130)             // back up
        {
                spray_off();
                PORTB = 0x00;
                lcd_goto(2,32);
                lcd_string("        ");
                lcd_goto(2,36);
                lcd_string("Back");
                go_reverse();
                _delay_ms(750);
                turn_left();
                _delay_ms(1000);        // turn around 1000 = 90 degrees

        }
        else if(IR_left > 130)          // turn right
        {
                spray_off();
                PORTB = 0x00;
                turn_right();
                lcd_goto(2,32);
                lcd_string("        ");
                lcd_goto(2,35);
                lcd_string("Right");
        }
        else if(IR_right > 130)         // turn left
        {
                spray_off();
                PORTB = 0x00;
                turn_left();
                lcd_goto(2,32);
                lcd_string("        ");
                lcd_goto(2,36);
                lcd_string("Left");
        }
```

```
                else                    // Going straight... painting
                {
                        while(IR_left < 130 && IR_right < 130)          // while no obs. in the way
                        {
                                go_forward();

                                // Incorporate some random turning...
                                if ((IR_left + IR_right) % 70 == 0)
                                {
                                        turn_left();
                                        lcd_goto(2,32);
                                        lcd_string("        ");
                                        lcd_goto(2,35);
                                        lcd_string("RLeft");
                                        _delay_ms(500);
                                        j = 0;                                  // reset j
                                }
                                if ((IR_left + IR_right) % 71 == 0)
                                {
                                        turn_right();
                                        lcd_goto(2,32);
                                        lcd_string("        ");
                                        lcd_goto(2,34);
                                        lcd_string("RRight");
                                        _delay_ms(500);
                                        j = 0;                                  // reset j
                                }



                                cds_avg = cds_average(2,3,4,5);
                                if (cds_avg > color_own + tolerance || cds_avg < color_own - tolerance)
        // if not on own color...
                                {
                                        if (cds_avg > color_env - tolerance && cds_avg < color_env +
tolerance)      // if inside standard env...
                                        {
                                                if (ETIFR & (1 << OCF3A))
                                                {
                                                PORTB ^= (1 << 0); // Toggle the LED
                                            ETIFR = (1 << OCF3A); // clear the CTC flag (writing a logic
one to the set flag clears it)
                                                if(PORTB & 0x01)
                                                        {       spray_on();     }
                                                        else
                                                        {       spray_off();    }
                                                }
                                        }
                                        else    // if not in environment, than over opposing gangs
colors!  Spray on full time
                                        {       spray_on();
                                                PORTB = 0x01;           }
                                }
                                else            // if ON its own color, turn off the spraying
                                {       spray_off();
                                        PORTB = 0x00;   }

                                lcd_goto(2,32);
                                lcd_string("        ");
                                lcd_goto(2,32);
                                lcd_string("Straight");
                                IR_left = analog(0);            // update analog values, shorter
                                IR_right = analog(1);
                                lcd_goto(1,12);                                         // print IR
values
                                lcd_string("   ");
                                lcd_goto(1,12);
```

```
                                lcd_int(IR_left);
                                lcd_goto(1,24);
                                lcd_string("   ");
                                lcd_goto(1,24);
                                lcd_int(IR_right);
                                lcd_goto(1,36);
                                lcd_string("    ");
                                lcd_goto(1,36);
                                lcd_int(cds_avg);
                        }
                }
                //PORTB = 0x00;
                //spray_off();
                }
        return 0;
}
```