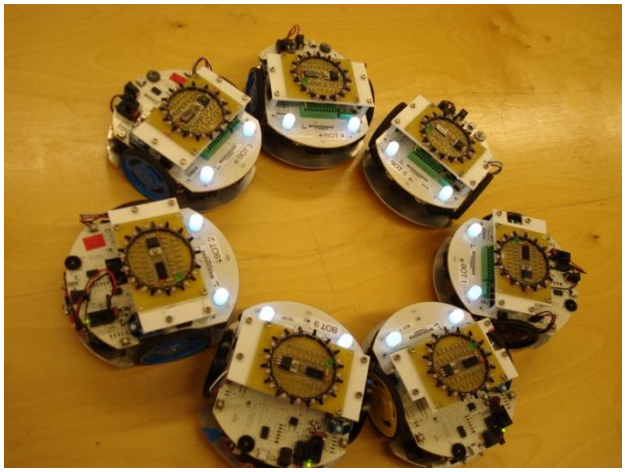
Robotics Club - Colony

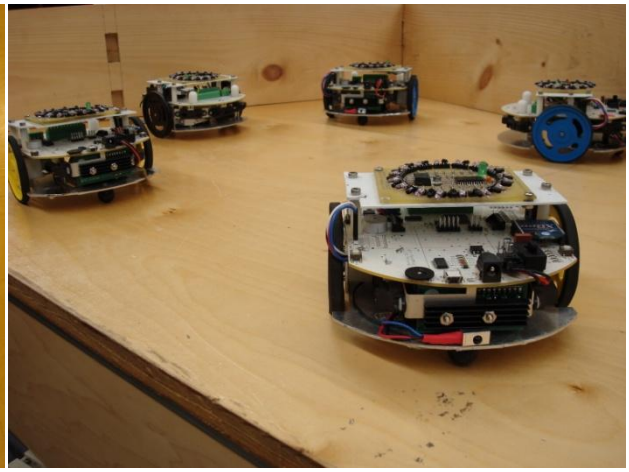
Introductory Lab #0 – Dance Competition

Release Date: 9/10/10

Demo Date: 9/24/10

Welcome to Colony!





This is a long document! Don't worry; you don't need to read it all right now, and hopefully having all the info in one place will mean you don't have to look up functions on our website. Here's a short table of contents to get you started:

1. **Colony: the Disco Years** – a short intro to the colony bots and what they can do
2. **Lab0 Objective and Documentation** – how to get started making your robot do a little dance, how to set up your computer for programming and get the programs you write onto the robots
3. **Appendix A: Colony Library Documentation** – no need to read this now! As you start working, scan it for more specific information on different things you can program
4. **Appendix B: C Documentation** – comments, logic, and stuff that goes with programming in C
5. **Appendix C: Sample Code** – want to see what simple code look like? Check here

Colony: the Disco Years

You can skip ahead to the Lab0 Objective and Documentation if you like.

1. First Steps

So the title of this section is misleading. I'm not asking you to learn to walk your robot, I'm asking you to learn to run your robot. After all, that's what we'll be doing: running through all the basic functions that your robot can do.

Now I'll give you yet another test, since I'm a jerk and you're the one who needs the knowledge that I'm to impart upon you.

Read these lines of code:

```
motor_r_set (FORWARD, 200);
motor_l_set (BACKWARD, 200);
```

While I can't say I enjoy reading C syntax (but for the record I probably do), this should still be easy to comprehend. You can basically say this out loud: "The right motor will be going forward at a speed of 200, while the left motor backward at the same speed".

This is exactly what your bot will do. It will start spinning in a circle at some arbitrary speed defined by your 200.

Read this line of code:

```
orb1_set_color(BLUE);
```

Now I don't expect you to know what an orb is. I do expect you to easily read the above as "set orb 1's color to blue". So you're setting an orb, whatever it is, to blue.

Now that you see how simple the language is, let's start moving faster!

2. Time to Learn Your Bot

Alright, I admit the above was a way to make you say things to my amusement. I did say I was a jerk. Now I'm a reformed man, so you can trust me to not make you do silly things again.

Basic Movement

If you remember, we control the bots by their individual motors. This means you get twice the control over your robot! Or does math not work like that? Nonetheless, you basically use this simple format:

```
motor_r_set (DIRECTION, SPEED);
```

`DIRECTION` is simple. Your motor can only go `FORWARD` or `BACKWARD`. `SPEED` is also straightforward. Of course, the problem with speed is that there's no actual unit of measurement that corresponds to the number you put in. Let's just say that 170 is slow and 255 is fast.

If you want to use the other motor, then you invoke `motor_l_set`. You already know that, don't you? Of course you do.

Orb Light Shows

It's time to customize your bot to your liking, which means Pretty Colors!

```
orb1_set_color(RED);
orb2_set_color(GREEN);
```

Your robot, when given this code, will have the holiday cheer. The above code is self explanatory: you are giving colors to orb1 and orb2. They will light up with your given colors and stay that way until you tell your robot otherwise.

You can also use:

```
orb1_set( R, G, B );
```

If you so happen to know your computer graphics, this ought to be simple to read. Otherwise, fiddle with the numbers. R stands for amount of red, G stands for amount of green, B stands for amount of blue, and they all range from 0 to 255. Have fun finding the perfect shade of color for your bot!

Dial One for Revolution

Let's say you want your bot to respond to your wishes, so that it could go on a rampage when you press one button but still greet you with a friendly beep when you press another. Colony robots have two buttons you can use, and you program them like this:

```
button2_click();
```

This returns "true" when button 2 is pressed while your bot is executing your program. This means you can do interesting things like:

```
if(button1_click())
{
      orb1_set_color(GREEN);
}
```

This should be easy to read as well. If button 1 is pressed, then orb 1 turns green! If you want the bot to continuously check for the button, you'll have to put it in a loop. Like so:

```
while(1)  /* the stuff in these braces runs forever! */
{
      if(button1_click())
      {
            orb1_set_color(GREEN);
            orb2_set(20,30,40);
      }
}
```

Spinning Faster and Faster

We are almost at the finish line. Hang in there!

The second to last aspect of the colony bot that we will be discussing is the potentiometer. I'm not quite sure what the word means. It probably measures the potential of something. Yeah.

Anyway, basically the potentiometer is the black circle on your bot. You can turn it to change the reading that the bot gets from it. Basically, all the way clockwise is 255 and all the way counter-clockwise is 0. We get this number using the function `wheel()`.

```
int wheel_value = wheel();
motor_r_set(BACKWARD, wheel_value);
motor_l_set(FORWARD, wheel_value);
```

This just allows you to adjust the speed of your bot's spin by twisting the potentiometer.

Wait for it...

We're at the end! However, I have one last thing for you. Perhaps it'll be the most important thing. Or not.

```
delay_ms( time );
```

`delay_ms` takes in a number. This number means that the bot will do whatever you told it to do before `delay_ms` for `time` milliseconds. This allows us to now do code like:

```
motor_r_set (FORWARD, 200);
motor_l_set (BACKWARD, 200);
delay_ms(10000);
motor_r_set (BACKWARD, 200);
motor_l_set (FORWARD, 200);
```

This has the bot spin counter-clockwise for 10 seconds (10000 milliseconds), then starts spinning clockwise. Now your bot can do more than just go forwards forever!

3. Let's Dance!

So here's the challenge: you're going to make your bot dance. I personally prefer the California Hustle, but you can do any sort of dance you want for your bot. Perhaps you would like your bot to do ballet and base your dance off the spinning code that you have been given. Whatever dance you want your bot to do, it's your choice. Just please don't let it dance off the table.

Lab 0 Objective and Documentation

Learn how to program a Colony Bot to dance like there's no tomorrow. Work with a team to outdance the competition.

Demo date: 9/24/10

1. Setting up your computer to program the robots

Linux

1. Install gcc-avr, avr-libc, and avrdude using your favorite package manager. (`synaptic` or `sudo apt-get install gcc-avr avr-lib avrdude`)
2. In the Code folder open the Makefile in a text editor, and near the top, find the section labeled `USB PORT` and comment the line about Windows, and uncomment the line about Linux. # is the comment symbol at the beginning of a line.

Mac OSX

1. Download and install the crosspack app: http://www.obdev.at/downloads/crosspack/CrossPack-AVR-20100115.dmg
2. Download and install the serial drivers: http://www.ftdichip.com/Drivers/VCP/MacOSX/FTDIUSBSerialDriver_v2_2_14.dmg
3. In the Code folder open the Makefile in a text editor, and near the top, find the section labeled `USB PORT` and comment the line about Windows, and uncomment the line about OSX. # is the comment symbol at the beginning of a line.

Microsoft Windows

1. Download: http://sourceforge.net/projects/winavr/files/WinAVR/20100110/WinAVR-20100110-install.exe/download
2. Install the above file. This gives your Programmer's Notepad, which you should use to program.
3. Note that you may have to edit your Makefile to change the COM port number.

2. Compiling Your Code and Programming your Robot

Extract `colony_lab0` from the zip archive. Copy the folder `colony_lab0` to a directory on your computer where you want it to live. Change directory into `Code`, and open `lab0.c` in your favorite text editor.

Linux/OS X

1. Write your code in your favorite text editor. Note for later: vim is the best.
2. Open a terminal, and cd into `colony_lab0/Code/`
3. Type `make` <Enter> to compile.
    a. If it complained, you need to fix your code.
4. Connect the robot to the computer
    a. Plug in the USB cable at both ends.
    b. Turn the robot off.

c. Hold down `button1` (BTN1) while turning it on to enter programming mode.
d. The Orbs should flash blue and green.

5. Type `make program` <Enter> to program the robot.
    a. If it complained, then reconnect the robot.
6. Restart the robot to run your code.

Windows

1. Write your code in Programmers Notepad.
2. In the Tools menu, select `[WinAVR] Make All`.
    a. If it complained, you need to fix your code.
3. Connect the robot to the computer
    a. Plug in the USB cable at both ends.
    b. Configure your serial ports with device manager: Start -> Right Click My Computer -> Manage -> System Tools -> Device Manager -> Ports (COM & LPT) -> USB Serial Port -> Properties -> port Settings -> Advanced -> COM Port Number -> COM4 (And/or ask a colony member for help.)
    c. Turn the robot off.
    d. Hold down `button1` (BTN1) while turning it on to enter programming mode.
    e. The Orbs should flash blue and green.
4. In the Tools menu, select `[WinAVR] Program`.
    a. If it complained, then reconnect the robot.
5. Restart the robot to run your code.

3. Writing a Colony Program

A Colony robot is only capable of running a single top level program (a "main" function) at a time. Therefore, all of your dance should be in this main function. However, because we don't have enough robots for everyone to have their own to program, you will have to team up to write your dance.

Each dance code will sit in the middle of the main function in lab0.c. Don't worry if you've never programmed C before, we've taken care of the hard stuff. There is a library of functions that takes care of making the robot work, so all you have to do is tell it what you want it to do.

In order to make your dance interesting (and to keep you from writing 5000 lines of code), you will probably want to put your dance routine in a loop. That way, it will keep dancing until the battery dies or you turn it off. Inside the main function, find the `YOUR CODE BEGINS HERE` line, and insert a few lines that look like this:

```
/* ****** YOUR CODE BEGINS HERE ******* */
while(1)
{
}
/* ****** YOUR CODE ENDS HERE ******* */
```

Done. Now whatever you put inside the curly braces { } will get repeated forever. While is a control structure that makes your code loop, and the 1 makes it go forever.

To make your robot dance, you can control two aspects of it: the motors and the orbs. The motors spin the wheels, and the orbs are the two white things on the front that light up. Start by having the robot spin by setting one motor forward and the other backward.

```
/* ****** YOUR CODE BEGINS HERE ******* */
while(1){
      motor_l_set(FORWARD,200);
      motor_r_set(BACKWARD,200);
}
/* ****** YOUR CODE ENDS HERE ******* */
```

Now your robot will spin clockwise. Forever. But that is boring. Lets mix it up. Add a delay of two seconds, and then have both motors drive forward, followed by another delay.

```
/* ****** YOUR CODE BEGINS HERE ******* */
while(1){
      motor_l_set(FORWARD,200);
      motor_r_set(BACKWARD,200);
      delay_ms(2000);
      motor_l_set(FORWARD,200);
      motor_r_set(FORWARD,200);
      delay_ms(1000);
}
/* ****** YOUR CODE ENDS HERE ******* */
```

Now your robot will spin clockwise for two seconds, and then drive forward for a second. And then repeat. Lets add color.

```
/* ****** YOUR CODE BEGINS HERE ******* */
while(1){
      motor_l_set(FORWARD,200);
      motor_r_set(BACKWARD,200);
      orb1_set_color(BLUE);
      orb2_set_color(RED);
      delay_ms(2000);
      motor_l_set(FORWARD,200);
      motor_r_set(FORWARD,200);
      orb1_set_color(LIME);
      orb2_set_color(MAGENTA);
      delay_ms(1000);
}
/* ****** YOUR CODE ENDS HERE ******* */
```

Now when your robot spins and drives, it will light up with colors. One last thing to add, control. The wheel is a sensor that reads a value that you can change while your program is running. You want to save it to a variable, and then use that variable in place of a number.

```
/* ****** YOUR CODE BEGINS HERE ******* */
int speed;
while(1)
{
      speed = wheel();
      motor_l_set(FORWARD, speed);
      motor_r_set(BACKWARD, speed);
      orb1_set_color(BLUE);
      orb2_set_color(RED);
      delay_ms(2000);
      motor_l_set(FORWARD, speed);
      motor_r_set(FORWARD, speed);
      orb1_set_color(LIME);
      orb2_set_color(MAGENTA);
      delay_ms(1000);
}
/* ****** YOUR CODE ENDS HERE ******* */
```

You just wrote a dance. Congrats. Now, be creative and make up your own dance (and add some flair). Read the next section to see all the functions that you can use to make your dance awesome.

Appendix A: Colony Library Documentation

This section lists all the functions that you can use to make your dance, as well as what they do. Try them all.

Motors
Sets the direction and speed of the motors. Motors keep their direction and speed until changed. Be aware that setting both motors at the same speed is no guarantee that your robot will drive straight.

```
motor_r_set (DIRECTION, SPEED)
motor_l_set (DIRECTION, SPEED)
```

DIRECTION: Either `FORWARD` or `BACKWARD`
SPEED: An integer value from 0-255. Motors may not work below 170.
Or one of the following values: `SLOW_SPD`, `HALF_SPD`, `NRML_SPD`, `FAST_SPD`, `FULL_SPD`.
return: nothing

Orbs
Sets the color of the Orbs. Orbs keep their color until changed. Calling these functions repeatedly without delay may cause Orbs to not light up.

```
orb1_set_color(COLOR)
orb2_set_color(COLOR)
orb1_set( R, G, B )
orb2_set( R, G, B )
```

COLOR: Any color from `RED`, `ORANGE`, `YELLOW`, `LIME`, `GREEN`, `CYAN`, `BLUE`, `PINK`, `PURPLE`, `MAGENTA`, `WHITE`, `ORB_OFF`
R,G,B: An integer value from 0-255. R = red value, G = green value, B = blue value
return: nothing

Buttons
Reads whether the buttons are being pressed or not.

```
button1_click()
button2_click()
```

return: 0 immediately if button not pressed. If button is pressed, returns 1 as soon as button is released.

Potentiometer
Reads the value of the potentiometer.

```
wheel()
```

return: the value of the potentiometer (wheel), as an integer from 0-255.

Delay
Delays code execution for the specified amount of time.

```
delay_ms(TIME)
```

TIME: Time in milliseconds (1/1000 seconds) that code execution will be delayed.
return: will return when TIME has elapsed.

Appendix B: C Documentation

Comments
Comments are text that is not executed as commands. They're just there to keep you organized.

```
//This is a legal single line comment. It is proceeded by //.
//Anything before // on this line is part of the code, anything
//after is part of the comment.
code // comment

/*This is a legal multi-line comment. Anything between the /* and
* / (without a space) is part of the comment. Anything else is
part of the code.  Notice it works over multiple lines.*/
code /* comment
comment
comment */ code
code
```

Control Structures
Control structures make decisions and make your program not progress in a linear fashion. The will cause blocks of code to be executed, skipped, or repeated based on tests. These tests are done with logic statements, that result in a 1 (true) or 0 (false) when evaluated. More detail on logic statements is below.

if (if-else)
Will run or skip a block of code depending on when test is true. Always use braces to delineate a block of code for an `if` or `if-else` statement.

```
if( test )
{
     /*code here will run if test is true*/
}
else
{
     /*code here will run if test is false*/
}
/*code here will always run*/
```

while
Loops through the code as long as test is true. Always use braces to delineate a block of code for a `while` loop.

```
while( test )
{
     /*code here will run if test is true*/
}/*when the execution reaches this line, it jumps back to the
while line */
/*code here will always run*/
```

for
A loop that iterates through a counter. Always use braces to delineate a block of code for a `for` loop.

```
int counter; /*you must declare your variable before the for loop*/
for( counter = 0; counter <= 250; counter++)
{
     /*code here will run once for each iteration of the loop*/
}/*when the execution reaches this line, it jumps back to the for line*/
/*code here will always run*/
```

True/false expressions:
Computer programs make decisions by testing whether a condition statement is true or false. In C, expressions that evaluate explicitly to false or to the number 0 are false, and all other expressions are true. As an example, 5 == 3 is an expression. The symbol == tests whether the number on the left equals the number on the right. This expression is false, because 3 does not equal 5. Other operators that may be used in the same way:

`a == b` true if a equals b
`a != b` true if a does not equal b
`a < b` true if a is less than b
`a > b` true if a is greater than b
`a <= b` true if a is less than or equal to b
`a >= b` true if a is greater than or equal to b

You may stick these expressions in the "test" area of the if, while, and for statements described below, and the expression will only execute if the expression evaluates to true.

Combining Logic Expressions
Once you've gotten that down, you may try combining expressions using the AND, OR, and NOT operators. They work like this:

`a && b` true if both expression a AND expression b are true
`a || b` true if either expression a OR expression b are true
`!a` true if expression a is false

An example: `(5 > 3) && (10 == 10)` is true, because 5 is greater than 3, and 10 is equal to 10.

A note should be made here on the appropriate use of parentheses. Operations inside parentheses will happen before operations outside. A brief example of the use of parentheses:
`(1 + 2) * 10` evaluates to 30, because the 1 + 2 happens first, then that 3 gets multiplied by 10.
`1 + 2 * 10` evaluates to 21, because the 2 * 10 happens first, then 1 is added to that 20.

If you followed all of that, it may interest you to know that large, complicated logical expressions may be created using AND, OR, NOT, and multiple layers of parentheses. It is unlikely that you will need to make heavy use of this capability to complete Lab 0.

Appendix C: Sample Code

```
/* spin.c */
while(1)
{
      if(button1_click())
      {
            orb1_set_color(GREEN);
            orb2_set_color(RED);
            delay_ms(2000);
      }
      orb1_set_color(BLUE);
      orb2_set_color(BLUE);

}
```

This has the bot display blue. If you press button one, the orbs go all Christmas on you. The `delay_ms(2000)` is time for you to see the changed colors.

```
/* controlled_spin.c */
int wheel_value;
while(1)
{
      wheel_value = wheel();
      motor_r_set(BACKWARD, wheel_value);
      motor_l_set(FORWARD, wheel_value);
      delay_ms(100);
}
```

This code will make the bot spin. You can adjust the speed with the potentiometer.

```
/* color_changer.c */
int color;
while(!button2_click())
{
      color = wheel();
      orb1_set((color + 30) % 255, (color + 78) % 255, color);
      orb2_set(color, (color + 10) % 255, (color + 50) % 255);
      delay_ms(10);
}
```

I have no clue what colors will come out. You spin the potentiometer to change the colors of the orbs.