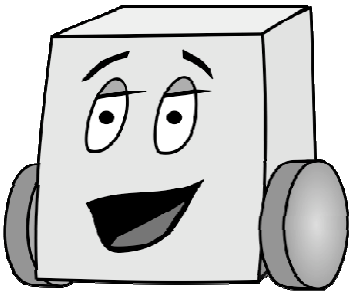


E11: Autonomous Vehicles

Fall 2010

Harris & Lape with Keeter & Ong

PS 1: Welcome to Arduino

This is the first of four programming problem sets. In this assignment you will learn to program the Arduino board that you recently built. If you need help a good reference containing all of the commands you will need to use is the Arduino website at http://arduino.cc/en/Reference/HomePage.

You are welcome to do the problem sets on your own or with a partner; working with a partner is highly encouraged and choosing a different partner each week is even better. You'll need to choose a teammate for the robot competition and it is good to learn who you work well with. If you work with a partner, turn in a single copy of your work with both of your names on it.

Part 0: Installing and Testing the Arduino Software

You may do the problem sets on your own computer or in the PC or Mac labs (e.g. in Linde or Sprague). If you have a laptop running Windows, MacOS, or Linux, you may wish to do the assignments there so that you can bring it to class for later labs.

The HMC labs are already setup with the Arduino software. If you prefer to work on your own computer, you will first need to install the Arduino software. The software is located at \\Charlie\Courses\Engineering\E11\Tools. Choose your operating system and grab the corresponding files. Uncompress them into a folder on your computer. If you have trouble, bring your laptop to tutoring hours or make friends with a computer-savvy neighbor.

Begin by opening Arduino. Files in Arduino are organized in folders called sketches. Arduino should open a new sketch immediately, with the date as the temporary title. Re-name this sketch by saving it as ps10 when prompted, where xx are your initials. This will create a `ps10` folder with `ps10.pde` as the only file inside it. Arduino will default to saving all of your sketch folders in a directory named `Arduino` in your `Documents` folder. This is fine if you are working on your own laptop. It's also convenient because Arduino will easily find your sketches when you select File -> Sketchbook. However, if you are working on a HMC lab computer, be sure to save your sketch in your personal Charlie folder instead so that you'll still be able to access it if you log into a different computer.

Type (or copy-and-paste!) the following program into your sketch, substituting your own name and date. Save it.

```
// ps10
// David_Harris@hmc.edu 5 August 2010
// Test the serial port

void setup()
{
  Serial.begin(9600);

  Serial.println("Hello world!");
}

void loop()
{
}
```

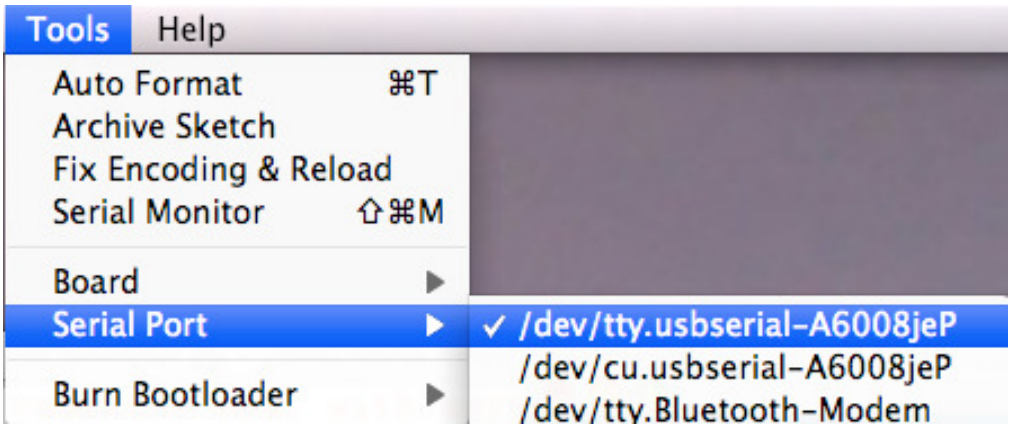
All Arduino programs must contain the two functions `setup()` and `loop()`. Remember that `setup()` will run just once at the beginning of your program. After that, as the name suggests, `loop()` will continue looping ad infinitum. **Be careful** when you put code in your `loop()` function: if you have not thought it through and make an error it will run erroneously forever – or, at least, until you unplug your Arduino board!

Be sure to put the line `Serial.begin(9600);` into your `setup()` function. This line tells the Mudduino to initiate a Serial connection at 9600 baud which will allow you to send information back to your computer over the USB cable[1]. Once you have this, you can use the function `Serial.print()` which will print whatever is passed to it or `Serial.println()` which will do the same, but with a newline at the end.

Plug your Arduino board into a USB port on the computer. If you are using your own computer, you may be prompted to install the USB drivers. Accept the default options. Be sure your Main Power jumper is set to get power from the USB port. Insert a spare LED in the LED header pins, with the long pin going in the + header. If the board is getting power, the LED should light up.

Now, make sure Arduino is using the correct port using Tools -> Serial Port. Try choosing the first or second port listed. On a Mac, it will have a funny name such as `/dev/tty.usbserial-A6008jeP` (where `A6008jeP` is a seemingly random string of numbers and letters), as shown below. On Windows, it will have a name like COM7.

[1] Arduino supports faster baud rates such as 115200. However, the serial monitor defaults to 9600 baud, so it's easiest to leave your program at 9600 rather than having to remember to change the serial monitor.



Click the Upload button in the Arduino window.

You can also do this by pressing Command+U (Mac) or Ctrl-U (PC). You should see a message reading

```
Binary sketch size: 2190 bytes (of a 30720 byte maximum)
```

Then you should see the status bar saying "Uploading to I/O Board…" and eventually "Done uploading." If you get an error, check that your board is powered up and then try a different serial port. If the only options for Serial Port that are shown have Bluetooth in the name, ask one of the lab assistants for help.

To check the results in the Serial monitor, click on its icon:

The Serial Monitor window will open up for you. You will need to re-open the Serial Monitor every time you upload.

Part 1: The Infamous Harris Numbers

In this section, you will compute and print the infamous Harris numbers. Many of you may be familiar with the Fibonocci numbers, 1, 1, 2, 3, 5, 8, 13, …, described by Leonardo of Pisa in 1202, but only a select few know about the numbers devised by his evil cousin, Harris of Parsons, in 2010. The first two Harris numbers are 1 and 3; each subsequent number is the sum of the last two.

Create a new sketch and save it as ps11. Always include your name, email address, date, and purpose of the file in comments at the beginning of a program.

We don't want the program to run forever, so only print the Harris numbers that are less than 1000. Before you begin, write out at least the first few expected numbers on paper.

Write your program and test it. All programs large enough to be interesting have bugs in them when first written; thus, please expect to have bugs! Bugs are an opportunity to get a deeper understanding – or perhaps just a reminder – of how your system works. Debug your program until it matches your expectations.[2]

Part 2: LEDs!

Your next task is to control the LEDs built into your Mudduino board using commands typed into the Serial Monitor. Specifically, your program should respond to the following six integers when sent over the serial port. For example, if you enter a 3 into the Serial Monitor, the yellow LED should turn on.

1. Turns the red LED on
2. Turns the red LED off
3. Turns the yellow LED on
4. Turns the yellow LED off
5. Turns the green LED on
6. Turns the green LED off

For this problem make a new sketch named `ps12`.

If you study the schematic of your Mudduino board at the end of Lab 1, you'll see that the board has 20 digital input/output pins, D0 – D19, for interacting with the external world. D0 – D7 are tapped out to the header pins on the left side of the board. Also, D5 – D7 are connected to the red, yellow, and green LEDs, respectively.[3]

To drive a pin from your program, you need to configure it as an output using the command `pinMode(pin, OUTPUT);` Include three lines in your setup function to configure pins 5, 6, and 7 as outputs. Don't forget to put `Serial.begin(9600);` in your `setup()` function as well.

To repeatedly check for user input, include the following `if` statement in the `loop()` function:

```
int pressed;
if (Serial.available())
{
```

[2] Note that it is wise to have written down your expectations in advance so that you don't fool yourself into believing the program is good when it is not!

[3] Note that pin numbers in the Arduino refer to the logical pin, not the physical pin. For example, pin 0 in an Arduino program refers to logical pin D0, which is actually pin number two on the 28-pin package.

```
  pressed = Serial.read() - 48;
  // now do something with it
}
```

Serial reads its input as characters, and so the integer that it will return is the ASCII value (http://en.wikipedia.org/wiki/ASCII#ASCII_printable_characters) of that character.

The ASCII value 48 corresponds to the character `0`, 49 to `1`, 50 to `2`, and so on. Thus, subtracting 48 from the ASCII value gives the number that was pressed.

You can manipulate the LEDs with `analogWrite(pin, value)`, where `value` should be an `int` from `0` - `255`. Setting `value` as `0` will turn the LED off, and setting `value` to `255` will turn the LED completely on. Intermediate values give intermediate brightnesses using a technique called ***Pulse Width Modulation*** (PWM). PWM rapidly and repeatedly turns the output on and off, such that the output is on for a specified fraction of the total time.

NOTE: Another option is to use `digitalWrite(pin, value)`, where value is `HIGH` or `LOW`. digitalWrite turns the pin completely on or off.

Write, upload, and test your program. Enter your command at the top of the Serial Monitor window and press Send.

Part 3: Randomness

Modify your previous program to make the LEDs blink on and off randomly. Save it as ps13.

Instead of taking input from the keyboard, choose a random command using the `random(min, max)` function. Note that the lower bound is inclusive but the upper bound is exclusive! Thus `random(0,5)` would randomly choose a number from 0 through *4*, not 0 through 5.

If the LEDs change too rapidly, your eye will just see a blur. Put a `delay(timeInMilliseconds)` between commands so that the you can see what is happening.

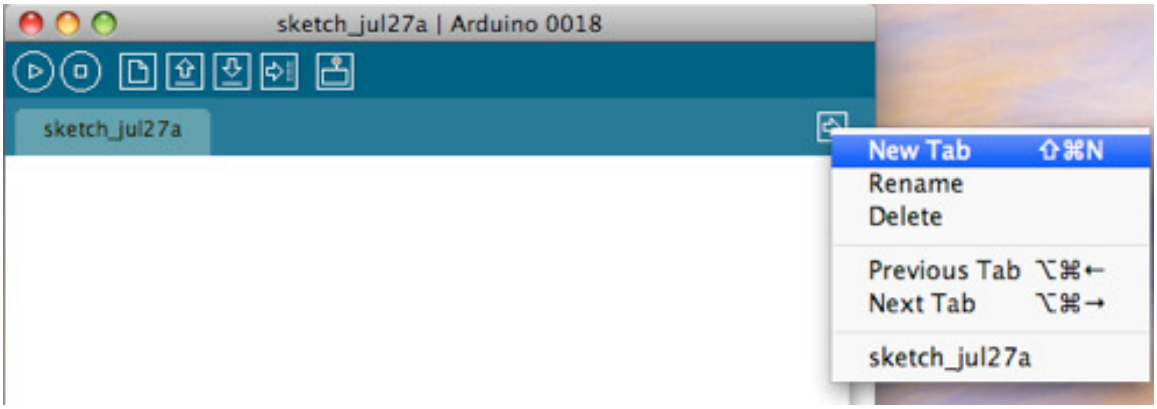
Again, test your program before moving on. Perhaps you could ask a neighbor to watch your randomly-blinking lights until they enter a hypnotic stupor?

Part 4: Reaction Times

Your final assignment is to build a reaction timer game using the Mudduino's buzzer and distance sensor. You will play a note on the buzzer for a random length of time, then switch the note, then time how long it takes to move your hand in reaction to the changed note.

The Arduino software comes with example code to play a tone. You'll copy and modify part of this code. Choose File -> Examples -> Digital -> toneMelody to open the `toneMelody` example. Click on the tab named `pitches.h`. This file defines the frequencies (in Hz) corresponding to various notes of the piano keyboard. Select All and copy the note definitions.

Open a new sketch and save it as ps14. Create a new tab by clicking no the arrow at the right side of the window:



Then paste everything into this new tab, name the tab `pitches.h` and save it. You will then need to add the line `#include "pitches.h"` near the top of your ps14 file. You may now close the `toneMelody` example, as you will no longer be using it.

The `tone(pin, pitch, noteLength)` command plays a note if the pin connected to a speaker. On your board, pin D4 is connected to the buzzer. Pitch is the pitch (in Hz) or note name that you wish to play (any of the note variables created in the `pitches.h` file may be used here), and `noteLength` is the length of time in milliseconds for which you wish the note to play. For example, `tone(4, NOTE_A4, 250)` will play a 440 Hz A note for a quarter second. The command `notone(4)` will stop playing a tone on the speaker. Also, you will want to add a delay for at least as long as the duration of the note after you set it playing: otherwise, the code will simply proceed and play the next note right away. Test this out to see how it works.

Using the distance sensor

Your distance sensor should be connected to the analog pin A0. Use this program to make sure it is working: it will simply print the raw value that is read from the distance sensor every tenth of a second:

```
int DIST_PIN = 0;      // the range sensor is on A0
int raw_distance = 0; // variable for raw range readings

void setup()
{
  Serial.begin(9600);
```

```
  Serial.println("Hola, mundo!");
}

void loop()
{
  raw_distance = analogRead(DIST_PIN);
  Serial.println(raw_distance);
  delay(100);
}
```

When you upload and run this, you should be able to see the values read from the range sensor. Set up the sensor so that it is facing upward with nothing above it (except the ceiling, perhaps!) Move your hand over the sensor to get a *sense* of how the raw values correspond to distances.

Creating a reaction-time game

You will now combine the speaker and range sensor to write a program to test your reaction time. To do this you will play a note through your Mudduino's speaker for a random length of time, then switch the note, and finally time how long it takes you to move your hand in reaction to the changed note. You can detect the movement of your hand using your distance sensor. Remember that the output of the distance sensor peaks at around 15 cm – usually corresponding to a raw sensor value of above 500 -- you will use this peak as the sensitive region in your program.

Remember to have `Serial.begin(9600);` in your `setup()` function. The rest of your code can go in `loop()`.

The operation of the program should be as follows: the user puts his or her hand into the range around 15 cm from the distance sensor where the readings are above 480, say. Once this happens your program begins to play a note for a random length of time, then it switches notes and times how long it takes for the distance reading to dip below 350. The easiest thing to do is thus to put your hand in the path of the distance sensor at the correct spot and when the note changes to simply pull it out of the path.

Since you only want to begin the reaction testing routine once your hand is in place, you will want to have an `if` statement in the `loop()` function test whether the distance reading is above 480. The rest of your code will then go inside of this `if` statement. Once the reading is above 480, you will once again need to use `random()` to determine the length of time to play the first note (make sure you make the duration of this note reasonable!).

You will then want to use the `millis()` command to grab the current time in milliseconds. When storing this value it will be necessary to use a `unsigned long` variable instead of an `int` variable, because milliseconds grow very rapidly and can exceed the storage capacity of an `int` very quickly. You can then begin playing a different note for a few seconds and taking distance readings over and over until a

reading registers as under 350. To do this you should use a `while` loop. Since you want this to continue looping until the reading is less than 350, you can employ the following loop:

```
// here is the code for playing two notes...

start_time = millis();  // start_time is declared above

while (analogRead(0) > 350)
{
   // Do nothing. This ensures that the following code
   // won't run until a reading less than 350 comes in.
}

end_time = millis();  // end_time is declared above, too

// Put code here to measure and print out total time.
```

Your code for measuring and printing the total reaction time can be fairly simple: the measurement is just a subtraction, and you have already worked with printing things to Serial. Remember that you can use `Serial.print()` and `Serial.println()` in combination to print out strings and variables on the same line. Finally, at the bottom of your `loop()` function it will be useful to put a small delay so that the game is played are taken with some time in between them, perhaps 50 milliseconds.

NOTE: When you wish to print strings to the Serial Monitor, you will need to put them in ***double quotes***, like so: `"This is my string to print"`. This will ensure that your string is printed verbatim.

Part 4 Bonus: Cheating!

The system we have described above is flawed: it is very easy to cheat and register a reaction time of 0. If the user simply places his or her hand in front of the distance sensor to start the first note and then removes it before that note is done playing, then the reaction time will be recorded as 0 milliseconds.

As an extra challenge, think of a way that you could automatically detect this kind of cheating (and print a message that they've been caught!) You can simply change your ps14 file in order to reflect this new capability, if you do try it… .

Deliverables

You are responsible for turning in 4 Arduino files:

- `ps11.pde`
- `ps12.pde`
- `ps13.pde`
- `ps14.pde`