Programming an Arduino

10/24/17

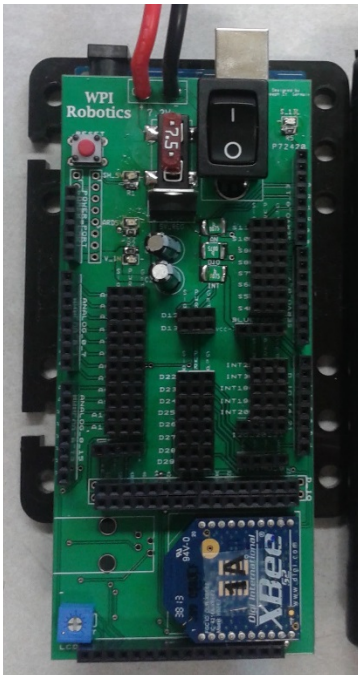


Figure 1: The Arduino Shield

Figure 2: USB Programming Cable

1. Installing the programming environment

The Arduino is capable of being programmed using Eclipse, though that will be covered in a separate document. This document will cover the installation and use of the Arduino software, which has been pre-installed on all of the robotics lab computers. The Arduino programming

environment is open source software which can be freely downloaded at http://www.arduino.cc/.

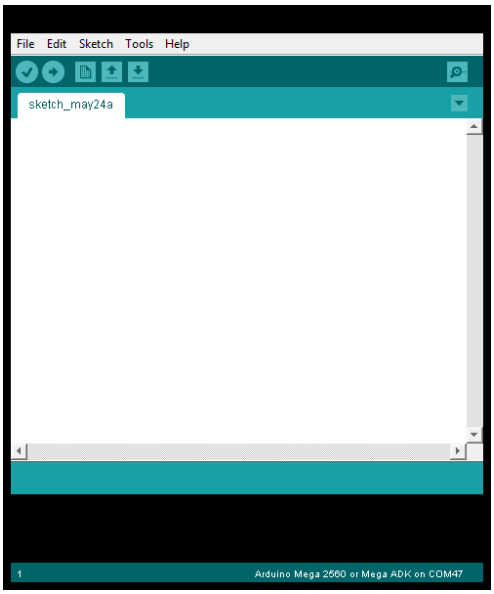
To download, visit http://arduino.cc/en/Main/Software and select the corresponding operating system. (WARNING: as of this date, many of the required Linux libraries and repositories are out of date. This may cause additional problems not covered within this document – instructions are given for a Windows operating system). This folder contains everything necessary to run the software; simply place the folder where you like on your computer. The software will be run using the "Arduino.exe" file – you may want to create a desktop shortcut for this.

If using a personal computer, it is likely that your computer will not recognize the Arduino when you first plug it in. To get your computer to interact with the Arduino, you will have to install the drivers. To do this, you will want to go into your "devices and printers" section in the windows start bar. The bottom-most section is labeled "unspecified" and should contain an unknown device (the Arduino). Right click on the device, and navigate properties -> hardware -> properties -> change settings -> driver -> update driver -> browse my computer. You should get a pop-up window; browse to your Arduino install folder and select the "drivers" folder and hit ok. This will install the drivers and allow your computer to interface with the board. The current installer adds the drivers automatically when the Arduino software is installed.

It is important to note that some of the examples in this guide were custom-built for this guide – you will not have those on your computer. If you would like to use these on your personal computer simply replace the "libraries" folder in your Arduino install folder with the one downloaded from MyWpi.

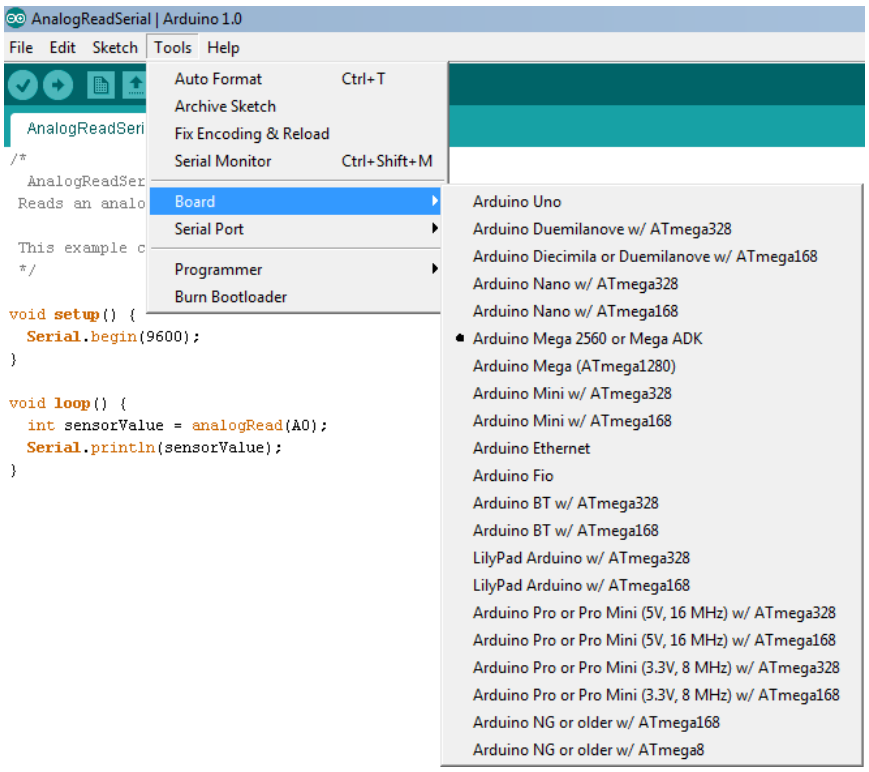
2. Using the software

When the software is run, it will open a window resembling the image below.

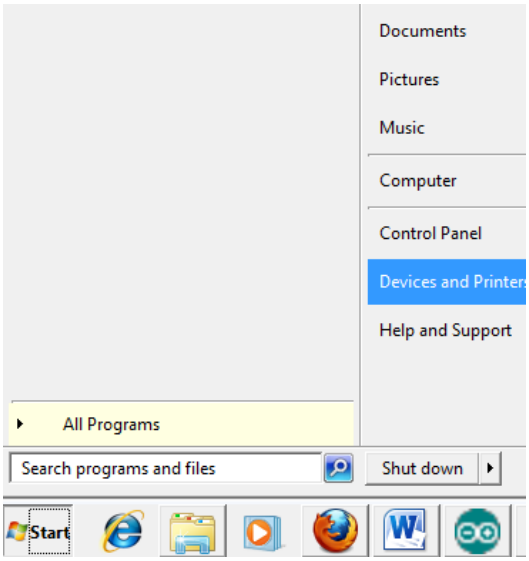


This is the graphical programming interface. The Arduino language is unique, though it is extremely similar to C++. For those with more programming experience, you may note that it is customary to use multiple files when programming. In the Arduino environment, this is difficult; for the purposes of this document everything will be contained within one file. Further "advanced" instructions are located in the "using libraries in Arduino" document on Blackboard.

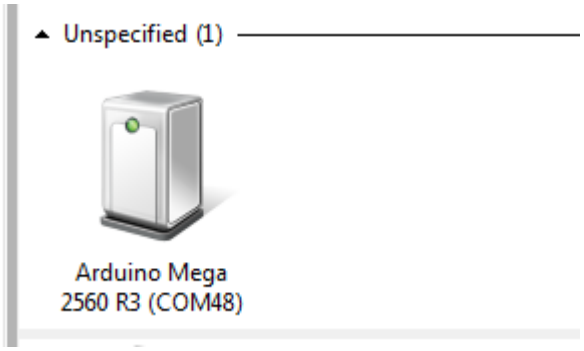
Once you have working code, you will eventually want to run it on the Arduino. The Arduino contains local memory to store the code and run it; it does not need to be connected to the computer in order to operate. To send code to the Arduino, connect the USB cable to the board and to the computer. Once this is done, the software must be aware of where to actually send the code, and to what. First, select the type of Arduino being used (for the RBE labs and final project, this is the Mega 2560; for homework assignments it will be the Uno) as shown in the image below.



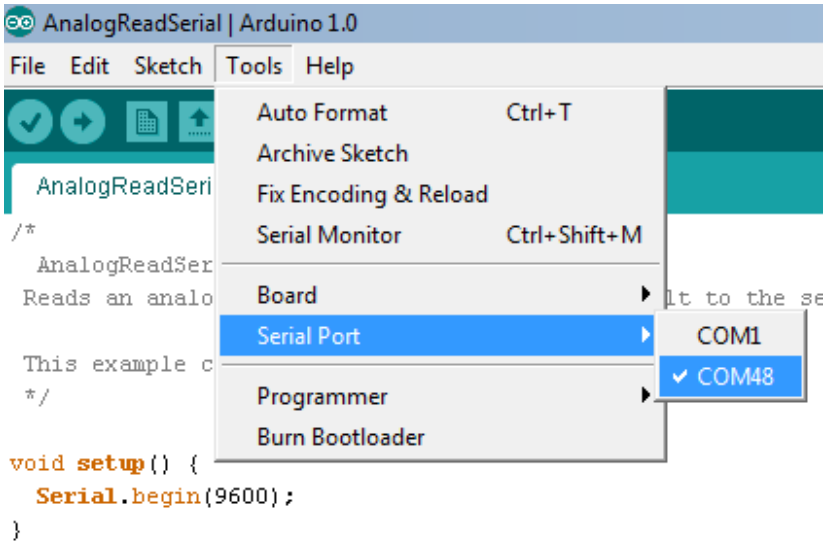
Then select the Serial port the Arduino is connected to. This is listed as a device in windows – to find it, enter the Devices and Printers menu in windows.



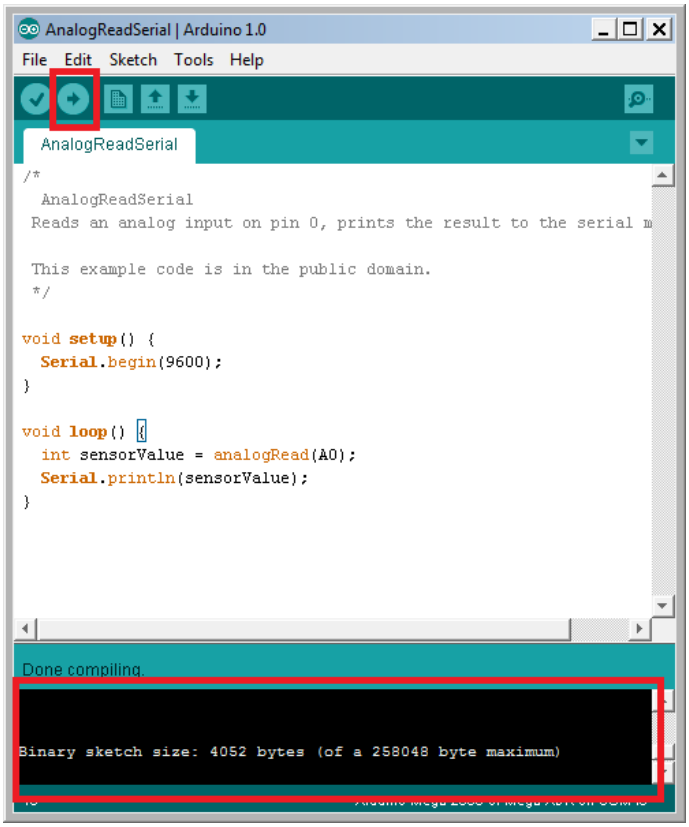
Then look for the Arduino board. The com port is listed clearly at the end of the name.



Using this com port number, select it from the Arduino menu.



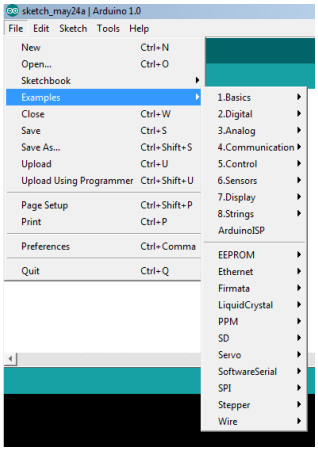
Then, the only remaining step is to actually send the code to the Arduino. You may do this by clicking the arrow button in the top left of the application.



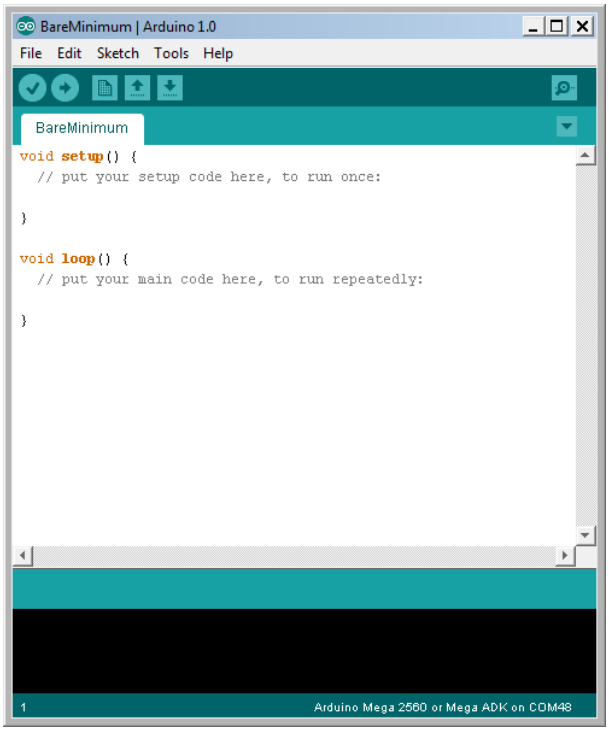
If any errors occur while compiling your code, they will appear in the box at the bottom.

3. The Code

When writing code, it is often very difficult to get started, especially for those who are not experienced programmers. Fortunately, the Arduino environment contains many pre-written and operational bits of example code. You may access these at any time by navigating to the examples option:



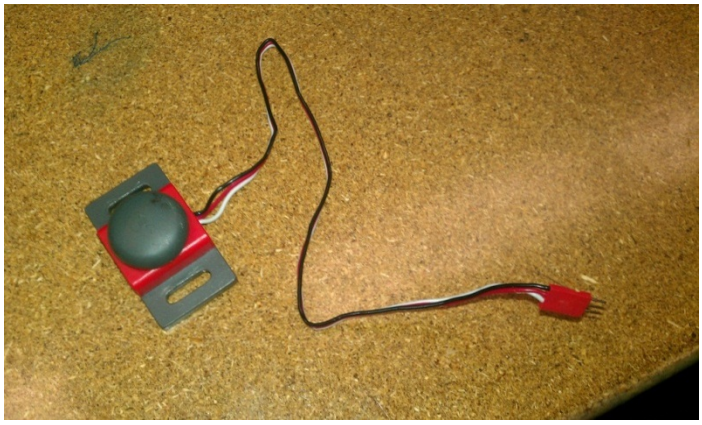
At this point, you should try loading the “BareMinimum” example within the “Basics” option. You should get a new window that looks like this:



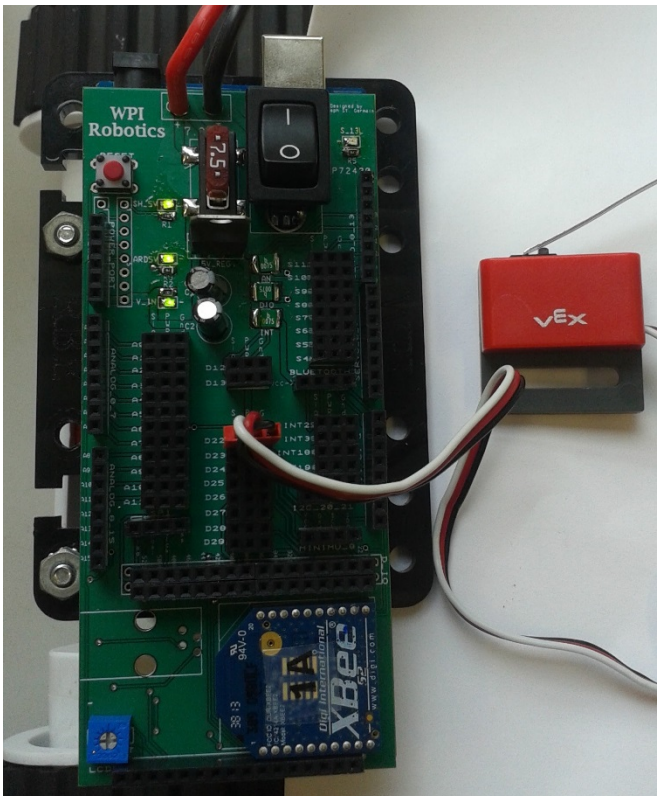
This shows the basic format of any Arduino program. Instead of a main() function like other common programming languages, the Arduino program must always contain setup() and loop() functions. Setup is a place for “one-time” code, such as declaring variables, or setting pins as input/output. The loop section is where the operational code goes: everything you want the Arduino to actually “do” will be placed in this function, and will repeat indefinitely.

a. Digital Sensors

One very important function of the Arduino is to read digital and analog sensors. For this tutorial, we will use a vex button and a bump switch.

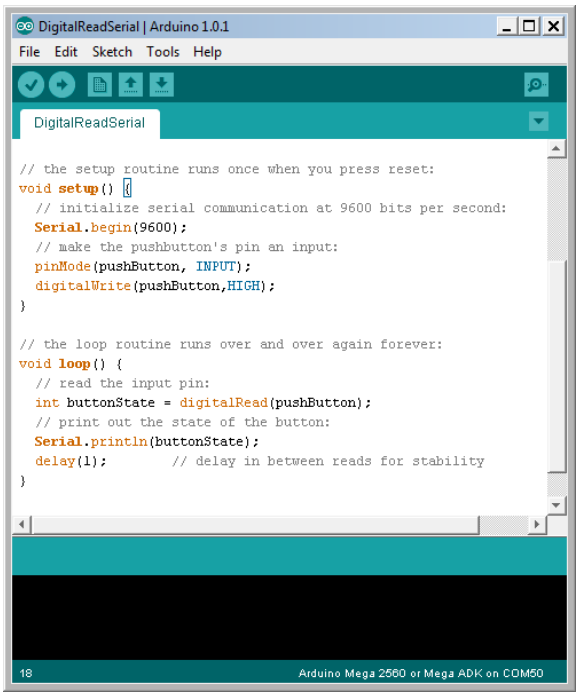


First, we must give the board power. Find your (charged) 7.2V battery, and plug it into the battery input on your board. Once you do this, you will notice a series of lights appear on the shield. If no lights appear, then try using the power switch on the board. You will also want the board connected to the computer using the USB cable. Once it is fully connected, it should look like this:

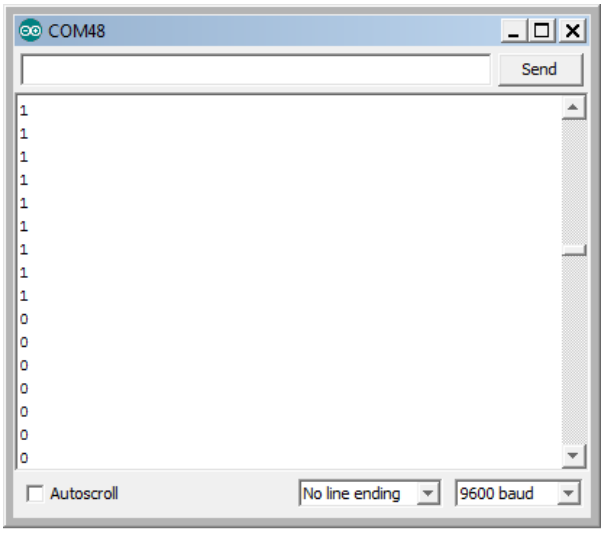


Now, you will want to plug in either the bump switch or the button into “D22”. Be very careful to insert it correctly: a backwards connection could cause serious damage to the sensor or the Arduino. The board is labeled “Sig” “PWR” and “Gnd”. These signals correspond to white, red, and black connections respectively. Open the Arduino programming environment. Using the method described earlier to load examples, find “DigitalReadSerial” under “Basics”. Then modify the pushbutton variable from 2 to 22. The vex pushbuttons require an external pullup resistor to function reliably. upload it to the Arduino.

The Arduino environment has the capability to show serial data directly to the computer to which it is connected. You may view this serial data by clicking the small magnifying glass in the top right of the software.



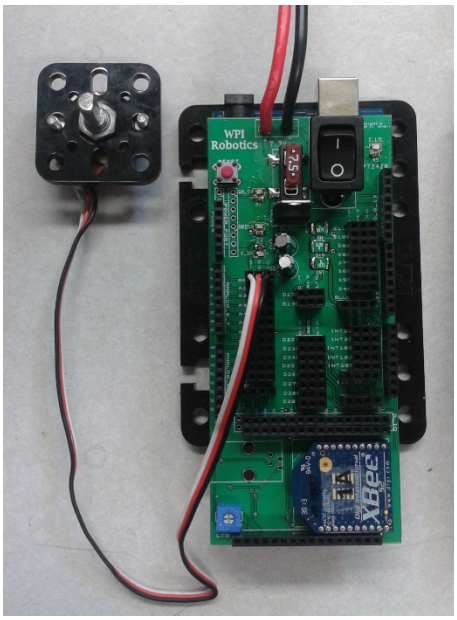
This will show the serial monitor.



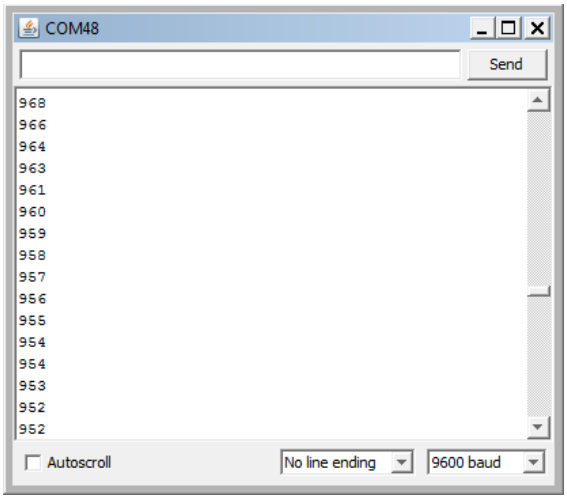
As you push the button and release it you will notice the value change from 1 to 0 in real time. This “on-off” functionality is very useful for certain robotics applications. Now try this again with the other sensor, and get a feel for the differences between the two.

b. Analog Sensors

A similar process may be used to read an analog sensor, such as a variable resistor (potentiometer). Find your potentiometer and insert it into port A0, following the same color conventions as previously described. Be sure to attach a vex axle so you can turn the potentiometer.

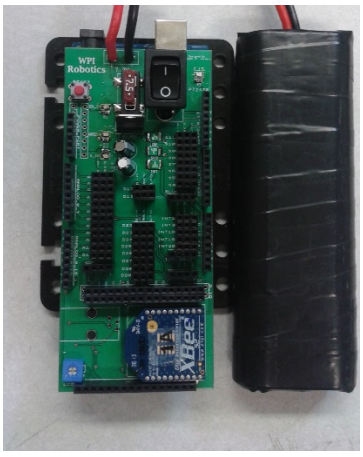


Open up the “AnalogReadSerial” example under “Basics”, load it and send it to the Arduino. This will operate similarly to the previous example, though it reads an analog input instead of digital: instead of seeing one of two binary values (0 or 1) the input will now be in the range of 0 – 1023. Open the serial monitor and rotate the potentiometer – you will notice the values change as you do so. These numbers are proportional to the voltage measured on the wiper of the potentiometer, where 0 = 0 volts, and 1023 = 5 volts (the voltage supplied to the potentiometer by the shield board).

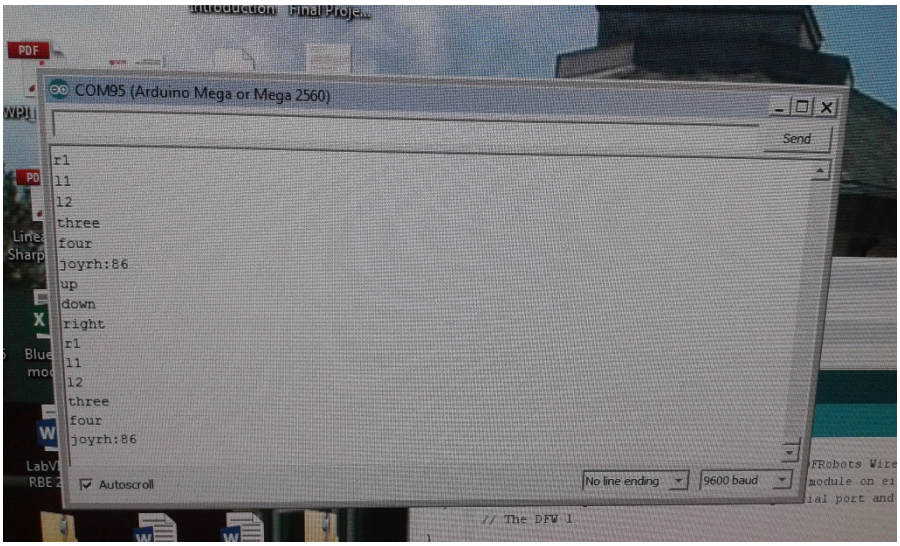


c. The Remote Control

You may wish to drive your robot via remote control. You can accomplish this using the DF robots Gamepad and Xbee modules. It is important that the Arduino receive and interpret the signals from the Xbee receiver module correctly. This code has already been written for you, all that remains is to load it and use it. The first thing is to make sure there is an Xbee module plugged into the Arduino shield. You also want to make sure the number on the xbee module matches the number on the gamepad.



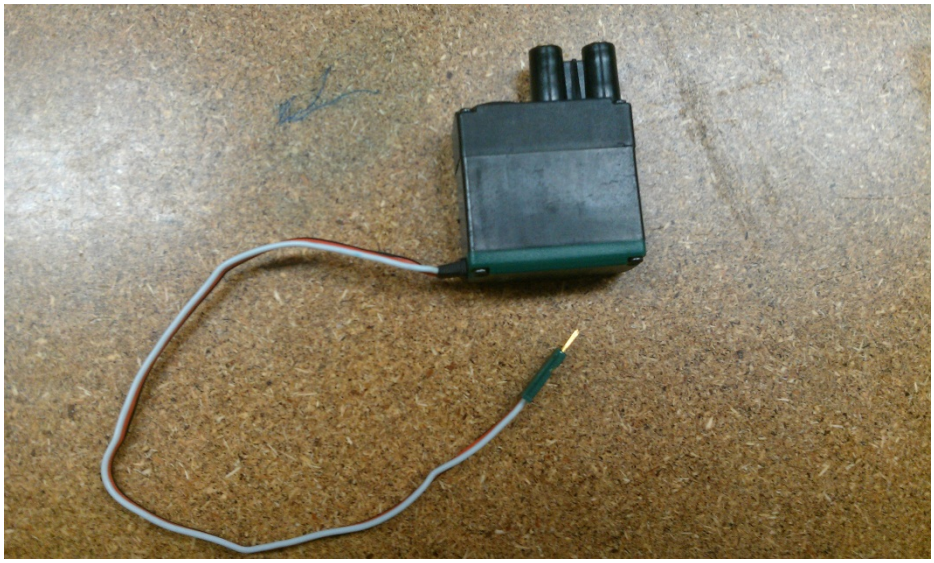
By loading the “DFWTest” example code, you may view the values from each joystick or button in real-time. Try this, and experiment with the operation. At first, you will notice that the serial monitor will show nothing until the xbee modules connect. This can take up to 12 seconds to complete. In the bottom right corner of the serial monitor you must change the baud rate to 9600.



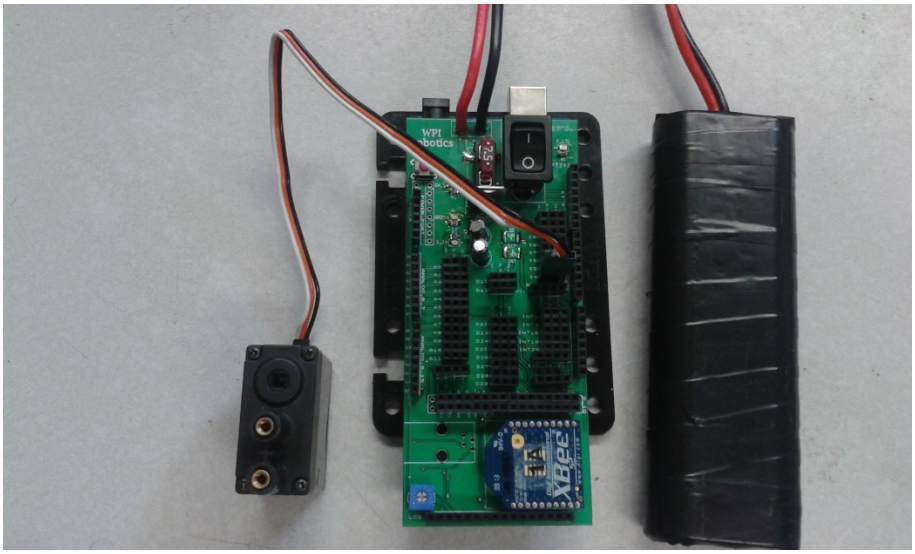
This will allow readable output from the program. When a button is pressed the button label will be printed to the screen. When a joystick is moved from the neutral position the value of the joystick will be printed. Joystick values range from 0-180 with 90 being neutral. Buttons are set as 1 by default and switch to 0 when pressed. The test program simply polls the data using the prewritten functions. It is very important to add a delay of more than 20ms in your loop for stable control. It is also advisable to not delay more than 500ms between reads. There is a 1 second timeout built into the library. If the Arduino does not receive a signal within 1 second, the control values will return to their default value until the next update is received and pin 13 will be set high.(LED 13 will turn on)

Motors and Servos

The last functionality you will need to master is the control of motors and servos. A primary difference between servos and motors is the operation commands and their function. A servo will take a value between 0-180, and will rotate to an angle corresponding to that number of degrees. The motor will take this value and do something slightly different; giving the command 90 will force the motor to stop. Commands above 90 will spin the motor in one direction, and commands below 90 will spin in the other. The farther from 90 the command is, the faster the motor will spin. In short, sending 0 will spin the motor full speed in one direction, and sending 180 will spin it full speed in the other. The motors and servos are both identical by shape and appearance, though are named on the green panel.



By loading the “sweep” example under “servos” we may examine the differences in operation between servos and motors. This example will send values ranging 0-180 in order, then repeat. Plug a motor into port S9, as shown.



Run the code example. Watch as the motor spins in one direction, slows to a stop, and then speeds up in the other direction. If the motor is not stopping and starting modify the line

myservo.attach(9); to myservo.attach(9,1000,2000);

Turn off the Arduino board using the switch, and replace the motor with a servo. Turn on the Arduino and then watch as the servo operates. Note the differences, and consider some operations that may benefit from each device.

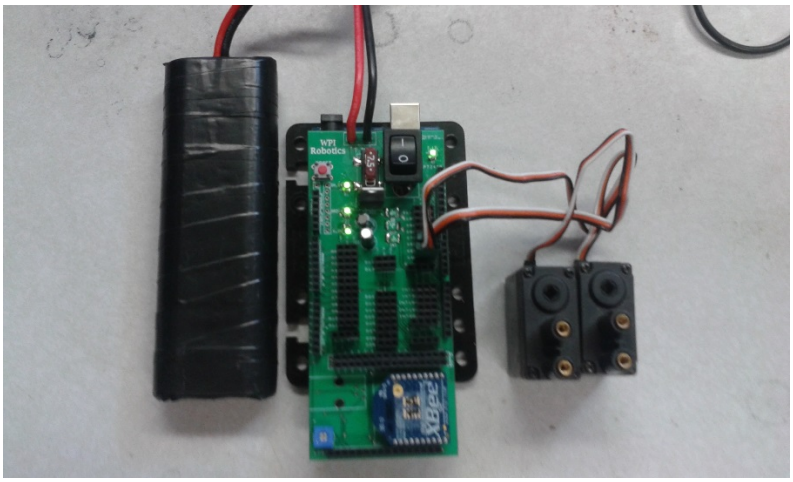
Now we will have a chance to actually change the code and watch what it does. Modify your sweep example by editing the “pos” value. The “pos” variable controls the operation of the motor, based on the 0-180 scale described earlier. Try to add a second servo object, and experiment with the motion of the basebot, re-sending the code to the Arduino with each

change. Feel free to attempt to program some simple motions, as they may be of some help later in the course.

d. Tank Drive

The operation of your robot will require you to be able to drive your basebot with a remote control. This requires integration between the motors and the remote control.

Load the “DFWTank” example under “DFW” onto your Arduino. Make sure your xbee module is plugged in and has the same number as the gamepad.[1] Attach your left motor to port S4, and your right motor to port S5. When fully connected, it should look something like this:



Once the Arduino has been turned on, you will be able to drive your basebot with your remote control. Remember the connection process can take as long as 12 seconds. The left joystick corresponds to the left wheels, and the right joystick corresponds to the right wheels. If you push both joysticks forward, the basebot should drive straight forward. Be sure to let everyone in your team experience the “tankdrive” functionality, and become accustomed to it.

Update History:

Revision	Date	By	Description
1	5/14/12	Corey Russell	Original version
2	9/6/12	Craig Putnam	Minor formatting changes; updates to reflect new shield board
3	9/10/13	Joseph St. Germain	Pictures, note that the new installer includes the drivers.
4	7/31/15	Joseph St. Germain	Update to Xbee and Gamepad controllers.