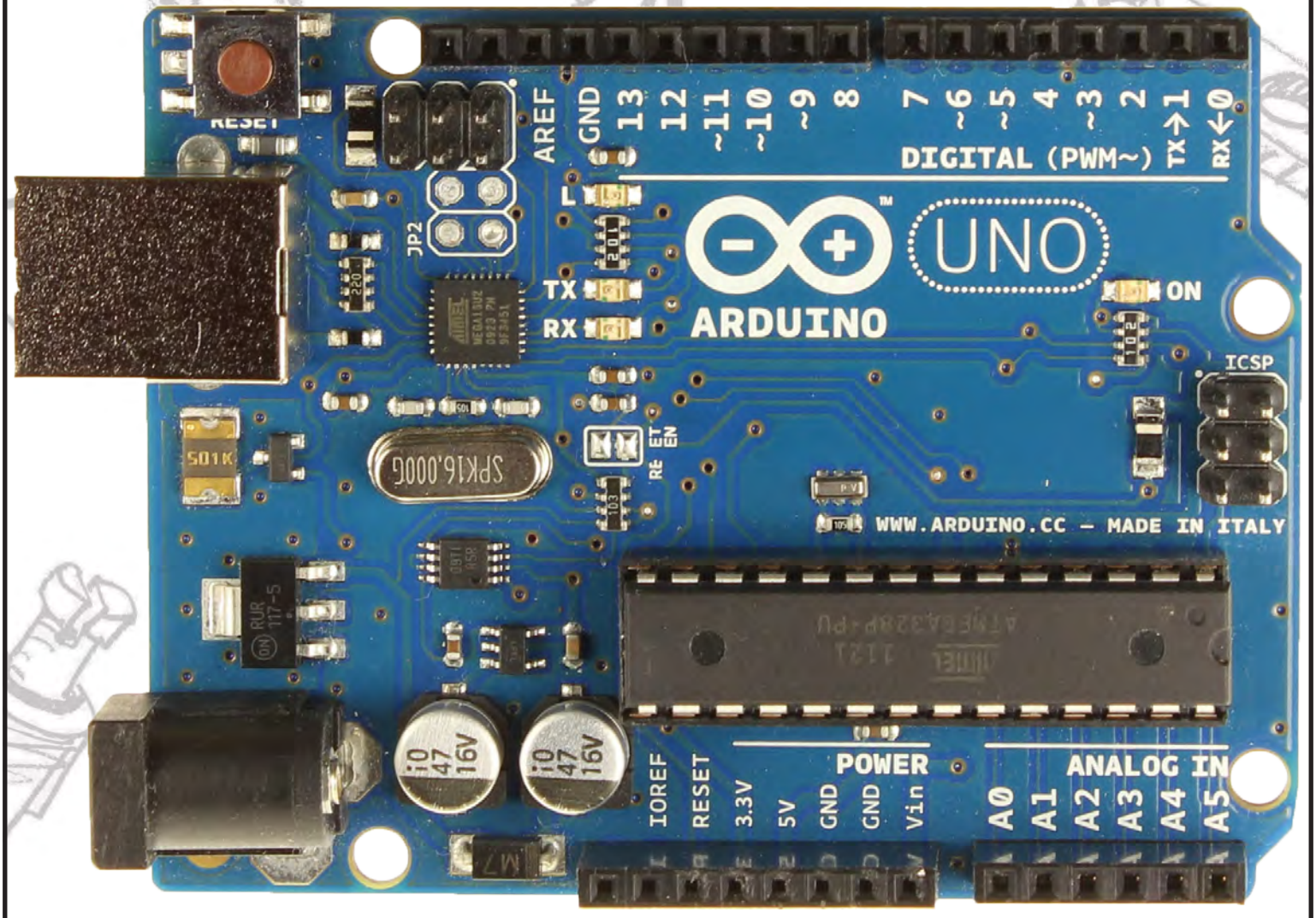
For Use with MINDS-i

Arduino

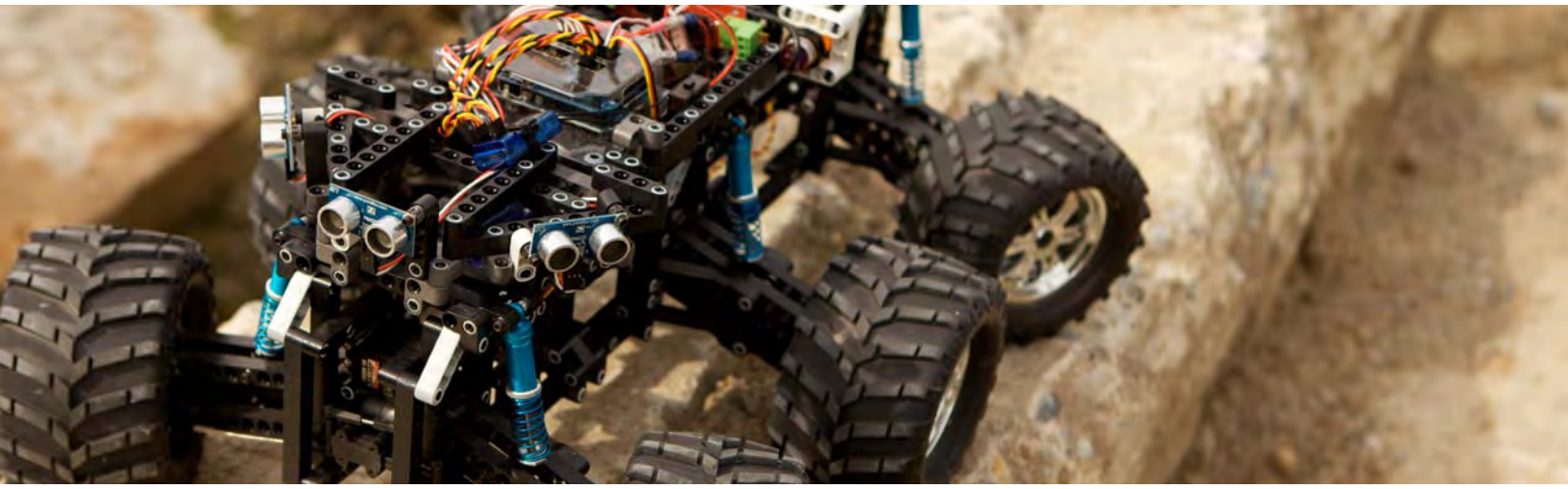
Users Guide



Helpful instructional videos available at: www.mymindsi.com

Table of contents

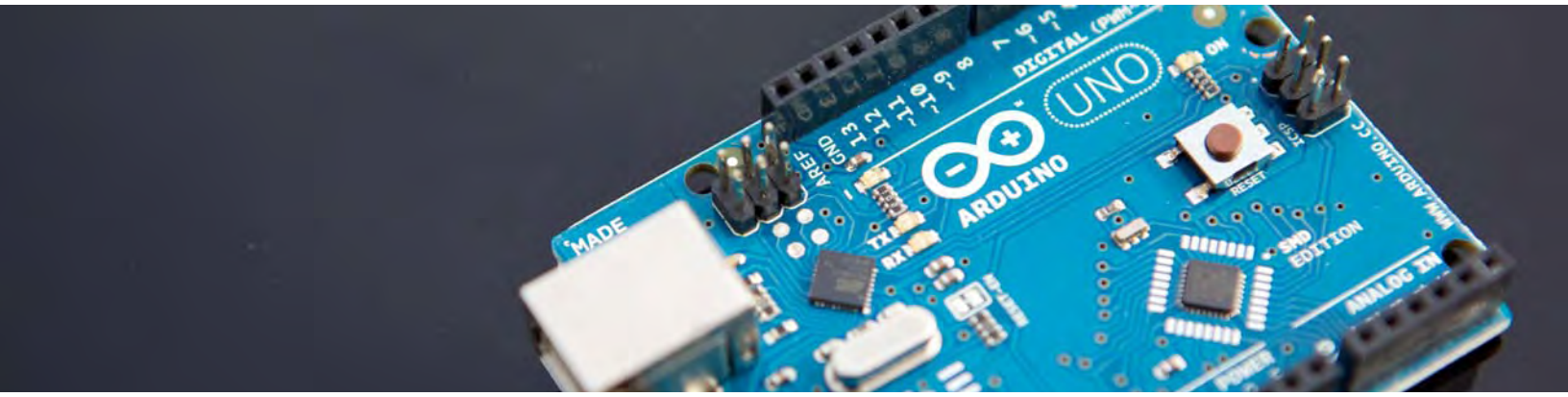
Welcome to the future of robotics



The machines of yesterday have evolved into a Robotics system that is able to handle the tough terrain of reality. For years robotics have been kept inside controlled environment situations, but today a new line of robotics is ready to face the challenge. MINDS-i all terrain robots go anywhere you do.

Unleash your MINDS-i. Unleash your Creativity.

This guide book has been adapted to teach beginners, experts, teachers and students the versatility of the MINDS-i system with the open source Arduino microcontroller. Included are in depth guides that show you step by step how to use Arduino, how to program your robot, and how to conquer various challenges.



Are you Ready?

MINDS-i®

Your Feedback is Important to us

This version of the guide is in it's early stages of development, and there may be small errors throughout. Your feed back is important to the design of this manual so that together we may make this guide book a success. If there are any topics that may need more explanation, or if there are additional topics that may need to be added your feedback will be greatly appreciated and adapted to improve future additions of this guide.

Contact Information

For technical questions or to place an order:

Voice: 1 (509) 252 - 5767
Fax: 1 (509) 924 - 2219

Email us at: info@mymindsi.com

Write to ATTN: MINDS-i Inc.
22819 East Appleway Avenue
Liberty Lake, Washington 99019

For the latest from MINDS-i visit

mindsirobotics.com

For the updated and lasted edition of the code and MINDS-i Library go to

mindsirobotics.com/code

Introduction

What is Arduino?

Arduino is a tool for making computers that can sense and control more of the physical world than your desktop computer. It's an open-source physical-computing platform based on a simple microcontroller board, and a development environment for writing software for the board.

Arduino can be used to develop interactive objects, taking inputs from a variety of switches or sensors, and controlling a variety of lights, motors, and other physical outputs. Arduino projects can be stand-alone, or they can be able to communicate with software running on your computer (e.g. Flash, Processing, and MaxMSP.) The boards can be assembled by hand or purchased pre-assembled; the open-source IDE can be downloaded for free.
The Arduino programming language is an implementation of Wiring, a similar physical-computing platform, which is based on the Processing multimedia-programming environment.

Why Arduino?

Arduino simplifies the process of working with microcontrollers, and offers some advantages for teachers, students, and interested amateurs over other systems:

- Cross-platform - The Arduino software runs on Windows, Macintosh OSX, and Linux operating systems. Most microcontroller systems are limited to Windows.

- Simple, clear programming environment - The Arduino programming environment is easy-to-use for beginners, yet flexible enough for advanced users. For teachers, it's conveniently based on the Processing programming environment, so students learning to program in that environment will be familiar with the look and feel of Arduino

- Open source and extensible software - The Arduino software is published as open source tools, available for extension by experienced programmers. The language can be expanded through C++ libraries, and people wanting to understand the technical details can make the leap from Arduino to the AVR C programming language on which it is based. Similarly, you have the capacity to add AVR-C code directly into your Arduino programs if you want to.

- Open source and extensible hardware - The Arduino is based on Atmel's ATMEGA8 and ATMEGA168 microcontrollers. The plans for the modules are published under a Creative Commons license, so experienced circuit designers can create their own version of the module, extending it and improving it. Even relatively inexperienced users can build the breadboard version of the module in order to understand how it works and save money.

Getting Started with Arduino

1 | Get an Arduino board and USB cable

In this tutorial, we assume you are using an Arduino Uno.

You also need a USB cable (A plug to B plug): the kind you would connect to a USB printer, for example.

2 | Download the Arduino environment

You will need the flash drive found in your MINDS-i kit.

Drag and drop the files onto your computer.

You can also download the latest version of the software and library from the code page.
http://www.mymindsi.com/code

When the download finishes, unzip the downloaded file. Make sure to preserve the folder structure. Double-click the folder to open it. There should be a few files and sub-folders inside.

For Macs copy the Arduino application into the Applications folder (or elsewhere on your computer). Since you're using an Arduino Uno you don't have any drivers to install. Continue to step 3 than Skip Step 4 install the drivers.

3 | Connect the board

The Arduino Uno automatically draws power from either the USB connection to the computer or an external power supply.

Connect the Arduino board to your computer using the USB cable. The green power LED (labeled PWR) should turn on.

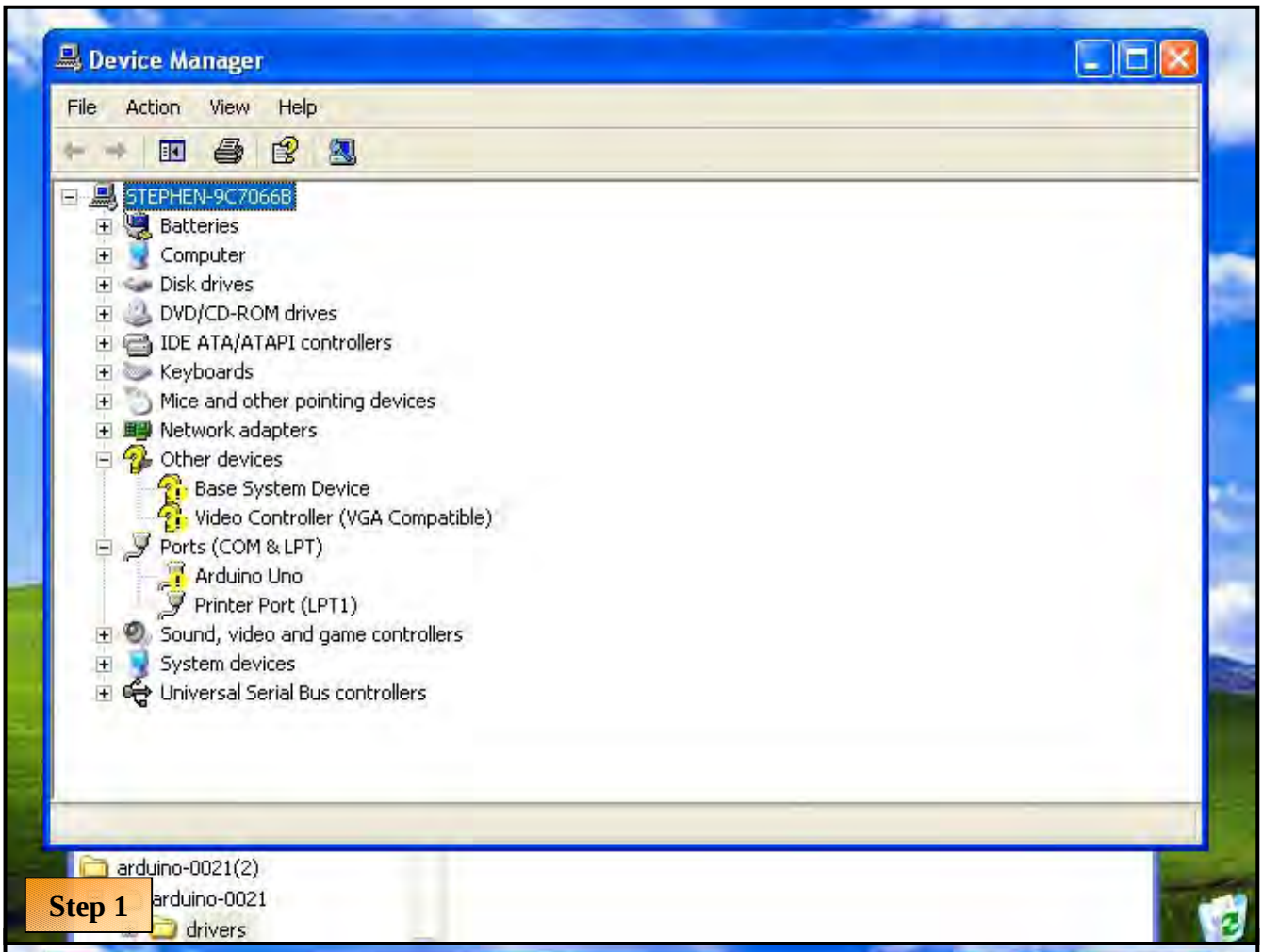
4 | Install the drivers

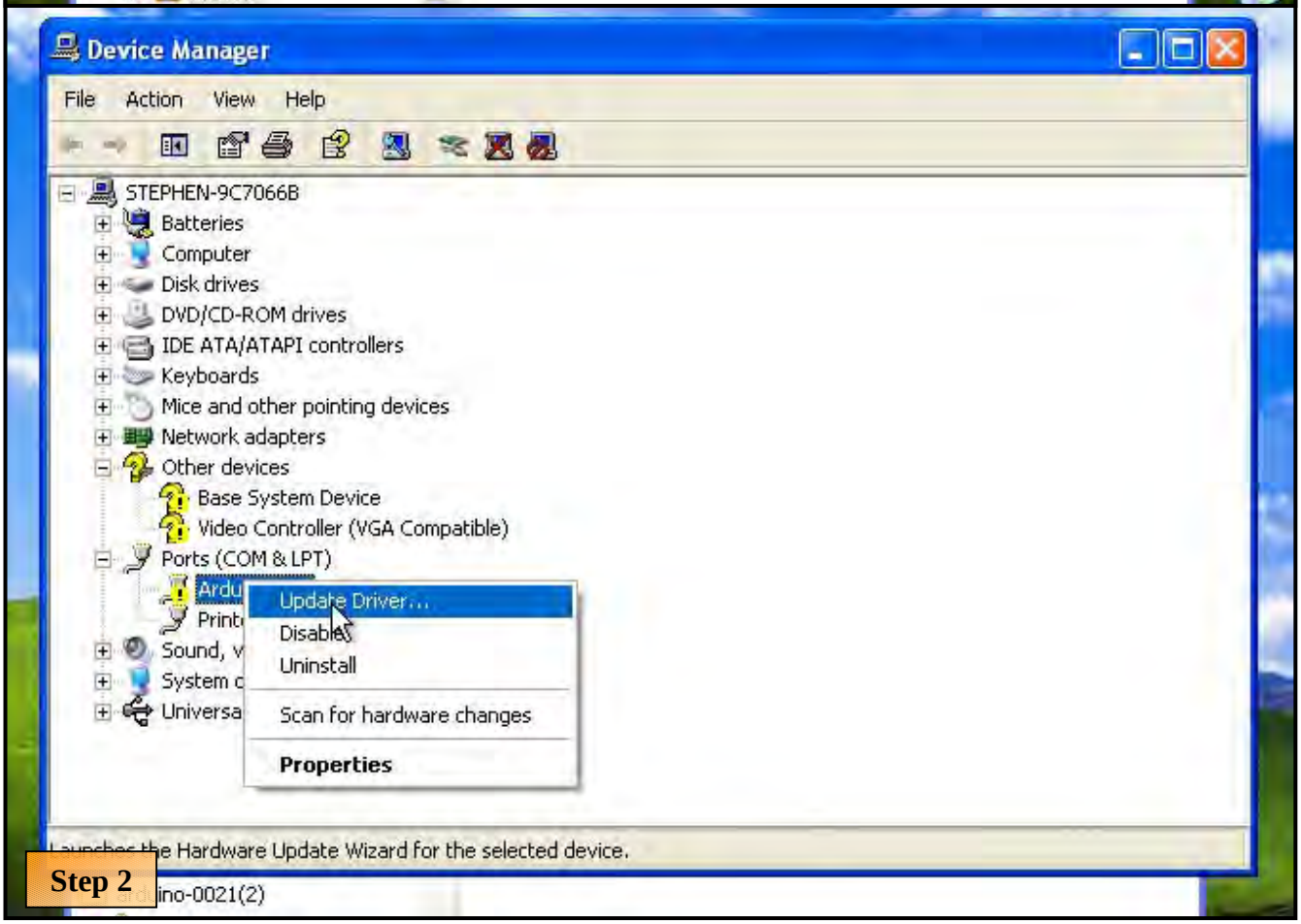
Installing drivers for the Arduino Uno with Windows7, Vista, or XP:

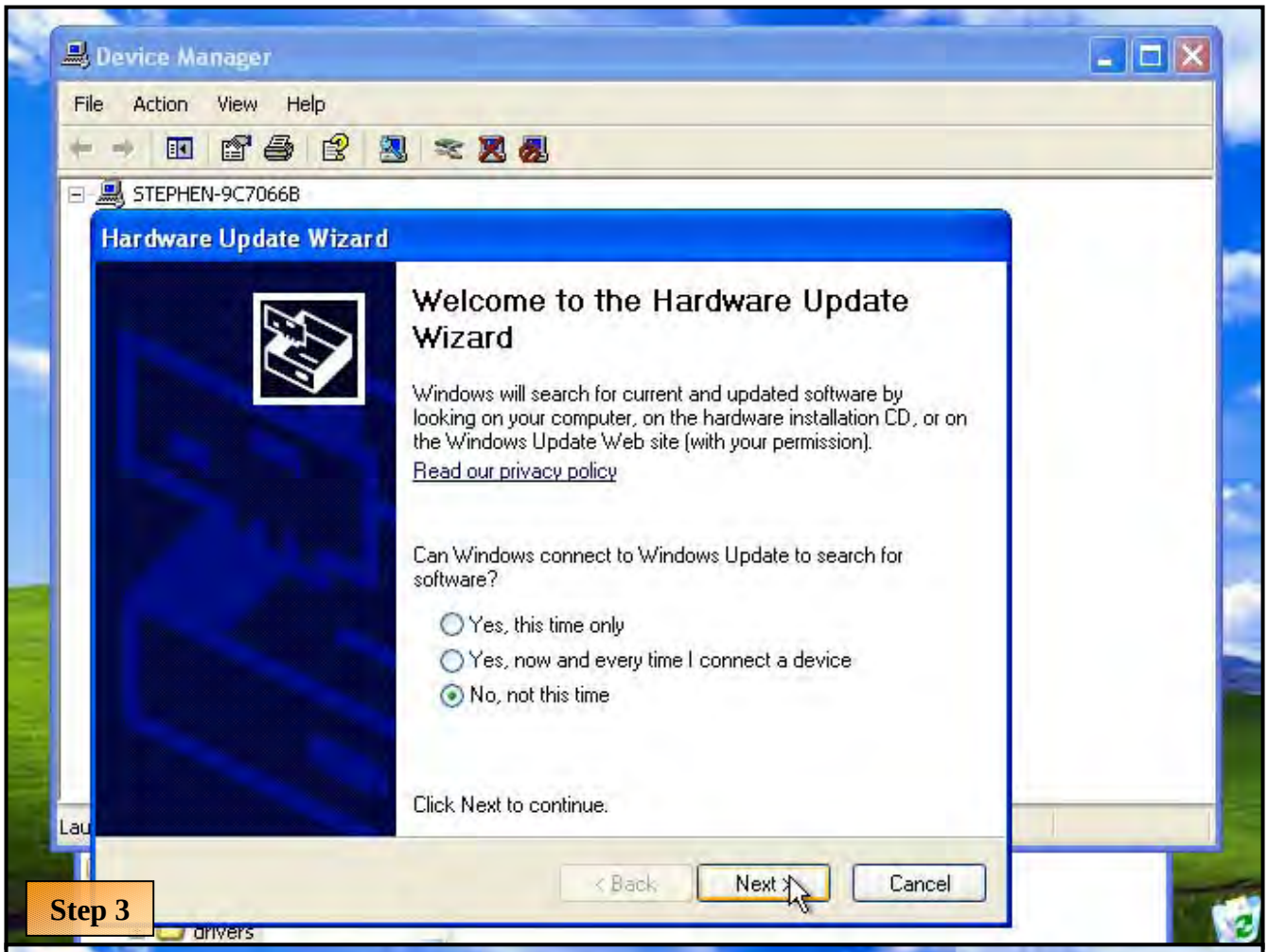
- Plug in your board and wait for Windows to begin its driver installation process. After a few moments, the process will fail, despite its best efforts
- Click on the Start Menu, and open the Control Panel.
- While in the Control Panel, navigate to System and Security. Next, click on System. Once the System window is open, double click on the Device Manager.
- Look under Ports (COM & LPT). You should see an open port named "Arduino UNO (COMxx)"
- Right click on the "Arduino UNO (COMxx)" port and choose the "Update Driver Software" option.
- Next, choose the "Browse my computer for Driver software" option.
- Finally, navigate to and select the Uno's driver file, named "ArduinoUNO.inf", located in the "Drivers" folder of the Arduino Software download (not the "FTDI USB Drivers" sub-directory).
- Windows will finish the driver installation from there.

You can check that the drivers have been installed by opening the Windows Device Manager (in the Hardware tab of System control panel). Look for a "USB Serial Port" in the Ports section; that is the Arduino board.

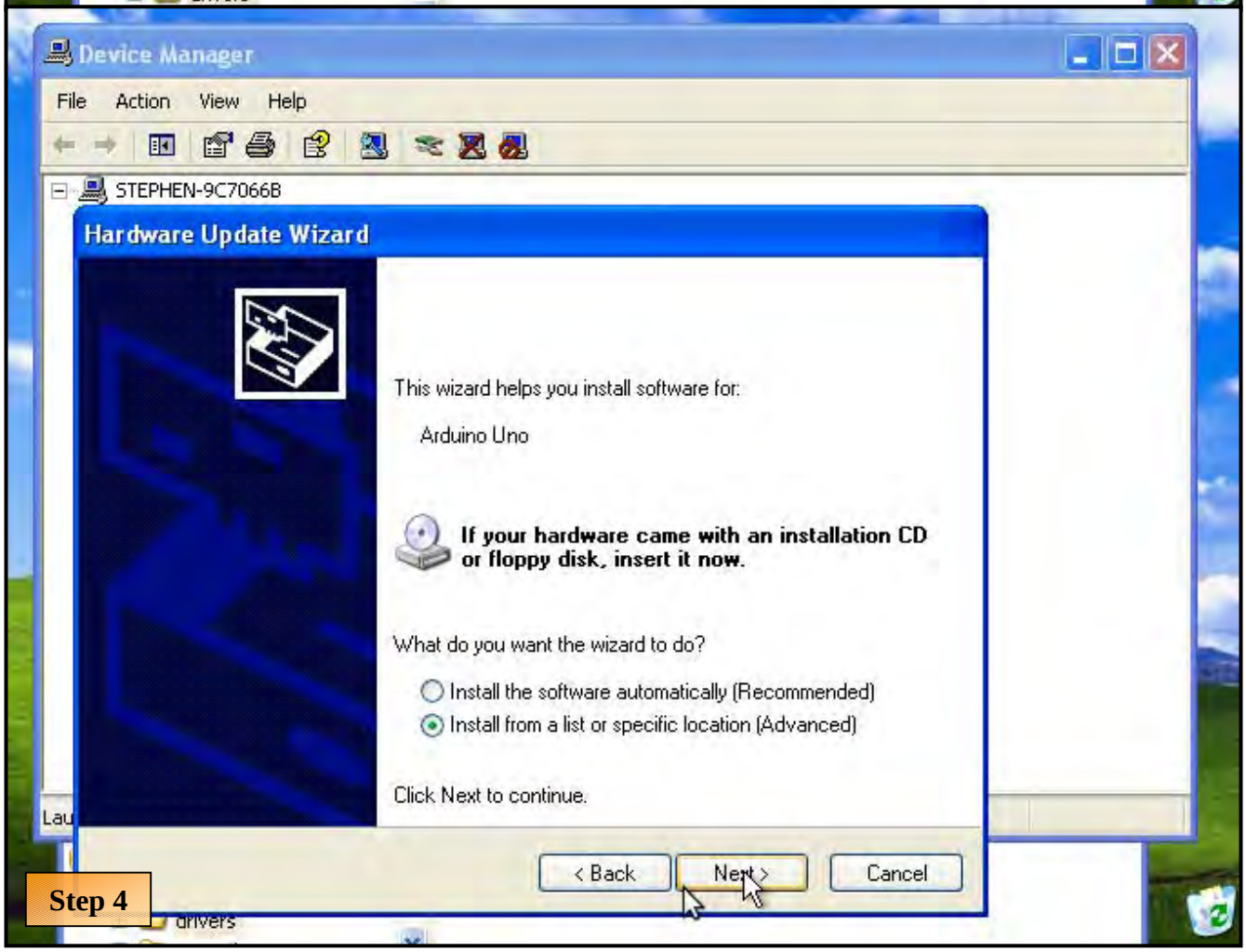
Follow the step-by-step screenshots for installing the Uno under Windows XP.


Device Manager
File Action View Help
STEPHEN-9C7066B
Batteries
Computer
Disk drives
DVD/CD-ROM drives
IDE ATA/ATAPI controllers
Keyboards
Mice and other pointing devices
Network adapters
Other devices
Base System Device
Video Controller (VGA Compatible)
Ports (COM & LPT)
Arduino Uno
Printer Port (LPT1)
Sound, video and game controllers
System devices
Universal Serial Bus controllers
arduino-0021(2)
arduino-0021
drivers
Step 1

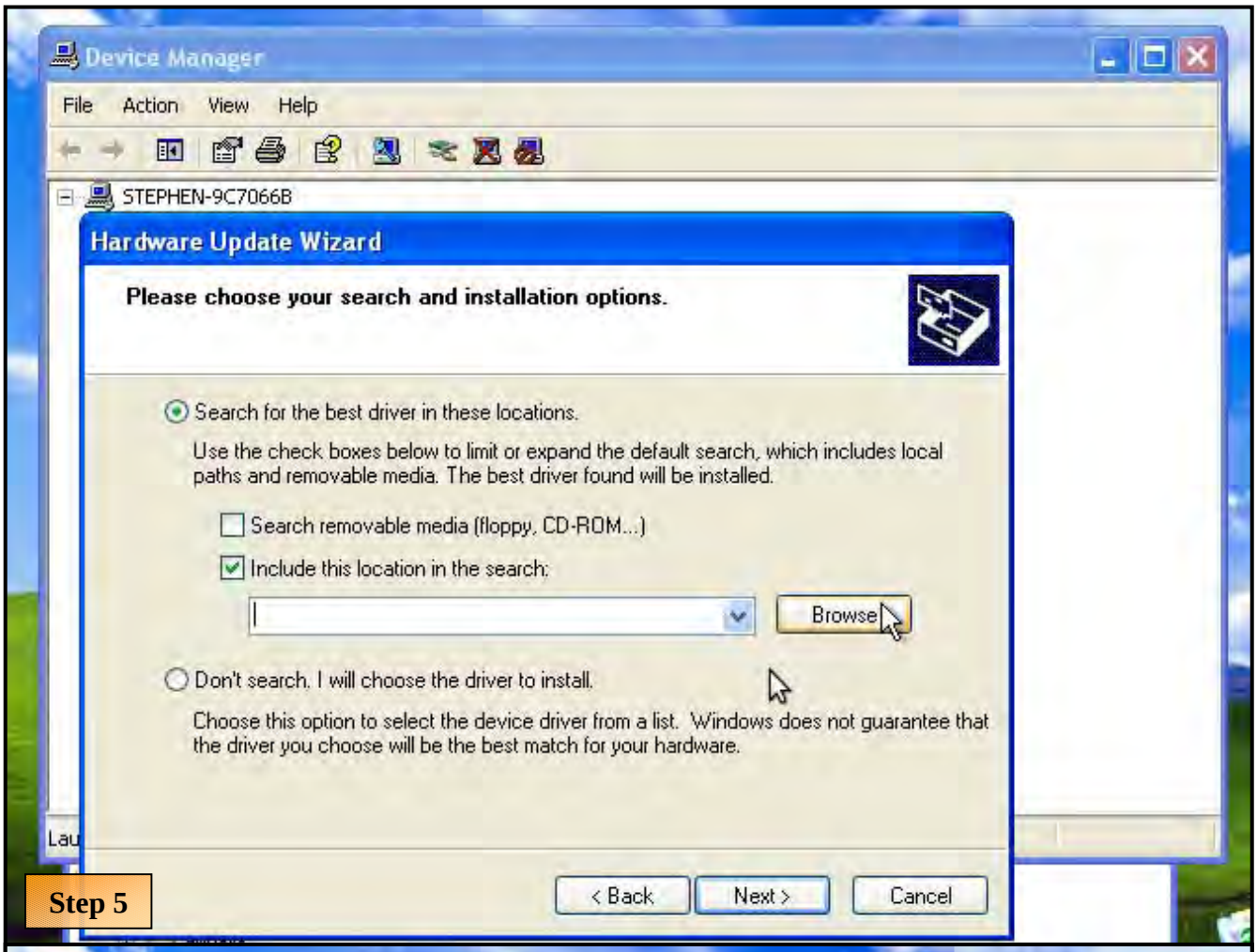

Device Manager
File Action View Help
STEPHEN-9C7066B
Batteries
Computer
Disk drives
DVD/CD-ROM drives
IDE ATA/ATAPI controllers
Keyboards
Mice and other pointing devices
Network adapters
Other devices
Base System Device
Video Controller (VGA Compatible)
Ports (COM & LPT)
Update Driver...
Disable
Uninstall
Scan for hardware changes
Properties
Launches the Hardware Update Wizard for the selected device.
Step 2


Device Manager
File Action View Help
STEPHEN-9C7066B
Hardware Update Wizard
Welcome to the Hardware Update Wizard
Windows will search for current and updated software by looking on your computer, on the hardware installation CD, or on the Windows Update Web site (with your permission).
Read our privacy policy
Can Windows connect to Windows Update to search for software?
Yes, this time only
Yes, now and every time I connect a device
No, not this time
Click Next to continue.
< Back
Next >
Cancel
drivers

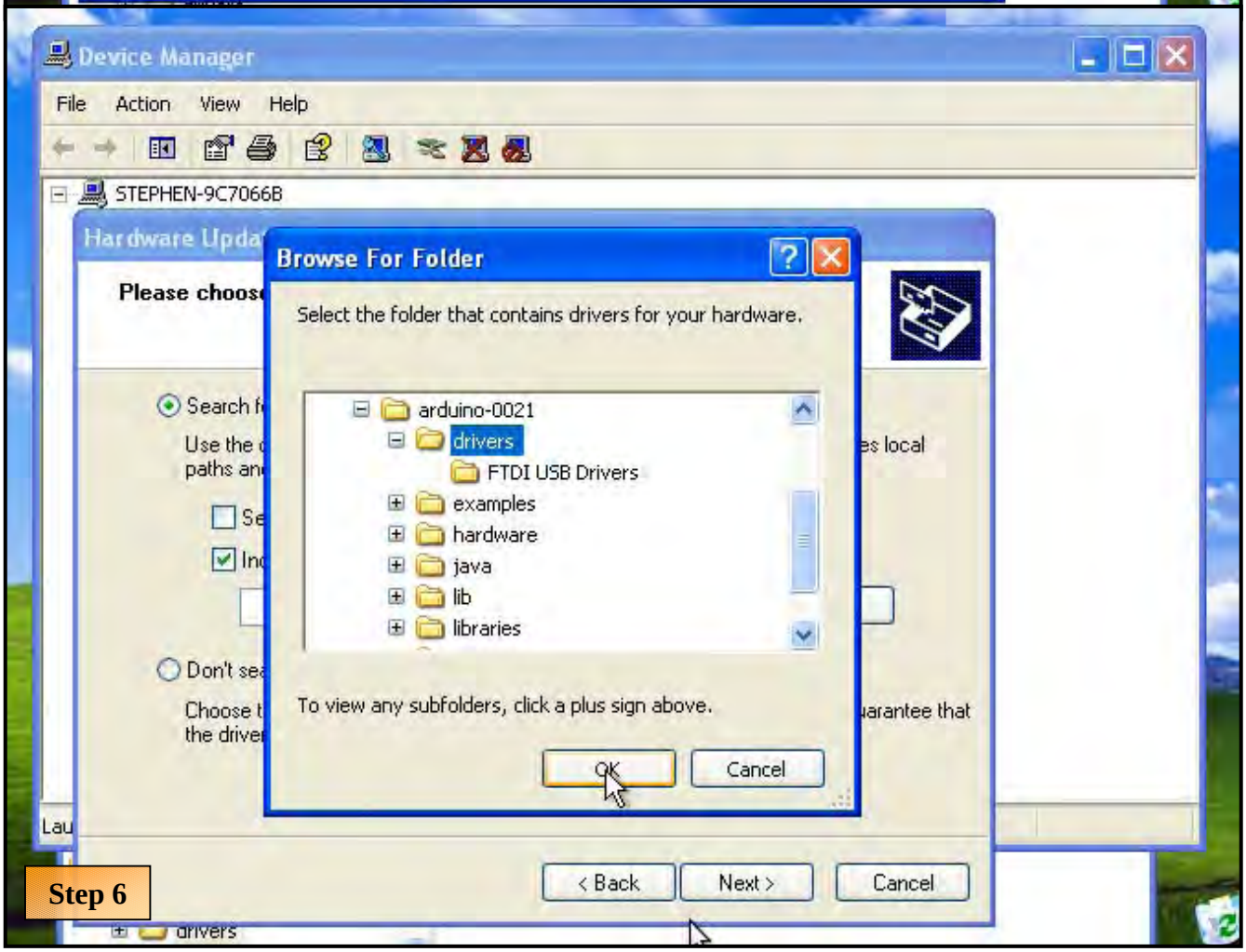
Step 3


Device Manager
File Action View Help
STEPHEN-9C7066B
Hardware Update Wizard
This wizard helps you install software for:
Arduino Uno
If your hardware came with an installation CD or floppy disk, insert it now.
What do you want the wizard to do?
Install the software automatically (Recommended)
Install from a list or specific location (Advanced)
Click Next to continue.
< Back
Next >
Cancel
drivers

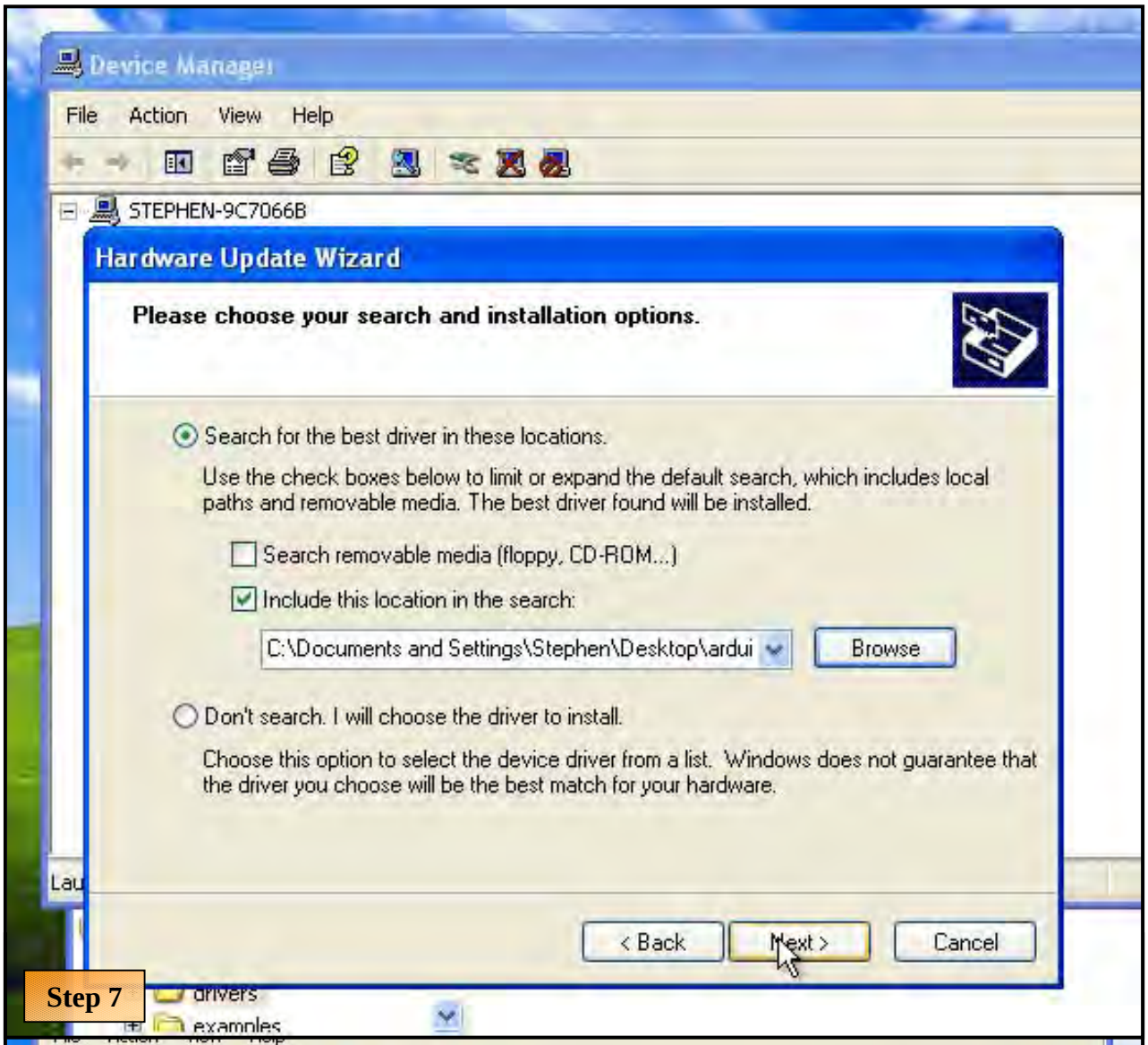
Step 4


Device Manager
File Action View Help
STEPHEN-9C7066B
Hardware Update Wizard
Please choose your search and installation options.
Search for the best driver in these locations.
Use the check boxes below to limit or expand the default search, which includes local paths and removable media. The best driver found will be installed.
Search removable media (floppy, CD-ROM...)
Include this location in the search:
Browse
Don't search. I will choose the driver to install.
Choose this option to select the device driver from a list. Windows does not guarantee that the driver you choose will be the best match for your hardware.
< Back
Next >
Cancel

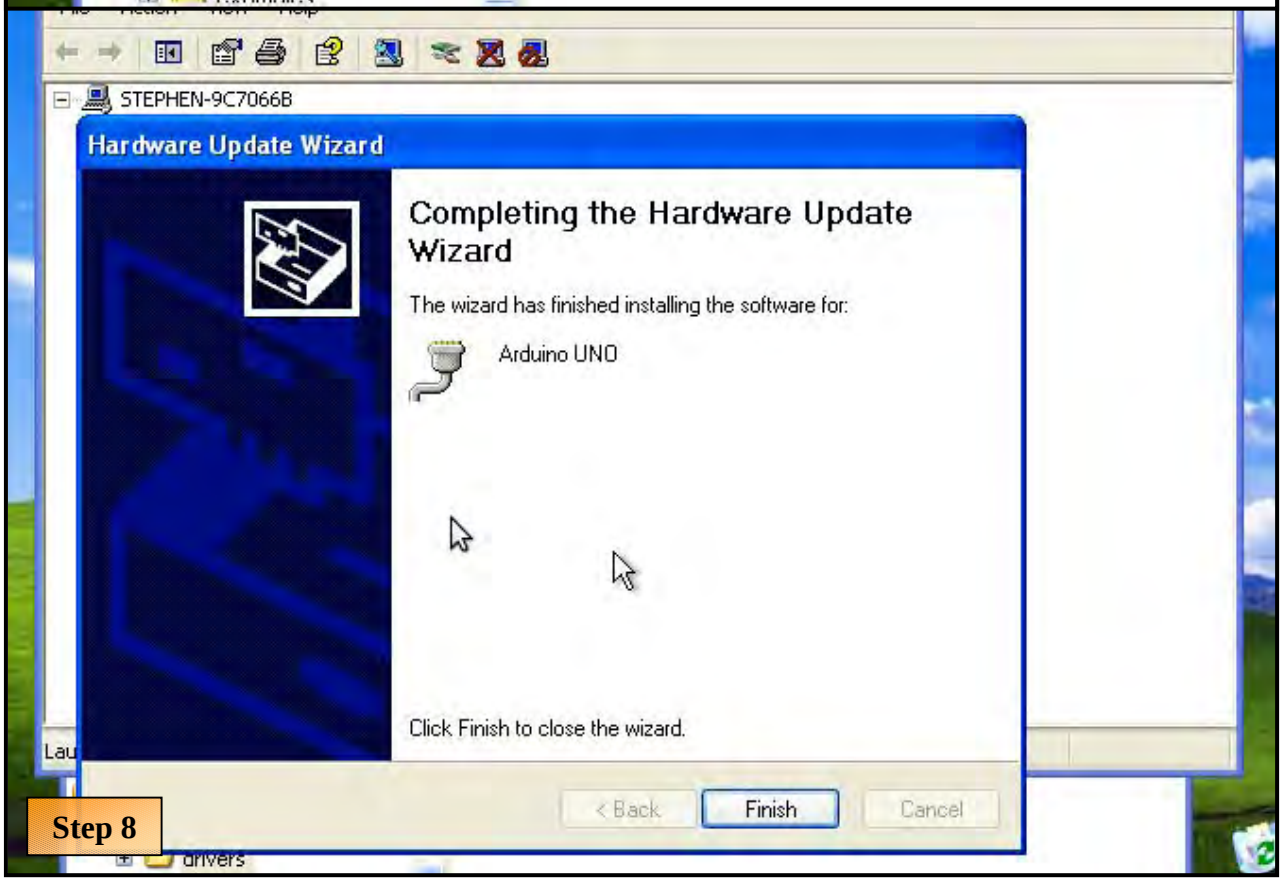
Step 5


Device Manager
File Action View Help
STEPHEN-9C7066B
Browse For Folder
Select the folder that contains drivers for your hardware.
arduino-0021
drivers
FTDI USB Drivers
examples
hardware
java
lib
libraries
To view any subfolders, click a plus sign above.
OK
Cancel
< Back
Next >
Cancel

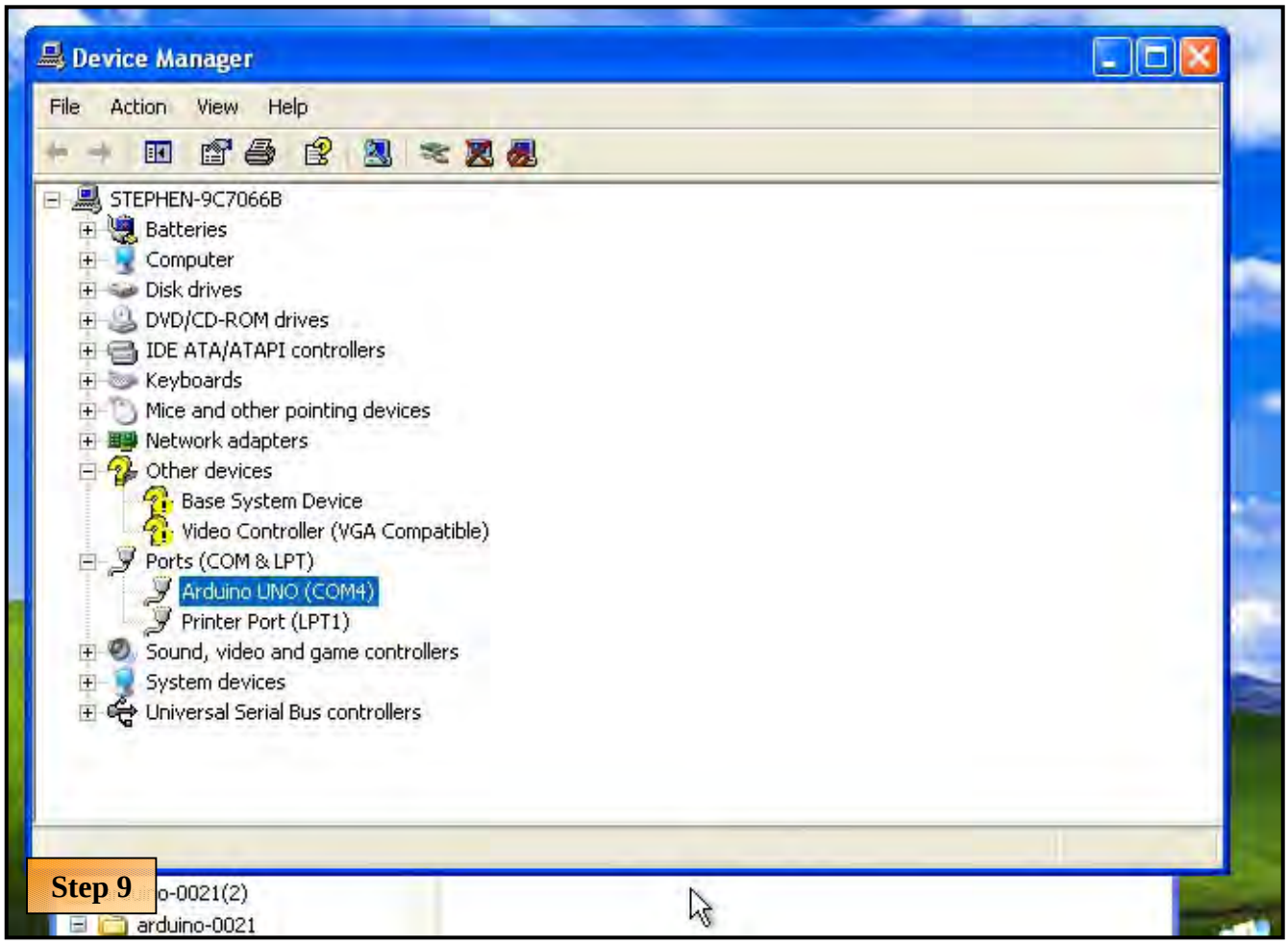
Step 6


Device Manager
File Action View Help
STEPHEN-9C7066B
Hardware Update Wizard
Please choose your search and installation options.
Search for the best driver in these locations.
Use the check boxes below to limit or expand the default search, which includes local paths and removable media. The best driver found will be installed.
Search removable media (floppy, CD-ROM...)
Include this location in the search:
C:\Documents and Settings\Stephen\Desktop\ardui
Browse
Don't search. I will choose the driver to install.
Choose this option to select the device driver from a list. Windows does not guarantee that the driver you choose will be the best match for your hardware.
< Back
Next >
Cancel
drivers
examples

Step 7


STEPHEN-9C7066B
Hardware Update Wizard
Completing the Hardware Update Wizard
The wizard has finished installing the software for:
Arduino UNO
Click Finish to close the wizard.
< Back
Finish
Cancel
drivers

Step 8

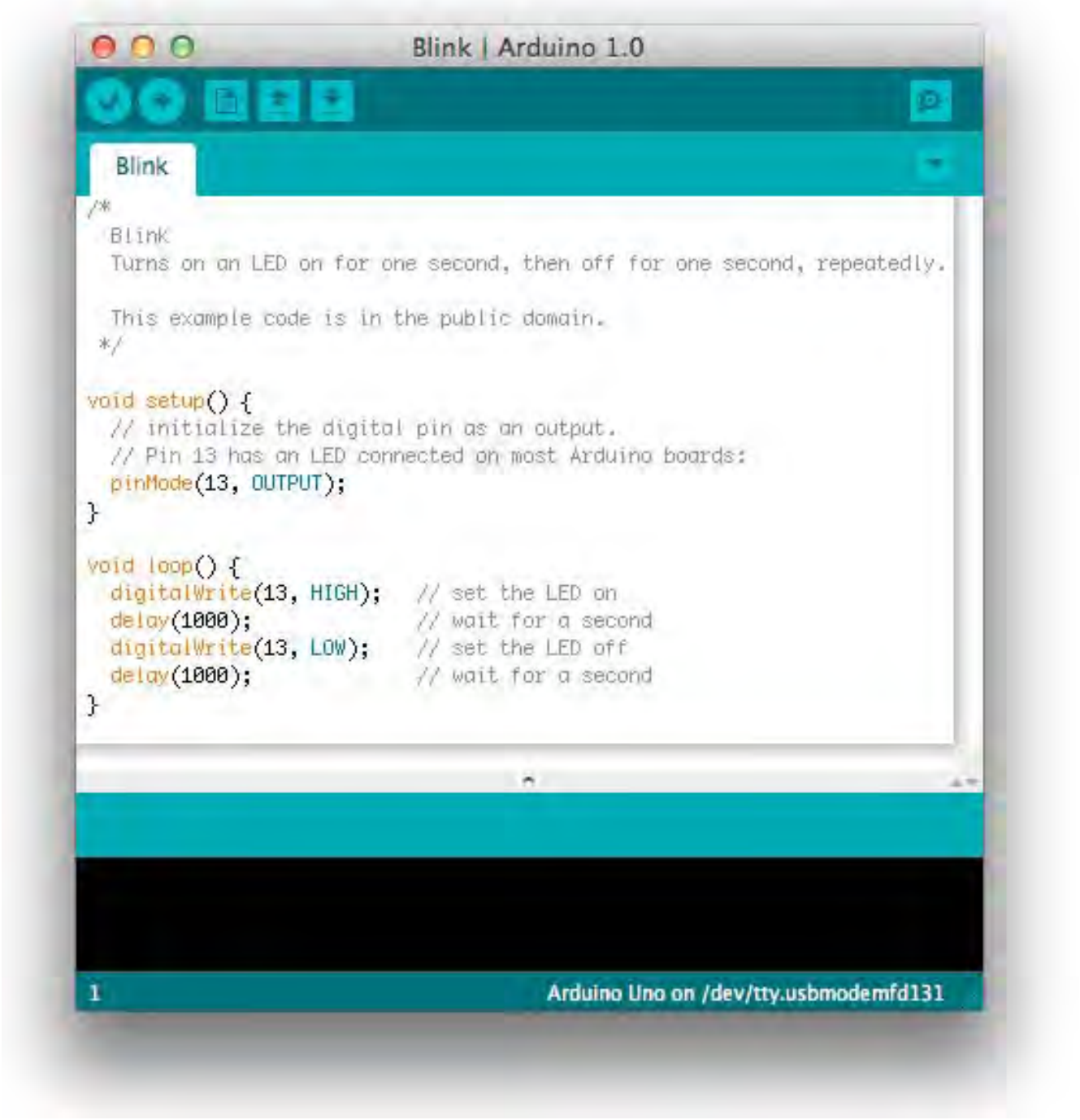

Device Manager
File Action View Help
STEPHEN-9C7066B
Batteries
Computer
Disk drives
DVD/CD-ROM drives
IDE ATA/ATAPI controllers
Keyboards
Mice and other pointing devices
Network adapters
Other devices
Base System Device
Video Controller (VGA Compatible)
Ports (COM & LPT)
Arduino UNO (COM4)
Printer Port (LPT1)
Sound, video and game controllers
System devices
Universal Serial Bus controllers
Step 9
o-0021(2)
arduino-0021

5 | Launch the Arduino application

Double-click the Arduino application. (Note: if the Arduino software loads in the wrong language, you can change it in the preferences dialog, Ctrl + Comma)

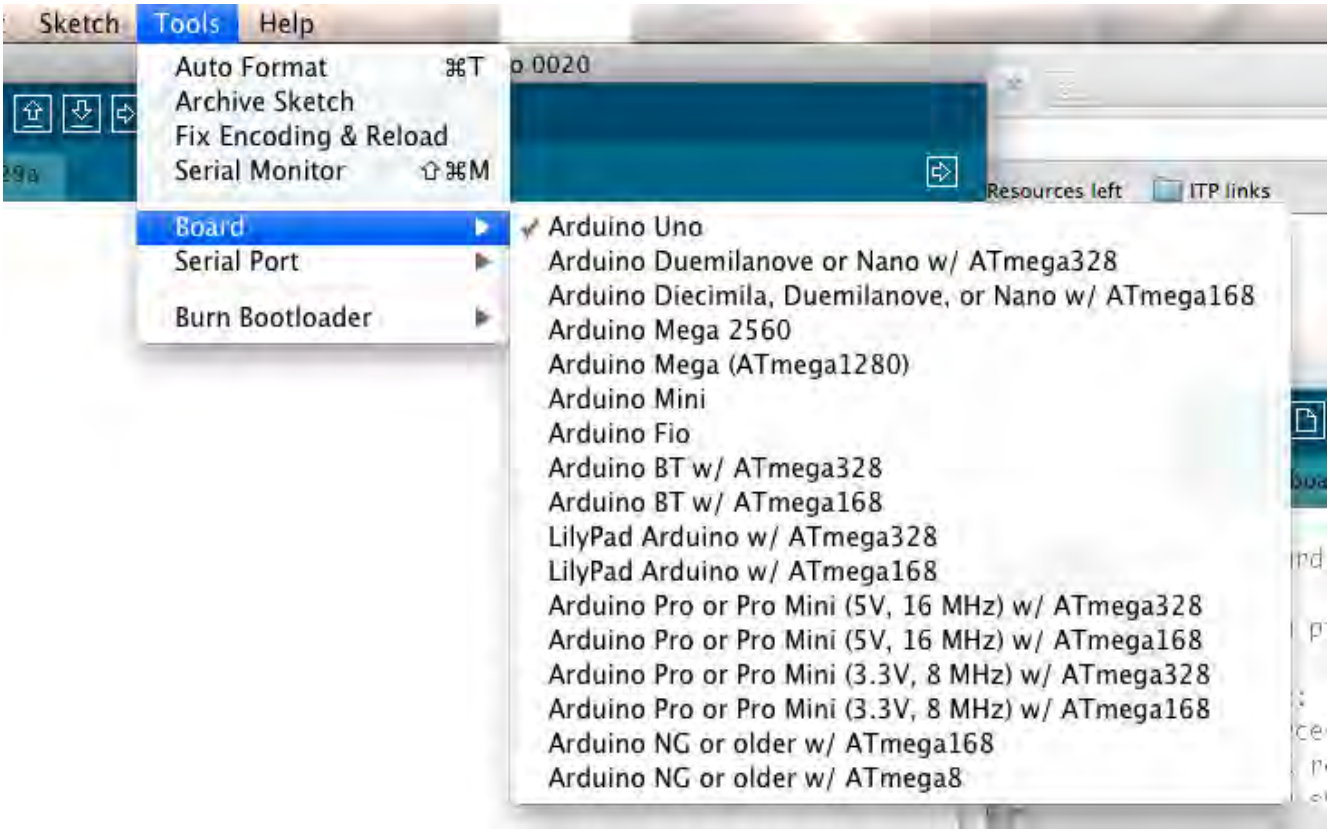
6 | Open the blink example

Open the LED blink example sketch: File > Examples > 1.Basics > Blink.



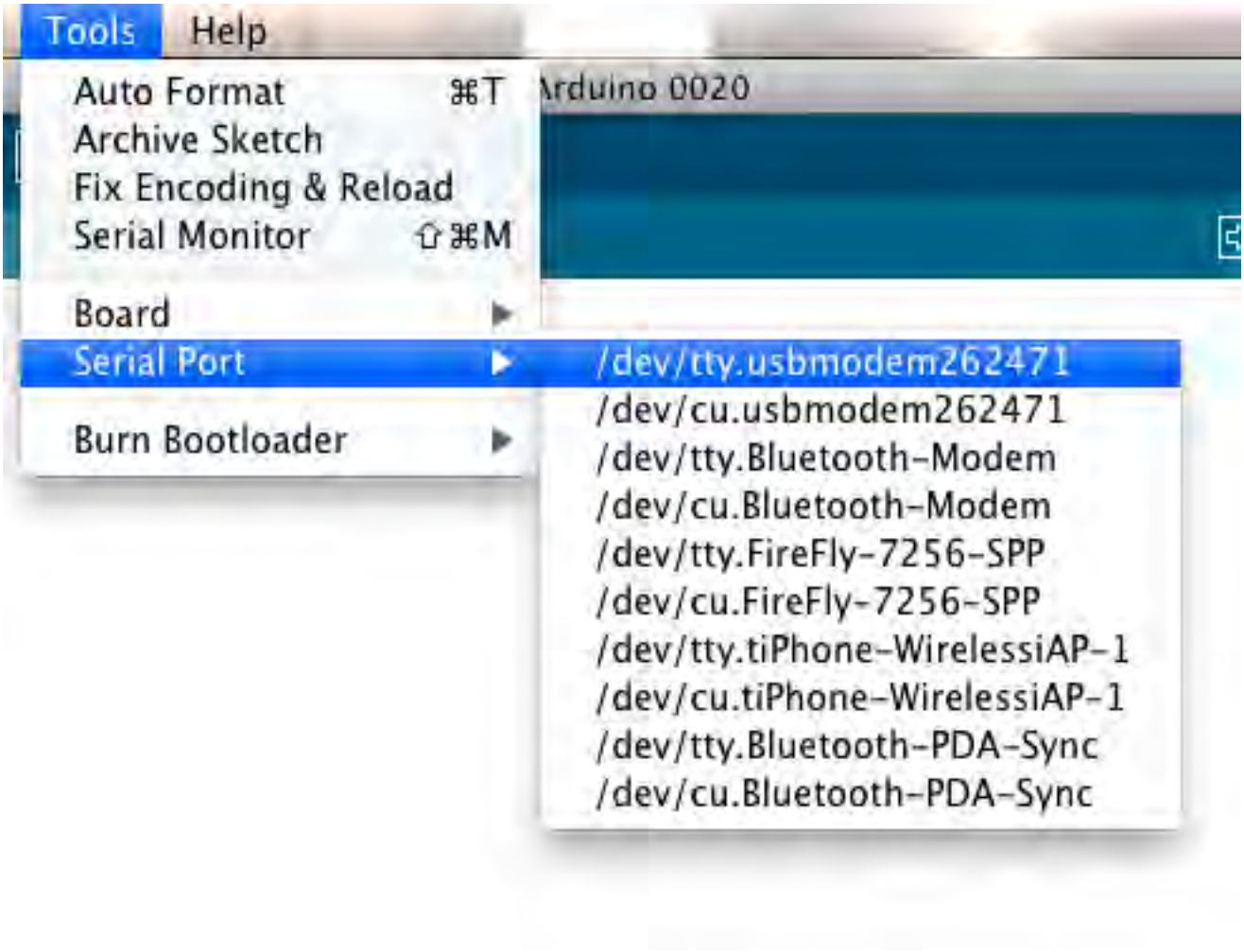
7 | Select your board

You'll need to select the entry in the Tools > Board menu that corresponds to your Arduino.



8 | Select your serial port

Select the serial device of the Arduino board from the Tools | Serial Port menu. This is likely to be COM3 or higher (COM1 and COM2 are usually reserved for hardware serial ports). To find out, you can disconnect your Arduino board and re-open the menu; the entry that disappears should be the Arduino board. Reconnect the board and select that serial port.



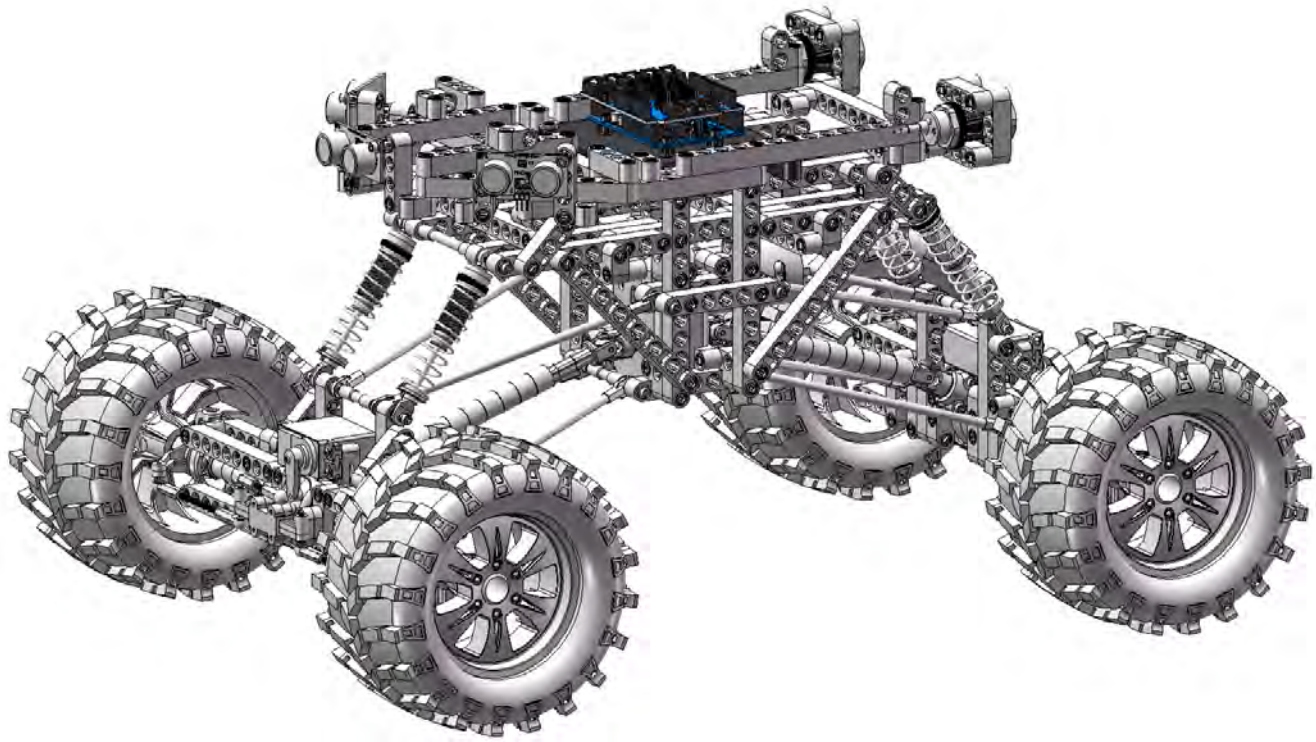
9 | Upload the program

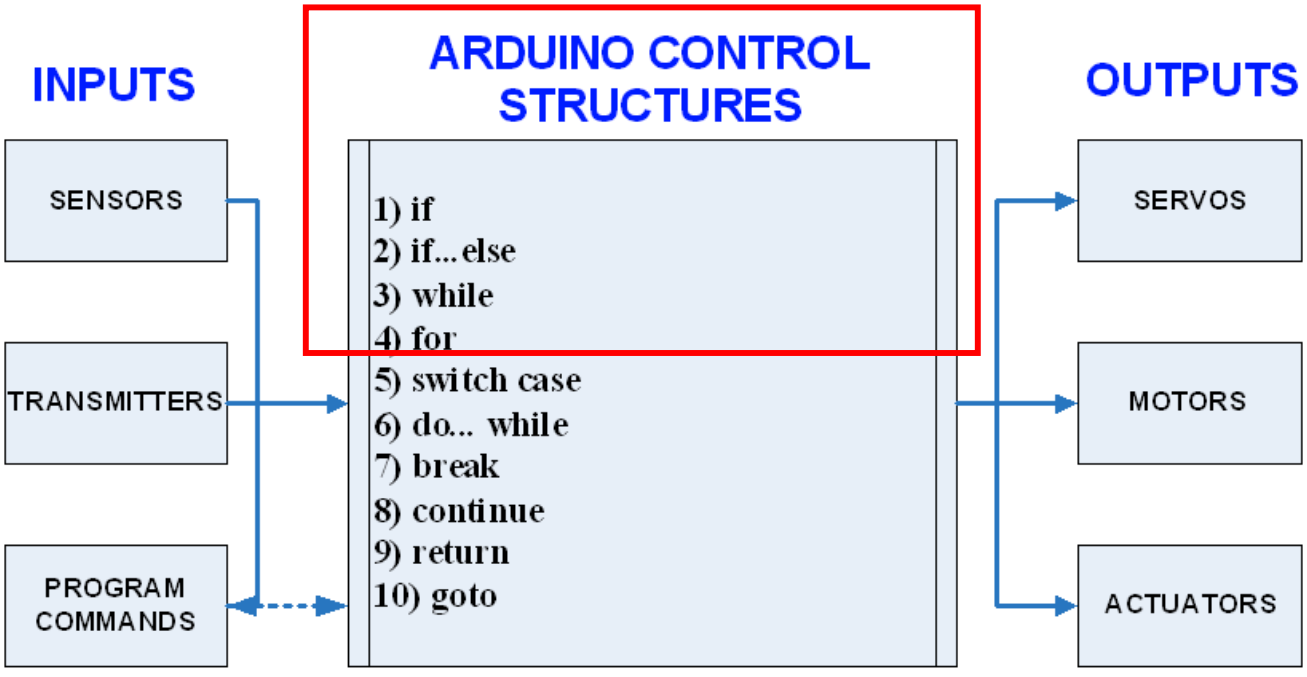
Now, simply click the "Upload" button in the environment. Wait a few seconds - you should see the RX and TX LED's on the board flashing. If the upload is successful, the message "Done uploading." will appear in the status bar.

A few seconds after the upload finishes, you should see the pin 13 (L) LED on the board start to blink (in orange). If it does, congratulations! You now have Arduino up-and-running.

If you have problems, please see the troubleshooting guide or
http://arduino.cc/en/Guide/Troubleshooting

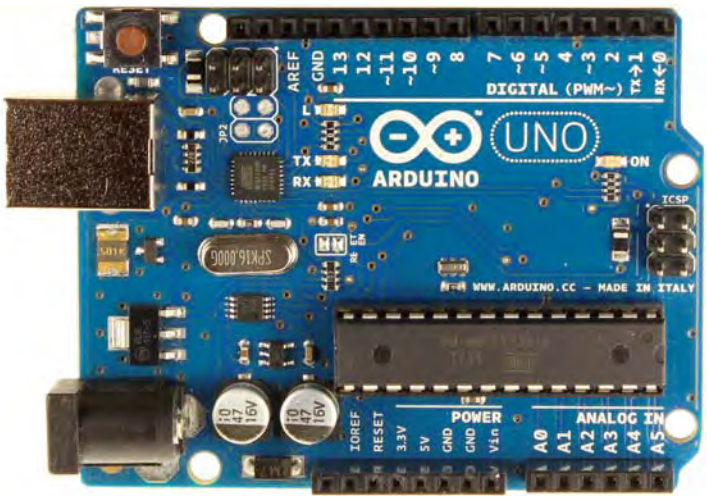
Arduino the Micro-controller

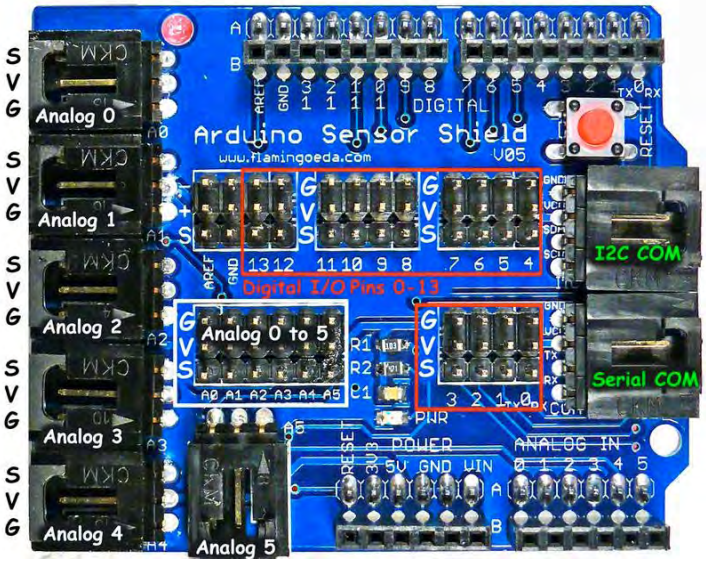




Arduino Uno R3

Throughout this guide we will be using the Arduino Uno R3 and the Sensor Shield. The Arduino Uno is a microcontroller board based on the ATmega328. It has 14 digital input/output pins (of which 6 can be used as PWM outputs), 6 analog inputs, a 16 MHz crystal oscillator, a USB connection, a power jack, an ICSP header, and a reset button. It contains everything needed to support the microcontroller; simply connect it to a computer with a USB cable or power it with a AC-to-DC adapter or battery to get started.



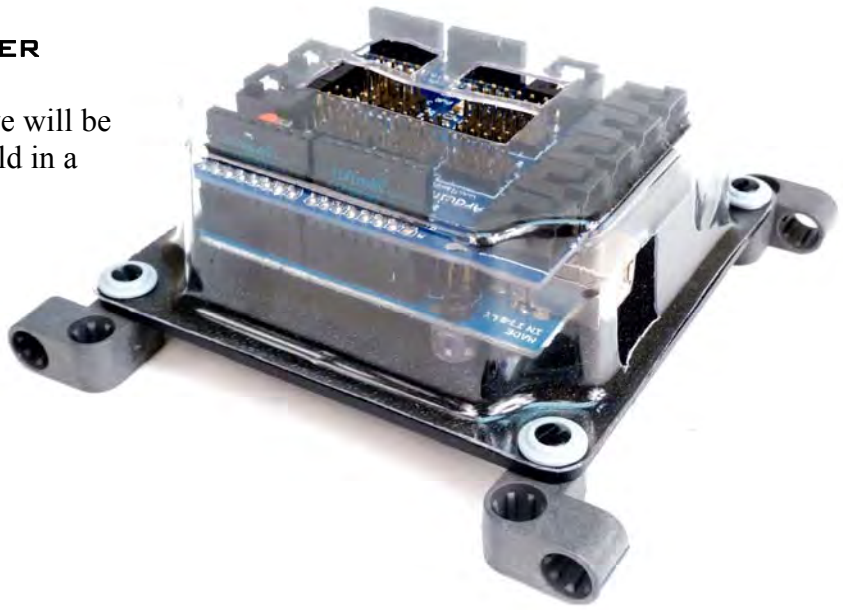


Sensor Shield

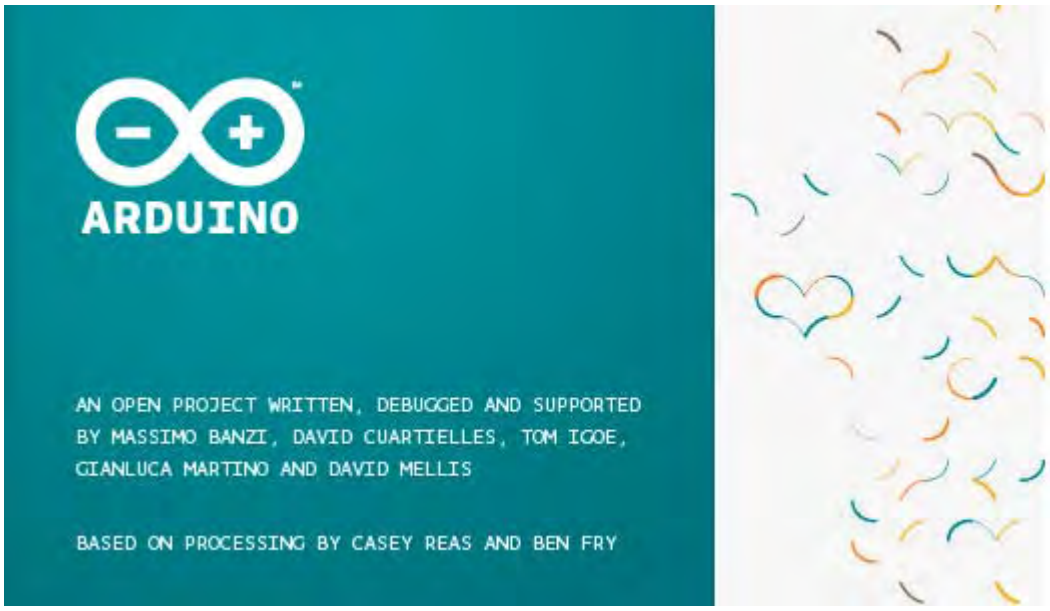
The Sensor Shield's purpose is to make it easy to connect cables and devices to the correct Arduino pins. It simply connects the Arduino pins to many connectors that are ready to use to connect to various devices like Servos and Sensors with simple cables. Each Port has 3 pins which are connected to (Ground), (Vcc + 5 V) and (Signal). See the GVS labels below Cables are normally color coded so you know the right way to plug them in: Ground Black, Voltage Red, and Signal White or Yellow.

Mounting Plate and Cover

In most of the exercises in this guide we will be using the Arduino with the Sensor shield in a stack. MINDS-i includes a mounting plate that will connect to the system. A protective cover is also included and it is suggested to use for protection of dirt and other debris.



A Few Terms
To Get You Started



The Arduino is programmed in the C language. This is a quick guide to help you understand some of the key concepts. If you find the concepts a bit daunting, don't worry you can start going through the examples and pick up most of it along the way. For a more in-depth guide, see the Reference Guide section. Also see the Arduino.cc website for more information.

Structure

Each Arduino program also called a "sketch", has two required functions also called "routines"

void setup( ) { }

The code in the "void setup" between the two curly brackets will only run once when your Arduino program begins.

void loop ( ) { }

The "void loop" is run after the setup has finished. After it has ran once it will be run again, and again, until power is removed

Syntax

One of the slightly frustrating elements of C is its formatting requirements (this also makes it very powerful). If you remember the following you should be all right.

//
(Single line comment)

It is often useful to write notes to yourself as you go along about what each line of code does. To do this type two forward slashes at the beginning of your comment and everything until the end of the line will be ignored by your program.

/* */
(Multi line comment)

If you have a lot to say you can span several lines as a comment. Everything between these two symbols will be ignored in your program.

{ }
(Curly brackets)
Used to define when a block of code starts and ends (used in control structures as well as loops).

;
(Semicolon)
Each line of code must be ended with a semicolon (a missing semicolon is often the reason for a program refusing to compile).

Data Types

A program is nothing more than instruction to move numbers around in an intelligent way. Variables are used to do the moving.

int (integer)
The main workhorse; stores a number in 2 bytes (16bits). Has no decimal places and will store a value between –32,768 and 32,767

long (long)
Used when an integer is not large enough. Takes 4 bytes (32 bits) of RAM and has a range between -2,147,483,648 and 2,147,483,647.

Boolean (Boolean)
A simple True or False variable. Useful because it only uses one bit of RAM.

Float (float)
Used for floating point math (decimals). Takes 4 bytes (32 bits) of RAM and has a range between –3.4028235E+38 and 3.4028235E+38.

char (character)
Stores one character using the ASCII code (ie 'A' = 65). Uses one byte (8 bites) of RAM. The Arduino handles strings as an array of char's.

Arithmetic Operators

Operators used for manipulating numbers. (They work like simple math).

= (assignment) makes something equal to something else (es. X = 10 * 2 (x now equals 20))

+ (addition) adds one value to another (ex. delay(1000) + delay(500) (delay now equals 1500))

- (subtraction) subtracts one value from another (ex. delay(1000) - delay(500) (delay now equals 500))

* (multiplication) Multiplies one value by another (ex. delay(1000) * delay(500) (delay now equals 500,000))

/ (division) Divides one value by another (ex. delay(1000) / delay(500) (delay now equals 2))

% (modulo) gives the remainder when one number is divided by another (ex. 12 % 10 (gives 2))

Comparison Operators

Operators used for logical comparison.

= = (equal to) ex. 12 = = 10 is FALSE or 12 = = 12 is TRUE

! = (not equal to) ex. 12 ! = 10 is TRUE or 12 ! = 12 is FALSE

< (less than) ex. 12 < 10 is FALSE or 12 < 12 is FALSE or 12 < 14 is TRUE

> (greater than) ex. 12 > 10 is TRUE or 12 > 12 is FALSE or 12 > 14 is FALSE

Control Structure

Programs are reliant on controlling what runs next, here are the basic control elements (There are many more, see the Reference Guide)

If else

If(condition){ }
Else if(condition) { }
Else { }

This will execute the code between the curly brackets if the condition is true, and if not it will test the else if condition, if that is also false the else code will be executed.

Example:

```
if (pinFiveInput < 500)
{
  // do Thing A
}
else if (pinFiveInput >= 1000)
{
  // do Thing B
}
else
{
  // do Thing C
}
```

For

For (int I = 0; i< #repeats; i++) { }

Used when you would like to repeat a chunk of code a number of times (can count up i++ or down i- - or use any variable).

Example:

```
// Dim an LED using a PWM pin
int PWMpin = 10; // LED in series with 470 ohm resistor on pin 10

void setup()
{
  // no setup needed
}

void loop()
{
   for (int i=0; i <= 255; i++){
      analogWrite(PWMpin, i);
      delay(10);
   }
}
```

Digital

pinMode (pin, mode);

Used to set a pin's mode, pin is the pin number you would like to address 0-19 (analog 0-5 are 14-19). The mode can either be INPUT or OUTPUT.

digitalWrite (pin, value);

Once a pin is set as OUTPUT it can be set either HIGH (pulled to +5 volts) or LOW (pulled to ground) (ex. To turn a LED on you would digitalWrite whatever pin it was connected to HIGH)

Int digitalRead (pin);

Once a pin is set as an INPUT you can use this to return whether it is HIGH (pulled to +5 volts) or LOW (pulled to ground). (ex. Having a switch activated will return a value of HIGH)

Analog

The Arduino is a digital machine but it has the ability to operate in the analog realm. Here's how to deal with things that aren't digital.

Int analogWrite (pin, value);
Some of the Arduino's pins support pulse width modulation (3, 5, 6, 9, 10, 11). This turns the pin on and off very quickly making it act like an analog output. The value is any number between 0 (0% duty cycle ~0v) and 255 (100% duty cycle ~5 volts).

Int analogRead (pin);
When the analog input pins are set to input you can read their voltage. A value between 0 (for 0 volts) and 1024 (for 5 volts) will be returned. Pins A0 – A5 are used for analogRead.

TIME

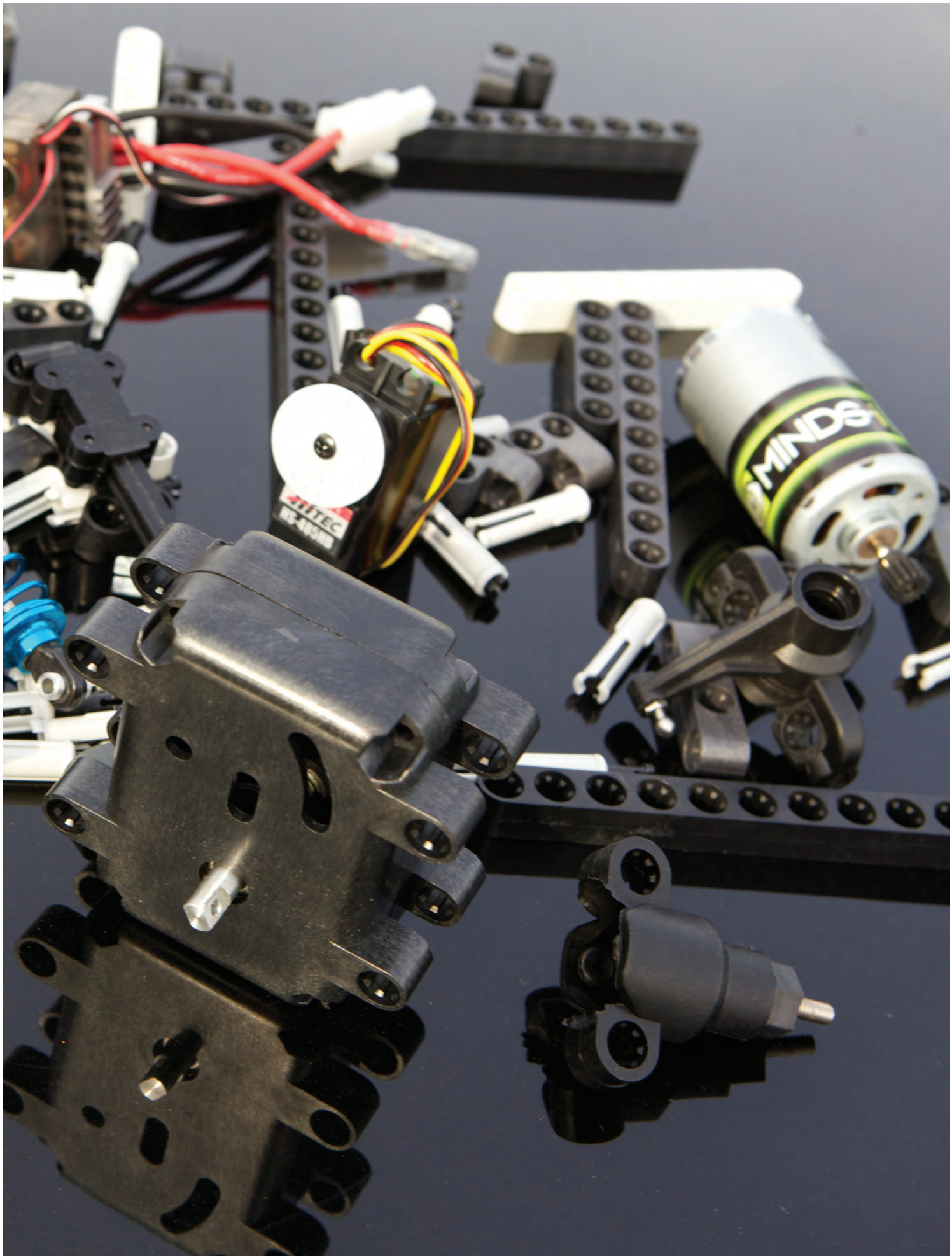
Millis( )

Returns the number of milliseconds since the Arduino board began running the current program. This number will overflow (go back to zero), after approximately 50 days.

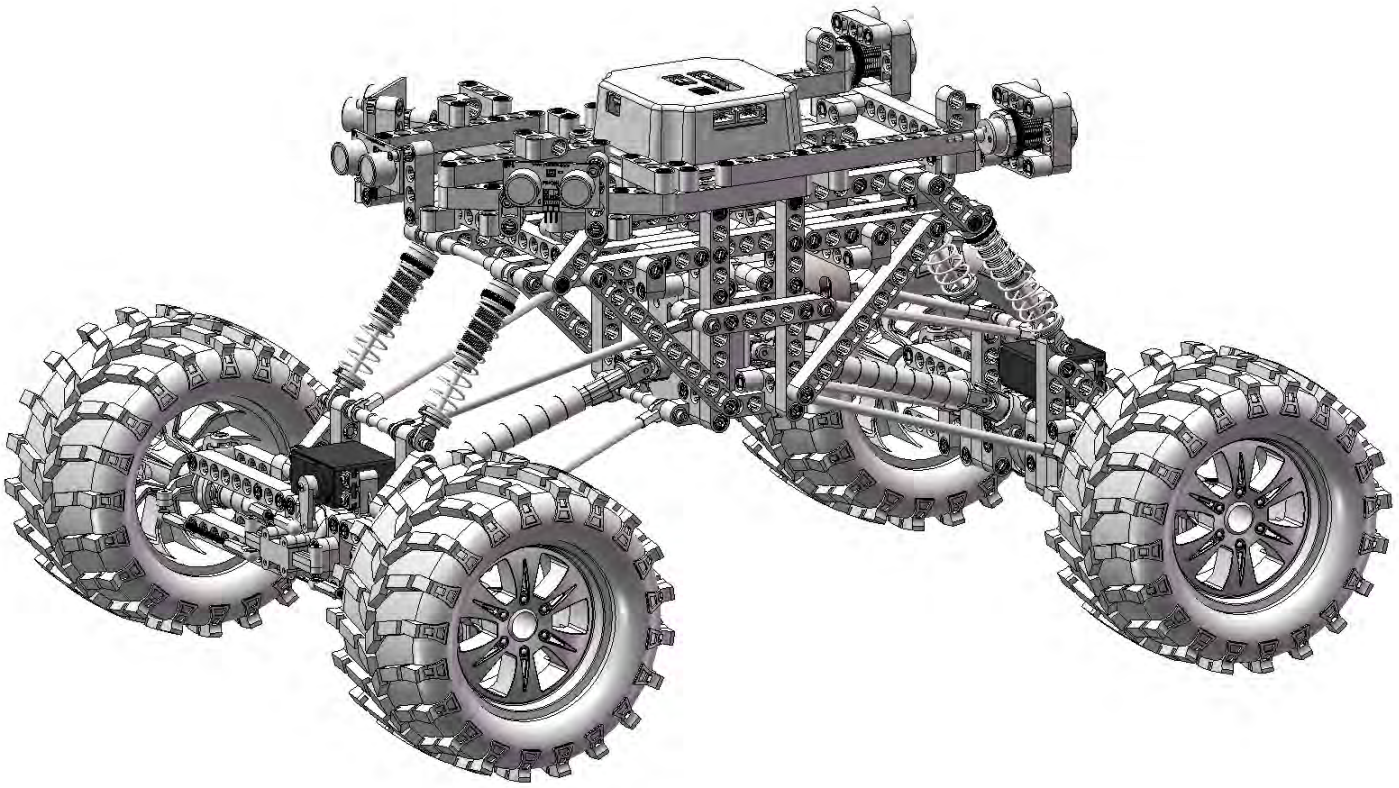
Example

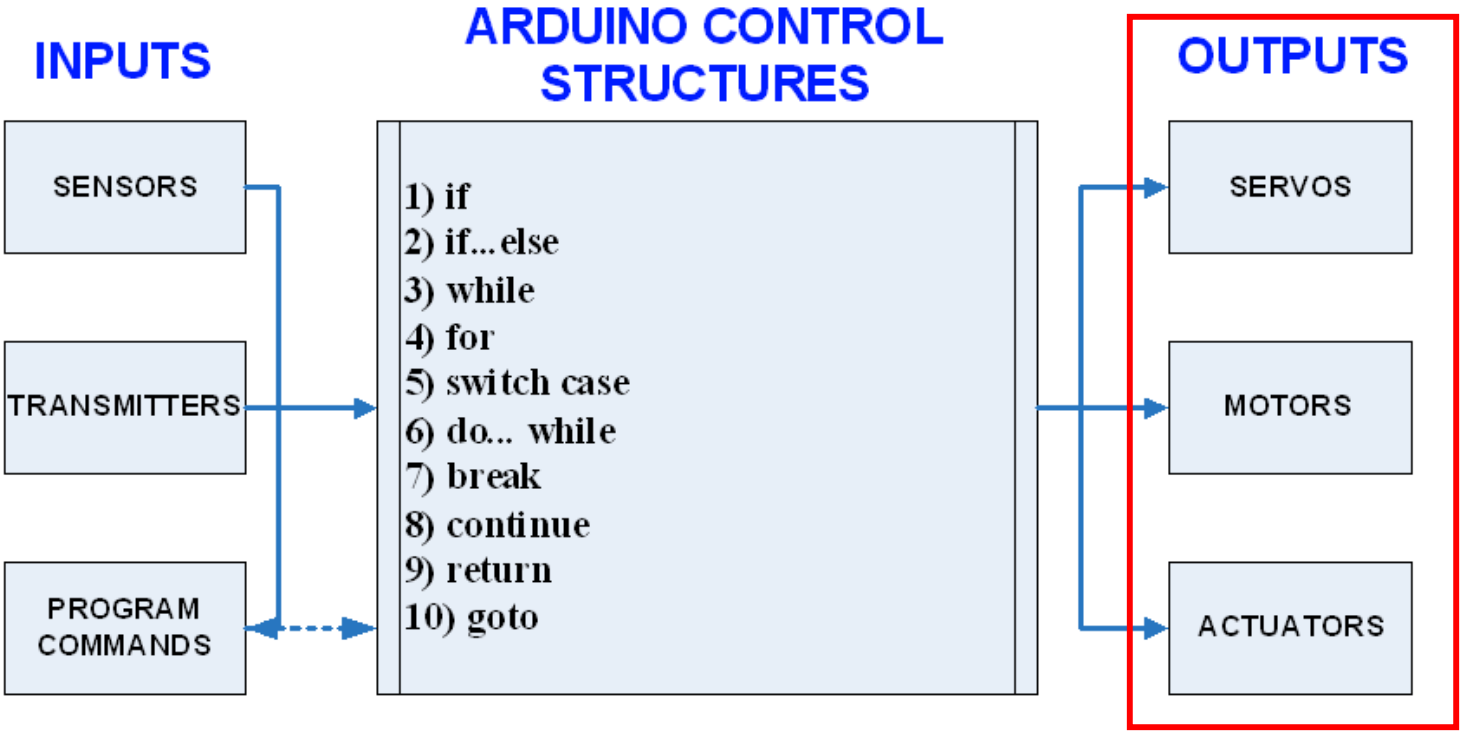
```
unsigned long time;

void setup( ){
  Serial.begin(9600);
}
void loop( ){
        Serial.print("Time: ");
        time = millis( );
        //prints time since program started
        Serial.println(time);
        // wait a second so as not to send massive amounts of data
        delay(1000);
}
```


HITEC

Outputs




INPUTS
ARDUINO CONTROL STRUCTURES
OUTPUTS
SENSORS
TRANSMITTERS
PROGRAM COMMANDS
1) if
2) if...else
3) while
4) for
5) switch case
6) do... while
7) break
8) continue
9) return
10) goto
SERVOS
MOTORS
ACTUATORS

Servo

Servos allow for specific motion control. A standard servo is capable of 180 degrees, 0 to 179. Some allow for continuous rotation. Servos also may be referred to as rotary actuators. Applications may include, steering, arms, legs, wheels and other components on a robot that involve motion.

In the code

Include the servo library:

```
#include <Servo.h>
```

Define the servo, reference the servo library by typing "Servo" and then write in the name that will be used to reference the servo in the code. I'll call it "myservo".

```
Servo myservo;
```

In the "void setup section" attach it to a pin on the Arduino, 0 to 13 or A0 to A5. I'll attach mine to digital pin 5.

```
void setup( )
{
        myservo.attach(5);
}
```

Set the value to send to the servo. We can use any value from 0 to 179. (See figure 1.1) Neutral or center is 90. You can figure what value centers your servo for yourself. To do that, write this in the "void loop" section:

```
void loop( )
{
        myservo.write(90); //neutral
}
```

To check if your code is correct press the verify button. 

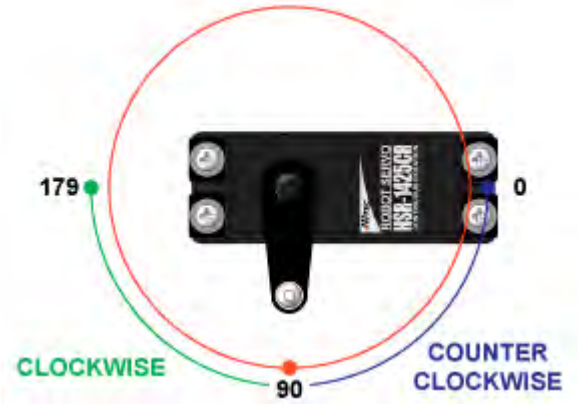
1.1 Challenge: **Center your servo.**

Open up the Arduino program, Go to File > Examples > 0. Minds-i > 1-Calibration > Servo. Plug your Arduino into your computer via USB and plug your servo into pin 5. Look for this line of code in the void loop section "myservo.write(90);" Change the value of 90 to any value from 0 to 179 to center your servo. The value should be close to 90. When you're done press upload. 

Mechanical hint: You can also change the direction your servo horn is mounted on your servo. Unscrew it, then try attaching your servo horn at a different angle. When using a servo for steering, a good test of whether your servo is center is when your robot will drive in a straight line.

Figure 1.1: Servo

Figure 1.2: Continuos Rotation Servo



Now that you know center, find the end points. Write the following in the "void loop" section, make corrections to the comments as needed:

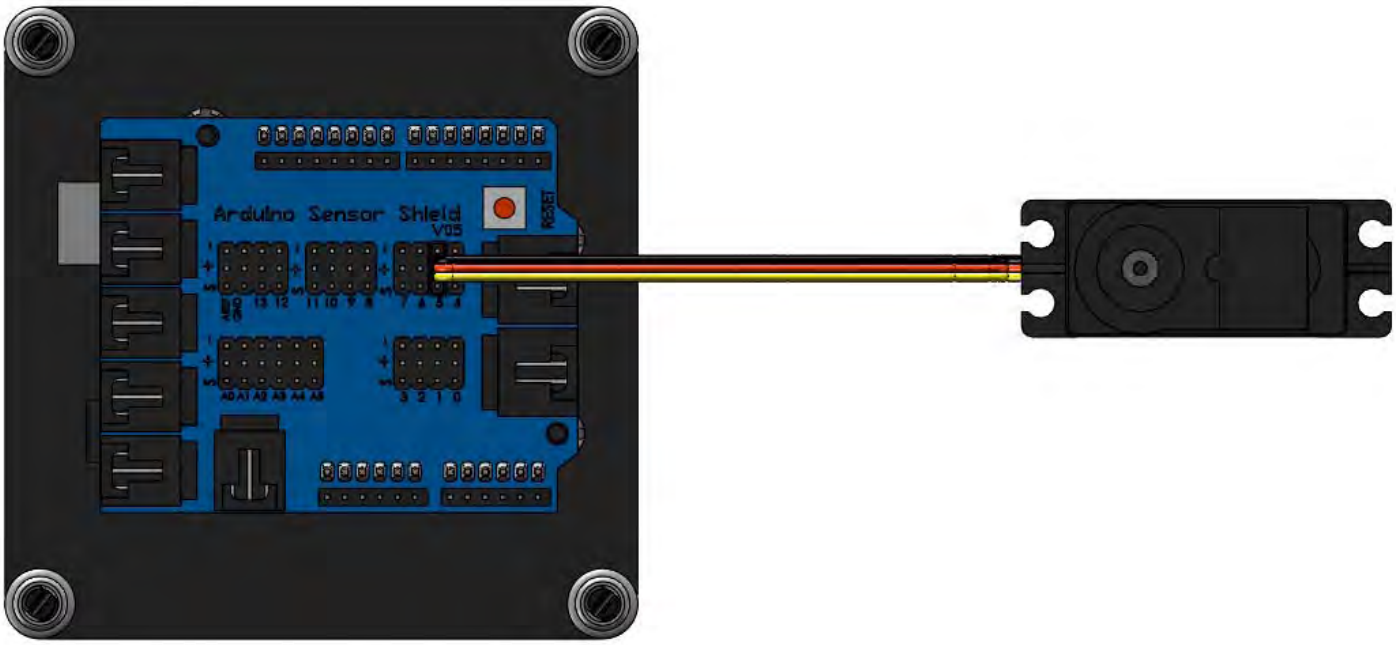
```
void loop()
{
        myservo.write(0); //left
        delay(1000);
        myservo.write(90); //neutral
        delay(1000);
        myservo.write(180); //right
        delay(1000);
}
```

1.2 Challenge Find Right and Left, Forward and Reverse

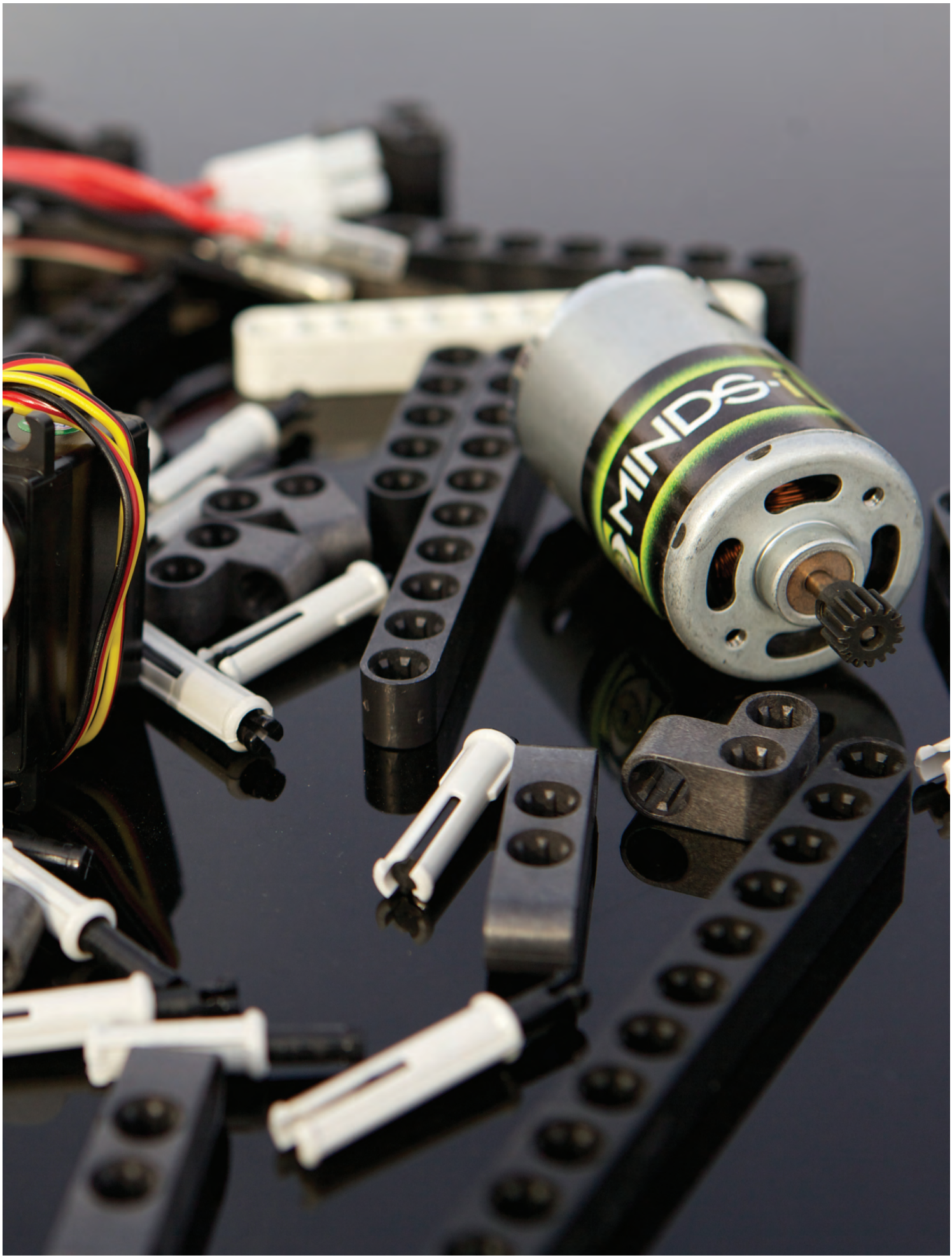
The value of right and left or forward and reverse can be different for each servo. It depends on how the servo is mounted on your robot and may even vary from manufacturers.

Open the Arduino program, Go to File > Examples > 0. Minds-i > 1-Calibration > Servo. Plug your Arduino into your computer via USB and plug your servo into pin 5. Look for this line of code in the void loop section "myservo.write(90);" Change the value of 90 to 0 or 179. Watch which direction your servo rotates or moves. Write a comment in the code using the "//" for future reference. ex // 179 = right. When you're done press upload.

Wiring Diagram for the Servo



Notes:


MINDS-I

ESC and Motor

The Electronic Speed Controller (ESC) is used to regulate voltage and current to a DC motor. It is similar to a continuous rotation servo in that entering values of 0 to 179 change the direction of the motor. However, the motor and ESC are for higher torque applications, and contain an electronic brake making them ideal in drive systems. With MINDS-i it will mostly be used as a drive motor, but other applications could be arms, winches, and turntables.

In the Code

Include the MINDS-i library and the Servo library:

```
#include <MINDSi.h>
#include <Servo.h>
```

Define the ESC, reference the servo library by typing “servo” and then write in the name that will be used to identify the ESC in the code. I’ll call it “motor”.

```
Servo motor;
```

In the “void setup” section, attach it to a pin on the Arduino, 0 to 13 or A0 to A5. I’ll attach mine to pin 4.

```
void setup()
{
        motor.attach(4);
}
```

Set the value to send to your ESC. We can use any value from 0 to 179, 90 being neutral. In the initial start up of the ESC, it will make tone meaning it is arming the speed controller, whatever value is sent to it initially from the microcontroller will be set a neutral. We want to set the value at 90 so the ESC will set that as neutral. We also need to add a 2 second delay (2000 milliseconds) to arm the speed controller.

```
Void setup ( );
{
        motor.attach(4);
        motor.write(90);
        delay (2000);
}
```

To check if your code is right click the verify button. 

1.1 Challenge: Setting Up Your ESC

Open up the Arduino Program, Go to File > Examples > 0.Minds-i >1.Calibration > ESC. Plug your arduino into your computer via USB and plug your ESC into pin 4. Plug the ESC into the motor. Plug the battery into the ESC. Turn "off" the ESC switch. In the "void setup" section of the code look for "motor.write(90);" This value can be set from 0 to 179. The ESC will set that as the value for neutral when it is armed. For example, if I plug in 90, it will be set as neutral. 0-89 will be reverse and 91 to 179 will be forward. Then Upload the code.  Unplug the USB cable and turn your ESC "on".

Helpful hint: In the top of your code in the above the void setup section define a new object as neutral and set the value for it.
Int neutral;
Now everywhere in the code that you use 90 to put your motor at neutral replace with "neutral"
Motor.write(90);
Motor.write(neutral);

Now that we set up the ESC, we can find forward, reverse and use the electronic brake. I'll assume that we are using a MINDS-i Robot with one drive motor, a steering servo, an Arduino and Sensor Shield. We'll control the robot autonomously through the Arduino and Sensor shield to drive forward, apply the electronic brake, and then drive in reverse.

Include the MINDS-i library and the Servo library:

```
#include <MINDSi.h>
#include <Servo.h>
```

Define the ESC, and steering servo, I'll use "drive" for the ESC and Motor and "steer" for the steering servo.

```
Servo drive, steer;
```

In the "void setup" section, attach the ESC and the Servo to pins on the Arduino, 0 to 13. I'll attach my ESC to pin 4 and my Servo to pin 5.

```
Void setup( )
{
        drive.attach(4);
        steer.attach(5);
```

We have to set both at 90, that will center our servo and set our ESC at neutral. We also need to add a delay for 2 seconds (2000 milliseconds) to allow the ESC to arm.

```
        drive.write(90);
        steer.write(90);

        delay(2000);
}
```

Now in the “void loop” section we’ll program the robot to drive forward one second, apply the brake, and then go in reverse for one second. It will continue to “loop” this process over and over.

Since we set our steering servo at 90 in the void setup it should be centered and drive in a straight line. (This may need additional adjusting to center your servo. See 1.1 Challenge: Center your servo)

As described above 0 to 89 is reverse 90 is neutral and 91 to 180 is forward. The Arduino accomplishes this by varying the signal it sends to the ESC. Setting the ESC at 100 will send your robot barley chugging along and 180 will send your robot of at full speed and most likely crash into the nearest object. As a rule of thumb I suggest starting at the lowest possible speed and then increasing it little by little.

The ESC uses an electronic brake that is activated by sending it into reverse after the robot is driving forward. The change from forward to instantly in reverse applies the electronic brake. It will then need to be sent a value for neutral to disengage the brakes, and then after it can be sent in reverse or forward.

Write the following in the “void loop” section. I’ll set the speed at 100, this is about as slow as the robot can go. (As the battery loses charges it will go slower). Then add a delay for 1 second. The robot will drive forward for one second.

```
Void loop( )
{
        drive.write(100); //drive forward one second
        delay(1000);
```

To apply the brake, write in the value for reverse(0), and add a half a second delay to give it time to engage. Then disengage the brakes by sending it a value of neutral, I’ll use 95 just to be on the safe side.

```
        drive.write(0); //brake and wait
        delay(500);
        drive.write(95); //disengage brakes
```

With the brakes disengaged now we can power it in reverse. Set the delay for 1500 milliseconds to drive in reverse for a second and a half.

```
        drive.write(80); //drive backward for one and a half seconds
        delay(1500);
}
```

To check if your code is right click the verify button. 

1.2 Challenge: Understanding Forward, Reverse, and the Electronic Brake.

Program your robot to drive forward, apply the brakes then go in reverse.

I'll assume you are using a MINDS-i robot with one drive motor and a steering servo. Plug your Arduino into your computer via USB. Plug your ESC into pin 4. Turn "off" the ESC switch. Plug your steering servo into pin 5.

Open up the Arduino program, Go to File > Examples > 0. Minds-i > 2-Application > ESC.

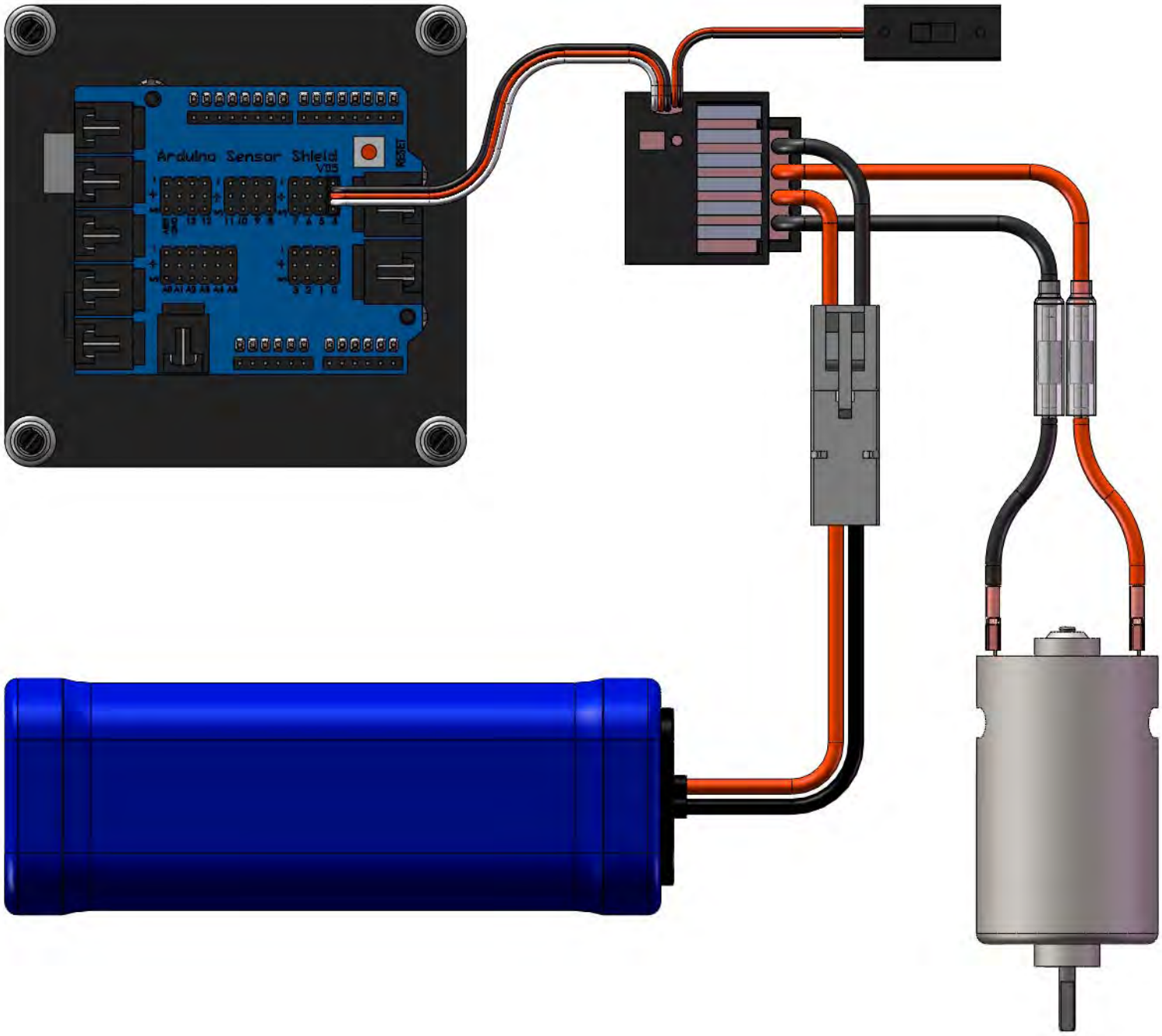
Make sure your ESC and servo are attached to their corresponding pins in the code "drive and steer". Upload the program.  Unplug the USB cable and Turn your ESC "on".

In the "void loop" section look for "motor.write(X);" This value can be set from 0 to 179. Change the value to see what your robot does. WARNING: 179 and 0 are full speed and fast. Start by increasing and decreasing the values a few numbers at a time. Change the delays as well to find to optimal program for your situation.

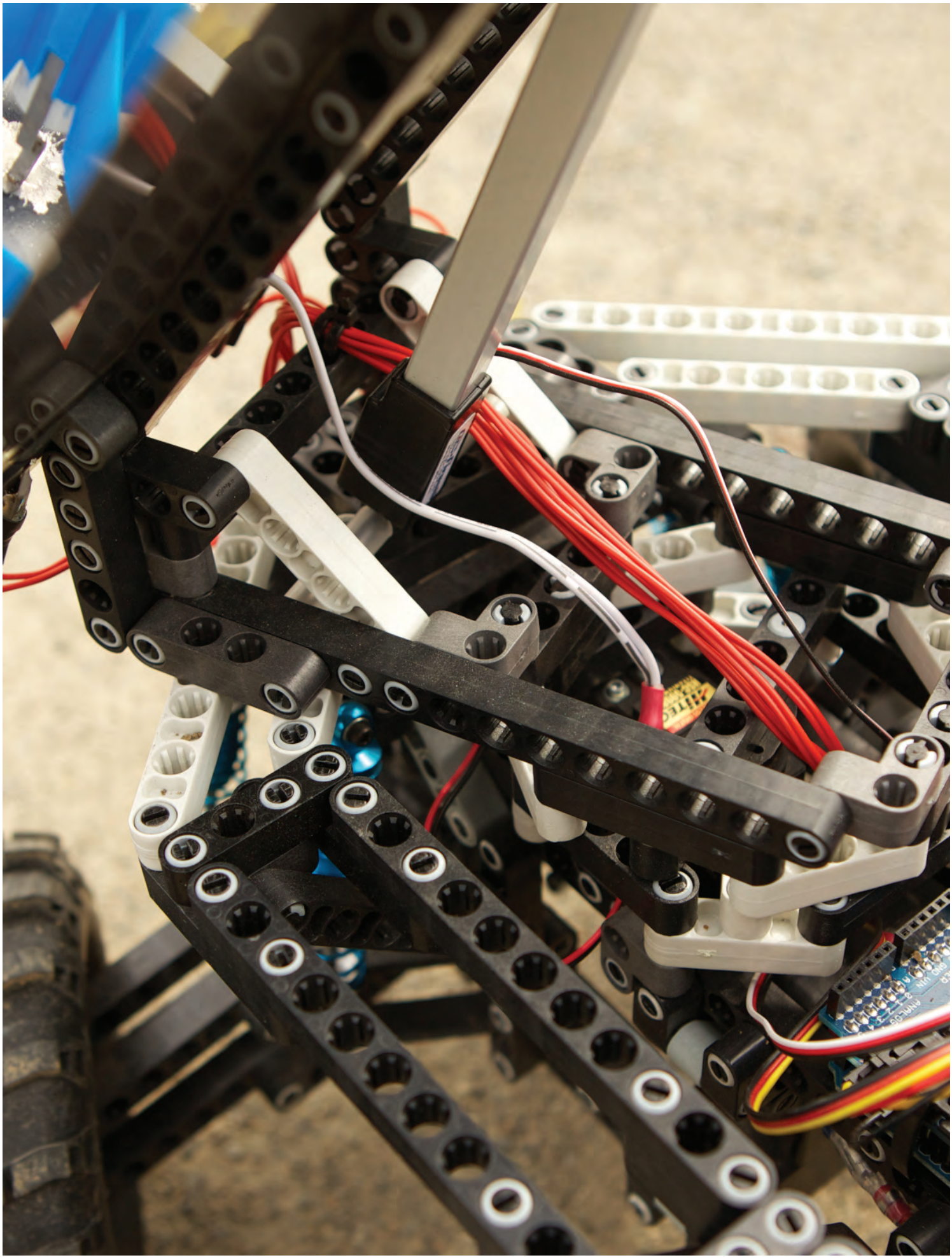
Helpful Hint: This ESC application code is also a good example code to check if your steering servo is center by seeing if your robot will drive in a straight line.

Additional Challenge: Using only the ESC, Servo, Arduino and Sensor shield, program your robot to drive from one end of the room to the other, maneuvering around objects as needed.

Wiring Diagram for the ESC and Motor



Notes:_____



Linear Actuator

Linear Actuators, like servos, allow for specific motion control. They come in different stroke lengths, speeds and many other options too. They may be used in a wide range of applications arms, legs, and other components on a robot that involve motion.

In the code

Include the servo library:

```
#include <Servo.h>
```

Define the linear actuator, reference the servo library by typing “Servo” and then write in the name that will be used to reference the linear actuator in the code. I’ll call it “myLinearActuator”.

```
Servo myLinearActuator;
```

In the “Void Setup section” attach it to a pin on the Arduino, 0 to 13 or A0 to A5. I’ll attach mine to digital pin 5.

```
void setup( )
{
        myLinearActuator (5);
}
```

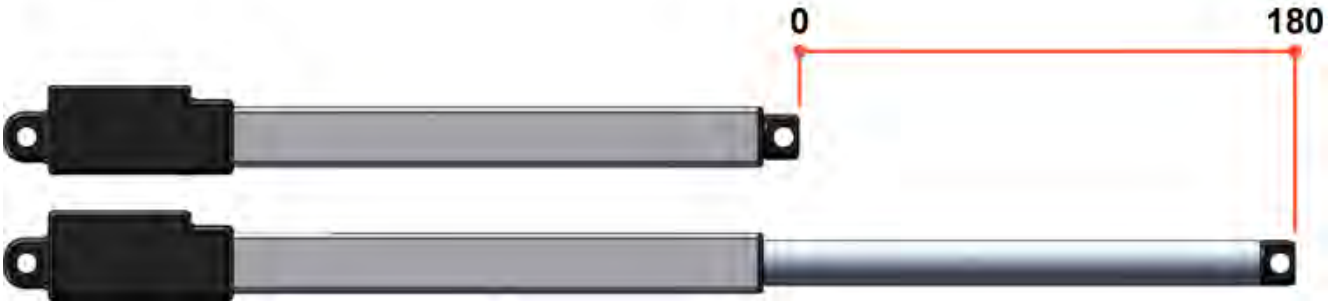
Set the value to send to the linear actuator. We can use any value from 0 to 180. (See figure 1.1) loop”

```
void loop( )
{
        myservo.write(90);
}
```

To check if your code is correct press the verify button.  Change the values to see how the linear actuator reacts. Try something like this: put a delay between each line to allow enough time for it to perform the full range of motion:

```
void loop( )
{
        myservo.write(0); //In
        delay(2000);
        myservo.write(90); //Middle
        delay(2000);
        myservo.write(180); //Out
        delay(2000);
}
```

Figure 1.1: Linear Actuator

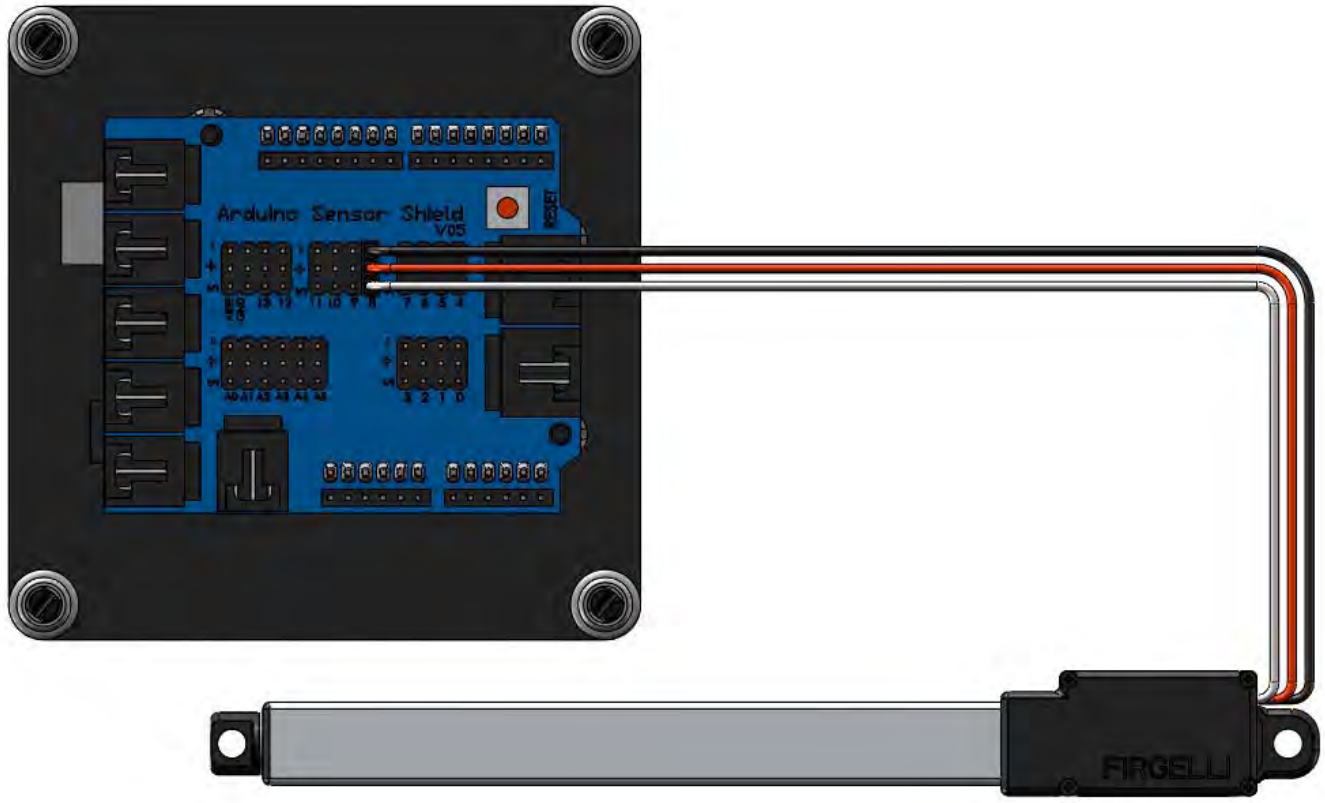


1.1 Challenge: **Using Your Linear Actuator**

Open the Arduino program, Go to File > Examples > 0. Minds-i > 1-Calibration > Servo. Plug your Arduino into your computer via USB and plug your linear actuator into pin 5. Look for this line of code in the void loop section "myservo.write(90);" Change the value of 90 to any value from 0 to 180 to adjust the linear actuator in and out. Change all references of "myservo" to "myLinearActuator". Make comments to yourself using the // to mark the "in", "middle", and "out" positions.

Notes:

Wiring Diagram for the Linear Actuator



Notes:

LED (LIGHT-EMITTING DIODE):
IS A SEMICONDUCTOR LIGHT SOURCE. LEDs ARE USED AS INDICATOR LAMPS IN MANY DEVICES AND ARE INCREASINGLY USED FOR OTHER LIGHTING.

VIBRATORY MICRO-MOTORS:
USED AS NON VISUAL INDICATORS IN MANY ELECTRONIC DEVISES SUCH AS CELLULAR PHONES OR HANDHELD GAMING DEVICES.

ELECTRONIC RELAY:
A RELAY IS AN ELECTRICALLY OPERATED SWITCH USED TO CONTROL HIGH CURRENT OR VOLTAGE WITH A LOW CURRENT LOW VOLTAGE INPUT.

Digital Outputs

A digital output is a device that is turned on or off. Most commonly an LED used as an indicator light. If you need to use a relay to control a higher amp device it will be a digital output. You can also control motors directly from the arduino if they are less than 40mA.

In the code

In the "void setup" section we will need to assign the "pinMode" as an "OUTPUT" and declare the pin number. I will use pin 12.

```
void setup( )
{
        pinMode(12, OUTPUT);
}
```

The Digital Output may be plugged into ports 0 through 13. I'll attach mine to pin 12. In the "void loop" section we will need to assign our pin "HIGH" to turn the device "on" or "LOW" to turn it "off". I will use "HIGH" to turn the device on.

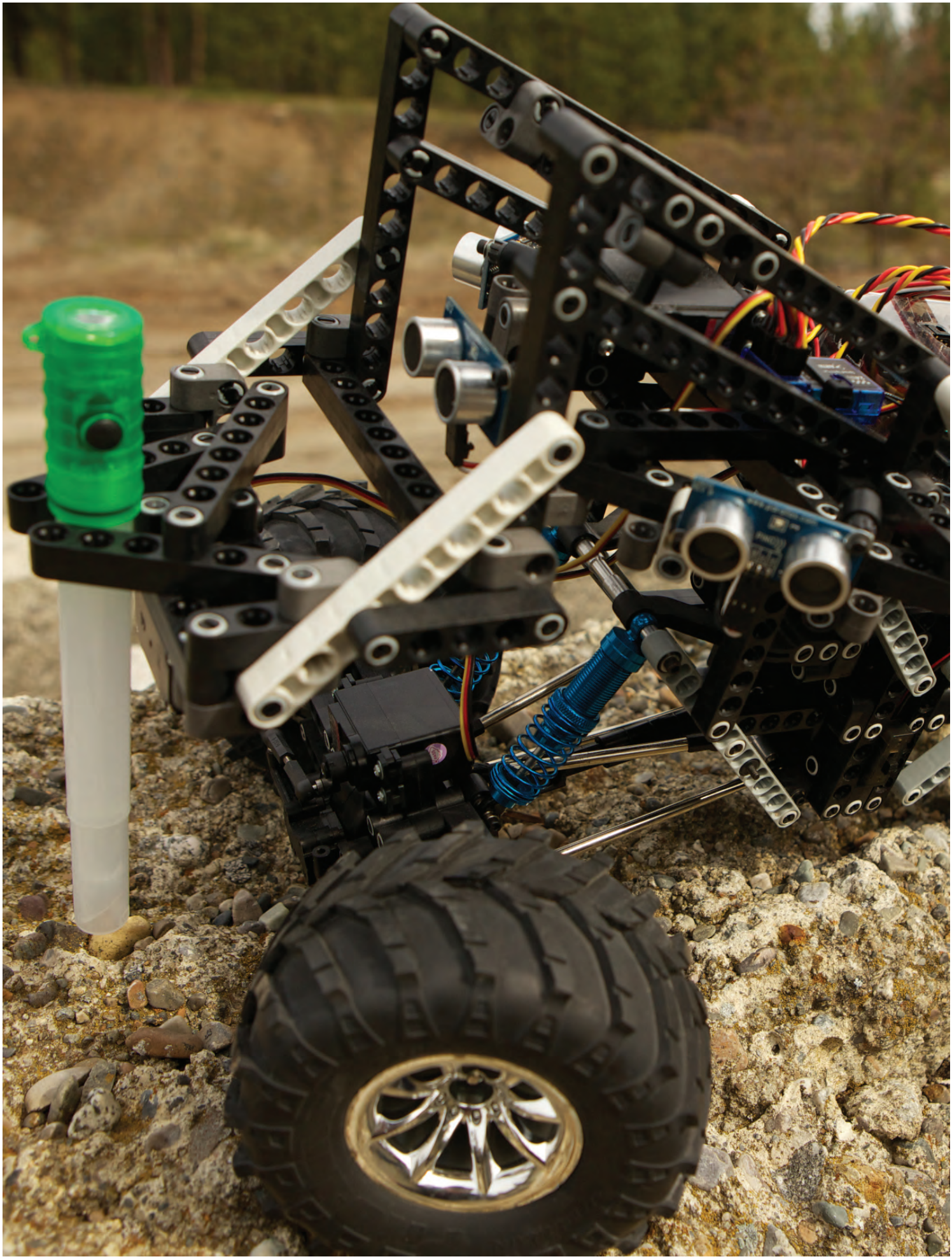
```
void loop( )
{
        digitalWrite(12, HIGH);
}
```

To check if your code is correct press the verify button. 

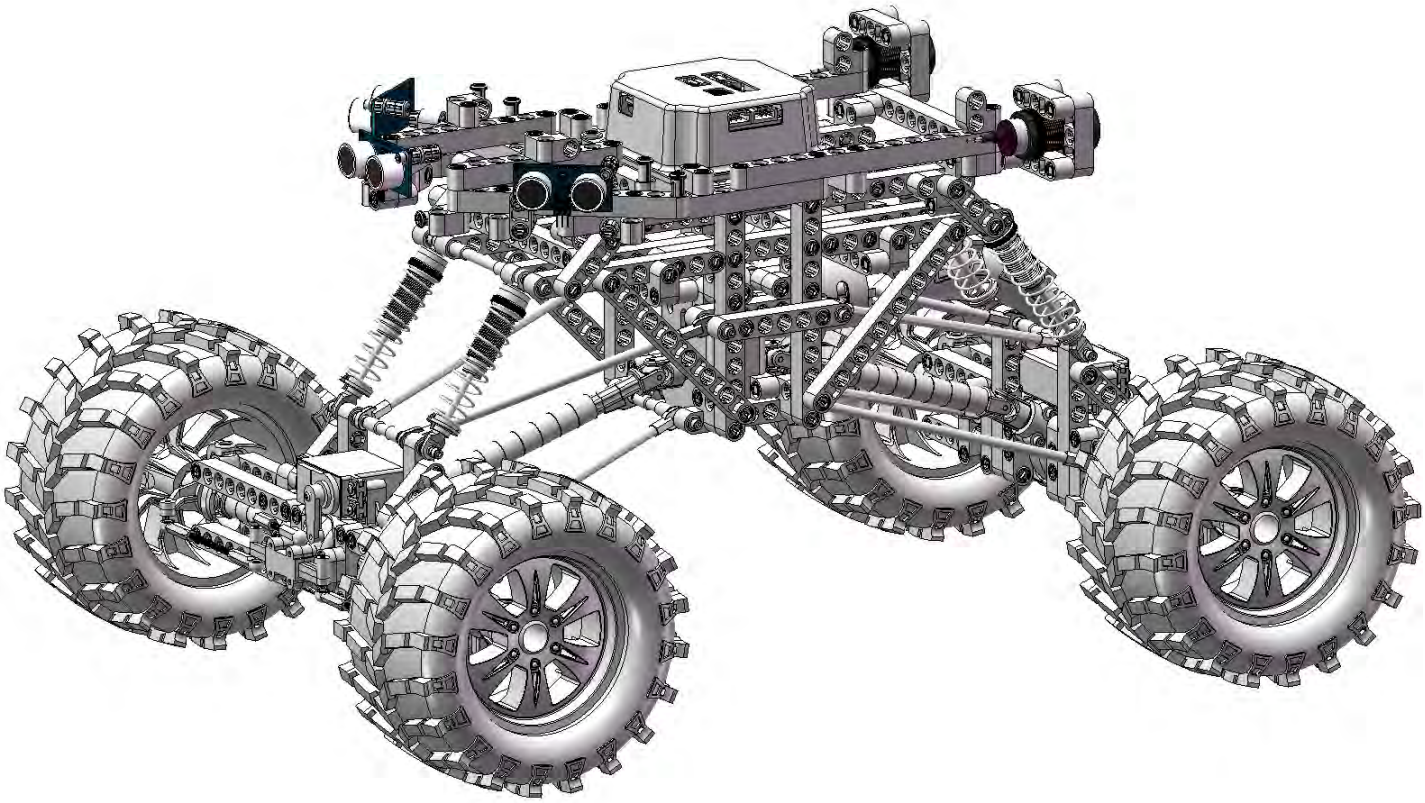
1.1 Challenge: **Turn your devise on**

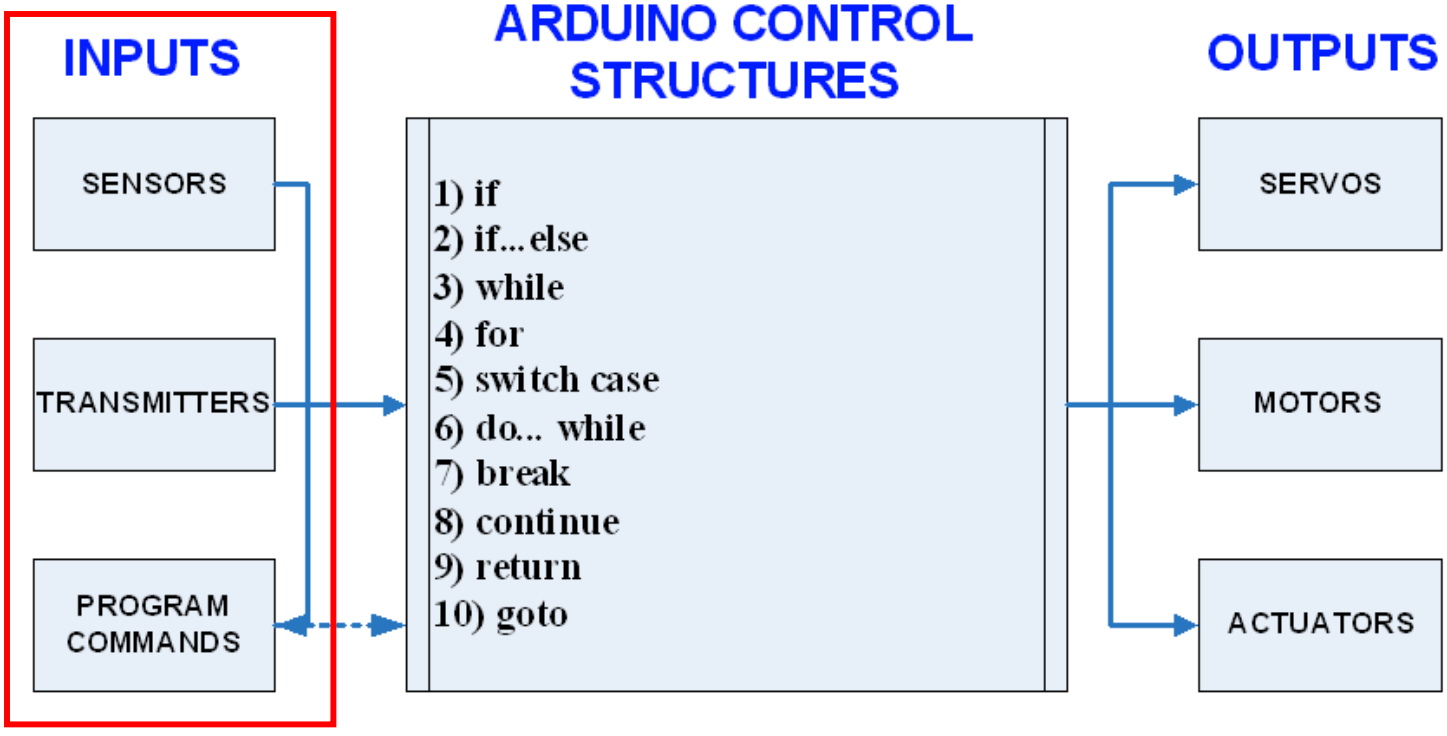
Open the Arduino program, Go to File > Examples > 0. Minds-i > 1-Applications > Digital_Outputs. Plug your Arduino into your computer via USB and plug your device into pin 12. When you're done press upload. 

Mechanical hint: You will need to make sure you have your device properly wired to activate it.



Inputs




INPUTS
SENSORS
TRANSMITTERS
PROGRAM COMMANDS
ARDUINO CONTROL STRUCTURES
1) if
2) if...else
3) while
4) for
5) switch case
6) do... while
7) break
8) continue
9) return
10) goto
OUTPUTS
SERVOS
MOTORS
ACTUATORS


venom
WWW.VENOM-GROUP.COM

Radio Transmitter

A radio transmitter or more commonly known as a Radio Control (RC), sends a signal from the transmitter (the controller) to the receiver. There are many different types of transmitters ranging from Am and Fm that use crystals, to 2.4 Ghz that sends a more powerful signal.

In the code

In this code we'll take a look at how to use the Arduino with the radio transmitter. We'll figure out how to get a reading from the radio and see how it can be used in the programming.

Include the MINDS-i library:

```
#include <MINDSi.h>
```

Define an integer "int" called "val". It will be used to store the value that will come from the radio.

```
int val;
```

In the "void setup" section, start a serial connection with the computer.

```
void setup( )
{
        Serial.begin(9600); //start a serial connection
}
```

In the "void loop" section get the radio signal using the "getRadio( )" function from the MINDS-i library. I'll plug my receiver into pin 2 on the Arduino. There are multiple channels on a radio so I'll pick one I want to get a reading from (Channel 1) and run a servo cable from the receiver channel 1 to pin 2. I'll assign getRadio to the value (val) integer I defined earlier. Then I use the serial command "Serial.println(val)". This will show the value from the radio in the Serial Monitor.

```
void loop( )
{
        val = getRadio(2);  //read the value being sent on pin 2
        Serial.println(val);  //send a string or value on the serial connection
}
```

To check if your code is correct press the verify button. 

1.1 Challenge: **Get Readings from your Transmitter.**

Open the Arduino program, Go to File > Examples > 0. Minds-i > 1-Calibration > RadioTest. Plug your Arduino into your computer via USB and plug a cable from channel 1 on the receiver to pin 2 on the Arduino. Upload the program and when it is done uploading open the Serial monitor. Move the control on the radio corresponding to channel 1 and watch the value change. The value should go from 0-179, if it does not adjust the trim on your radio.

Pistol Grip 2 Channel

In this example I'll be using a MINDS-i robot with one servo and one motor, I'll also need two extra servo cables, my receiver, and controller. I'll cover how to control the robot manually via Radio Control.

Include the MINDS-i and the Servo library. Define the ESC and Servo objects, “drive” and “steer”. In the “void setup” section attach the ESC to pin 4 and the servo to pin 5. Then set them at neutral and center (90), and add a delay to arm them.

```
#include <MINDSi.h>
#include <Servo.h>

Servo drive, steer;

void setup( )
{
        drive.attach(4); //set a pin for the ESC
        steer.attach(5); //steering servo

        drive.write(90); //set the output for the ESC/servo
        steer.write(90);

        delay(2000); //delay 2 second for arming
}
```

In the “void loop” section we’ll use getRadio and set the corresponding pins. Plug two cables from channel 1 and 2 into Pins 2 and 3 on the Arduino. Set the value that the radio gets to send that value to the ESC and servo, by using drive.write( ).

```
void loop( )
{
        drive.write( getRadio(2) );  //sets the ESC to the inbound radio value
        steer.write( getRadio(3) );  //sets the Servo to the inbound radio value
}
```

To check if your code is correct press the verify button. 

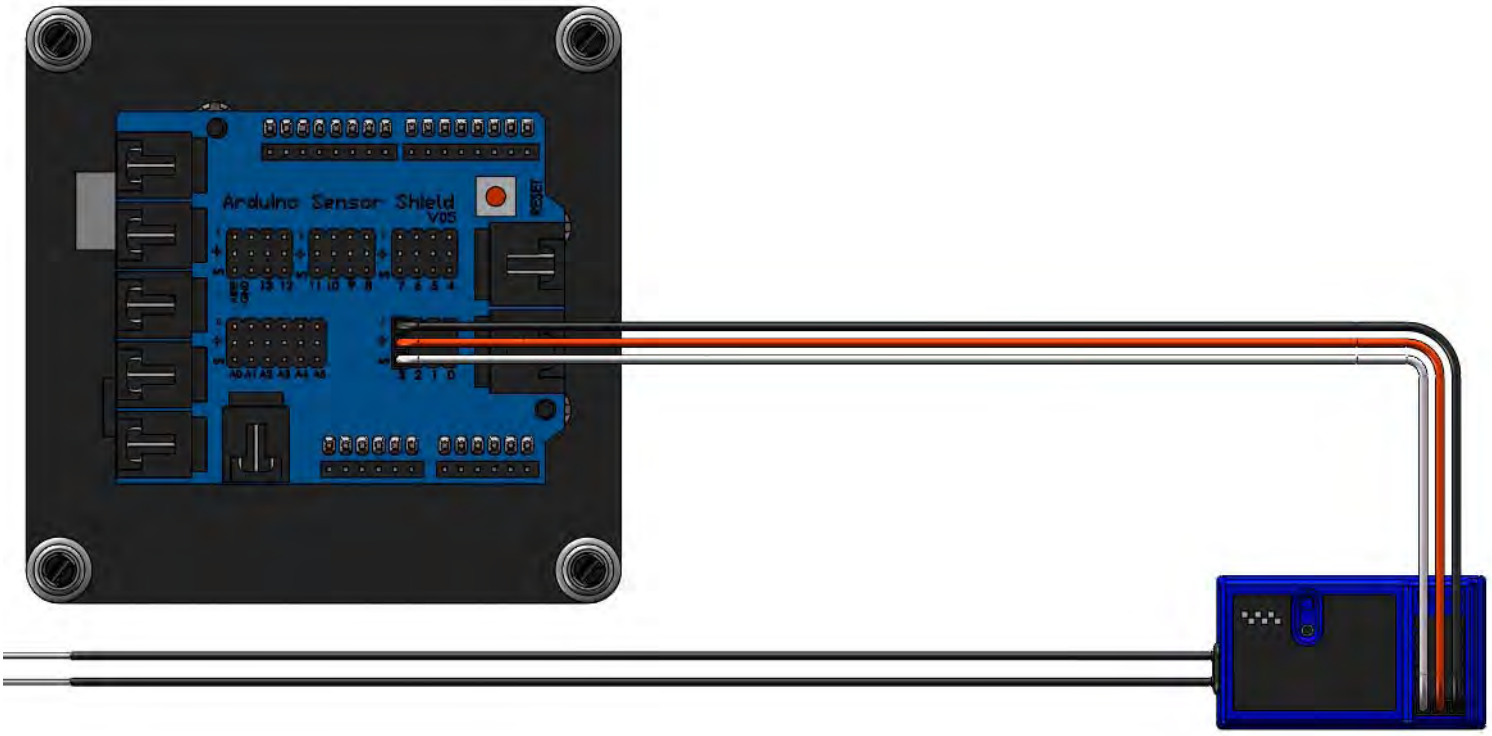
1.2 Challenge: Manual Control

Open the Arduino program, Go to File > Examples > 0. Minds-i > 1-Applications > SimpleRadioDrive. Plug your Arduino into your computer via USB, plug the ESC in Pin 4, the servo in Pin 5, and plug channel 1 and 2 on the receiver to pins 2 and 3 on the Arduino. Upload the code.  Turn your controller on first, then turn your robot on. (This prevents the robot from running away.) Your robot is now in manual drive.

Trouble? Make sure your battery has a full charge and the pins are plugged in correctly. If your still having issues see the Troubleshooting guide section.

Notes:

Wiring Diagram for the Radio Transmitter



Notes:

Push Button

Push Buttons allow physical interactions to be detected by a micro-controller. Most commonly they are used for human interactions and interfaces. They can also be used as limit switches to detect travel of an object. Push Buttons come in a couple common configurations, normally open, or normally closed with either a momentary or standard action.

In the code

In the "void setup" section we will start serial communications by typing "Serial.begin(9600);" Then we will define pin 12 as an input by typing "pinMode(12, INPUT);".

```
void setup( )
{
        Serial.begin(9600);
        PinMode(12, INPUT);
}
```

In the "void loop" section we will print the switch state to the serial monitor by typing "Serial.println(digitalRead(12));

```
void loop( )
{
        Serial.println(digitalRead(12) );
}
```

To check if your code is correct press the verify button. 

1.1 Challenge: Turn on the pin 13 led.

Open the Arduino program, Go to File > Examples > 0. Minds-i > 1-Calibration > Push Button. Plug your Arduino into your computer via USB and plug your Push Button into pin 12.

Now see what value is displayed by opening the "serial monitor"  . Make note of the value displayed when the button is being pressed _____ versus not pressed _____.

Now that you know how to get the switch state we will have you turn on and off a LED with the Push Button. In the “void setup” section we will define pin 12 as an “INPUT” and pin 13 as an “OUTPUT” by typing “pinMode(12, INPUT); and on the next line pinMode(13, OUTPUT).

```
void setup( )
{
        pinMode(12, INPUT);
        pinMode(13, OUTPUT);
}
```

In the “void loop” section we will set the switch input to the LED output by typing “digitalWrite(13, digitalRead(12));” In other words pushing the switch (input) to turning the LED on (output)

```
void loop( )
{
        digitalWrite(13, digitalRead(12) );
}
```

To check if your code is correct press the verify button. 

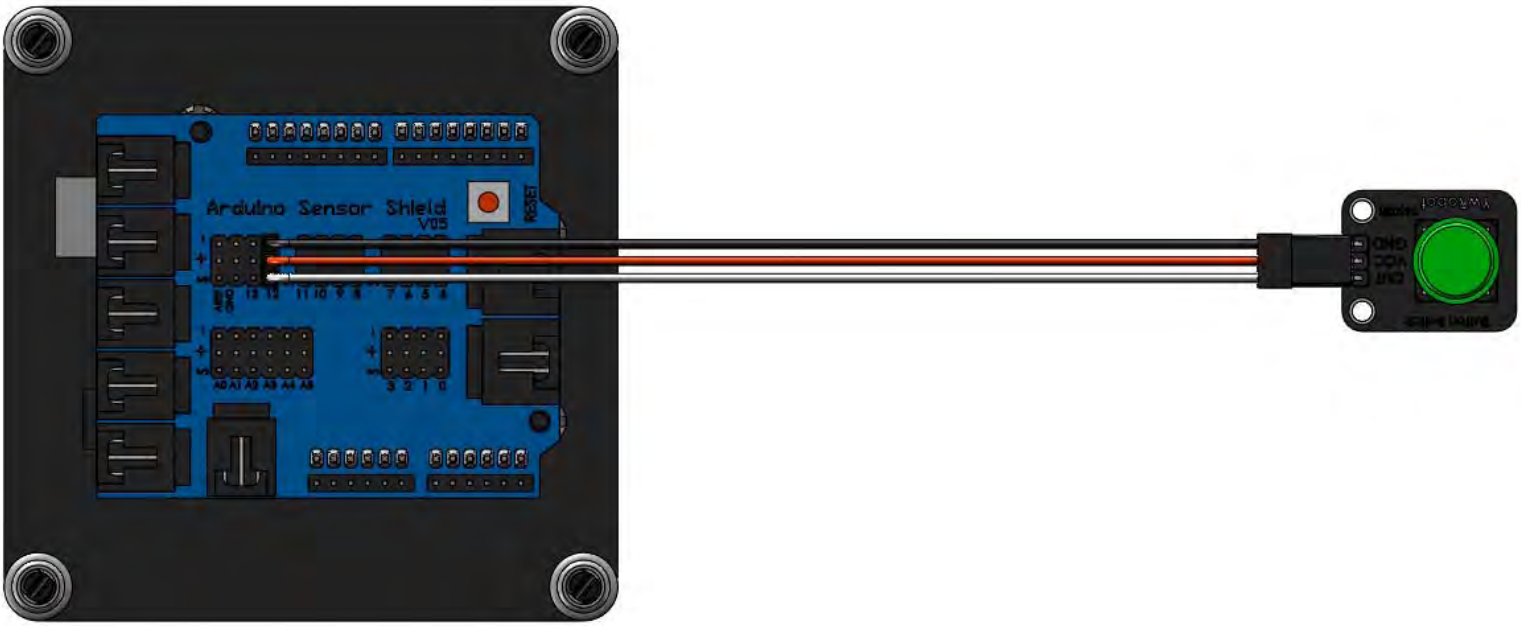
1.2 Challenge: Light up the pin 13 led

Open the Arduino program, Go to File > Examples > 0. Minds-i > 1-Application > Push Button. Plug your Arduino into your computer via USB and plug your Push Button into pin 12.

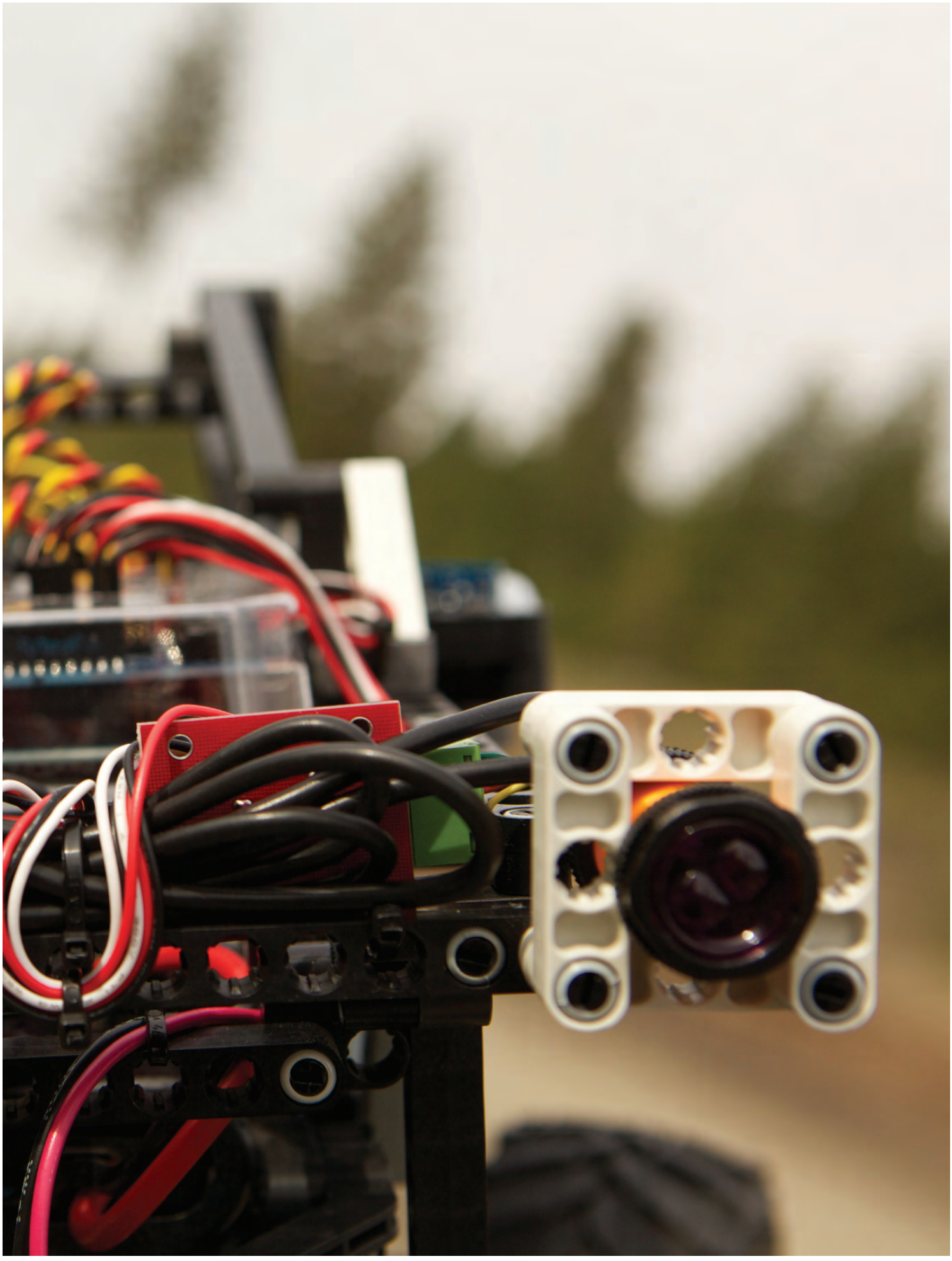
When you’re done press upload.  What happens when you press the Button? What type of switch is it? A momentary or standard, is it normally open or closed?

Mechanical hint: You will need to look for the orange LED in between the Arduino and the Shield, the LED is next to pin 13, it is marked by the letter “L” next to it.

Wiring Diagram for the Push Button



Notes:



Distance IR Sensor

The Infrared Reflectance Sensor Module carries a single infrared LED and phototransistor pair, along with modulation/demodulation electronics, in an inexpensive, small module that can be mounted almost anywhere and is great for obstacle detection of robot and home alert system. The optimal sensing distance is within 80cm (30 inches). This sensor has a screwdriver adjustment to set the desired distance, then gives a digital output when it detects something within that range. However it does not return a distance value like the Ping Sensor.

In the code

Before we can use our IR sensor we need to know what readings we can get from it. MINDS-i has an easy calibration program to help you get a reading from the sensor.

Define an integer "int" called "val". It will be used to store the value that will come from the radio.

```
int val;
```

In the "void setup" section we'll need to establish a serial connection between the computer and the Arduino through the USB cable. The connection should be set at 9600 baud (9600 bit/second). This allows us to send information to the serial monitor.

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

To read the value that the sensor is sending use "digitalRead" and to store that number assign it to the integer we defined earlier "val". Set the pin that the IR sensor is plugged into, I'll use 13. Then write "Serial.println(val);" to show the value in the serial monitor.

```
void loop( )
{
        val = digitalRead(13);  //read in the digital value on pin 13
        Serial.println(val);  //send a string or value on the serial connection
}
```

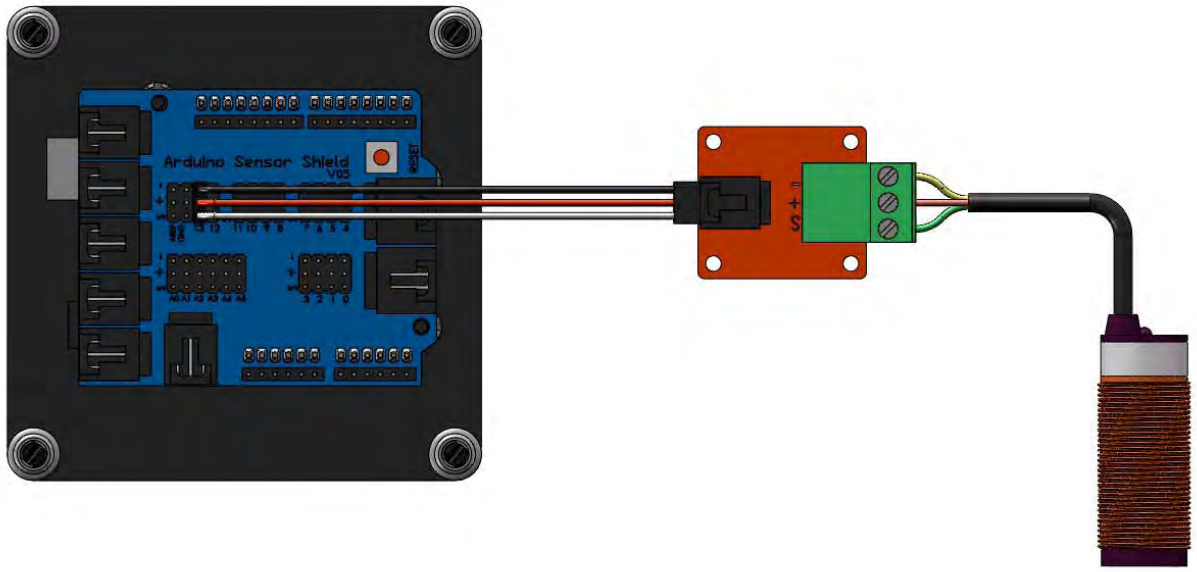
To check if your code is correct press the verify button. 

1.1 Challenge: **Calibrate Your IR Distance Sensor**

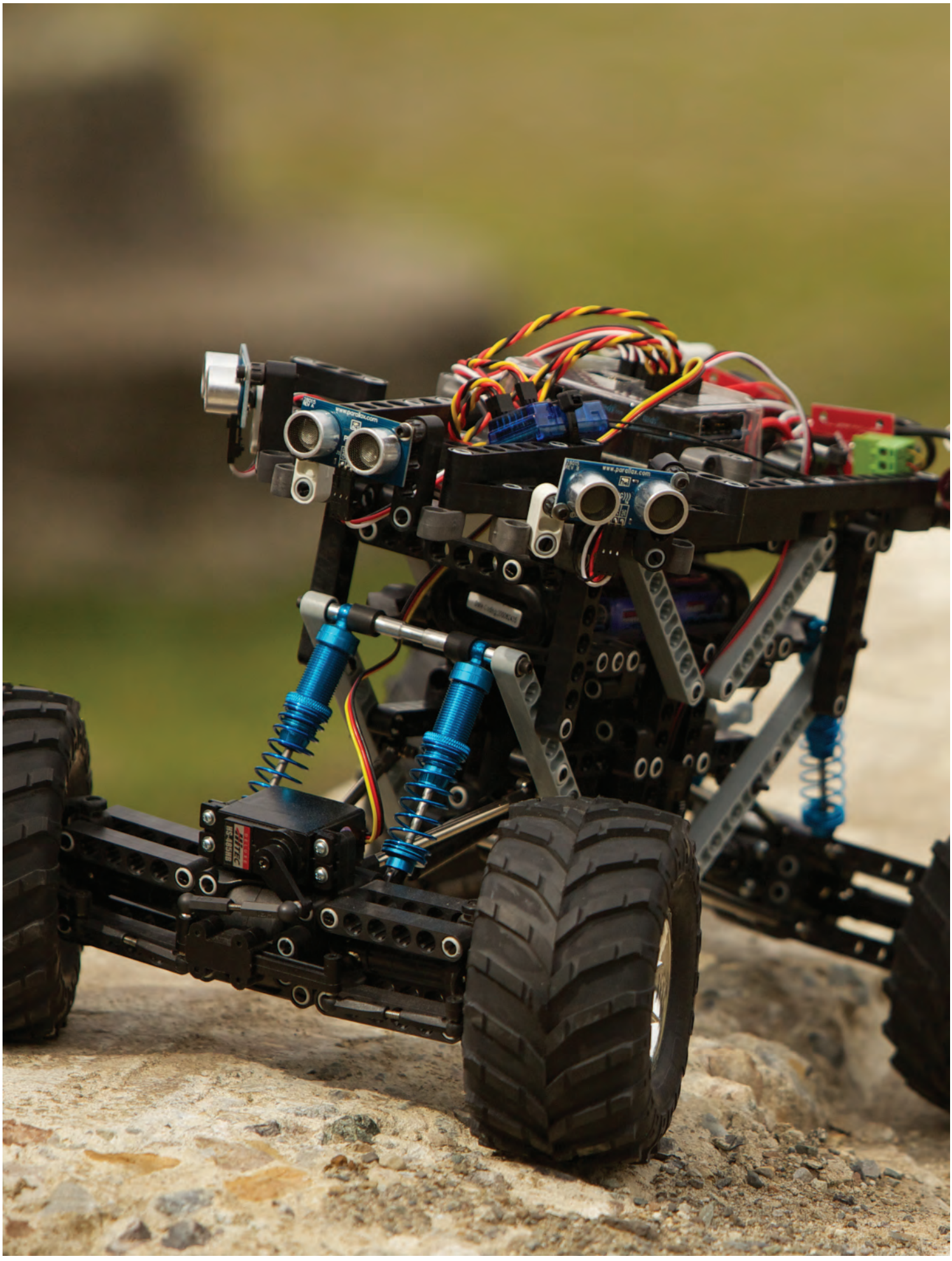
Open the Arduino program, Go to File > Examples > 0. Minds-i > 1-Calibration > DigitalSensor. Plug your Arduino into your computer via USB and plug your IR sensor into pin 13. Upload the code  and open the serial monitor  . To change the distance at which the sensor detects an object, adjust the screw on the back side of the sensor.

Notes:

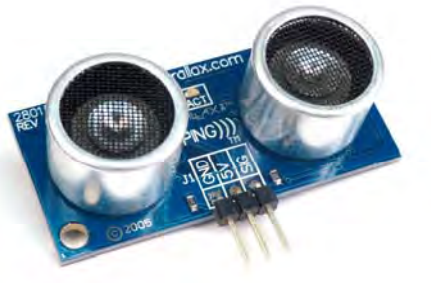
Wiring diagram for the Ir Distance Sensor



Notes:



Ultrasound Sensor



Parallax's PING)))™ ultrasonic sensor is an easy method of distance measurement and autonomous obstacle detection and avoidance. The Ping sensor measures distance using sonar; an ultrasonic (well above human hearing) pulse is transmitted from the unit and distance-to-target is determined by measuring the time required for the echo return.

The Ping sensor provides precise, non-contact distance measurements within a 2 cm to 3 m range. Burst indicator LED shows measurement in progress. 3-pin header makes it easy to connect using a servo extension cable.

In the code

Before we can use our Ping sensor we need to know what readings we can get from it. MINDS-i has an easy calibration program to help you get a reading from the ping sensor. Because we know the speed of sound, the theoretical value of one foot is about 1700 milliseconds.

Include the MINDS-i library:

```
#include <MINDSi.h>
```

In the “void setup” section we’ll need to establish a serial connection between the computer and the Arduino through the USB cable. The connection should be set at 9600 baud (9600 bit/second). This allows us to send information to the serial monitor.

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

Use the serial command to print the line to the serial monitor. Reference the Ping sensor by using the command “getPing”, this is provided by the MINDS-i library, which makes it easier to control the Ping sensor by handling the complicated code. Set the Pin, I’ll pick pin 7.

```
void loop( )
{
        Serial.println(   getPing(7)   );
}
```

To check if your code is correct press the verify button. 

1.1 Challenge: **Calibrate Your Ping Sensor**

Open the Arduino program, Go to File > Examples > 0. Minds-i > 1-Calibration > PingSensor. Plug your Arduino into your computer via USB and plug your ping sensor into pin 7 on the sensor shield. Look for this line of code in the "void loop" section "Serial.println(getPing(7));" The "7" indicates the pin the sensor is plugged in to. Pins 0-13 will work. We'll use pin 7. Upload the code.  You will see that the Ping Sensor LED lights up meaning the sensor is working. Open the "Serial Monitor"  . This shows us the value the sensor is reading in milliseconds. When you move an object in front of the sensor the value changes, it increases as the object is further away. Use a tape measure or a yardstick to mark the values. Then make a note to yourself in the code. (see figure 1.1 for reference of the Ping sensors' range.)

example.

```
Serial.println( getPing(7) ); // front ping sensor, 12 inches =  1800 milliseconds
                              //  right ping sensor, 12 inches = 1600 milliseconds
                              //  left ping sensor, 12 inches = ?
```

Now that the Ping Sensors are calibrated let's write a code for object detection. I'll assume that we are using a MINDS-i Robot with one drive motor, a steering servo, an Arduino and Sensor Shield and now our Ping sensor on the front of the robot. (refer to the Arduino Autonomous Upgrade Module on how to mount the Ping Sensor, steps 10-12) We'll control the robot autonomously through the Arduino to drive forward, detect an object and stop.

Include the MINDS-i library and the Servo Library:

```
#include <MINDSi.h>
#include <Servo.h>
```

Define the ESC as "drive" and the steering servo as "steer". In the "void setup" section, attach the ESC to pin 4 and the Servo to pin 5. Set both at 90 to set the ESC at neutral and center the servo. Add a delay for 2 seconds to arm the ESC.

```
Servo drive, steer;

void setup( )
{
        drive.attach(4);
        steer.attach(5);

        drive.write(90);
        steer.write(90);
        delay(2000);
```

Instead of starting and driving straight let's add something different by having it turn to begin with. The servo will always stay at last position that it was given in the code until a new command is given. Don't forget your semicolon (;) and end bracket }

```
        steer.write(179);
}
```

In the "void loop" section, we'll write an if /else statement. We'll need to write it so that the Ping sensor detects an object at a distance and stops before it hits the object. You'll want to figure out on your own the stopping distance of the Robot compared to the speed and the distance from the object that the Ping sensor will read. For brevity, I'll use 500 as the value the ping sensor reads to trigger it to stop. I'll attach the Ping sensor to pin 7.

```
void loop()
{
        if( getPing(7) < 500 )
```

Now that we have our condition, we need to write what it will do. We want it to stop, I'll set it at neutral to coast to a stop.

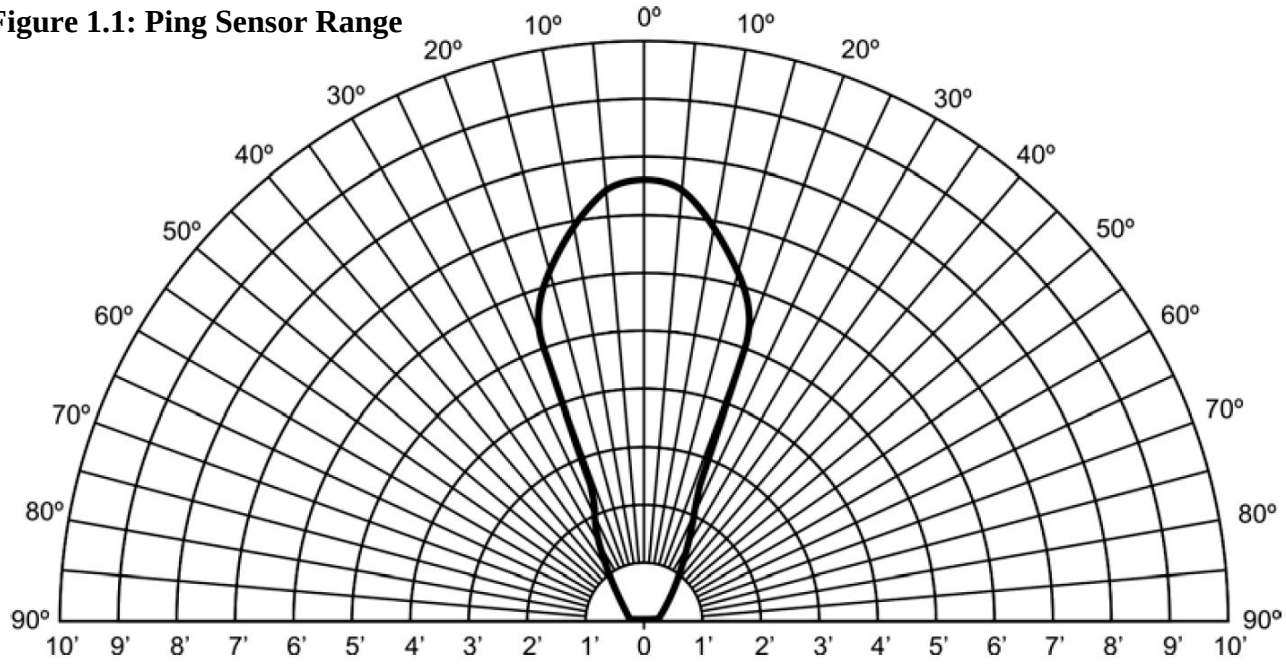
```
        drive.write(90);
```

The next part of the if/else statement is… else. We need to write what it will do if it does not see an object.

```
else
        drive.write(100);
}
```

To check if your code is right click the verify button. 

Figure 1.1: Ping Sensor Range



1.2 Challenge: Ultrasound Object Detection

I'll assume you are using your MINDS-i robot with one drive motor and one steering servo.

Open the Arduino program, Go to File > Examples > 0. Minds-i > 2-Application > PingSensor. Plug your Arduino into your computer via USB and plug your ping sensor into pin 7 on the sensor shield. Look for this line of code in the "void loop" section "if(getPing(10) < 500)" The number in parenthesis (10) after the getPing designates the pin the sensor is attached to. Make the change in the code to match your wiring. Upload the code. 

Change the sensor detection value (500). What happens if that number is 250? What happens if that number is 5000?

```
if( getPing(7) < 250 ) // Did it crash into the wall?
```

or

```
if( getPing(7) < 5000) // What will it do?
```

With the sensor detection value at 5000, change the speed of your robot (drive.write(100);) Increase the number by 5 each time and see what it does. Make a note to yourself using the comments //.

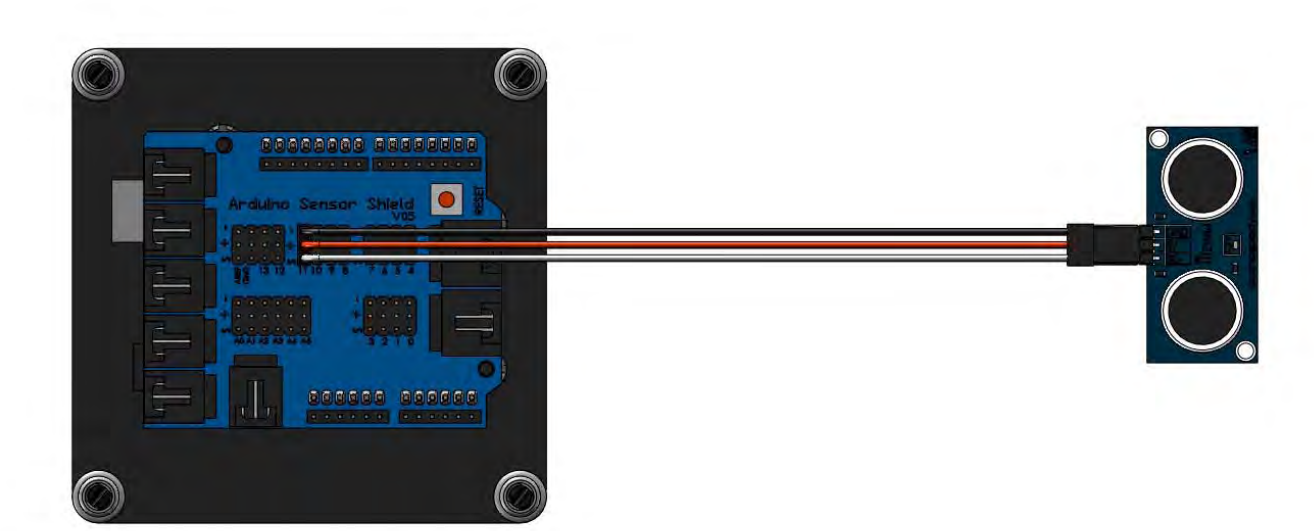
```
else
        drive.write(105); //What happens?
```

or

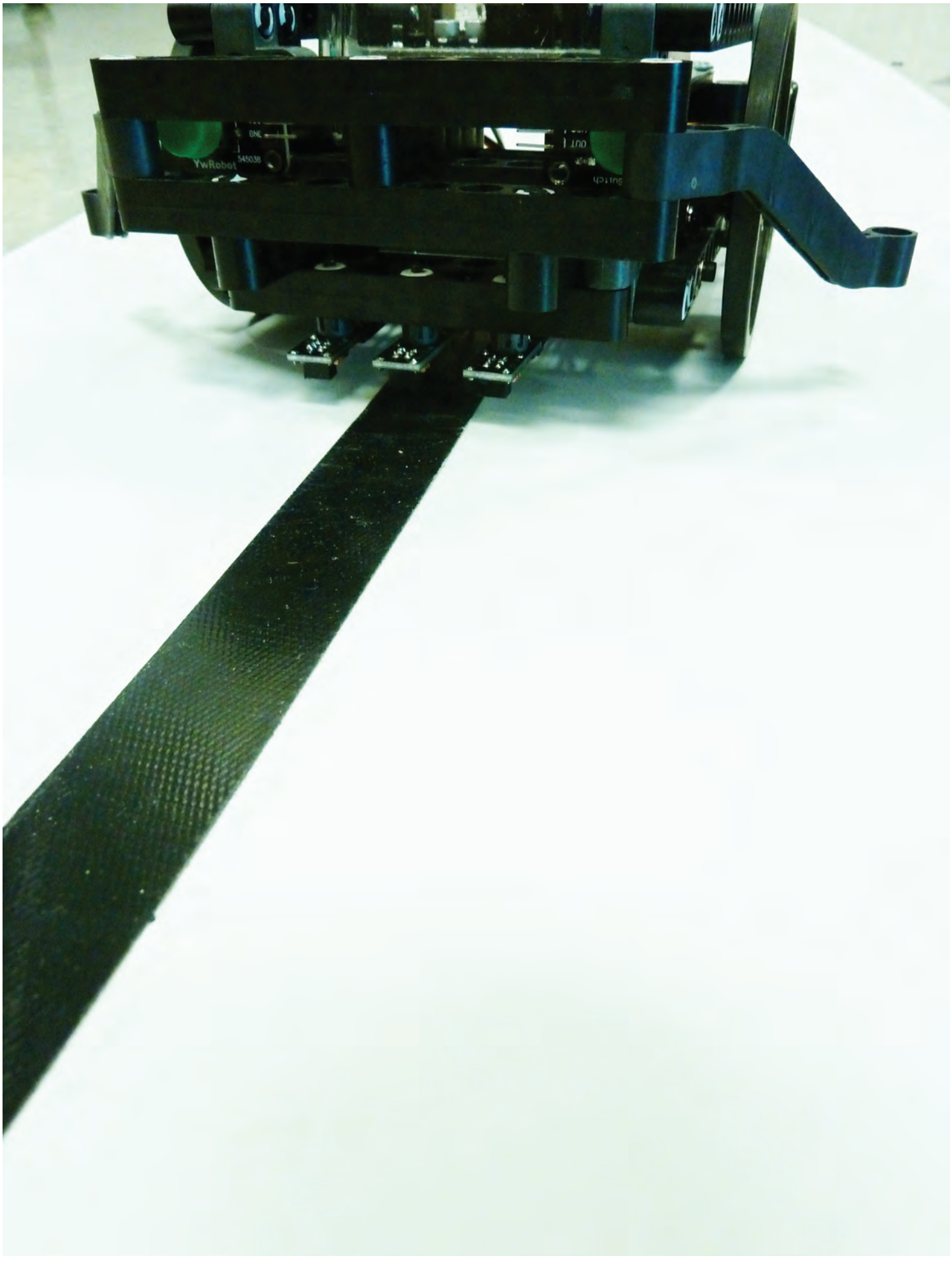
```
        drive.write(110); //What happens?
```

How can you make the code better? What if you used the electronic brake? Continue modifying your code and robot until it will successfully avoid objects.

Wiring Diagram for the Ping Ultrasound Sensor



Notes:


YwRobot
545038
BNC
OUT
Switch

QTI Sensor

The QTI sensor is an infrared emitter/receiver that is able to differentiate between a dark surface (with low IR reflectivity) and a light surface (with high IR reflectivity). These sensors are used for line following, maze navigation, or sensing the outer rim of a SumoBot ring. We use them as an analog sensor to detect different shades of gray. A daylight filter is built into the sensor.

In the code

Before using the QTI sensor we need to know what readings we can get from it.
MINDS-i has an easy calibration program to help you get a reading from the QTI sensor.

Include the MINDS-i library:

```
#include <MINDSi.h>
```

With this sensor we need to store the value received before we can display it. To store the value we create an “int” (integer) then define it; I’ll call it “val” for value.

```
int val;
```

In the “void setup” section we will need to establish a serial connection between the computer and the Arduino through the USB cable. The connection should be set at 9600 baud (9600 bit/second). This allows us to send information to the serial monitor.

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

The QTI sensor may be plugged into ports A0 through A5. I’ll attach mine to Analog pin A0. In the “void loop” section we will assign our integer “val” to our sensor reading from “QTI(A0)”. We will then send our value to the serial monitor by typing “Serial.println(val);”

```
void loop( )
{
        val = QTI(A0);
        Serial.println(val);
}
```

To check if your code is correct press the verify button. 

1.1 Challenge: **Calibrate Your QTI Sensor**

Open the Arduino program, Go to File > Examples > 0. Minds-i > 1-Calibration > QTI. Plug your Arduino into your computer via USB and plug your QTI into pin A0. Click the Upload button  , when it says "done uploading" open the serial monitor  . With the serial monitor open, pass your sensor over the "gray scale" below (See figure 1.1). Record what values the sensor returns for white and for black by using the comments in the Arduino program.

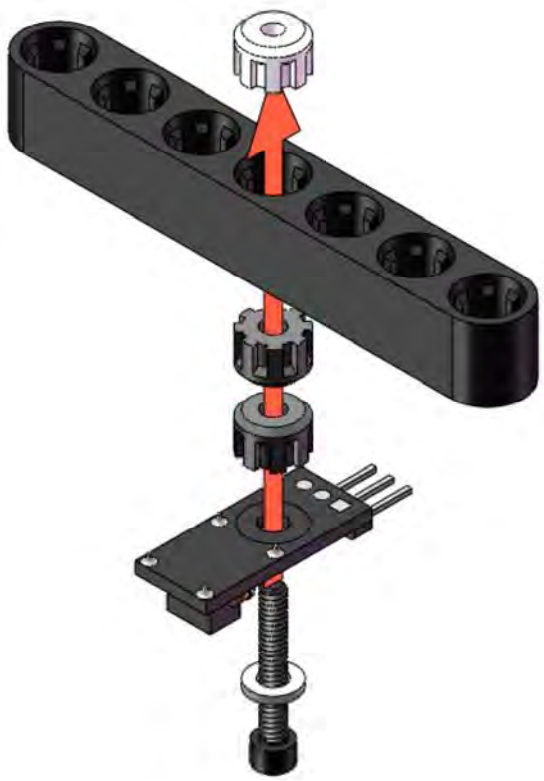
Example:

```
val = QTI(A0); //white = 300, Black = 2000
```

Mechanical hint: You will need to keep the sensor about half an inch away from the paper to get a consistent reading. Mount it on a stand or chassis to keep it at the same distance from the paper.

Figure 1.1: Gray scale

Now that you can get a reading from your QTI sensor, we can write a program that integrates the QTI with our MINDS-i Robot. I'll assume that we are using a MINDS-i Robot with one drive motor, a steering servo, an Arduino and Sensor Shield and our three QTI sensors mounted on the front of the robot. Follow this step to mount the QTI sensors on a beam:



In this program we'll control the robot autonomously through the Arduino to follow a line.

Include the MINDS-i library and the Servo Library:

```
#include <MINDSi.h>
#include <Servo.h>
```

In this code we'll use the value that we got for white when we ran the calibration. If you didn't do that already go back to the top and follow the steps. I'll need to use this value multiple times in the code so instead of typing it in every line I can make a integer (int) that stores the value for my white threshold. I'll just call it "threshold". The value that I got for white when I calibrated my sensor is 300.

```
int threshold = 300;
```

Define the ESC as "drive" and the steering servo as "steer". In the "void setup" section, attach the ESC to pin 4 and the Servo to pin 5. Set both at 90 to set the ESC at neutral and center the servo. Add a delay for 2 seconds to arm the ESC.

```
Servo drive, steer;

void setup( )
{
        drive.attach(4);
        steer.attach(5);

        drive.write(90);
        steer.write(90);

        delay(2000);
```

Make sure the robot stays at a slow speed, we can set that up in the "void setup" section. The ESC/motor will stay at the last position that it was given in the code.

```
drive.write(100);
}
```

In the "void loop" section I'll use an if/else statement with another if else/statement in the first if. If the middle sensor is off of the black line, reading white (less then our threshold of 300) the code will then check the left and the right sensors if they are on the line and steer toward the line.

```
void loop()
{
        if( QTI(A1) < threshold) //if the center sensor is off
        {
                if( QTI(A0) > threshold) //check if the left is on
                {
                        steer.write(45); // if it is, turn left
                }
                else if(QTI(A2) > threshold) // check if the right is on
                {
                        steer.write(135); //if it is, turn right
                }
        }
```

In the else statement, meaning that the middle QTI sensor is on the black line, it will drive straight.

```
        else // if the sensor is not off that means it's on the line
        {
                steer.write(90);  //center the servo to drive straight
        }
}
```

To check if your code is correct press the verify button. 

1.2 Challenge: Follow a Line

Program your robot to follow a black line on white paper.

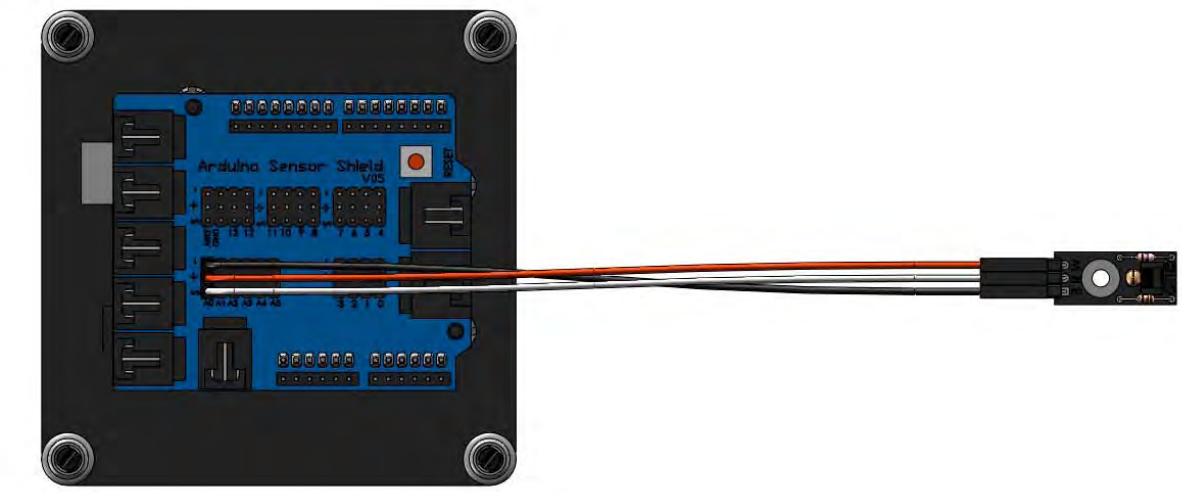
Open the Arduino program, Go to File > Examples > 0. Minds-i > 2-Application > QTI. Plug your Arduino into your computer via USB and plug your three QTI sensors into A0(left), A1(center), and A2(right). Plug your ESC into pin 4 and your servo into pin 5. Look for the line of code near the top that says "int threshold = 300" This will be your value for white. Change the number according to the reading you got from your QTI. (If you need to re-calibrate your sensor see 9.1 Challenge: Calibrate Your QTI Sensor). You may have use the calibration code a few times to double and triple check the readings and then see how well the code works. Adjust also the speed of the rover in "void setup( )" section look for the line "drive.write(100);" I'd recommend not exceeding 120, remember we want the robot to go as slow as it can to help it stay on the line and getting the correct sensor reading. Write a comment in the code using the "//" for future reference.

Ex.

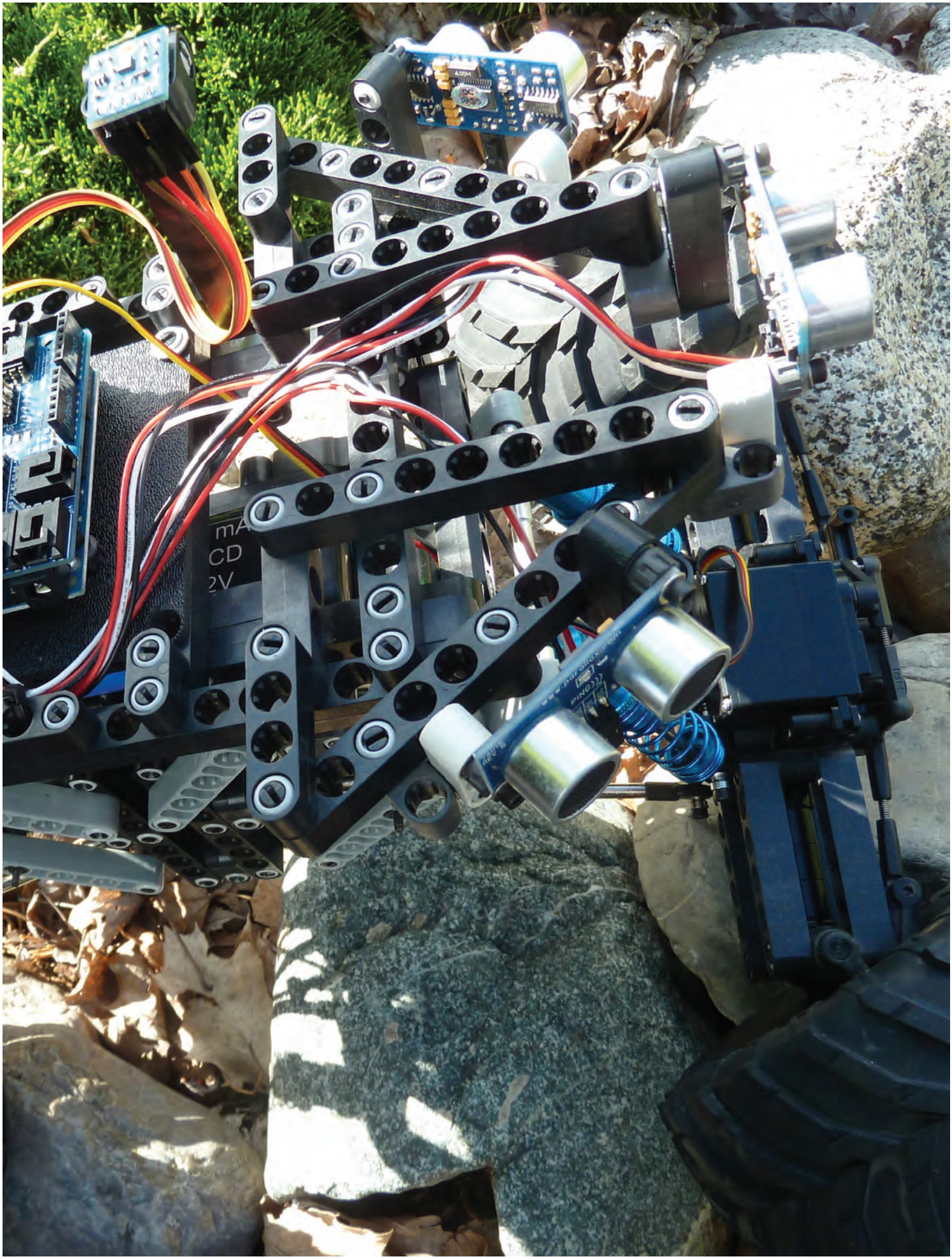
```
int threshold = 300 // white is 189 to 287, black is 2034 to 3467.
drive.write(100) // 108 is my optimal speed
```

Mechanical Hint: Generally, the darker the line is the great the value the QTI sensor gets, and the lighter the line is the lower the value is. The value the sensor gets also changes depending on how fast the rover is moving. Try running the calibration code while moving the rover on a big white sheet of paper or on a white floor and comparing the results to when it was stationary.

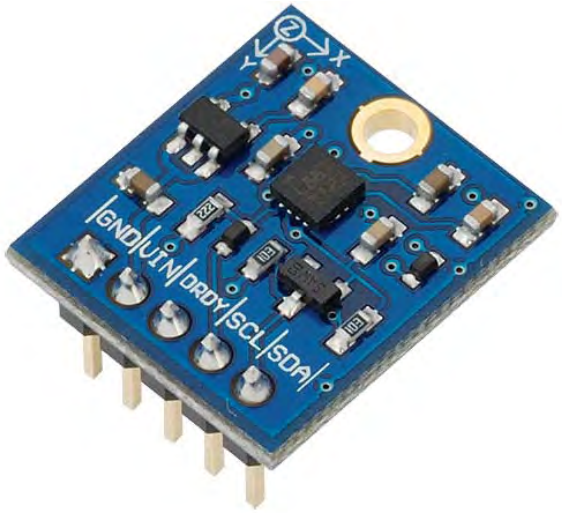
Wiring Diagram for the QTI Sensor



Notes:



Compass Sensor



Used to detect magnetic fields such as the Earth's magnetic field. The sensor converts any magnetic field to a differential voltage output on 3 axes. This voltage shift is the raw digital output value, which can then be used to calculate headings or sense magnetic fields coming from different directions.

In the code

Include the Wire library, Servo library and MINDS-i library:

```
#include <Wire.h>
#include <MINDSi.h>
#include <Servo.h>
```

We will need to define a "float" (a number with a decimal point see the Reference Guide Section) I'll call it "val".

```
float val;
```

In the "void setup section" we will start Serial communications by typing "Serial.begin(9600);" then we will initialize the compass by typing "beginCompass( );" on the next line.

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

In the "void loop" section we will assign the compass reading to our "float" by typing "val = getHeading();". On the next line we will print the compass reading to the serial monitor by typing "Serial.println(val,DEC);". We have to add the "DEC" so when it prints, it is a decimal number.

```
void loop( )
{
        val = getHeading( );
        Serial.println(val,DEC);
}
```

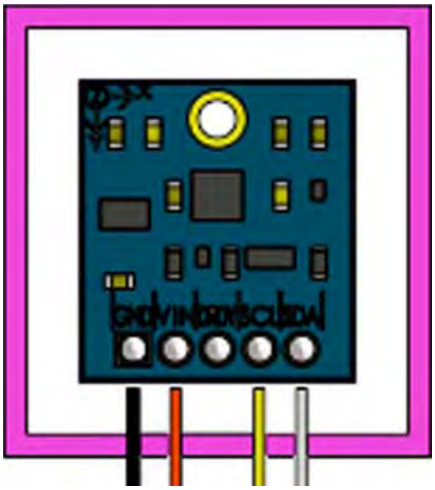
To check if your code is correct press the verify button. 

1.1 Challenge: **Get a reading.**

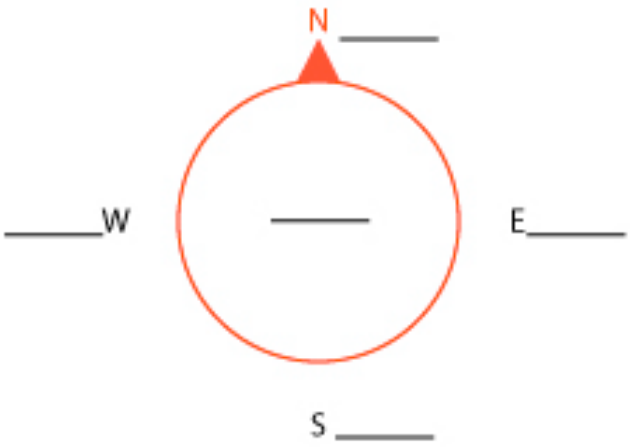
Open the Arduino program, Go to File > Examples > 0. Minds-i > 1-Calibration > Compass. Plug your Arduino into your computer via USB and plug your compass into the I2C port (refer to Figure 1.1 for proper wiring polarity).

Mechanical hint: You will need to keep the compass relatively flat to get a consistent reading.

Figure 1.1: Compass Wiring Polarity



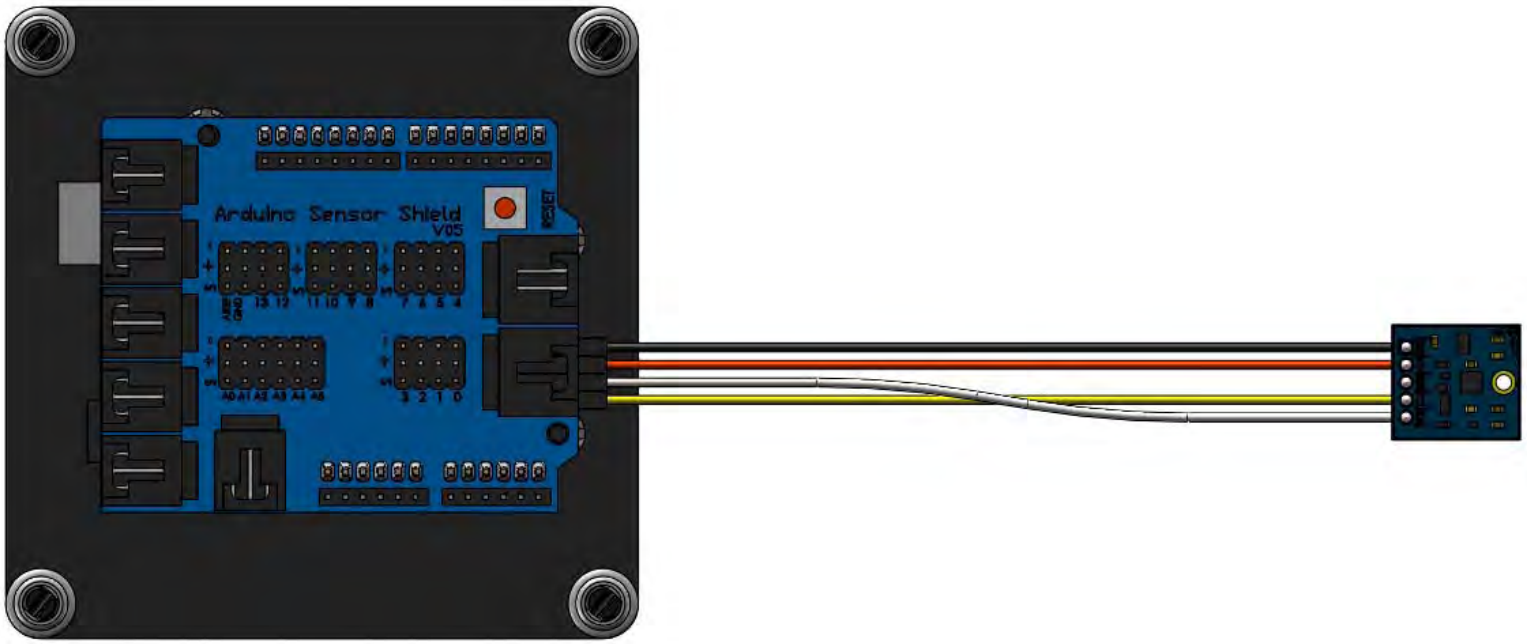
Now that you know to get a reading, you will need to map out what reading represents North, South, East and West.



1.2 Challenge: Get a rover to drive and face a set direction

Open the Arduino program, Go to File > Examples > 0. Minds-i > 1-Application> Compass. Plug your Arduino into your computer via USB and plug your Compass into the I2C port. Look for this line of code in the void loop section "if(fabs(getHeading()) < .05) drive.write(90);" Change the value of .05 to one of the values you figured out from Challenge 1.1.

Wiring diagram for Compass Sensor

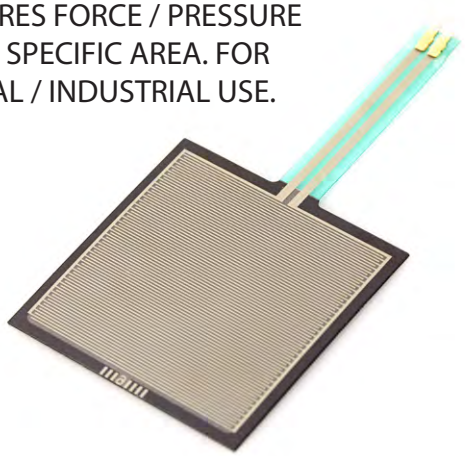


Notes:

O2 SENSOR:
MEASURES OXYGEN LEVELS IN ENGINE EXHAUST. FOR AUTOMOTIVE USE.

CO2 SENSOR:
MEASURES CARBON DIOXIDE LEVELS IN AIR. FOR MEDICAL / INDUSTRIAL USE.

FORCE SENSITIVE RESISTOR:
MEASURES FORCE / PRESSURE OVER A SPECIFIC AREA. FOR MEDICAL / INDUSTRIAL USE.



JOY STICK CONTROLLER:
USES ANALOG POTENTIOMETERS TO GIVE FEEDBACK BASED ON STICK MOVEMENT. FOR MILITARY / MEDICAL / INDUSTRIAL / ENTERTAINMENT USE.

Analog Sensors

Analog Sensors return a varied voltage usually from 0 to 5 volts (sometimes 0 to 3 volts). The voltage returned by the sensor is directly proportional to the resistance in the sensor itself. The main advantage is the fine definition of the analog signal, which has the potential for an infinite amount of signal resolution. Most Micro-controllers measure analog signals as a number between 0 and 1024.

In the code

With this sensor we need to store the value received before we can display it. To store the value we create an "int" (integer) then define it; I'll call it "val" for value.

```
Int val;
```

In the "void setup" section we will need to establish a serial connection between the computer and the Arduino through the USB cable. The connection should be set at 9600 baud (9600 bit/second). This allows us to send information to the serial monitor.

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

The Analog sensor may be plugged into ports A0 through A5. I'll attach mine to Analog pin A0. In the "void loop" section we will assign our integer "val" to our sensor reading from "analogRead(A0)". We will then send our value to the serial monitor by typing "Serial.println(val);"

```
void loop( )
{
        val = analogRead(A0);
        Serial.println(val);
}
```

To check if your code is correct press the verify button. 

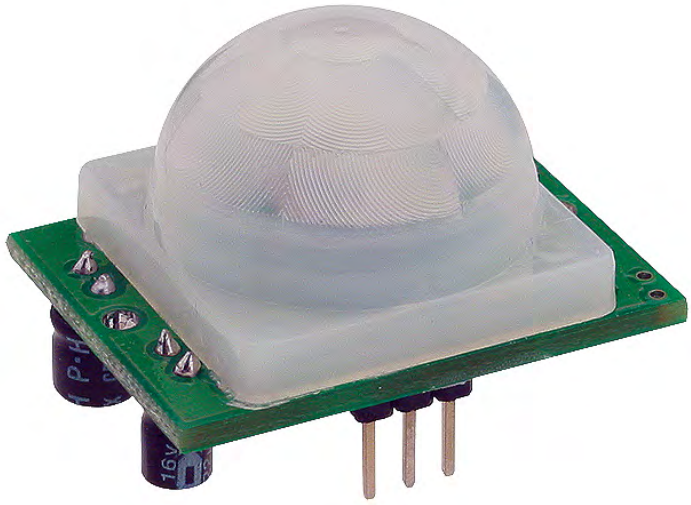
1.1 Challenge: **Get a reading**

Open the Arduino program, Go to File > Examples > 0. Minds-i > 1-Calibration > AnalogSensor. Plug your Arduino into your computer via USB and plug your sensor into pin A0. When you're done press upload. 

Mechanical hint: You will need to make sure you have your sensor properly wired to get a correct reading.

MASS AIRFLOW SENSOR:
MEASURES INTAKE AIRFLOW, BAROMETRIC PRESSURE AND SOMETIMES TEMPERATURE. FOR AUTOMOTIVE USE.

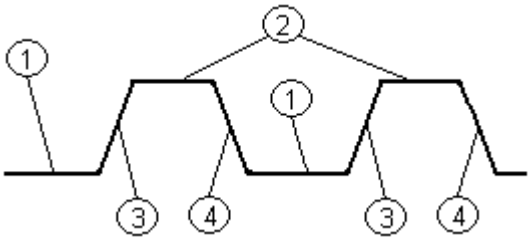
PIR (MOTION) SENSOR:
MEASURES CHANGING IFRA-RED HEAT TO DETECT MOVEMENT. FOR INDUSTRIAL / SECURITY USE.



TOGGLE SWITCH:
THE SIMPLEST TYPE OF DIGITAL SENSOR, RETURNS EITHER A 1 OR 0. USED ON ALMOST EVERY PIECE OF MACHINERY, EQUIPMENT, OR DEVICE.

Digital Sensors

A digital sensor is an electronic or electrochemical sensor, where data conversion and data transmission are done digitally. Digital sensors use a varied on / off pulse to send information back to a Micro-controller. The time on vs. the time off is measured and corresponds to the reading taken.



In the code

With this sensor we need to store the value received before we can display it. To store the value we create an "int" (integer) then define it; I'll call it "val" for value.

```
Int val;
```

In the "void setup" section we will need to establish a serial connection between the computer and the Arduino through the USB cable. The connection should be set at 9600 baud (9600 bit/second). This allows us to send information to the serial monitor.

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

The Digital sensor may be plugged into ports 0 through 13 or A0 through A5. I'll attach mine to pin 13. In the "void loop" section we will assign our integer "val" to our sensor reading from "digitalRead(13)". We will then send our value to the serial monitor by typing "Serial.println(val);"

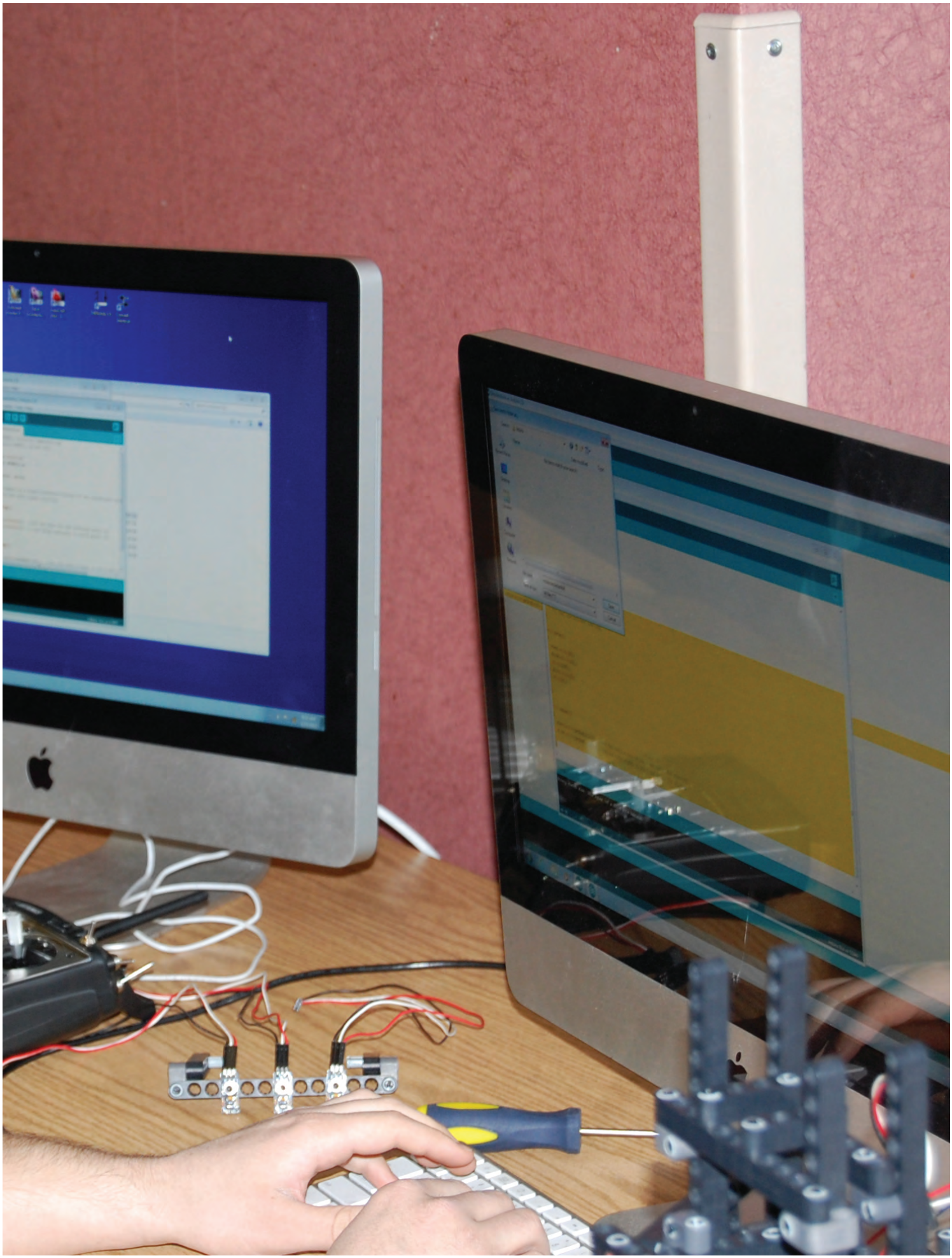
```
void loop( )
{
        val = digitalRead(13);
        Serial.println(val);
}
```

To check if your code is correct press the verify button. 

1.1 Challenge: **Get a reading**

Open the Arduino program, Go to File > Examples > 0. Minds-i > 1-Calibration > DigitalSensor. Plug your Arduino into your computer via USB and plug your sensor into pin 13. When you're done press upload. 

Mechanical hint: You will need to make sure you have your sensor properly wired to get a correct reading.



Reference Guide

This reference guide will provide you with a more in depth approach at the various terms used in the Arduino program. Most of the terms that are used throughout this manual and in the MINDS-i example code will be found here, but there are many other terms that can be used, for further reference see Arduino.cc. Arduino Programs can be divided into three main parts, structures, values (variables and constants) and functions.

STRUCTURE

setup( )

The setup( ) function is called when a sketch starts. Use it to initialize variables, objects, pin modes, serial connection, set up servos, etc. The setup function will only run once, after each power up or reset of the Arduino board.

Example

```
Int buttonPin = 3;

Void setup( )
{
        Serial.begin(9600);
        pinMode(buttonPin, INPUT);
}

void loop( )
{
        //…
}
```

loop( )

After creating a setup( ) function, which initializes and sets the initial values, the loop( ) function does precisely what its name suggests, and loops consecutively, allowing your program to change and respond. Use it to actively control the Arduino board.

Example

```
int buttonPin = 3;

// setup initializes serial and the button pin

void setup( )
{
        Serial.begin(9600);
        pinMode(buttonPin, INPUT);
}

// loop checks the button pin each time,
// and will send serial if it is pressed

void loop( )
{
        if (digitalRead(buttonPin) == HIGH) // HIGH is equivalent to TRUE or 1.
        serial.println("H");
        else
        serial.println("L");

        delay(1000);
}
```

Control Structures

if (conditional) and ==, !=, <, > (comparison operators)

'if ', which is used in conjunction with a comparison operator, tests whether a certain condition has been reached, such as an input being above a certain number. The format for an 'if ' test is:

```
if (someVariable > 50)
{
        // do something here
}
```

The program tests to see if someVariable is greater than 50. If it is, the program takes a particular action. Put another way, if the statement in parentheses is true, the statements inside the brackets are run. If not, the program skips over the code.

The brackets may be omitted after an 'if ' statement. If this is done, the next line (defined by the semicolon) becomes the only conditional statement.

```
if (x > 120) digitalWrite(LEDpin, HIGH);

if (x > 120)
digitalWrite(LEDpin, HIGH);

if (x > 120){ digitalWrite(LEDpin, HIGH); }

if (x > 120)
{
        digitalWrite(LEDpin1, HIGH);
        digitalWrite(LEDpin2, LOW);
}

// all are correct
```

The statements being evaluated inside the parentheses require the use of one or more operators:

Comparison Operators:

x == y (x is equal to y)
x != y (x is not equal to y)
x < y (x is less than y)
x > y (x is greater than y)
x <= y (x is less than or equal to y)
x >= y (x is greater than or equal to y)

Warning:

Beware of accidentally using the single equal sign (e.g. if (x = 10)). The single equal sign is the assignment operator, and sets x to 10 (puts the value 10 into the variable x). Instead use the double equal sign (e.g. if (x == 10)), which is the comparison operator, and tests whether x is equal to 10 or not. The latter statement is only true if x equals 10, but the former statement will always be true.

This is because C evaluates the statement if (x=10) as follows: 10 is assigned to x (remember that the single equal sign is the assignment operator), so x now contains 10. Then the 'if ' conditional evaluates 10, which always evaluates to TRUE, since any non-zero number evaluates to TRUE. Consequently, if (x = 10) will always evaluate to TRUE, which is not the desired result when using an 'if ' statement. Additionally, the variable x will be set to 10, which is also not a desired action.

'if ' can also be part of a branching control structure using the 'if...else' construction.

if / else

‘if/else ’ allows greater control over the flow of code than the basic ‘if ’ statement, by allowing multiple tests to be grouped together. For example, an analog input could be tested and one action taken if the input was less than 500, and another action taken if the input was 500 or greater. The code would look like this:

```
if (pinFiveInput < 500)
{
        // action A
}
else
{
        // action B
}
```

‘else’ can proceed another ‘if ’ test, so that multiple, mutually exclusive tests can be run at the same time.

Each test will proceed to the next one until a true test is encountered. When a true test is found, its associated block of code is run, and the program then skips to the line following the entire ‘if/else’ construction. If no test proves to be true, the default ‘else’ block is executed, if one is present, and sets the default behavior.

Note that an ‘else if ’ block may be used with or without a terminating else block and vice versa. An unlimited number of such else if branches is allowed.

```
if (pinFiveInput < 500)
{
        // do Thing A
}
else if (pinFiveInput >= 1000)
{
        // do Thing B
}
else
{
        // do Thing C
}
```

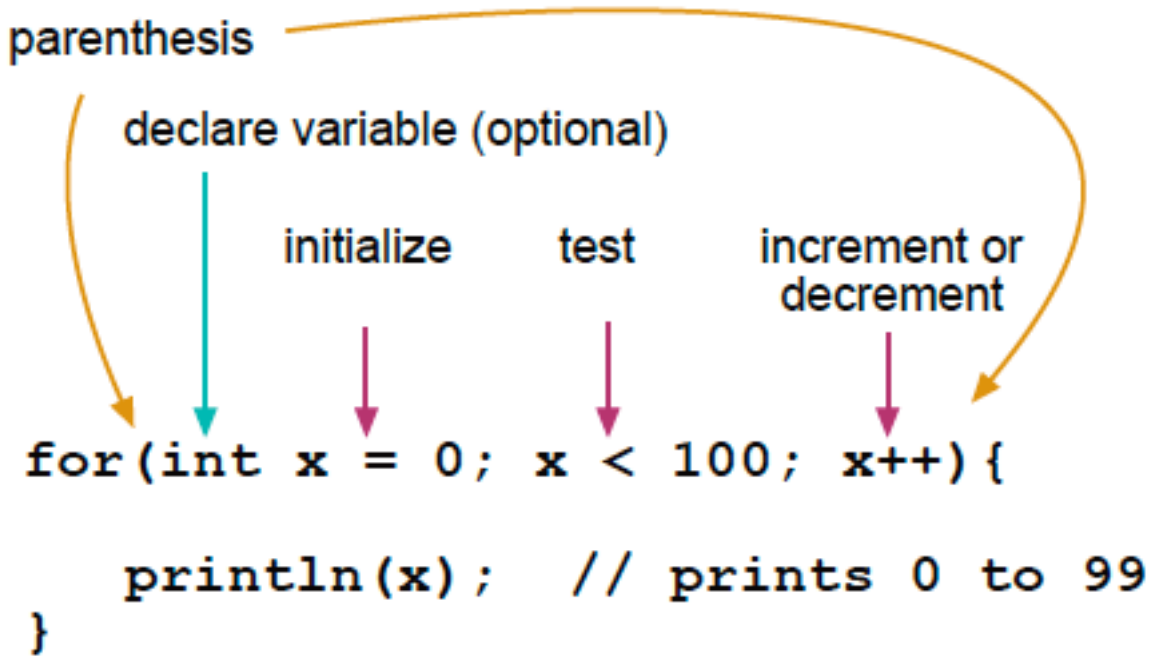
Another way to express branching, mutually exclusive tests, is with the ‘switch case’ statement.

For statements

The ‘for’ statement is used to repeat a block of statements enclosed in curly braces. An increment counter is usually used to increment and terminate the loop. The ‘for’ statement is useful for any repetitive operation, and is often used in combination with arrays to operate on collections of data/pins.

There are three parts to the ‘for’ loop header:

```
for (initialization; condition; increment)
{
//statement(s);
}
```



The initialization happens first and exactly once. Each time through the loop, the condition is tested; if it's true, the statement block, and the increment is executed, then the condition is tested again. When the condition becomes false, the loop ends.

Example

```
// Dim an LED using a PWM pin
int PWMpin = 10; // LED in series with 470 ohm resistor on pin 10

void setup( )
{
        // no setup needed
}
```

```
void loop( )
{
        for (int i=0; i <= 255; i++)
        {
                analogWrite(PWMpin, i);
                delay(10);
        }
}
```

Coding Tips

The C ‘for’ loop is much more flexible than ‘for’ loops found in some other computer languages, including BASIC. Any or all of the three header elements may be omitted, although the semicolons are required. Also the statements for initialization, condition, and increment can be any valid C statements with unrelated variables, and use any C data types including floats. These types of unusual ‘for’ statements may provide solutions to some rare programming problems.

For example, using a multiplication in the increment line will generate a logarithmic progression:

```
for (int x = 2; x < 100; x = x * 1. 5)
{
        println(x);
}
```

Generates: 2, 3, 4, 6, 9, 13, 19, 28, 42, 63, 94

Another example, fade an LED up and down with one for loop:

```
void loop( )
{
        int x = 1;
        for (int i = 0; i > -1; i = i + x)
        {
                analogWrite(PWMpin, i);
                if (i == 255) x = -1;           // switch direction at peak
                delay(10);
        }
}
```

switch / case statements

Like ‘if ’ statements, ‘switch/case’ controls the flow of programs by allowing programmers to specify different code that should be executed in various conditions. In particular, a ‘switch’ statement compares the value of a variable to the values specified in ‘case’ statements. When a ‘case’ statement is found whose value matches that of the variable, the code in that ‘case’ statement is run.

The ‘break’ keyword exits the ‘switch’ statement, and is typically used at the end of each case. Without a ‘break’ statement, the ‘switch’ statement will continue executing the following expressions ("falling-through") until a ‘break’, or the end of the ‘switch’ statement is reached.

Example

```
switch (var)
{
        case 1:
                //do something when var equals 1
                break;
        case 2:
                //do something when var equals 2
                break;
        default:
                // if nothing else matches, do the default
                // default is optional
}
```

Syntax

```
switch (var)
{
        case label:
                // statements
                break;
        case label:
                // statements
                break;
        default:
                // statements
}
```

Parameters

var: the variable whose value to compare to the various cases

label: a value to compare the variable to

while loops

'while' loops will loop continuously, and infinitely, until the expression inside the parenthesis, ( ) becomes false. Something must change the tested variable, or the while loop will never exit. This could be in your code, such as an incremented variable, or an external condition, such as testing a sensor.

Syntax

```
while(expression)
{
        // statement(s)
}
```

Parameters

expression - a (boolean) C statement that evaluates to true or false

Example

```
var = 0;
while(var < 200)
{
        // do something repetitive 200 times
        var++;
}
```

do - while

The 'do' loop works in the same manner as the while loop, with the exception that the condition is tested at the end of the loop, so the do loop will always run at least once.

```
do
{
        // statement block
}
 while (test condition);
```

Example

```
do
{
        delay(50);          // wait for sensors to stabilize
        x = readSensors( );  // check the sensors
}
while (x < 100);
```

break

'break' is used to exit from a 'do', 'for', or 'while' loop, by passing the normal loop condition. It is also used to exit from a 'switch' statement.

Example

```
for (x = 0; x < 255; x ++)
{
        digitalWrite(PWMpin, x);
        sens = analogRead(sensorPin);
         if (sens > threshold) // bail out on sensor detect
        {
                x = 0;
                break;
        }
        delay(50);
}
```

Further Syntax

#Define

#define is a useful C component that allows the programmer to give a name to a constant value before the program is compiled. Defined constants in Arduino don't take up any program memory space on the chip. The compiler will replace references to these constants with the defined value at compile time.

This can have some unwanted side effects though, if for example, a constant name that had been #defined is included in some other constant or variable name. In that case the text would be replaced by the #defined number (or text).

In general, the const keyword is preferred for defining constants and should be used instead of #define.

Arduino defines have the same syntax as C defines:

Syntax

```
#define constantName value                    //Note that the # is necessary.
```

Example

```
#define ledPin 3
// The compiler will replace any mention of ledPin with the value 3 at compile time.
```

Tip

There is no semicolon after the #define statement. If you include one, the compiler will throw cryptic errors further down the page.

```
#define ledPin 3;    // this is an error
```

Similarly, including an equal sign after the #define statement will also generate a cryptic compiler error further down the page.

```
#define ledPin  = 3  // this is also an error
```

#include

#include is used to include outside libraries in your sketch. This gives the programmer access to a large group of standard C libraries (groups of pre-made functions), and also libraries written especially for Arduino.

Note that #include, similar to #define, has no semicolon terminator, and the compiler will yield cryptic error messages if you add one.

Example

This example includes the MINDS-i library that is used to simplify the messy details of the code so you don't have to.

```
#include <MINDSi.h>
```

Boolean Operators

These can be used inside the condition of an if statement.

&& (logical 'and')

True only if both operands are true, e.g.

```
if (digitalRead(2) == HIGH  && digitalRead(3) ==
HIGH)
{
       // read two switches
       // ...
}
```

is true only if both inputs are high.

|| (logical 'or')

True if either operand is true, e.g.

```
if (x > 0 || y > 0)
{
       // ...
}
```

is true if either x or y is greater than 0.

! (not)

True if the operand is false, e.g.

```
if (!x)
{
       // ...
}
```

is true if x is false (i.e. if x equals 0).

Warning

Make sure you don't mistake the boolean AND operator, && (double ampersand) for the bitwise AND operator & (single ampersand). They are entirely different beasts.

Similarly, do not confuse the boolean || (double pipe) operator with the bitwise OR operator | (single pipe).

The bitwise not ~ (tilde) looks much different than the boolean not ! (exclamation point or "bang" as the programmers say) but you still have to be sure which one you want where.

Examples

```
if (a >= 10 && a <= 20){}     // true if a is between 10 and 20
```

Compound Operators

++ (increment) / -- (decrement)

Increment or decrement a variable

Syntax

```
x++;  // increment x by one and returns the old value of x
++x;  // increment x by one and returns the new value of x

x-- ;   // decrement x by one and returns the old value of x
--x ;   // decrement x by one and returns the new value of x
```

Parameters

x: an integer or long (possibly unsigned)

Returns

The original or newly incremented / decremented value of the variable.

Examples

```
x = 2;
y = ++x;      // x now contains 3, y contains 3
y = x--;      // x contains 2 again, y still contains 3
```

Constants

Constants are predefined variables in the Arduino language. They are used to make the programs easier to read. We classify constants in groups.

Defining Logical Levels, true and false (Boolean Constants)

There are two constants used to represent truth and falsity in the Arduino language: true, and false.

false

false is the easier of the two to define. false is defined as 0 (zero).

true

true is often said to be defined as 1, which is correct, but true has a wider definition. Any integer which is *non-zero* is true, in a Boolean sense. So -1, 2 and -200 are all defined as true, too, in a Boolean sense.
Note that the *true* and *false* constants are typed in lowercase unlike HIGH, LOW, INPUT, & OUTPUT.

Defining Pin Levels, HIGH and LOW

When reading or writing to a digital pin there are only two possible values a pin can take/be-set-to: HIGH and LOW.

HIGH

The meaning of HIGH (in reference to a pin) is somewhat different depending on whether a pin is set to an INPUT or OUTPUT. When a pin is configured as an INPUT with pinMode, and read with digitalRead, the microcontroller will report HIGH if a voltage of 3 volts or more is present at the pin. A pin may also be configured as an INPUT with pinMode, and subsequently made HIGH with digitalWrite, this will set the internal 20K pullup resistors, which will *steer* the input pin to a HIGH reading unless it is pulled LOW by external circuitry. This is how INPUT_PULLUP works as well When a pin is configured to OUTPUT with pinMode, and set to HIGH with digitalWrite, the pin is at 5 volts. In this state it can *source* current, e.g. light an LED that is connected through a series resistor to ground, or to another pin configured as an output, and set to LOW.

LOW

The meaning of LOW also has a different meaning depending on whether a pin is set to INPUT or OUTPUT. When a pin is configured as an INPUT with pinMode, and read with digitalRead, the microcontroller will report LOW if a voltage of 2 volts or less is present at the pin.
When a pin is configured to OUTPUT with pinMode, and set to LOW with digitalWrite, the pin is at 0 volts. In this state it can *sink* current, e.g. light an LED that is connected through a series resistor to, +5 volts, or to another pin configured as an output, and set to HIGH.

Defining Digital Pins, INPUT, INPUT_PULLUP, and OUTPUT

Digital pins can be used as INPUT, INPUT_PULLUP, or OUTPUT. Changing a pin with pinMode( ) changes the electrical behavior of the pin.

Pins Configured as INPUT

Arduino (Atmega) pins configured as INPUT with pinMode( ) are said to be in a high-impedance state. Pins configured as INPUT make extremely small demands on the circuit that they are sampling, equivalent to a series resistor of 100 Megohms in front of the pin. This makes them useful for reading a sensor, but not powering an LED.
If you have your pin configured as an INPUT, you will want the pin to have a reference to ground, often accomplished with a pull-down resistor (a resistor going to ground) as described in the Digital Read Serial tutorial.

Pins Configured as INPUT_PULLUP

The Atmega chip on the Arduino has internal pull-up resistors (resistors that connect to power internally) that you can access. If you prefer to use these instead of external pull-down resistors, you can use the INPUT_PULLUP argument in pinMode( ). This effectively inverts the behavior, where HIGH means the sensor is off, and LOW means the sensor is on. See the Input Pullup Serial tutorial for an example of this in use.

Pins Configured as OUTPUT

Pins configured as OUTPUT with pinMode( ) are said to be in a low-impedance state. This means that they can provide a substantial amount of current to other circuits. Atmega pins can source (provide positive current) or sink (provide negative current) up to 40 mA (milliamps) of current to other devices/circuits. This makes them useful for powering LED's but useless for reading sensors. Pins configured as outputs can also be damaged or destroyed if short circuited to either ground or 5 volt power rails. The amount of current provided by an Atmega pin is also not enough to power most relays or motors, and some interface circuitry will be required.

int

Integers are your primary datatype for number storage, and store a 2 byte value. This yields a range of -32,768 to 32,767 (minimum value of -2^15 and a maximum value of (2^15) - 1).

Int's store negative numbers with a technique called 2's complement math. The highest bit, sometimes refered to as the "sign" bit, flags the number as a negative number. The rest of the bits are inverted and 1 is added.

The Arduino takes care of dealing with negative numbers for you, so that arithmetic operations work transparently in the expected manner. There can be an unexpected complication in dealing with the bitshift right operator (>>) however.

Example

```
int ledPin = 13;
```

Syntax

```
int var = val;
```

var - your int variable name
val - the value you assign to that variable

Coding Tip

When variables are made to exceed their maximum capacity they "roll over" back to their minimum capacitiy, note that this happens in both directions.

unsigned int

Unsigned ints (unsigned integers) are the same as ints in that they store a 2 byte value. Instead of storing negative numbers however they only store positive values, yielding a useful range of 0 to 65,535 (2^16) - 1).

```
int x
   x = -32,768;
   x = x - 1;       // x now contains 32,767 - rolls over in neg. direction

   x = 32,767;
   x = x + 1;       // x now contains -32,768 - rolls over
```

The difference between unsigned ints and (signed) ints, lies in the way the highest bit, sometimes refered to as the "sign" bit, is interpreted. In the Arduino int type (which is signed), if the high bit is a "1", the number is interpreted as a negative number, and the other 15 bits are interpreted with 2's complement math.

Example

```
unsigned int ledPin = 13;
```

Syntax

```
unsigned int var = val;
```

var - your unsigned int variable name
val - the value you assign to that variable

Coding Tip

When variables are made to exceed their maximum capacity they "roll over" back to their minimum capacitiy, note that this happens in both directions

```
unsigned int x
  x = 0;
  x = x - 1;       // x now contains 65535 - rolls over in neg direction
  x = x + 1;       // x now contains 0 - rolls over
```

long

Long variables are extended size variables for number storage, and store 32 bits (4 bytes), from -2,147,483,648 to 2,147,483,647.

Example

```
long speedOfLight = 186000L;   // 'L' to force the constant into a long data format.
```

Syntax

```
long var = val;
```

var - the long variable name
val - the value assigned to the variable

float

Datatype for floating-point numbers, a number that has a decimal point. Floating-point numbers are often used to approximate analog and continuous values because they have greater resolution than integers. Floating-point numbers can be as large as 3.4028235E+38 and as low as -3.4028235E+38. They are stored as 32 bits (4 bytes) of information.

Floats have only 6-7 decimal digits of precision. That means the total number of digits, not the number to the right of the decimal point. Unlike other platforms, where you can get more precision by using a double (e.g. up to 15 digits), on the Arduino, double is the same size as float.

Floating point numbers are not exact, and may yield strange results when compared. For example 6.0 / 3.0 may not equal 2.0. You should instead check that the absolute value of the difference between the numbers is less than some small number.

Floating point math is also much slower than integer math in performing calculations, so should be avoided if, for example, a loop has to run at top speed for a critical timing function. Programmers often go to some lengths to convert floating point calculations to integer math to increase speed.

Examples

```
float var = val;
```

Syntax

```
float myfloat;
float sensorCalbrate = 1.117;
```

var - your float variable name
val - the value you assign to that variable

Example Code

```
int x;
   int y;
   float z;

   x = 1;
   y = x / 2;            // y now contains 0, ints can't hold fractions
   z = (float)x / 2.0;   // z now contains .5 (you have to use 2.0, not 2)
```

Time

millis( )

Returns the number of milliseconds since the Arduino board began running the current program. This number will overflow (go back to zero), after approximately 50 days.

Parameters

None

Returns

Number of milliseconds since the program started (*unsigned long*)

Example

```
unsigned long time;

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

void loop( )
{
        Serial.print("Time: ");
        time = millis();
        //prints time since program started
        Serial.println(time);
        // wait a second so as not to send massive amounts of data
        delay(1000);
}
```

Tip:

Note that the parameter for millis is an unsigned long, errors may be generated if a programmer tries to do math with other datatypes such as ints.

Communication

println( )

Prints data to the serial port as human-readable ASCII text followed by a carriage return character (ASCII 13, or '\r') and a newline character (ASCII 10, or '\n'). This command takes the same forms as Serial.print().

Syntax

```
Serial.println(val)
Serial.println(val, format)
```

Parameters

val: the value to print - any data type

format: specifies the number base (for integral data types) or number of decimal places (for floating point types)

Returns

size_t (long): println() returns the number of bytes written, though reading that number is optional

Example:

```
int analogValue = 0;    // variable to hold the analog value

void setup( )
{
        Serial.begin(9600); // open the serial port at 9600 bps:
}
void loop( )
{
        analogValue = analogRead(0);  //read the analog input on pin 0
        Serial.println(analogValue);       // print as an ASCII-encoded decimal
        Serial.println(analogValue, DEC);  // print as an ASCII-encoded decimal
        Serial.println(analogValue, HEX);  // print as an ASCII-encoded hexadecimal
        Serial.println(analogValue, OCT);  // print as an ASCII-encoded octal
        Serial.println(analogValue, BIN);  // print as an ASCII-encoded binary
        delay(10); // delay 10 milliseconds before the next reading:
}
```

Additional Links and Resources

I. The Arduino Uno

The Arduino Uno is a micro-controller board based on the ATmega328. It has 14 digital input/output pins (of which 6 can be used as PWM outputs), 6 analog inputs, a 16 MHz crystal oscillator, a USB connection, a power jack, an ICSP header, and a reset button. It contains everything needed to support the micro-controller; simply connect it to a computer with a USB cable or power it with a AC-to-DC adapter or battery to get started.

A) **The Adruino Uno Board:** Open the following link to access user reference and instructional materials that are necessary for gaining understanding of the workings of the Arduino Uno Board and its various capabilities.

Link: http://arduino.cc/en/Main/ArduinoBoardUno

1. Overview
2. Summary
3. Schematic & Reference Design
4. Power
5. Memory
6. Input and Output
7. Communication
8. Programming
9. Automatic (Software) Reset
10. USB Over-current Protection
11. Physical Characteristics

B) **The Shield:** The Sensor Shield's purpose is to make it easy to connect cables and devices to the correct Arduino pins. It simply connects the Arduino pins to many connectors that are ready to use to connect to various devices like Servos, Motors, and Sensors with simple cables. Open the following link to access user reference and instructional materials that are necessary for gaining understanding of the workings of the Shield and its various capabilities.

Link: http://arduino-info.wikispaces.com/SensorShield

1. Digital I/O Pins 0-13
2. Analog Input Pins A0 to A5
3. Communications Port

II. Software

The open-source Arduino environment makes it easy to write code and upload it to the I/O board. It runs on Windows, Mac OS X, and Linux. The environment is written in Java and based on processing, avr-gcc, and other open source software.

A) **Before Getting Started:** The Arduino Software can be found on the disk that was included in your MINDS-i kit. To install the program simply drag and drop the file named "arduino-1.0" onto your computer. The MINDS-i libraries and examples are already included.

Or to download the Arduino program from the Arduino website follow these instructions:

B) **Getting started:** Step-by-step instructions for setting up the Arduino software and connecting it to an Arduino Uno. Open the following link to access user references and instructional materials that are necessary for gaining understanding of the workings of the Arduino software and it's various capabilities.

Link: http://arduino.cc/en/Guide/HomePage

1. PC interface
2. Downloading program
3. Starting a new program

III. Familiarization with Menus and Functions

The Arduino programming environment is easy-to-use for beginners, yet flexible enough for advanced users as well. For teachers, it's conveniently based on the Processing-Programming environment, so students learning to program in that environment will be familiar with the look and feel of Arduino

A) **Arduino Introduction:** Arduino is a tool for making computers that can sense and control more of the physical world than your desktop computer. It's an open-source physical computing platform based on a simple micro-controller board, and a development environment for writing software for the board. Open the following link to access user reference and instructional materials that are necessary for gaining understanding of the workings of Arduino and its various capabilities.

Link: http://arduino.cc/en/Guide/Introduction

1. What is Arduino?
2. Why Arduino?
3. How do I use Arduino?

B) **Arduino Development environment:** The Arduino development environment contains a text editor for writing code, a message area, a text console, a toolbar with buttons for common functions, and a series of menus. It connects to the Arduino hardware to upload programs and communicate with the Arduino hardware. Open the following link to access user reference and instructional materials that are necessary for gaining understanding of the workings of the Arduino development environment and it's various capabilities.

Link: http://arduino.cc/en/Guide/Environment

Toolbar Buttons
Edit
Sketch
Tools
Sketchbook
Tabs, Multiple Files, and Compilation
7. Uploading
8. Libraries
9. Third-Party Hardware
10. Serial Monitor
11. Preferences
12. Boards

C) **Code and Commands:** This page contains explanations of some of the elements of the Arduino hardware and software and the concepts behind them. Open the following link to access user reference and instructional materials that are necessary for gaining understanding of the workings of the code and commands and it's various capabilities.

Link: http://arduino.cc/en/Tutorial/Foundations

4. Basics
   (a) **Sketch:** The various components of a sketch and how they work. Click on the following link to learn more.
   Link: http://arduino.cc/en/Tutorial/Sketch

5. Micro-controllers
   (a) **Digital pins:** Explains how digital pins can be configured. Click on the following link to learn more.
   Link: http://arduino.cc/en/Tutorial/DigitalPins

   (b) **Analog Input Pins:** A description of the analog input pins on an Arduino chip. Click on the following link to learn more.
   Link: http://arduino.cc/en/Tutorial/AnalogInputPins\

   (c) **PWM:** Pulse Width Modulation, or PWM, is a technique for getting analog results with digital means. Click on the following link to learn more.
   Link: http://arduino.cc/en/Tutorial/PWM

   (d) **Memory:** Learn the different types of memory on the Arduino Board. Click on the following link to learn more.
   Link: http://arduino.cc/en/Tutorial/Memory

6. Programming Technique
    (a) **Variables:** A variable is a place to store a piece of data. It has a name, a value, and a type. Click on the following link to learn more.
    Link: http://arduino.cc/en/Tutorial/Variables

    (b) **Functions:** The typical case for creating a function is when one needs to perform the same action multiple times in a program. Click on the following link to learn more.
    Link: http://arduino.cc/en/Reference/FunctionDeclaration

    (c) Libraries: Explains how to create a library for Arduino. Click on the following link to learn more.
    Link: http://arduino.cc/en/Hacking/LibraryTutorial

7. Circuits
    (a) Arduino on a Breadboard: how to build an Arduino compatible breadboard.
    Link: http://arduino.cc/en/Main/Standalone

D) **Examples:** Simple programs that demonstrate basic Arduino commands. These are included with the Arduino environment; to open them, click the Open button on the toolbar and look in the example folder. Click on the following link to see the examples that are found in the examples folder as well as additional examples.

Link: http://arduino.cc/en/Tutorial/HomePage

1. Core functions
    (e) Basics
    (f) Digital
    (g) Analog
    (h) Communication
    (i) Control Structures
    (j) Sensors
    (k) Display
    (l) Strings

2. Examples from other Libraries
    (a) EEPROM Library
    (b) Ethernet Library
    (c) Firmata Libraries
    (d) Liquid Crystal Library
    (e) SPI Library
    (f) Servo Library
    (g) Stepper Library
    (h) Wire Library
    (i) Arduino as ISP Programmer

E) **Language Reference:** A reference guide of the Arduino environment language and how to use them. Open the following link to access user reference and instructional materials that are necessary for gaining understanding of the workings of Arduino language and its various capabilities.

Link: http://arduino.cc/en/Reference/HomePage

F) **Arduino Troubleshooting:** Common problems and issues that occur with the Arduino and how to solve them. Open the following link to access to the Arduino troubleshooting guide.

Link: http://arduino.cc/en/Guide/Troubleshooting

IV. Arduino Libraries and Open Sources

A) **Arduino Libraries:** Libraries that are from Arduino's website.

Link: http://arduino.cc/en/Reference/Libraries

1. Standard Libraries
2. Contributed Libraries
    (a) Communication (networking and protocols)
    (b) Sensing
    (c) Displays and LED's
    (d) Frequency Generation and Audio
    (e) Motors and PWM
    (f) Timing
    (g) Utilities

3. Additional Libraries**:** Additional libraries are available at the Arduino Playground, which is a publicly edited wiki for Arduino.

    Link: http://arduino.cc/playground/Main/LibraryList

    (a) Audio
    (b) Interrupts
    (c) Color LCD
    (d) Timing
    (e) Schedulers
    (f) Input/ Output
    (g) Data Structures and Algorithms
    (h) StorageStorage
    (i) Communications
    (j) Testing and Utilities
    (k) Math
    (l) Strings

B) **Open Source Libraries and Downloads**

The Arduino fan-base is widespread, and therefore there are an almost infinite number of Open Sources across the Internet. These include web-sites, blogs, forums, code-downloads, and YouTube videos to name a few. Listed below are a few of our favorites that you can check out; but we highly recommend you go explore the availability of information at your fingertips across the web.

1.Websites
    (a) http://en.wikipedia.org/wiki/Arduino
    (b) http://www.ladyada.net/learn/arduino/
    (c) http://www.sparkfun.com/products/666
    (d) http://www.arduino.cc/cgi-bin/yabb2/YaBB.pl

(e) http://www.adafruit.com/products/50
(f) http://www.freeduino.org/
(g) http://oreilly.com/arduino/index.html
(h) http://littlebirdelectronics.com/collections/arduino-board

2. Blogs
   (a) http://hacknmod.com/hack/top-40-arduino-projects-of-the-web/
   (b) http://blog.makezine.com/arduino
   (c) http://www.instructables.com/technology/arduino/
   (d) http://www.practicalarduino.com/
   (e) http://www.instructables.com/id/Arduino-Projects/

3. Forums
   (a) http://www.arduino.cc/cgi-bin/yabb2/YaBB.pl
   (b) https://sites.google.com/site/projectsentrygun/home

4. Videos
   (a) Jeremy Blum's Arduino Tutorials:
   http://www.youtube.com/user/sciguy14#grid/user/A567CE235D39FA84
   (b) Make Magazine's video series:
   http://www.youtube.com/user/makemagazine#grid/user/C3A0E18A403665FE
   (c) The Arduino Documentary
   http://redux.com/stream/item/1909904/Arduino-The-Documentary-2010-English-HD
5. Code downloads
   (a) LCD I2C screen
   http://arduino-info.wikispaces.com/LCD-Blue-I2C
   (b) Simple keypad
   http://bildr.org/2011/05/arduino-keypad/
   (c) Additional serial ports
   http://arduiniana.org/libraries/NewSoftSerial/

V. Accessing Libraries

A) **MINDS-i Libraries**

For installation of the MINDS-i Libraries and Examples for the Arduino program follow these steps. If you installed the Arduino Program from the disk that was included in your MINDS-i kit the following steps were already completed.

To access the MINDS-i libraries from the Arduino Environment, you'll need to open the Arduino Program file where it is saved on your computer. If you do not know where the Arduino program file was located, then you can search for the program using the "search" function on your computer typing in the keyword "arduino".

To import the MINDS-i Library:

1. Open the folder named "arduino-1.0",
2. Then open the "libraries" folder therein.
3. Paste the "MINDSi" library folder located on the disk, into the "libraries" folder on your computer.

Now you can import the MINDS-i library by typing "#include <MINDSi.h>" at the top the open sketch or by clicking "Sketch" > "import library" > "MINDSi".

To import the MINDS-i examples:

1. Open the folder named "arduino-1.0",
2. Then open the "examples" folder therein.
3. Paste the "01. MINDS-i" examples folder located on the disk, into the "examples" folder on your computer.

Now to open a MINDS-i example click "File" > "examples" > "01. MINDS-i" and choose an example.

VEHICLE TROUBLESHOOTING GUIDE

PROBLEM	CAUSE	SOLUTION
Wheels rotate backwards when forwards throttle is applied.	1) One or more of your differentials are installed upside down.	1)Check the assembly instructions to make sure the differentials were installed correctly.
	2) The motor is spinning the wrong direction.	2) Check that the wires from the ESC to the motor are connected properly.
	3) The motor case was installed with the motor facing opposite of what the instructions show.	3) Check the assembly instructions to make sure the motor case was installed correctly.
	4) The "throttle" switch on your remote control may be switched to reverse.	4) Flip the switch to "Normal"
Front wheels rotate opposite of rear wheels when driving.	1) One of your differentials are installed upside down.	1) Check the assembly instructions for proper installation of the differentials.
Vehicle doesn't drive in a straight line with the steering channel centered.	1) Your steering trim is not centered.	1) Center your steering trim.
	2) Your servo horn was not installed on the servo while it was centered.	2) Remove the screw that retains the servo horn, then remove the servo horn. Turn the remote control on, center the trim on the remote, turn the vehicle on. Then reinstall the servo horn as the instructions show.
	3) The linkages that connect your steering bar to the front wheel knuckles are not the proper length or do not match each other.	3) Remove the linkages and check that they are the length indicated in your instructions.
	4) The "Y"harness for four wheel steering is not centered.	4) Follow step 2 for front and rear servo, making sure when you center the trim on the remote to also center the trim on the "Y"harness.
Servo does not operate when steering is applied.	1) Servo cable is plugged in upside down.	1) Check that the cable is plugged in according to the wiring polarity guide.
Servo steers only one direction or farther one way than the other.	1) Servo horn is not centered on the servo properly.	2) Remove the screw that retains the servo horn, then remove the servo horn. Turn the remote control on, center the trim on the remote, turn the vehicle on. Then reinstall the servo horn as the instructions show.
Vehicle has become under powered or sluggish.	1) Charge level of vehicle battery has dropped below useable level.	1) Remove battery from vehicle and charge with the recommended charger. Always follow proper safety procedures when charging a battery.
Vehicle stops unexpectedly or doesn't respond to remote.	1) Charge level of remote control batteries have dropped below useable level.	1) Check the remote control low battery indicator, if necessary replace or recharge batteries. Always follow proper safety procedures when charging a battery. Never recharge alkaline batteries.
	2) A wire may have come loose from the receiver.	2) Check that the wires are all properly plugged in and in good condition.
Vehicle does not move forward or reverse when throttle is applied but audible tone comes from motor	1) Something has become bound up in the drivetrain.	1) Check the drivetrain from motor to wheels for foreign objects and debris. Removing the drivelines that connect the motor case to the differential(s) then trying to spin the wheels will allow you to more easily locate the issue.
Motor spins freely when throttle is applied but vehicle doesn't move.	1) Gear mesh in motor case was not properly set.	1) Check that the gear mesh was set according to the instructions. Gears should be about the thickness of this sheet of paper apart for proper mesh.
	2) Pinion set screw was not Properly tightened.	2) Check that the pinion set screw was tightened according to the instructions.

ARDUINO TROUBLESHOOTING GUIDE

PROBLEM	CAUSE	SOLUTION
Vehicle is not behaving as programmed.	1) Charge level of vehicle battery has dropped below useable level.	1) Remove battery from vehicle and charge with the recommended charger. Always follow proper safety procedures when charging a battery.
	2) Inputs (sensors) or Outputs (servos, ESC, etc) are wired improperly.	2) Check that each device connected to the Arduino Sensor Shield is plugged in properly. Both in polarity and into the proper port the program states.
	3) Sensor is malfunctioning.	2) Check #2, Connect Arduino to computer using USB cable, run the calibration program for each sensor on the vehicle. If the sensor value is not present, uncharacteristic or erratic the sensor may need to be replaced.
Serial port "COM_" not found.	1) The Arduino is not connected to the computer.	1) Connect the supplied USB cable and try again.
	2) The proper serial port was not selected in the Arduino Program.	2) Disconnect, re-connect the USB Cable then go to Tools > Serial Port then select the Arduino Uno.
Light on Ping))) sensor is not cycling.	1) The Ping))) sensor is not wired properly.	1) Check the wiring polarity guide for proper installation.
	2) The Ping))) sensor in not connected to the proper port.	2) Check that the sensor is plugged into the port stated in the code.
QTI sensors not detecting line.	1) The QTI sensor is not wired properly.	1) Check the wiring polarity guide for proper installation.
	2) The QTI sensor in not connected to the proper port.	2) Check that the sensor is plugged into the port stated in the code.
Power light on Arduino Sensor Shield doesn't turn on when ESC is powered on.	1) One of the connections to the Sensor Shield is shorted.	1) Check that each device connected to the Arduino Sensor Shield is plugged in properly. Both in polarity and into the proper port the program states. If the problem persists disconnect one sensor at a time till the faulty connection (sensor, servo, ESC etc.) is found, then replace.
	2) No power source is connected to the Arduino	2) Check to make sure that it is either plugged into your computer with the USB cable or that you are using a charged battery and the ESC cable is properly connected.

http://arduino.cc/en/Guide/Troubleshooting