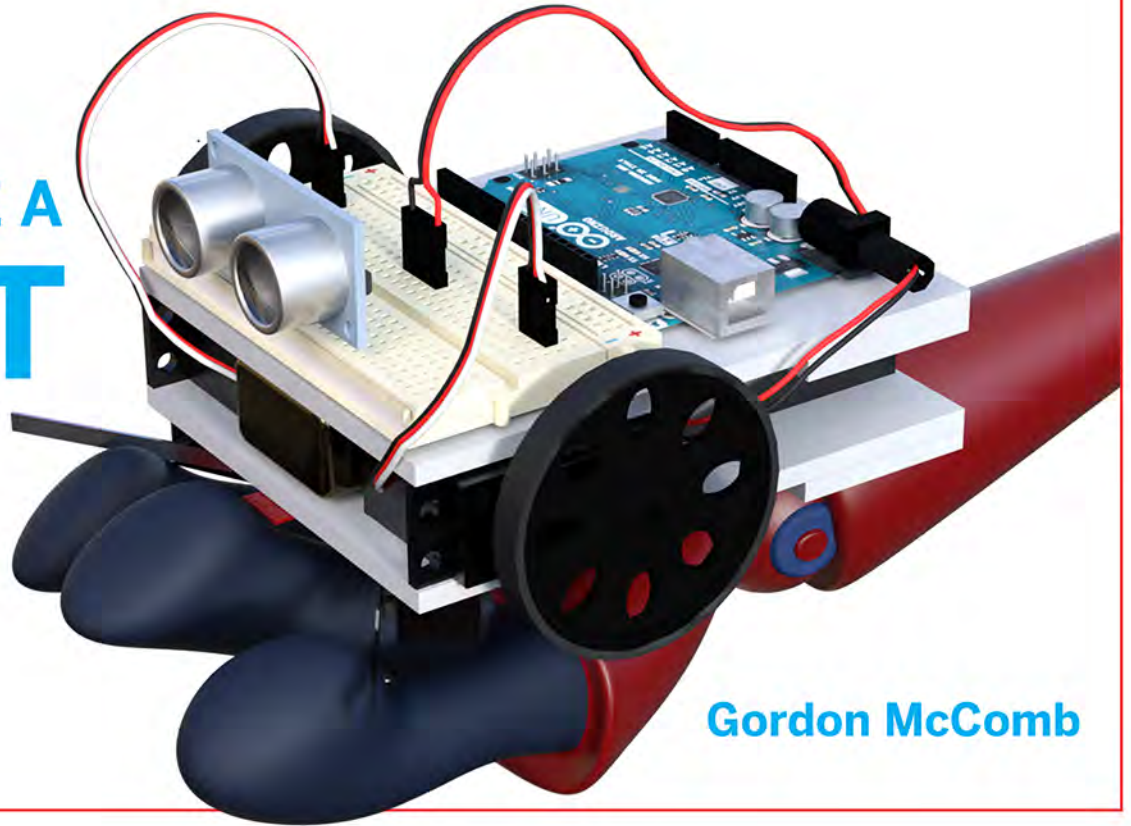


Make:

HOW TO MAKE A ROBOT



Gordon McComb

Handbook

How to Make a Robot

Gordon McComb

Maker Media, Inc.

San Francisco

How to Make a Robot

Gordon McComb

Copyright © 2018 Maker Media. All rights reserved.

Printed in the United States of America

Published by Maker Media, Inc., 1700 Montgomery Street, Suite 240, San Francisco, CA 94111

Maker Media books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (safaribooksonline.com). For more information, contact our corporate/institutional sales department: 800-998-9938 or corporate@oreilly.com. Customers may purchase kits, books, and more directly from us at Maker Shed.

Publisher: Roger Stewart

Illustration and Design: Gordon McComb

Proofreader: Jennifer Meredith

Technical Reviewer: Jon Hapeman

Cover Design: Julie Cohen

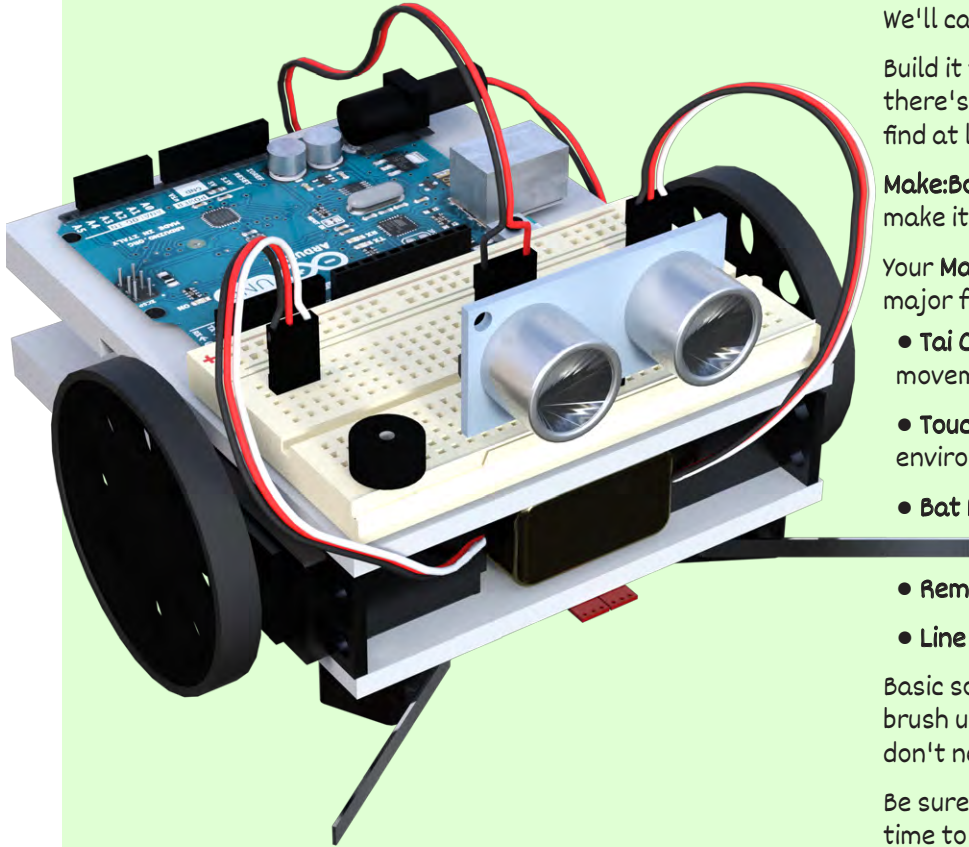
March 2018: First Edition

Revision History for the First Edition: 2018-3-15

Make:, Maker Shed, and Maker Faire are registered trademarks of Maker Media, Inc. The Maker Media logo is a trademark of Maker Media, Inc. *How to Make a Robot* and related trade dress are trademarks of Maker Media, Inc. Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. While the publisher and the author have used good faith efforts to ensure that the information and instructions contained in this work are accurate, the publisher and the author disclaim all responsibility for errors or omissions, including without limitation responsibility for damages resulting from the use of or reliance on this work. Use of the information and instructions contained in this work is at your own risk. If any code samples or other technology this work contains or describes is subject to open source licenses or the intellectual property rights of others, it is your responsibility to ensure that your use thereof complies with such licenses and/or rights.

978-1-680-45469-7 [LSI]

The author thanks OpenClipart, ShareCG, Daz, Turbosquid, FirstCAD, and other curators of open source and commercial art materials used in the preparation of this book,



Let's Build a Robot!

Here's how to make a real robot that follows your commands! We'll call it **Make:Bot**.

Build it from scratch using common electronic parts—there's a list on Page 50—plus tools and materials you can find at local craft and hardware stores.

Make:Bot uses the Arduino programmable microcontroller to make it smart. The Arduino is inexpensive and easy to use.

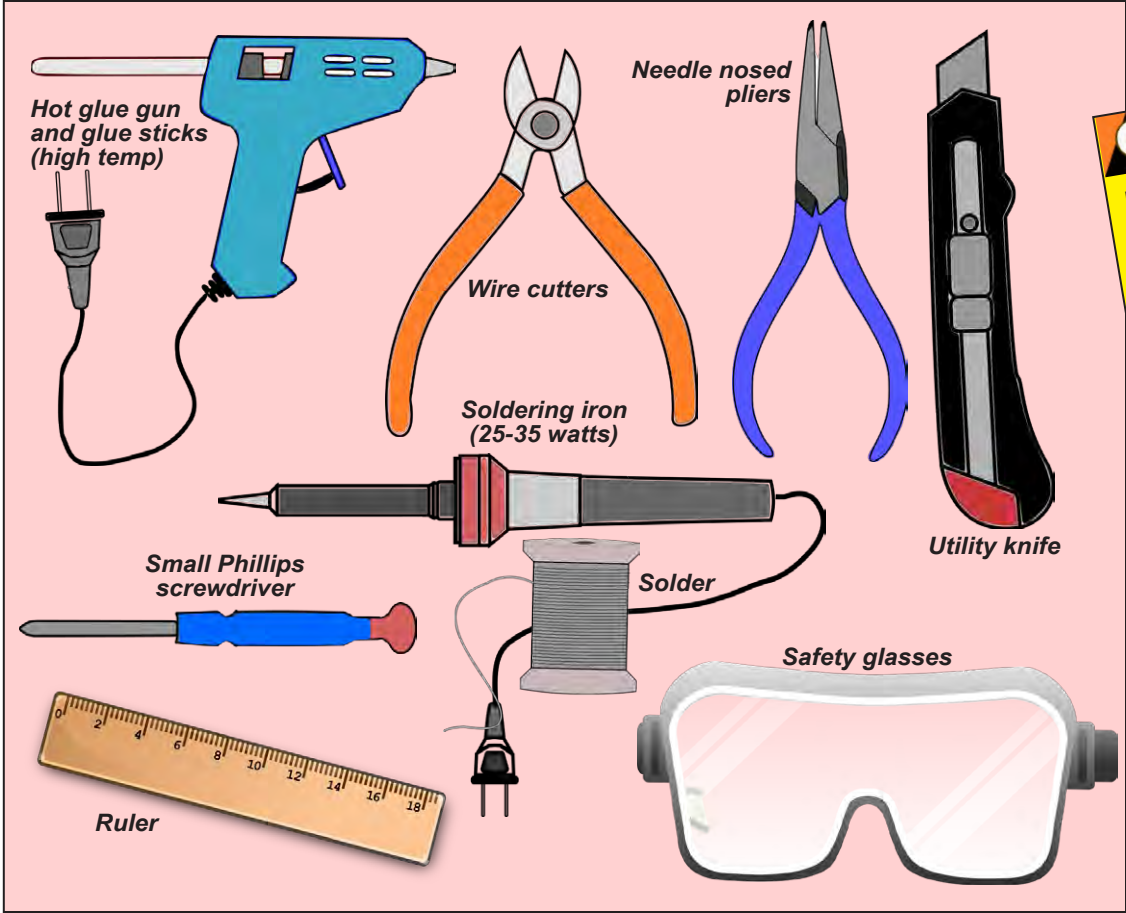
Your **Make:Bot** is really five robots in one, which show five major functions of robots:

- **Tai Chi Bot** replicates a set pattern of pre-programmed movements.
- **Touch-and-Go Bot** uses "whiskers" to navigate its environment.
- **Bat Bot** sends out sound waves to avoid obstacles.
- **Remote Bot** lets you use a TV remote to control it.
- **Line Bot** follows a line drawn on poster board.

Basic soldering skills are recommended, and you'll want to brush up on your Arduino programming skills. Otherwise, you don't need any prior experience to create your own **Make:Bot**!

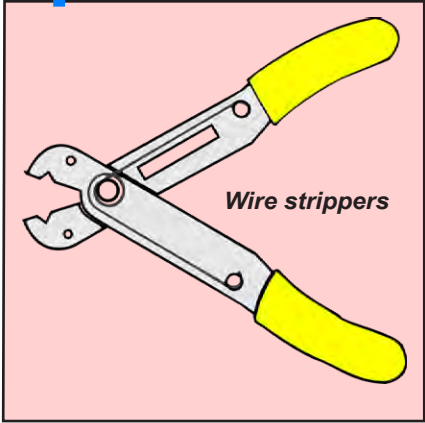
Be sure to read through each project before you start. Okay, time to begin!

Tools You'll Need



CAUTION
Utility knives are very sharp! Use only with a protective straight-edge. Cut on a scrap piece of cardboard to protect surfaces. Use safety glasses at all times.

Optional

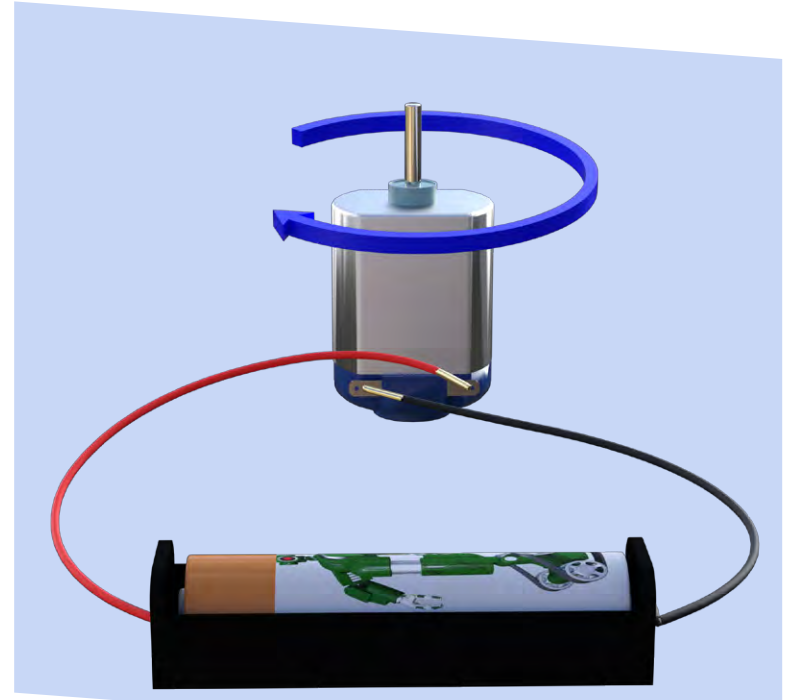
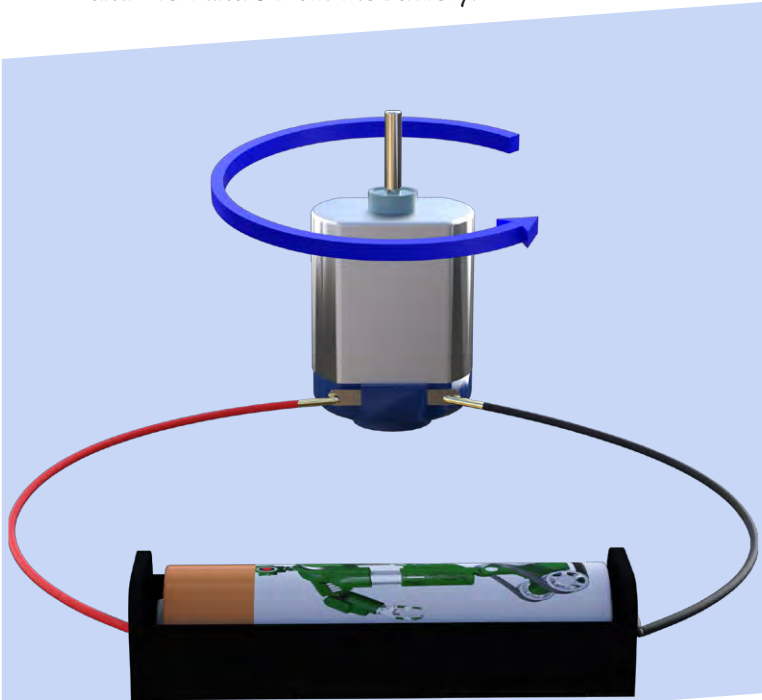


Project 0

Learn Robot Motor Basics!

The ability to travel from one place to another is an essential characteristic for many robots. This movement is often provided by **electric motors**. Applying **voltage** to a motor makes it spin, and this moves the robot.

A **battery** or **battery pack** provides power to the motor. To change the direction of the motor simply reverse the + and - terminals from the battery.



Two important aspects of motor operation are speed and torque.

Speed is the rotational velocity of the motor, typically specified in revolutions per minute (RPM). **Torque** is the amount of force the motor exerts.



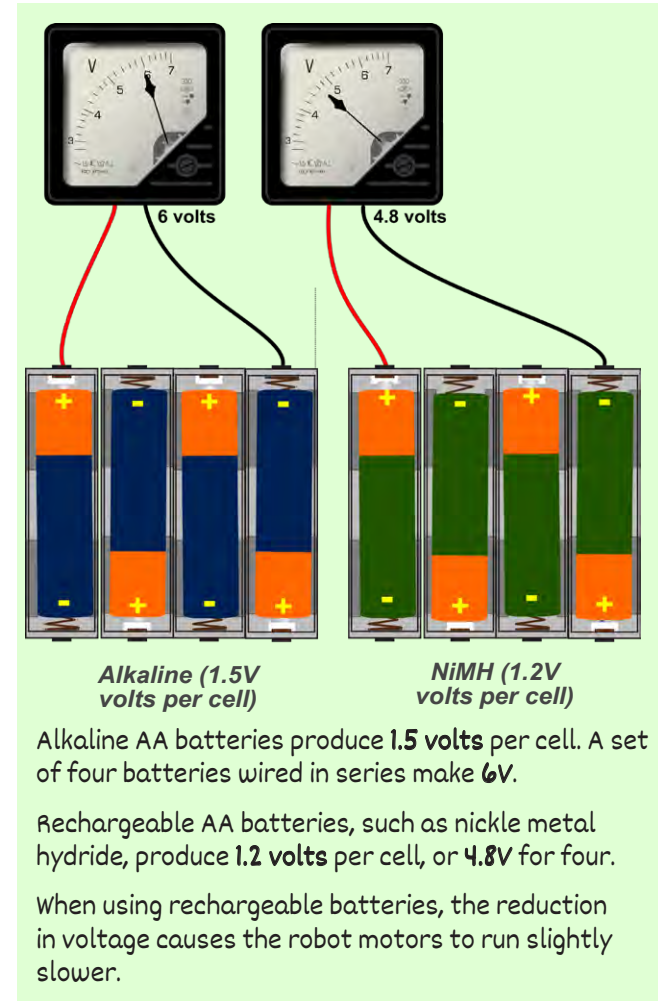
The best way to increase the speed and torque of a motor is to add more batteries in series.

Put four batteries in a four-cell AA holder, making sure the positive and negative terminals are facing the proper direction. Connect the motor to the battery holder.

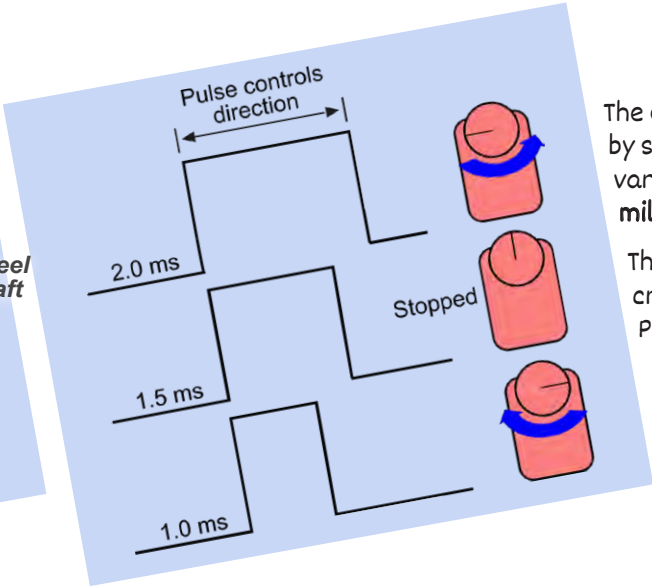
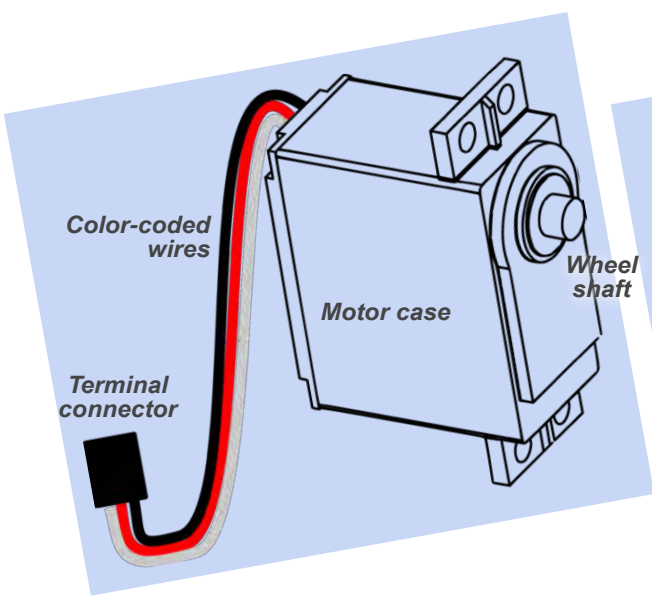
What happens?

The motor spins about four times faster because it's now getting 4X the voltage.

The motor also has **more torque**. You need the extra torque to move your robot around!

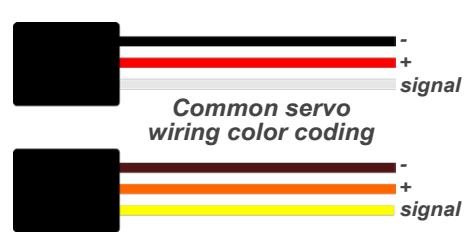


The Make:Bot uses a special type of motor, a **servo motor** intended for radio control (RC) models and toys. Unlike traditional RC servo motors, which are limited to turning only through a 90° (or so) arc, servos for robotics turn continuously in either direction.



The operation of the motor is set by sending a series of pulses of varying durations, from 1 to 2 **milliseconds**.

The control signal can be easily created using an Arduino. See Project 4.



CAUTION

Never EVER reverse the polarity of the battery connections to the servo, or damage may result!

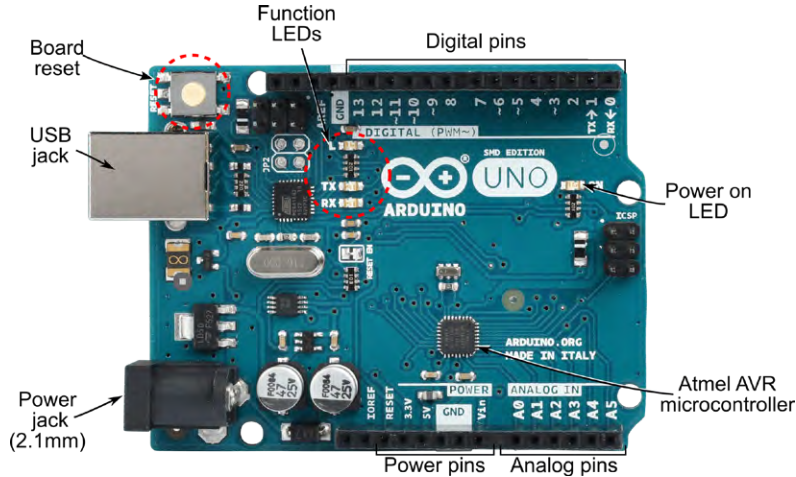
Core benefit of RC motors: They can be controlled directly from the Arduino, without the need for additional power circuitry.

It's also easy to attach wheels to RC servos, and to mount them to your robot.

Project 1

Setting Up the Arduino Brain

An **Arduino microcontroller** operates all the functions of the Make:Bot. Software running inside the Arduino determines what the robot is supposed to do. **Input/output** (or **IO**) **pins** on the Arduino provide connections to your robot's hardware. Separate **power** pins allow you to provide electrical juice to the components you add.



The Arduino is available in numerous models from different companies. I'm using an **Arduino UNO**. See Page 50 for a complete list of Make:Bot parts.

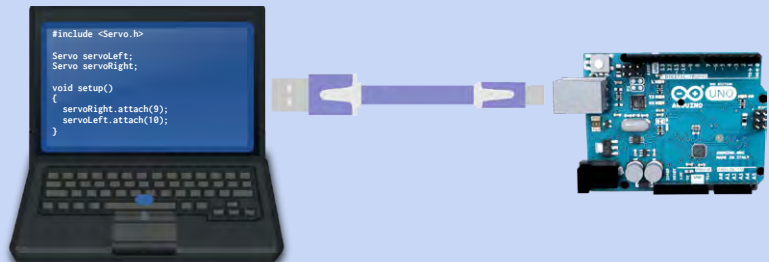
Placement of components on the Arduino can vary between models. Your board may look a little different than shown, but that's okay as long as it's UNO compatible.

The USB jack can differ among boards. Be sure to get the right kind of cable for your Arduino and computer!

The Arduino is programmed from your computer using a **USB cable**. The cable also provides power to the Arduino when not using batteries. Programming software for the Arduino is available for Windows, Macintosh, and Linux. Get a copy from

www.arduino.cc

and follow the directions provided for downloading and installing.



Before building your robot, connect and test your Arduino microcontroller, and get used to how it works.

- 1 If you haven't done so already, go to www.arduino.cc and download the Arduino software installer for your computer operating system.

TIP: Unless otherwise instructed, install the software **before** plugging in the Arduino.

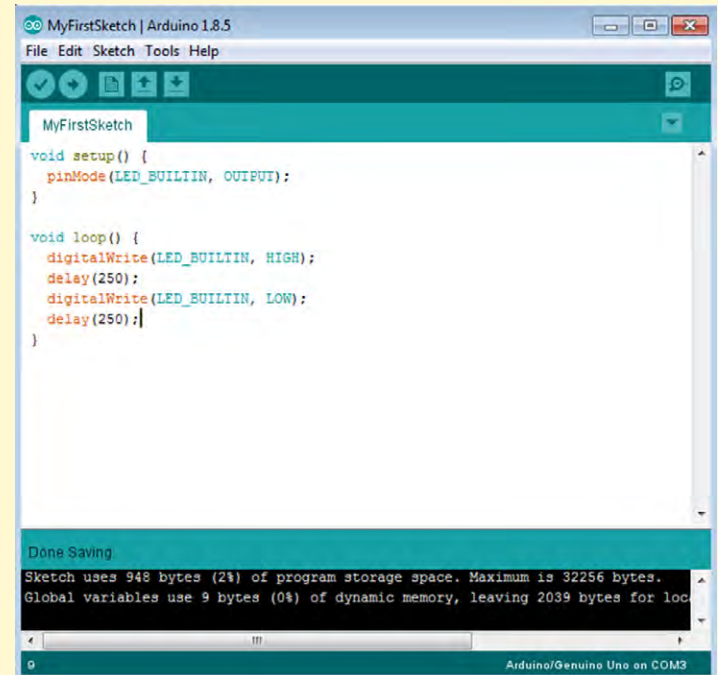
- 2 Install the software and follow any prompts that are displayed. You may need to also install one or more drivers.

- 3 Use the USB cable to plug your Arduino into your computer. You should see the Arduino's On light turn on.



- 4 Open the Arduino software and type in the sketch you see below. Be sure to type it EXACTLY!

```
void setup() {  
  pinMode(LED_BUILTIN, OUTPUT);  
}  
  
void loop() {  
  digitalWrite(LED_BUILTIN, HIGH);  
  delay(250);  
  digitalWrite(LED_BUILTIN, LOW);  
  delay(250);  
}
```



Before you can program your Arduino, you must first download and install the **Arduino desktop IDE software** (IDE stands for **I**ntegrated **D**evelopment **E**nvironment).

You use the IDE to create your robot's control programs, called **sketches**.

- 5 Click the **Save** button and name the sketch **MyFirstSketch**.
- 6 Click the **Verify** button to check your sketch for errors. If an error is reported, double-check your typing.
- 7 Click the **Upload** button in the IDE program. This "compiles," then uploads your new sketch into the Arduino's memory.
- 8 During upload, verify everything is working by watching the **TX** and **RX** lights flash.
- 9 When the upload is complete, you should see the **L** light rapidly flash on and off.



See the **Getting started** page on the Arduino Web site for important first-time installation tips and helpful resources for using the IDE.

Common IDE icons you should know:



Save



Verify



Upload

An error occurred while uploading the sketch

Copy error messages

```
avrdude: read from file 'C:\Users\USER\AppData\Local\Temp\arduino_build_126108/  
An error occurred while uploading the sketch
```

Uh oh! If you get an error instead of a successful upload, you'll need to determine the cause. Check these for starters:

- USB cable plugged in
- Correct port selected
- Correct board selected

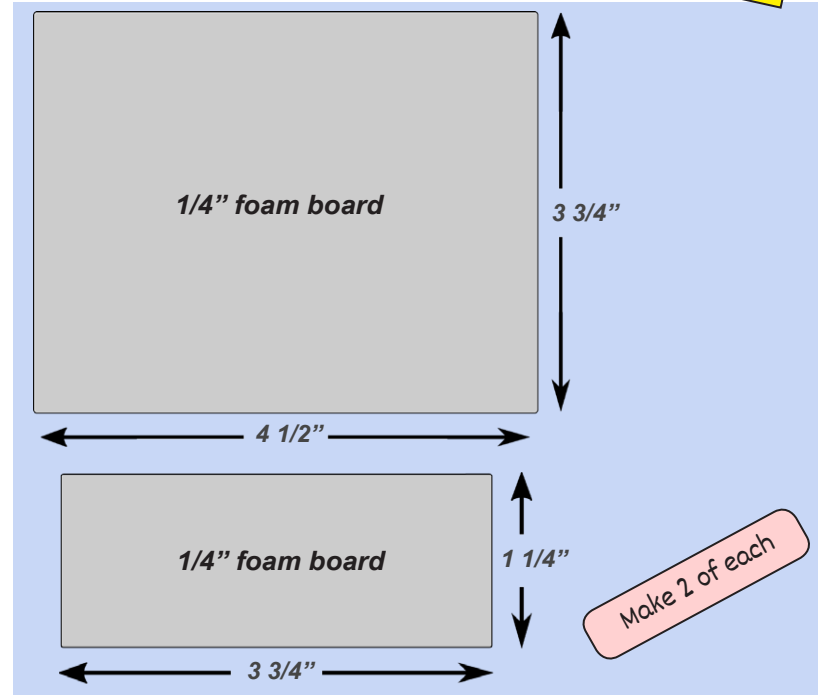
When using the IDE, be sure the correct **communications port** and **Arduino board model** are selected before you upload your sketch. Otherwise an error may occur.

Project 2

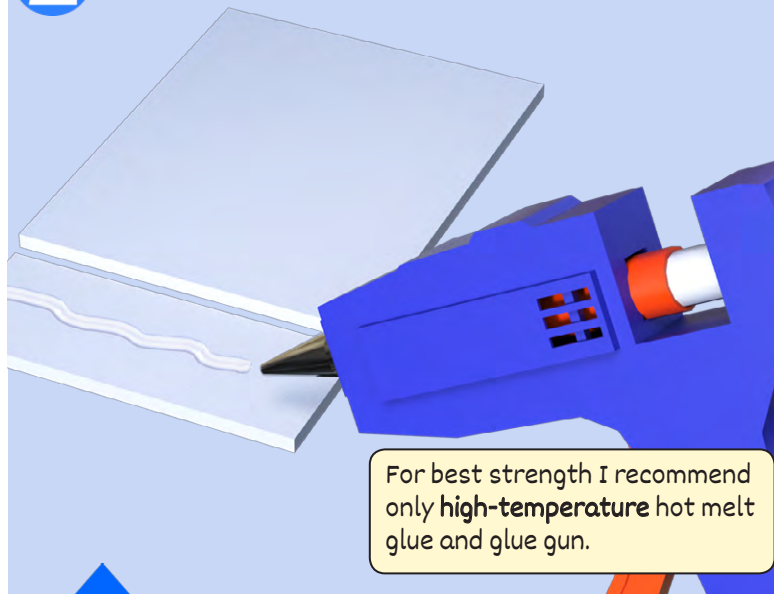
Make a Motorized Robot Base

The body of the Make:Bot uses commonly available 1/4" thick **foam board**. Cut the board with a utility knife and a metal or wood straight edge.

Everything is pieced together using **hot melt glue**, **double-sided tape**, and **Velcro** (or other hook-and-loop fastener).

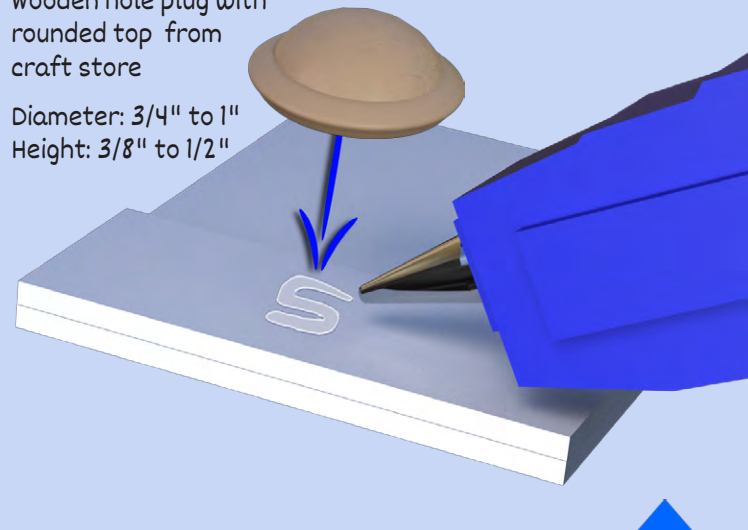


2 Assemble top and bottom bases



3 Attach tail skid

Wooden hole plug with rounded top from craft store
 Diameter: $\frac{3}{4}$ " to 1"
 Height: $\frac{3}{8}$ " to $\frac{1}{2}$ "



Apply a bead of hot glue to the smaller foam board. Sandwich the pieces together so the edges are flush.

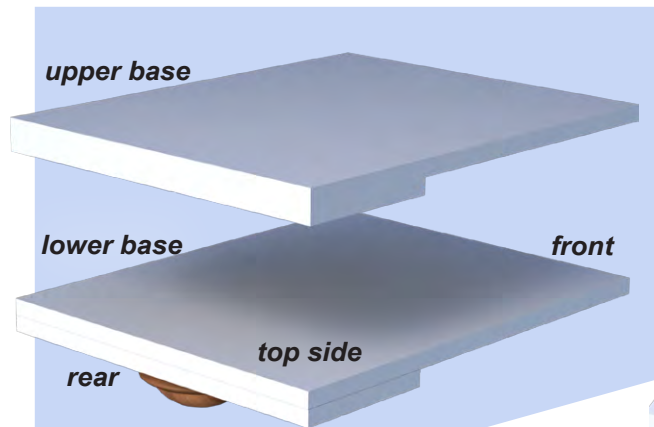
Repeat for the second set of pieces.

CAUTION

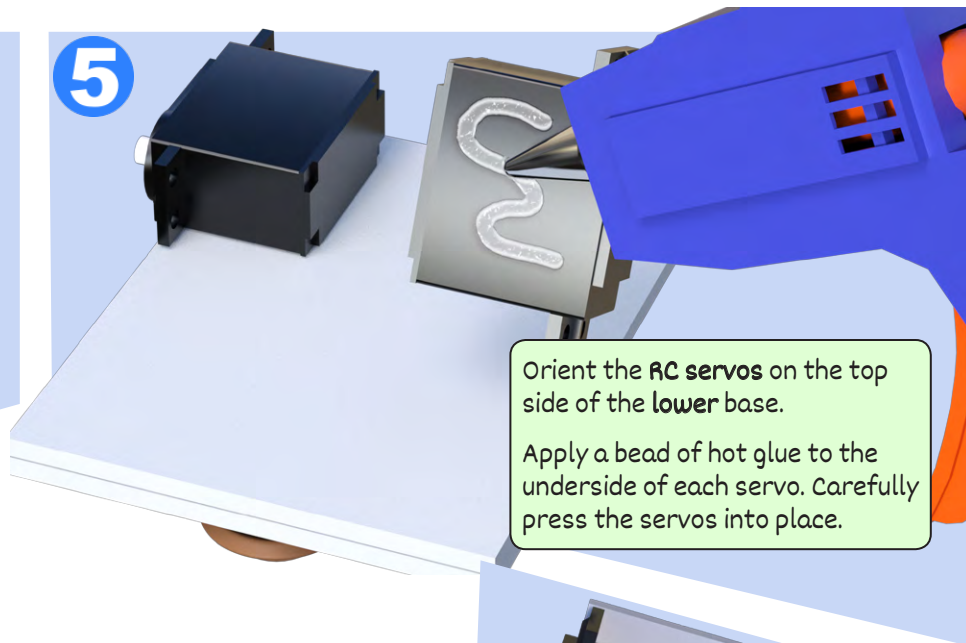
When gluing, wear eye protection, and remember that the glue is very HOT!

Using an assembled base piece, apply S-shaped bead of glue to the center of the board. Position the **wooden hole plug** and press for five seconds.

Do this for only one set of pieces.



5



Orient the **RC servos** on the top side of the **lower base**.

Apply a bead of hot glue to the underside of each servo. Carefully press the servos into place.

4

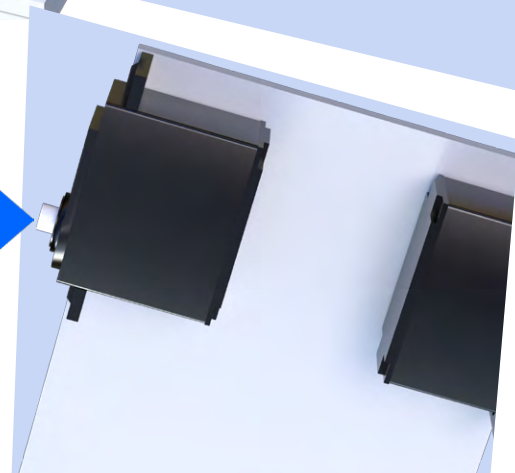
You should now have two bases: lower base with the wood plug, and upper base with nothing on it.

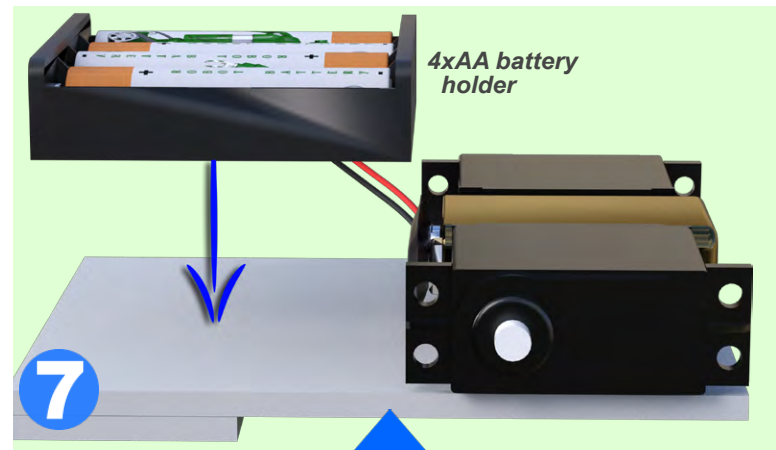
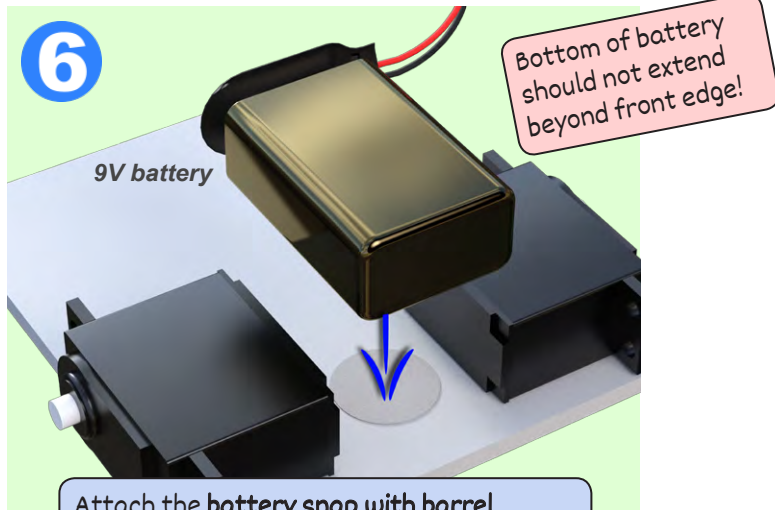
The Make:Bot is composed of two bases, a **lower base** (motors and batteries), and an **upper base** (Arduino and other electronics).

Everything is held together using **hot melt glue** or **double-sided tape** for the permanent stuff. **Velcro tabs** are used for the things that may need to be removed.

When gluing the servos make sure:

- The mounting flanges are **flush** with the front and sides of the base
- The white shaft is closest to the **rear** of the base (nearest to the wooden plug)





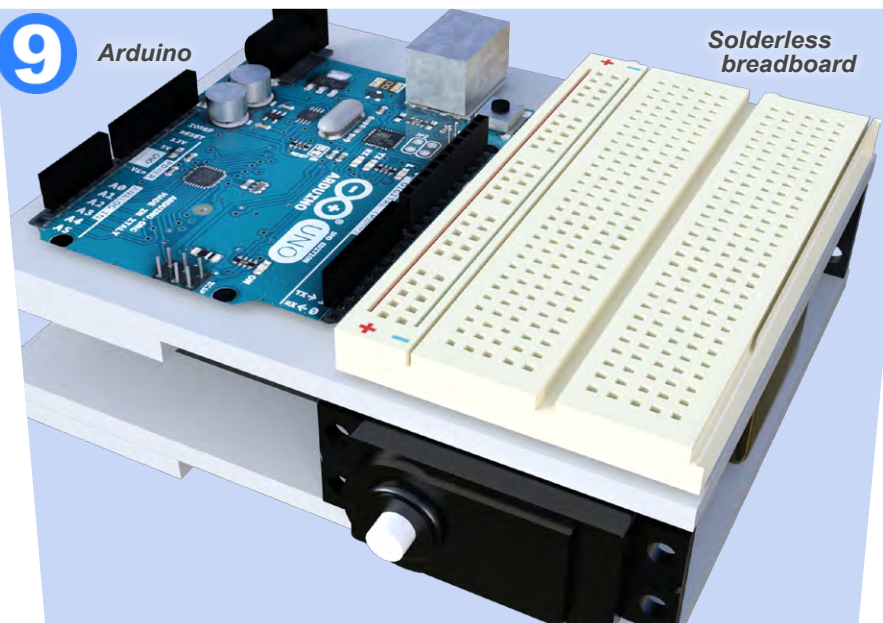
Peel off the backs of two 1" Velcro tabs. Stick the tabs to the **tops** of the servos.

Carefully position the upper base over the servos, so that its edges are even with the lower base. Press down firmly to seat the adhesive of the Velcro tabs.

9

Arduino

Solderless
breadboard

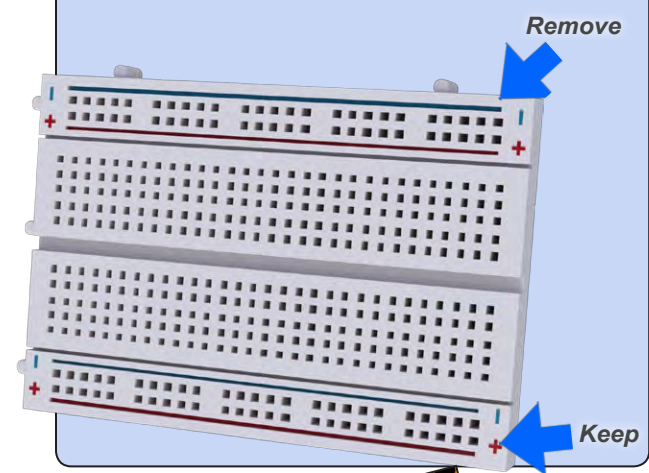


The **solderless breadboard** comes with its own double-sided tape. Peel off the backing of the tape. Apply the breadboard to the top of the upper deck.

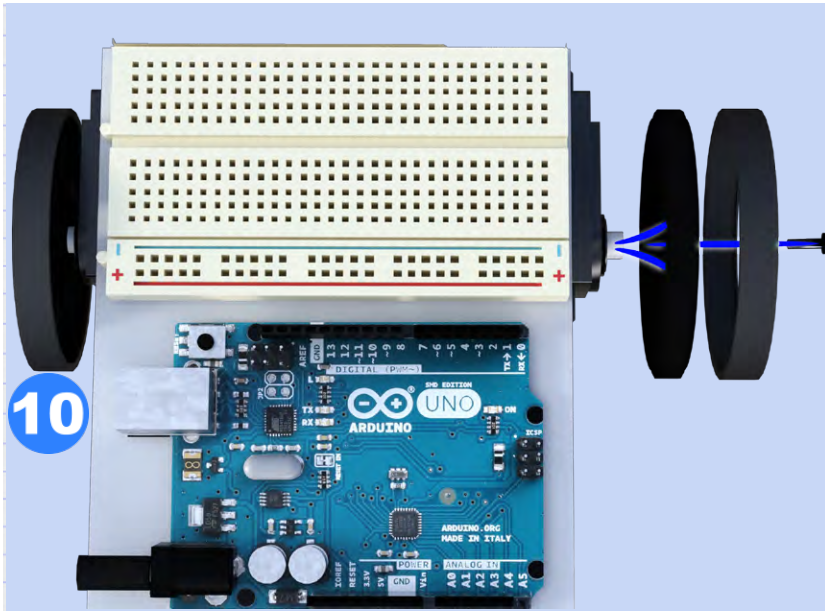
The breadboard should be flush with the front edge of the upper deck.

Apply a Velcro tab to the underside of the **Arduino**. Stick the Arduino to the upper base so its edge is about 1/8" from both the breadboard and the right side of the upper base..

The breadboard comes with two removable **power rails**. You'll only use one rail. Slide off the rail shown, and put it away for safekeeping.



CAUTION
Avoid taking off the double sided tape from the breadboard, or the breadboard may be damaged.
If the tape must be removed, use a scraper rather than pulling the tape.

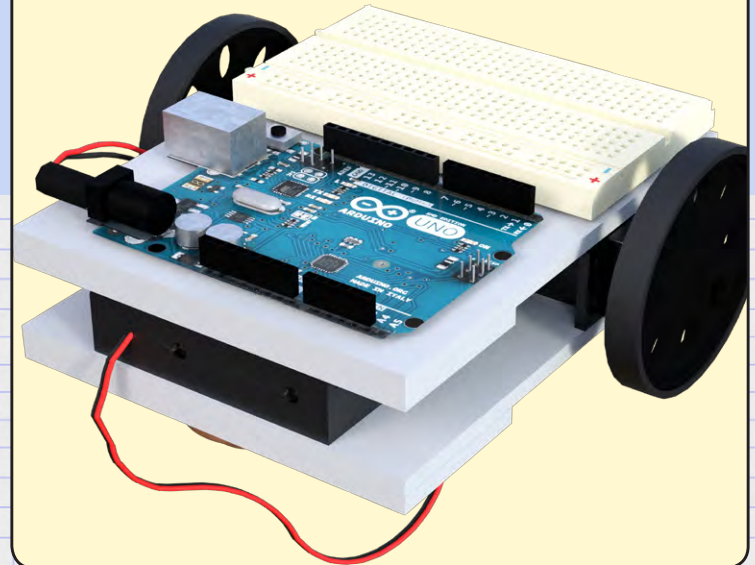


10

For each servo, place the **rubber tire** over the wheel. It's a tight fit.

Using the screw provided, mount the wheel and tire to the servo. The wheel and servo shaft are splined for a positive fit. Don't overtighten the screw!

At this point your Make:Bot will look like this:



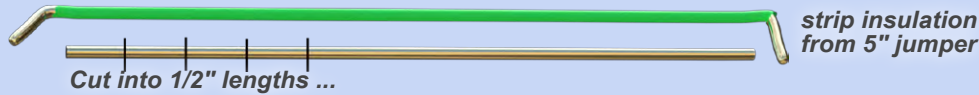
Your Make:Bot is almost complete. Congratulations!

To make it fully functional, you still need to wire up the batteries, servos, and Arduino.

That comes next, but this is a good time to **review your work**. Check for any loose parts, and fix problems now!

Project 3

Wire Your Robot's Motors and Batteries

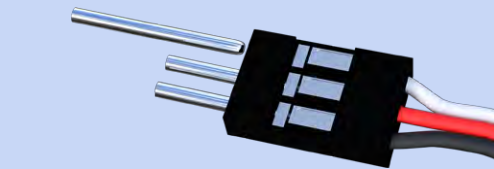
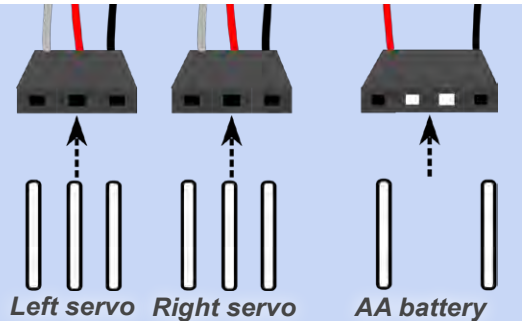


1

strip the insulation from a 5" solid-wire jumper. Cut into 1/2" lengths to make 8 (eight) short jumpers.

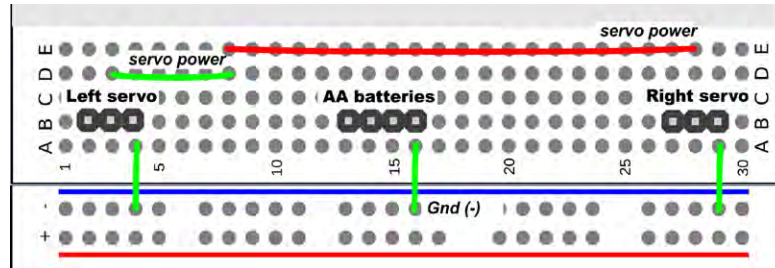
CAUTION

Be sure to observe the correct polarity of all connections! Otherwise your servos may be irreversibly damaged!



2

Use pliers to push the short jumpers into the servo and AA battery holder connectors.



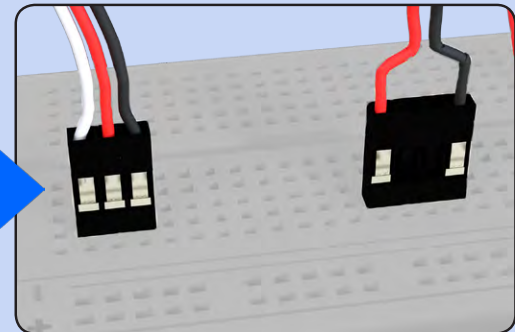
Plug the servo and AA battery connectors into the breadboard.

Double-check your work when you're through!

Wire color coding for power and servos is **SUPER-CRITICAL!**

The ground (black) wire is always on the right side of the connector.

3



Connect four male-to-male jumpers.

4

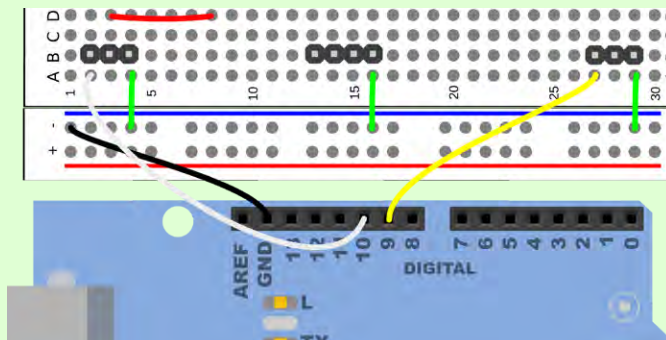
Color coding suggestions:

Left Servo: White (Arduino Digital Pin 10 to breadboard A2)

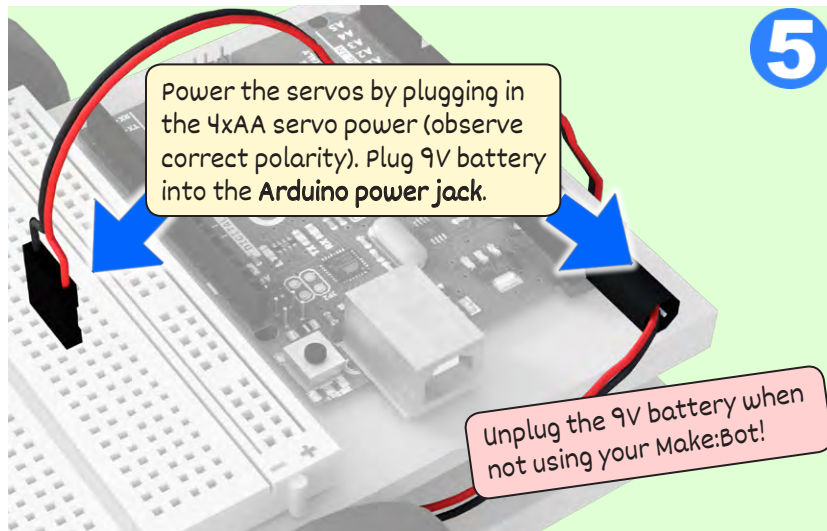
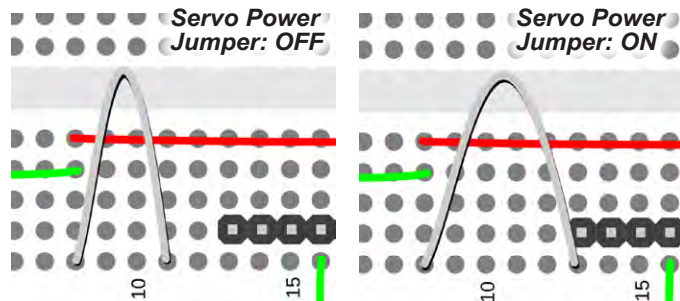
Right Servo: Yellow (Arduino Digital Pin 9 to breadboard A27)

Gnd (Ground): Black (Arduino GND to breadboard negative rail)

Servo Power Jumper: Gray (breadboard A8-A11 or A8-A13)

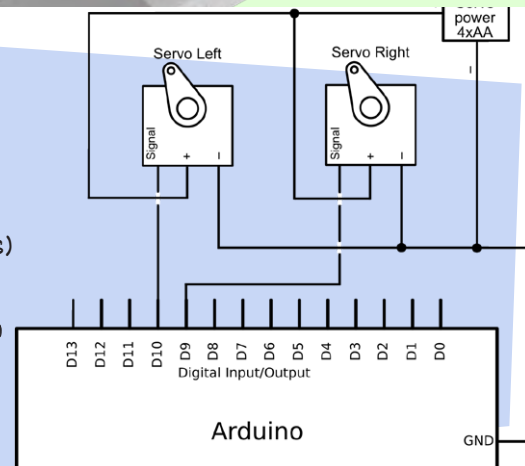


Use a jumper to connect the AA battery power for the servos. Disconnect when not using servos.



Make:Bot uses two battery sources: a **9V battery** that runs the Arduino, and **4xAA batteries** (4.8 or 6 volts) for the servos.

This **schematic diagram** shows how things are wired up.



Project 4

Program Your Robot to Repeat Steps

Command your **Make:Bot** to repeat basic movements—the first step in creating an autonomous robot.

ServoTest

```
#include <Servo.h>

Servo servoRight;
Servo servoLeft;

void setup()
{
  servoRight.attach(9);
  servoLeft.attach(10);
}

void loop()
{
  servoRight.write(0);    // Fwd
  servoLeft.write(180);
  delay(2000);

  servoRight.write(180);  // Rev
  servoLeft.write(0);
  delay(2000);

  servoRight.write(180);  // Right
  servoLeft.write(180);
  delay(2000);

  servoRight.write(0);    // Left
  servoLeft.write(0);
  delay(2000);
}
```

Arduino programming is based on **objects**. Objects extend the functionality of the Arduino. This code creates two **Servo** objects and gives them unique names.

All Arduino sketches have a **setup() function**. Code here is run once, when the Arduino is first powered on.

The code tells the Arduino you have two servos, connected to pins 9 and 10.

All sketches have a **loop() function**. Code repeats while the Arduino is powered.

Servos are controlled by specifying a direction (in degrees), either 0 or 180. By alternating the directions the robot moves forward and backward and turns.

// is a **comment**, a note just for you. The Arduino ignores it.

delay(2000) pauses sketch execution for 2000 milliseconds (2 seconds).

The Arduino programming language is **case sensitive**. An error will result if you use the wrong capitalization.

Try this sketch yourself!

1. Type the sketch in the Arduino IDE, and name it **ServoTest**.
2. Move the servo power jumper to the **OFF** position.
3. Upload the sketch to your Arduino, then move the servo power to the **ON** position.
4. Hold the robot in your hands and watch the servos change direction!

With a bit more code, you can extend the functionality of the basic **ServoTest** sketch. **ServoTaiChi** reorganizes some of the code and makes the sketch much more flexible! The sketch is available on the **Make:Bot Support Site**; see Page 50 for details.

ServoTaiChi

```
#include <Servo.h>

Servo servoRight;      // Define right servo
Servo servoLeft;       // Define left servo

void setup() {
  servoRight.attach(9); // Right servo pin 9
  servoLeft.attach(10); // Left servo pin 10
}

void loop() {
  goForward();
  delay(2000);
  goReverse();
  delay(2000);
  goRight();
  delay(2000);
  goLeft();
  delay(2000);
  allStop();
  delay(2000);
}

// Motion Routines: forward, reverse, turn, stop
void goForward() {
  servoRight.write(0);
  servoLeft.write(180);
}
```

A **function call** references a user-defined function, and can be used multiple times.

Statements are commands that tell the Arduino what to do. There's usually one statement per line.

The **semicolon** character is the **statement terminator**.

The **{** and **}** characters form a **block**, part of Arduino's coding **syntax**. Blocks contain multiple statements, and are treated as a unit.

ServoTaiChi, continued...

```
void goReverse() {
  servoRight.write(180);
  servoLeft.write(0);
}

void goRight() {
  servoRight.write(180);
  servoLeft.write(180);
}

void goLeft() {
  servoRight.write(0);
  servoLeft.write(0);
}

void allStop() {
  servoRight.write(90);
  servoLeft.write(90);
}
```

1. Upload sketch then remove USB cable from Arduino.

2. Connect 9V Arduino power and move servo power jumper to the ON position.

3. Depress Reset button, place your Make:Bot on floor (wood or linoleum is best), and release Reset button.

Methods are a special type of statement that uses an **object**, such as a servo. Methods are composed of the **object name**, a period, and the **action** to perform.

Many statements can be contained in a **user-defined function**.

When done experimenting, be sure to remove the power to the Arduino. Move the servo power jumper to the OFF position.

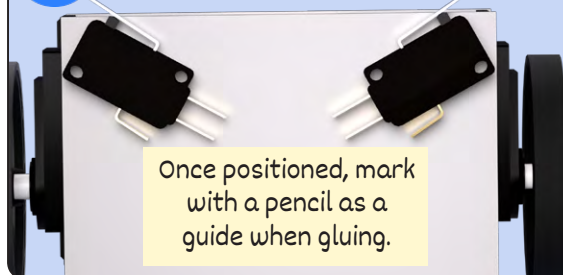
If your robot moves slowly when it should stop, one or both servos may need to be **calibrated**. See Frequently Asked Questions on Page 47 for more info.

Project 5

Build a Touch-and-Go Bumper Bot

By adding just two **switches**, your Make:Bot can explore its environment. Should the robot bump something, the switches will detect the collision. The robot stops, backs up, turns around, and heads in another direction!

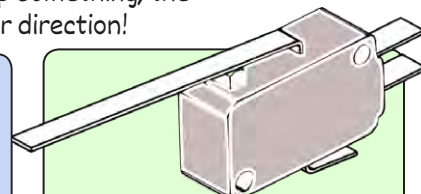
1



Position two snap-action switches on the underside of the lower base, facing the front.

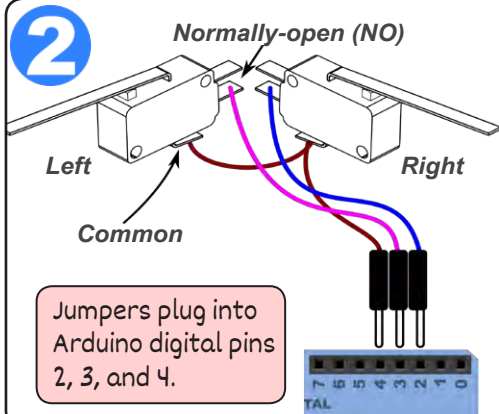
Tilt the switches so that the corners are flush with the front and side edges of the base.

The ends of the metal prongs should be roughly in front of the wheels.



The Touch-and-Go feature relies on **snap-action leaf switches**. The leaf is a metal prong that acts like a cat's whisker.

2

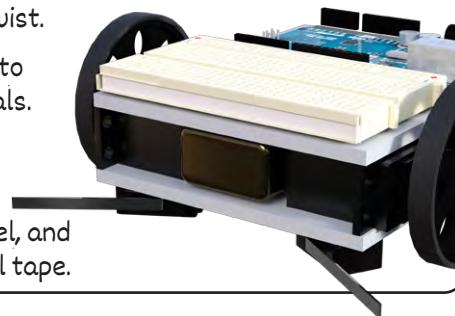


Select three 6" male-to-female jumpers, colors shown at left. For each wire:

1. Cut off just the female end.
2. Remove 1/4" of insulation from the cut end and twist.
3. Use a soldering iron to solder the bare wire ends to the switch **NO** (normally open) and **Common** terminals.
4. Bridge the two Common terminals of the switches with 3" solid wire jumper.
5. Route wires under right side of robot behind wheel, and plug into Arduino. Keep wires in place with electrical tape.

3

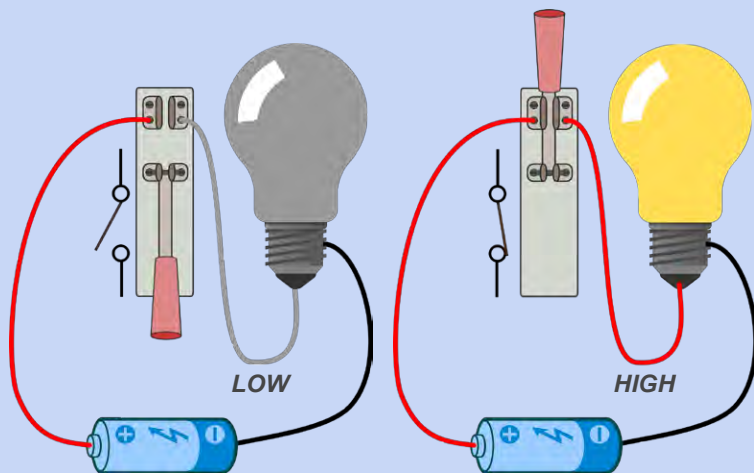
Glue the switches to the underside of the bottom base.



To understand how the Make:Bot detects collisions, it's important to know a little about **switches**.

The Arduino is a small computer that can connect to the outside world. The connection is made with **input/output** (or **I/O**) pins. Switches are an example of an **input**—data that comes **into** the Arduino.

Switches are **digital** inputs, meaning that they have only two states: **LOW** (or **0**) and **HIGH** (or **1**).



Circuit is open, the light is off.

Circuit is closed, the light is on.

On the Arduino UNO the **digital** input/output pins are along the top of the board. They are numbered 0 to 13. Pins 0 and 1 are used for communications between the Arduino and PC, so avoid using these.



The Arduino UNO also has **six analog I/O** pins along the bottom edge. You'll learn more about these in Project 9.



Switches are simple mechanical devices, consisting of a **button** or **lever**. Depressing this button **closes** the contacts inside the switch, which **completes a circuit**.

Check out Make:Bot's newfangled bumper switches with the **SwitchTest** sketch. The Arduino's **builtin LED** lights if either switch is depressed. The Arduino's integrated **pullup resistors** make the switch **HIGH** when it's not depressed; **LOW** when it is!

SwitchTest

```
void setup() {  
  pinMode(LED_BUILTIN, OUTPUT);
```

Set pin 13 (builtin LED) to output.

```
  pinMode(2, INPUT);      // Right switch  
  pinMode(3, INPUT);      // Left switch  
  pinMode(4, OUTPUT);     // Ground for switches
```

Set switch pins (2, 3) as inputs. Set pin 4 as an output so it can be used as a convenient ground connection.

```
  digitalWrite(2, HIGH);  
  digitalWrite(3, HIGH);
```

Enable Arduino's internal pullup resistors for switch pins.

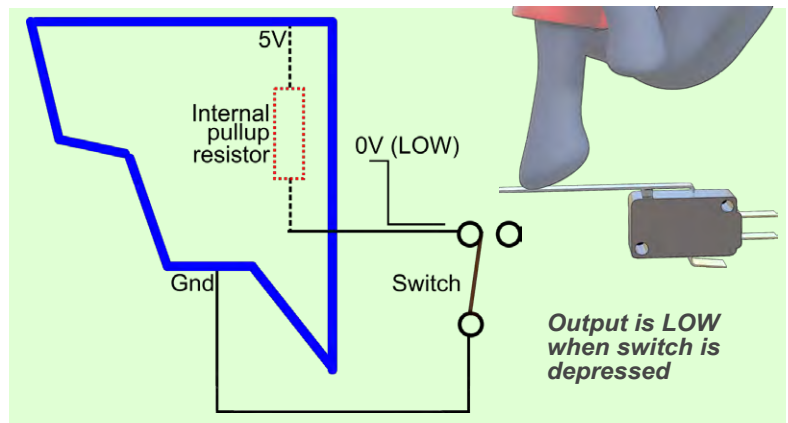
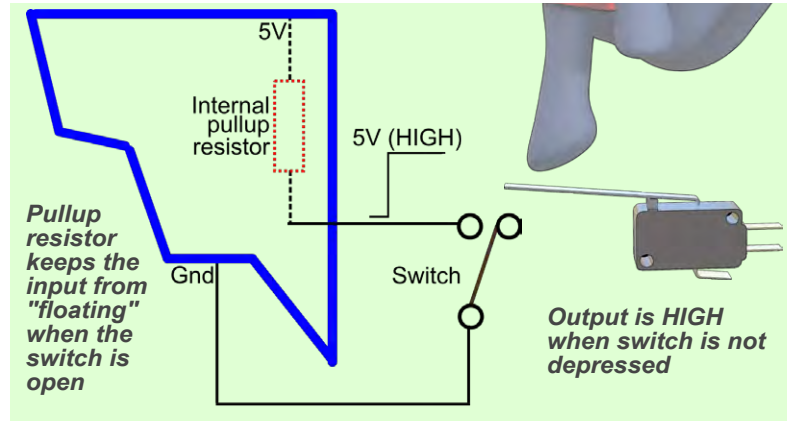
```
  digitalWrite(4, LOW);    // Set to 0 volts
```

```
}
```

Set pin 4 LOW (0 volts) to use as ground connection

```
void loop() {  
  if ((digitalRead(2) == LOW) || (digitalRead(3) == LOW))  
    digitalWrite(LED_BUILTIN, HIGH);  
  else  
    digitalWrite(LED_BUILTIN, LOW);  
  delay (100);  
}
```

Test if pin 2 **OR** pin 3 has gone LOW (0 volts). If so, light up the builtin LED on pin 13.



BumperBot wanders the room, and goes into a new direction if it hits things. After uploading the sketch, set the robot on the floor, and depress either switch to start movement.

Download the **BumperBot** sketch from the support site. See Page 50.

BumperBot

```
#include <Servo.h>

Servo servoRight;
Servo servoLeft;

volatile int pbLeft = LOW;    // Flag for left switch
volatile int pbRight = LOW;  // Flag for right switch
boolean started = false;     // True after first start

void setup() {
  // Set pin modes for switches
  pinMode(2, INPUT);
  pinMode(3, INPUT);
  pinMode(4, OUTPUT);

  // Set internal pull up resistors for switches
  digitalWrite(2, HIGH);    // Right switch
  digitalWrite(3, HIGH);    // Left switch

  digitalWrite(4, LOW);     // Pin 4 as ground

  servoRight.attach(9);     // Right servo to pin 9
  servoLeft.attach(10);     // Left servo to pin 10

  // Set up interrupts
  attachInterrupt(0, hitRight, FALLING);
  attachInterrupt(1, hitLeft, FALLING);

  started = true;           // Okay to start moving
}
```

This **#include statement** tells the Arduino you want to work with servos.

Variables are used to store information that changes when the sketch executes. The **pbLeft/pbRight** variables monitor the right and left "bumper" switches.

The left and right switches connect to pins 2, 3, and 4. Pins 2 and 3 are defined as **inputs** from the switches. Pin 4 is defined as an output. It provides a 0 volt (GND, or ground) connection to the switch.

Input pins need a **default digital state**. The Arduino has built-in **pullup resistors** that hold the pin HIGH until the switch is activated.

Make:Bot uses the Arduino's hardware interrupt feature, which allows the Arduino to immediately sense when a switch is hit.

We don't want the Make:Bot to do anything until all the setup() code has been run. The started variable is set to true when everything is ready.

BumperBot, continued...

```
void loop() {  
  if (pbLeft == HIGH) {           // If left bumper hit  
    goReverse();                  // Reverse for 1/2 sec  
    delay(500);  
    goRight();                    // Spin 1 sec  
    delay(1000);  
    goForward();                  // Go forward again  
    pbLeft = LOW;                 // Reset flag shows "hit"  
  }  
  
  if (pbRight == HIGH) {          // If right bumper hit  
    goReverse();  
    delay(500);  
    goLeft();  
    delay(1000);  
    goForward();  
    pbRight = LOW;  
  }  
}  
  
// Interrupt handlers  
void hitLeft() {  
  if (started)                    // Only if robot has started  
    pbLeft = HIGH;  
}  
void hitRight() {  
  if (started)                    // Same as left switch handler  
    pbRight = HIGH;  
}  
  
// Add Motion Routines from ServoTaiChi here
```

When the **left bumper switch** has been activated, the Make:Bot steers around the obstacle by backing up for 500 milliseconds (ms), spins right for 1000 ms, and heads forward again.

The same as above, but this happens when the **right bumper switch** is activated.

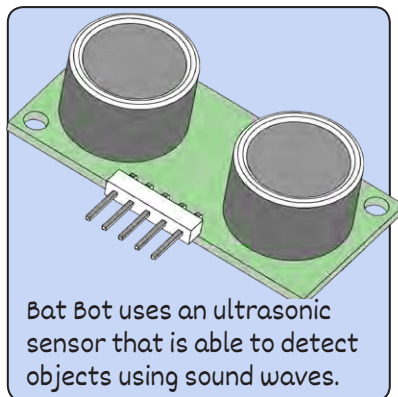
The Arduino is immediately interrupted when either switch is activated. The code simply sets the **pbLeft** or **pbRight** variable to **HIGH**, meaning "the switch has been struck!" This variable is subsequently checked inside the `loop()` function.

To save space the motion routines (`goForward`, etc.) are not shown here. Grab them from the ServoTaiChi sketch on Page 20.

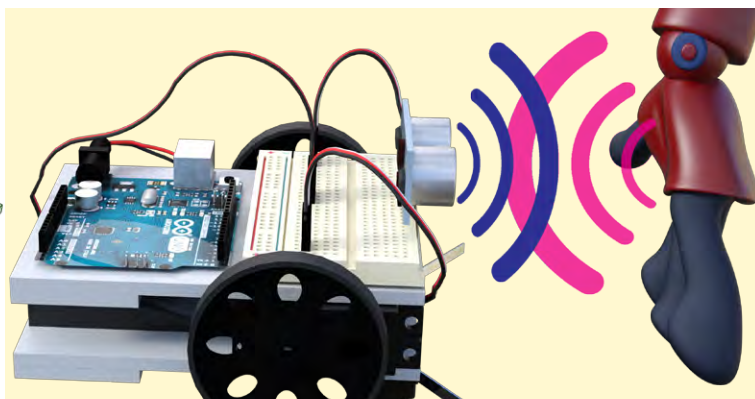
Project 6

Build an Ultrasonic "Bat Bot"!

An inexpensive ultrasonic sensor allows your Make:Bot to actually "see" its environment by using a series of very high frequency chirps, just like a bat! Unless you're a bat yourself you can't hear the chirping sounds!



Bat Bot uses an ultrasonic sensor that is able to detect objects using sound waves.



Ultrasonic sensors measure the time delay between a short high-frequency "ping" and its return echo.

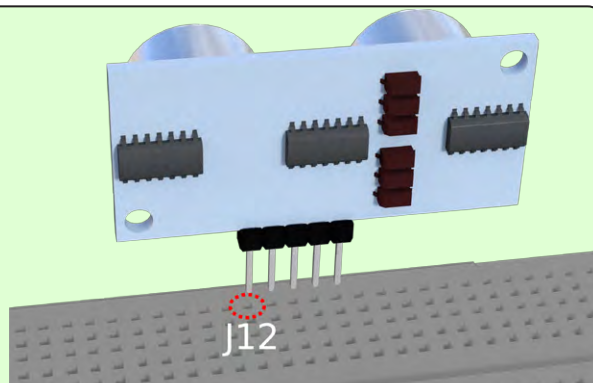
Echo times are measured in microseconds (μs , millionth of a second).

For a close object, the echo takes less time to come back.

1

Plug the ultrasonic sensor into the front row of the breadboard.

The left most pin (marked VCC) goes into socket J12.



Test the ultrasonic sensor to make sure it's working. The **BatBotTest** sketch reads the sensor and displays the result in the Arduino's Serial Monitor window (see Page 28).

BatBotTest

```
#define Trigger 7
#define Echo 8
```

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

  pinMode(Echo, INPUT);
  pinMode(Trigger, OUTPUT);
}
```

```
void loop(){
  digitalWrite(Trigger, LOW);
  delayMicroseconds(2);

  digitalWrite(Trigger, HIGH);
  delayMicroseconds(10);
  digitalWrite(Trigger, LOW);
```

```
  int distance = pulseIn(Echo, HIGH);
```

```
  distance = distance / 74 / 2;
  Serial.println(distance);
```

```
  delay(100);
}
```

Set up connections to Arduino: Trig (trigger) to pin 7, Echo to Pin 8

Establish Serial Monitor communications

Define "direction" of pins: Echo (pin 8) is an Input, Trigger (pin 7) is an Output

Reset Trigger by making pin LOW (0 volts) for 2 μ s (microseconds).

Send ping by making pin HIGH (5 volts) for 10 μ s.

Measure time until Echo pin changes to HIGH (echo is received)

Convert echo time to distance, in inches

Wait 100 ms before doing it again

FORMULA

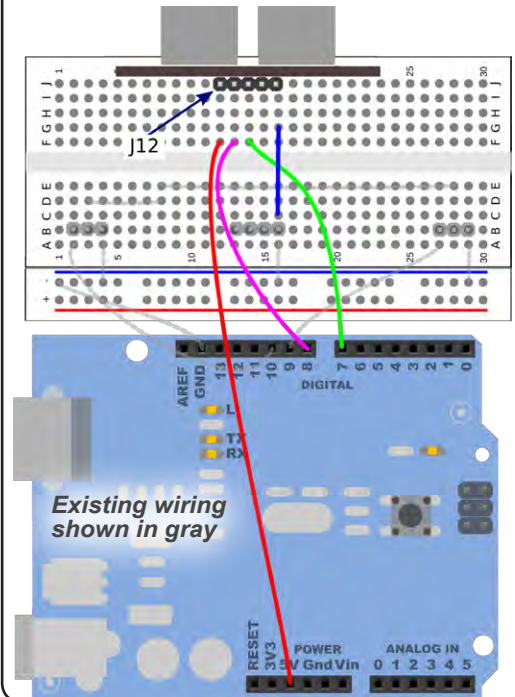
Sound travels 1" in about 74 microseconds. Echo times are always halved to calculate the round trip distance. Example: An echo that takes 750 μ s is roughly 5" away.

$5 = 750 / 74 / 2$ (result is rounded off)

Add wiring to connect the sensor to the Arduino.

2

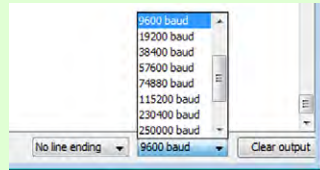
| Sensor | Arduino |
|--------|---------------------|
| VCC | 5V |
| Echo | Pin 8 |
| Trig | Pin 7 |
| GND | (breadboard ground) |



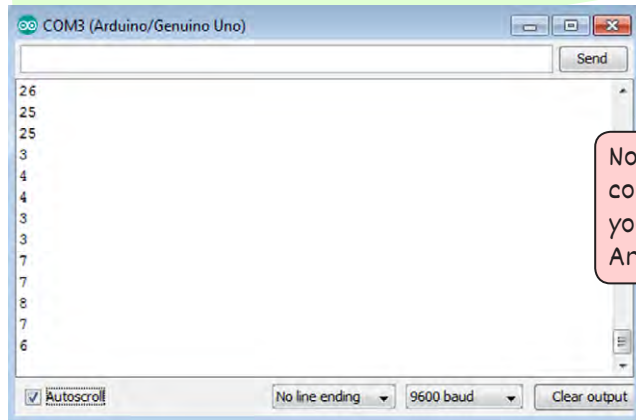
BatBotTest uses the Arduino's **Serial Monitor** to display the distance results. To open the Serial Monitor:

1. Upload the sketch.
2. After uploading is complete, click on the **Serial Monitor icon**.
3. Verify these settings:

Autoscroll: checked
Line ending: No Line ending
Baud: 9600 baud



Place your hand in front of the sensor and move it back and forth. Watch the values in the Serial Monitor change according to the distance.



Note: Keep the USB cable connected between your computer and the Arduino.

Now let's combine moving the Make:Bot while looking for objects with the ultrasonic sensor!

BatBotWander (partial only)

```
#include <Servo.h>
Servo servoRight;
Servo servoLeft;
#define Trigger 7
#define Echo 8

void setup() {
  servoRight.attach(9);
  servoLeft.attach(10);
  pinMode(Echo, INPUT);
  pinMode(Trigger, OUTPUT);
  randomSeed(analogRead(3));
  delay(200);
  goForward();
}

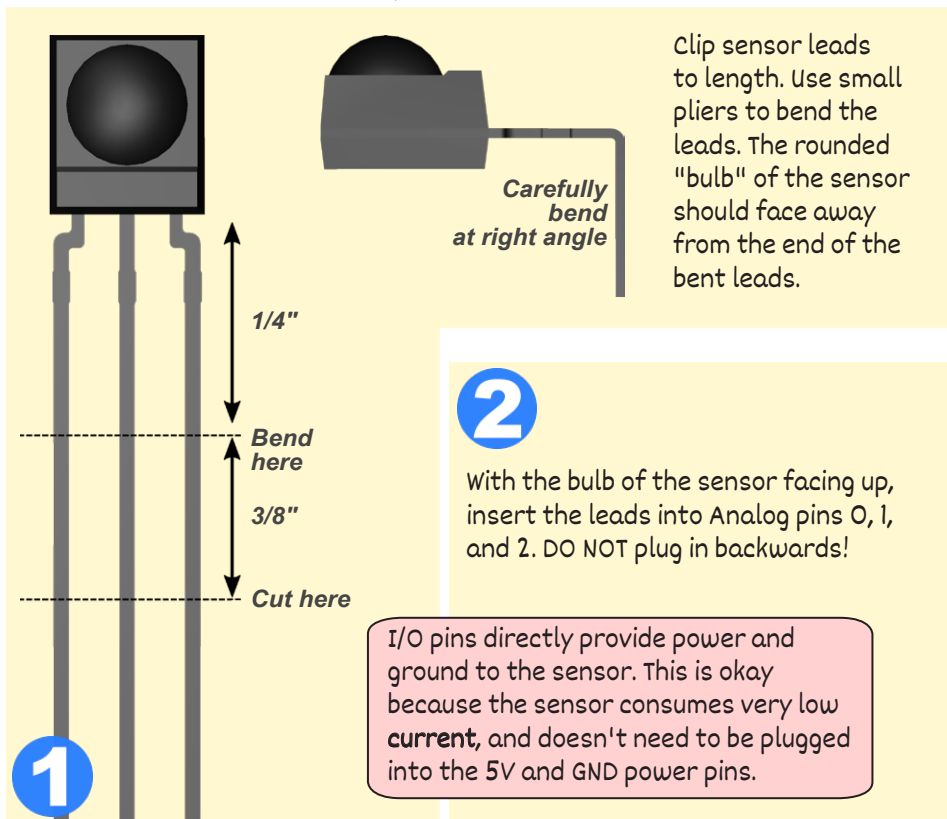
void loop() {
  int distance = doPing();
  if (distance <= 2)
    goReverse();
  delay(500);
  if (random(2) == 0)
    goRight();
  else
    goLeft();
  ...
}
```

Download the **BatBotWander** sketch from the support Site. see Page 50 more information.

Project 7

Build a Remote Controlled Robot

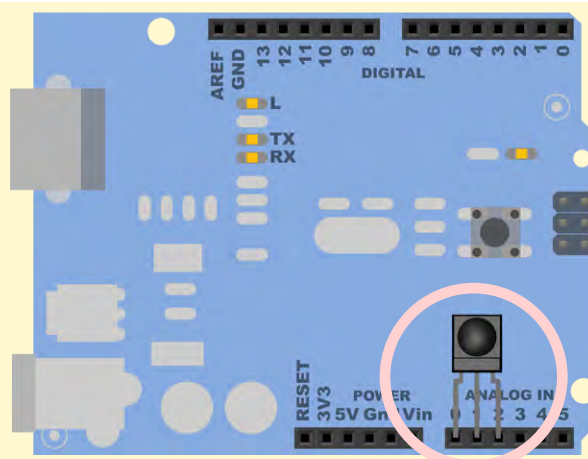
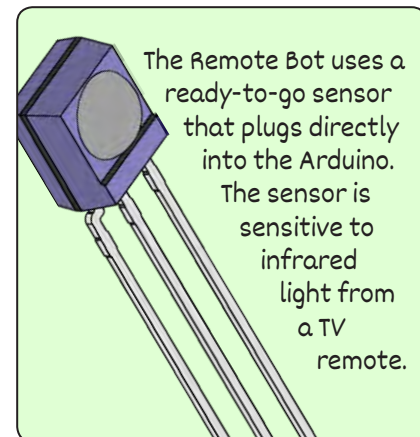
Command your robot from the comfort of your easy chair! Remote Bot uses an ordinary TV remote control and an inexpensive infrared sensor.



2

With the bulb of the sensor facing up, insert the leads into Analog pins 0, 1, and 2. DO NOT plug in backwards!

I/O pins directly provide power and ground to the sensor. This is okay because the sensor consumes very low **current**, and doesn't need to be plugged into the 5V and GND power pins.



3 Program a universal remote control to operate a *Sony TV*.

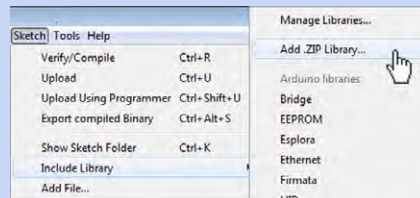
You need to look up **SONY TV** in the instruction manual for the remote, and enter the matching code.

Try different Sony TV codes until you find one that works with the **IRTest** sketch.



4 Add the **IRremote** third-party code library. The library comes with all the Remote Bot sketches provided on the Make:Bot Support Site.

1. Download the **IRTest** sketch and unzip it into your Arduino sketch folder.
2. In the Arduino IDE, choose Sketch→Include Library→Add .ZIP Library.



3. Navigate to the folder that contains the **IRTest** sketch you placed in Step 1.
4. Select the **IRremote.zip** file, click OK to add it to your IDE.
5. Exit the IDE and restart it.

5 Compile and upload the **IRTest** sketch to your Arduino.

Open the Serial Monitor window, and make sure the settings are the same as on Page 28.

Verify the actions of the remote by pressing the number buttons.



With the infrared sensor installed and tested it's time to program your Remote Bot. Use the number buttons on the remote control to drive your robot around the room. If the bot hits an obstacle, it'll automatically back up and steer in a new direction!

RemoteBot (portions only shown, based on BumperBot)

```
#include <IRremote.h>
```

```
int RECV_PIN = A0;           // Analog pin 0
IRrecv irrecv(RECV_PIN);
decode_results results;
```

Adds the **IRremote** library to your Arduino IDE.
(Remember, you **MUST** install this library before you can use this sketch; see Page 30).

```
pinMode(A1, OUTPUT);         // IR power, ground pins
pinMode(A2, OUTPUT);
digitalWrite(A1, LOW);       // IR ground; analog pin 1
digitalWrite(A2, HIGH);      // IR power; analog pin 2
...
irrecv.enableIRIn();         // Start the receiver
```

Turns pins A1 and A2 into power and ground for use by the infrared sensor, and then starts the IRremote process.

```
void loop() {
  if (pbLeft == HIGH) {      // If left bumper hit
    goReverse();
    delay(500);
  }
  ...
  if (pbRight == HIGH) {     // If right bumper hit
    goReverse();
    delay(500);
  }
}
```

Continuously loop: react to left or right switch.

RemoteBot, continued...

```
if (irrecv.decode(&results)) {
  switch (results.value) {
    case 0x10:
      Serial.println("1");    // Turn left forward
      turnLeftFwd();
      break;
  }
  ...
  irrecv.resume();           // Receive the next value
  delay(2);
}
```

Detects pressing buttons 1 thru 9 on the remote, and performs the associated action (e.g. left forward turn for button "1"). Then resets the IRremote process and waits 2 ms before repeating again.

```
void goForward() {
  servoRight.write(0);
  servoLeft.write(180);
}
void goReverse() {
  servoRight.write(180);
  servoLeft.write(0);
}
void goSpinRight() {
  servoRight.write(180);
}
...
```

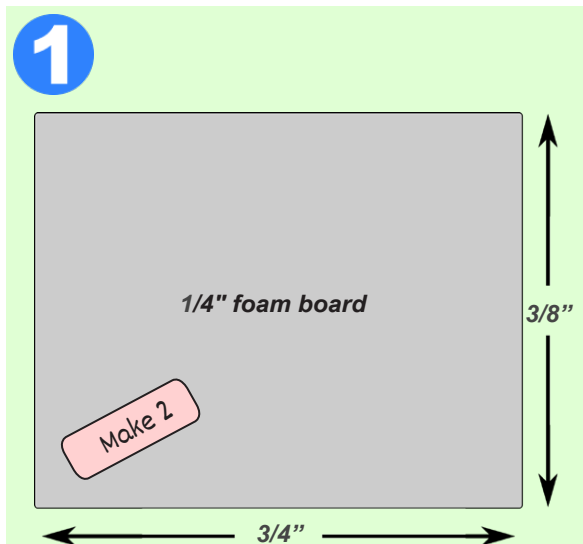
Download the **RemoteBot** sketch from the support site. See Page 50.

Motion routines for steering the robot. RemoteBot uses additional user-defined functions besides the ones in the BumperBot sketch.

Project 8

Make a Robot That Follows a Line

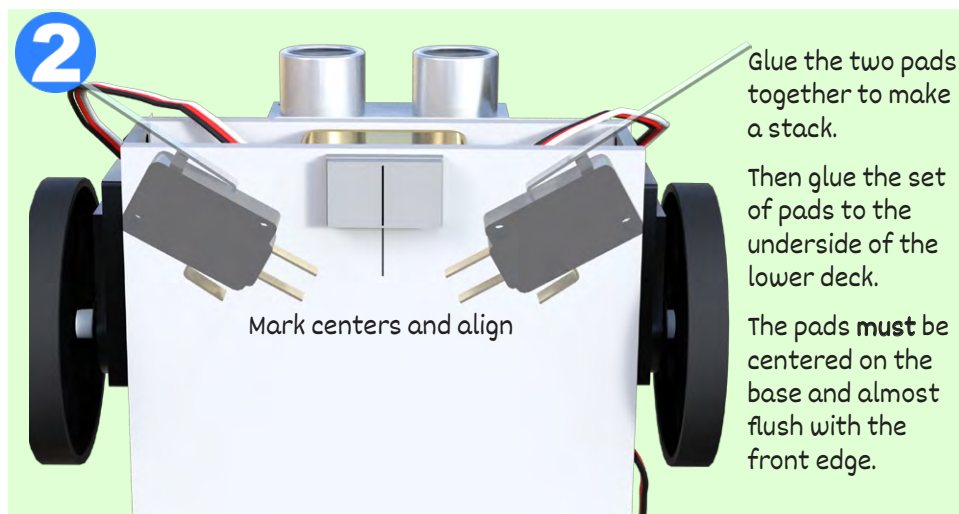
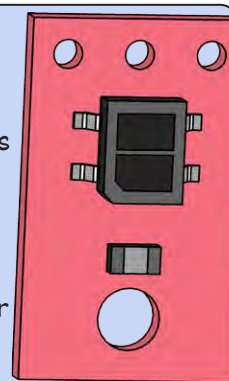
With just a pair of inexpensive sensors that detect reflected infrared light, your Make:Bot can dutifully follow a black line you've made on poster board—or white butcher paper, a kitchen floor, or any other white surface!

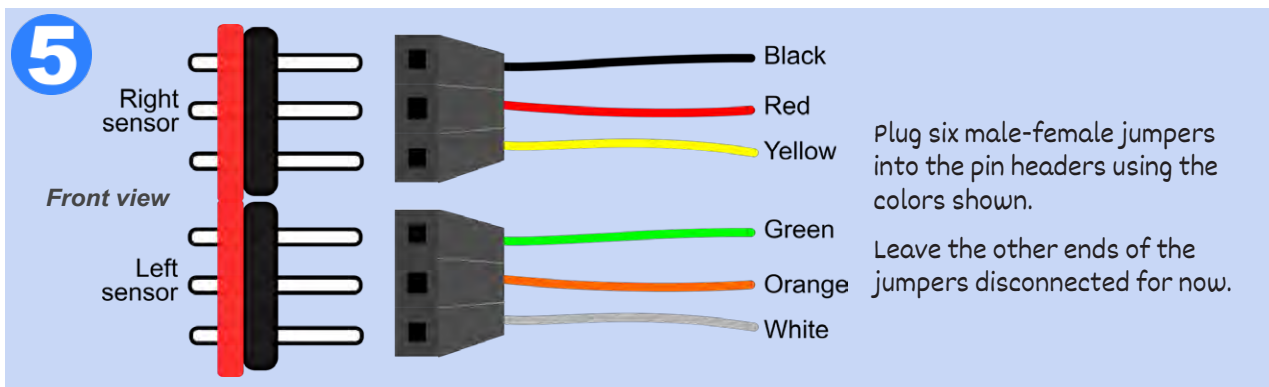
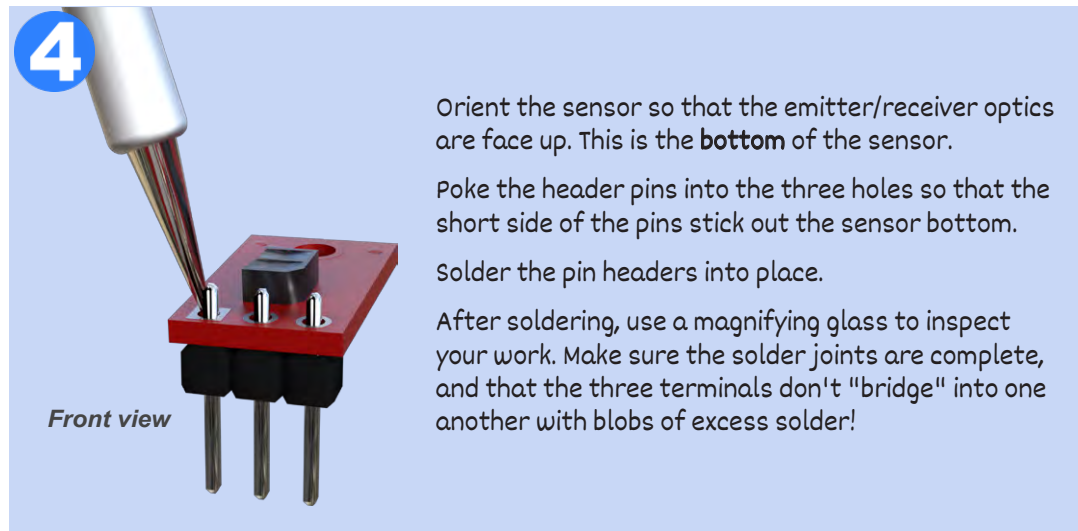
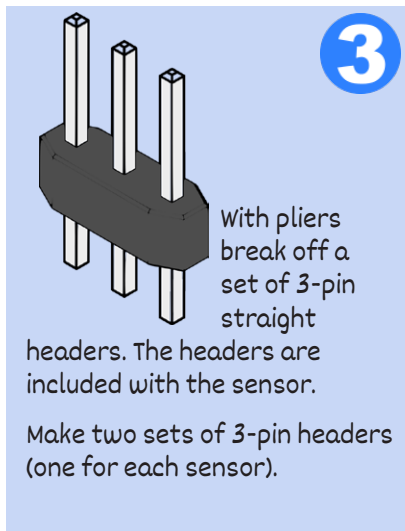


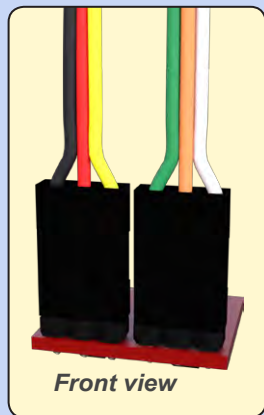
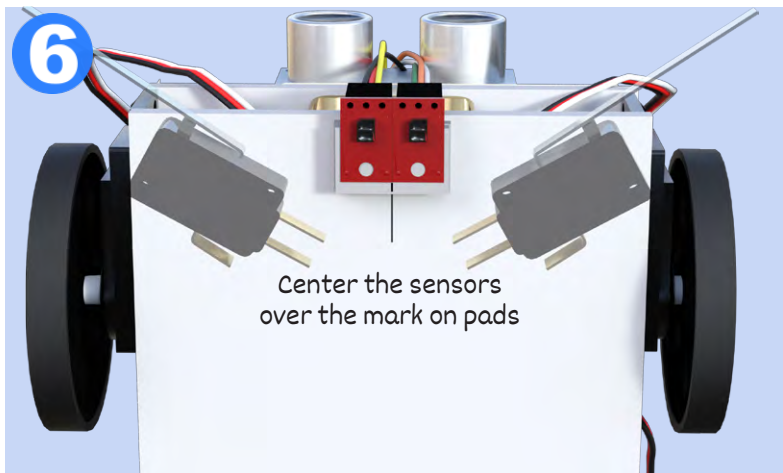
Use a utility knife to cut two IR sensor pads to 3/8" by 3/4". Exact size isn't critical, but the pads need to fit between the two switches on the underside of the Make:Bot.



Two miniature reflective light sensors allow the Make:Bot to follow a black line on white poster board.







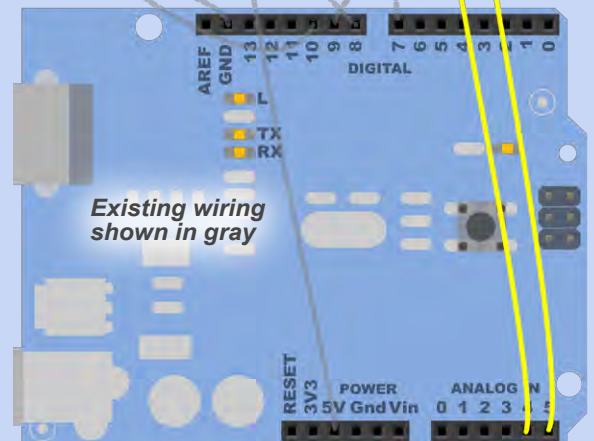
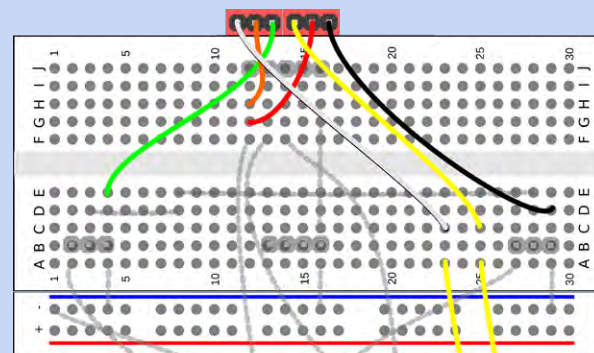
Cut a piece of double-sided tape to match the IR sensor pads. Stick the tape to the pads, then press the sensors onto the tape.

Position the sensors so that the connectors are flush against the front of the robot.

Use the connectors as a handle while you align them over the tape.

The sensors should be 1/8"-1/4" from the poster board. Tweak as needed.

Connect the six male-to-female jumpers from the sensors to the breadboard.



Test the two infrared sensors using the **LineSensorsTest** sketch. Upload the sketch as usual, and then open the Arduino Serial Monitor window. Wave your hand directly in front of the sensors. The readings should change. Expected values are 0-1023.

LineSensorsTest

```
#define lineLeft A4
#define lineRight A5
int irReflectL = 0;
int irReflectR = 0;
```

Sets up **variables** (temporary storage for data) for use elsewhere in the sketch.

#define is for coding convenience. The names represent the I/O pins connected to the reflective sensors.

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

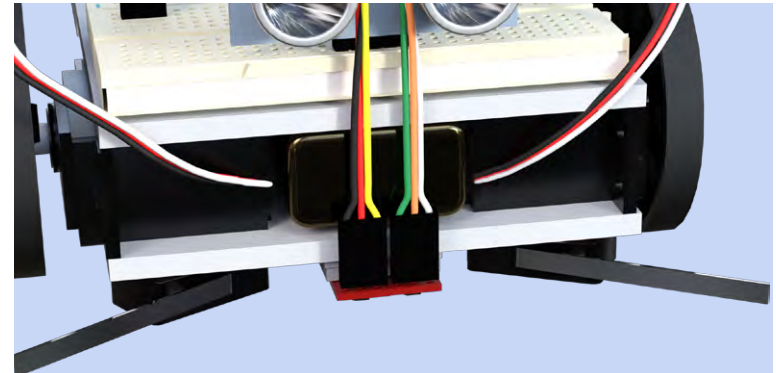
Prepares the Serial Monitor so you can see values.

```
void loop() {
  irReflectL = analogRead(lineLeft);
  irReflectR = analogRead(lineRight);
```

The instantaneous value of the right and left sensors are read and stored in variables.

The values that are captured range from 0 to 1023. These values represent an **analog** (continuously variable) voltage, from 0 to 5 volts.

A value of 1023 means the sensor detects pure black (no light reflected); 0 means white, lots of light reflected.



Verify that the infrared sensor wiring looks like this. Wrap the wires with electrical tape to bundle them together and tuck them between the ultrasonic sensor elements.

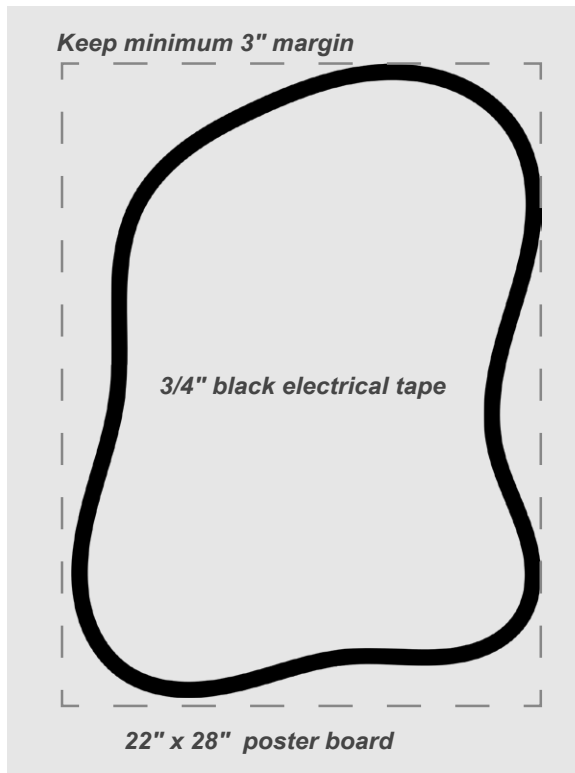
TestLineSensors, continued...

```
Serial.print ("Left:");
Serial.print ("\t");           // Makes a tab space
Serial.print (irReflectL);     // Inserts value for left
Serial.print ("\t");
Serial.print ("Right:");
Serial.println (irReflectR);   // Inserts value for right
delay(100);                   // and starts a new line
}
```

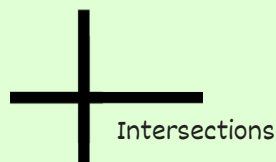
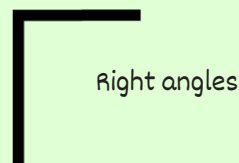
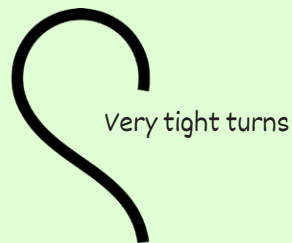
The sensor values are printed in the Serial Monitor window. To see the results, upload the sketch and open the Serial Monitor (see Page 28).

Make a line-following course using a 22" by 28" white poster board and 3/4" black electrical tape. High quality tape works best—the cheap stuff can wrinkle and "lift up" over time. You can also use white craft paper, even an all-white linoleum floor!

Use a pencil to lightly mark a border no less than 3" from the edge of the poster board. Start anywhere and apply the tape. Firmly press the tape into place as you go. Don't overstretch. Erase the pencil markings when done.



AVOID:



TRY THESE IDEAS:

- Make several track courses with different shapes. Use only one side of the poster.
- Race with a friend who has built her own Make:Bot!
- Remove the six jumper wires to the line sensors when using the ultrasonic sensor.

WHEN THINGS GO WRONG:

- If your bot "jumps the track" too much, try another course with simpler turns. Be sure the sensors are clean and properly aligned.
- Keep the sensor-to-poster distance between about 1/8" and 1/4".
- Try a lower threshold value in the **LineFollow** sketch if the robot can't stay on the line.
- Electrical tape can lift with age and humidity. Press it back down, or replace.

With the race course made, it's time to follow some lines! Upload the **LineFollow** sketch. Press the Arduino's **Reset** button while you place the bot over the black line. Release the button and go, baby, go!

LineFollow (partial)

```
...  
int thresh = 500;
```

The **thresh** variable defines a **threshold** for determining whether the reflective sensors are over a line. You can experiment with this value to see what works best. Try a lower number if your bots turns around a lot.

```
void loop() {  
  irReflectL = analogRead(lineLeft);  
  irReflectR = analogRead(lineRight);
```

The instantaneous values of the two sensors are captured every loop through the sketch.

```
  if (irReflectL >= thresh && irReflectR >= thresh) {  
    line_forward(); // on line  
  }  
  if (irReflectL >= thresh && irReflectR <= thresh) {  
    line_slipLeft(); // veering off right  
    delay(4);  
  }  
  if (irReflectL <= thresh && irReflectR >= thresh) {  
    line_slipRight(); // veering off left  
    delay(4);  
  }  
  if (irReflectL < thresh && irReflectR < thresh) {  
    line_spinRight(); // try to re-acquire line if lost  
    delay(20);  
  }  
}
```

LineFollow, continued...

A series of **conditional if** tests compare the values of the sensors to determine if the robot is straddling the black line or veering to the right or left. The code

```
irReflectL >= thresh && irReflectR >= thresh
```

compares the sensors to the threshold value. If both are greater than the threshold, the sensors "see" black, which means the robot is over the line.

Other **if** tests determine if the robot is veering to the right or left, so that small course corrections may be made.

```
void line_forward() {  
  servoRight.write(0);  
  servoLeft.write(180);  
}  
void line_slipRight() {  
  servoRight.write(90);  
  servoLeft.write(180);  
}  
void line_slipLeft() {  
  servoRight.write(0);  
  servoLeft.write(90);  
}  
...
```

User-defined functions provide steering for the robot.

Project 9

Learn About Digital and Analog Signals

Robots are all about the sensors connected to them. Without sensors, your creation is deaf and blind. Sensors attach to the Arduino using I/O pins and communicate with it using digital or analog signals.

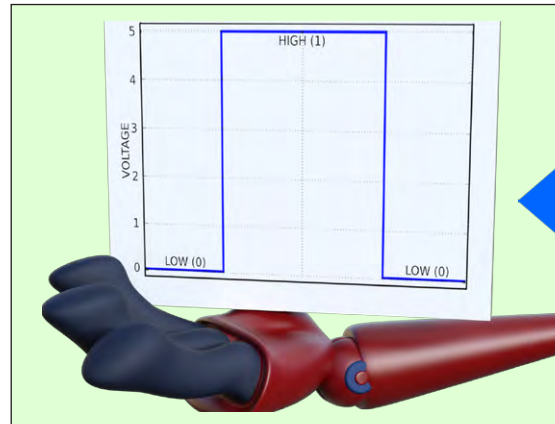
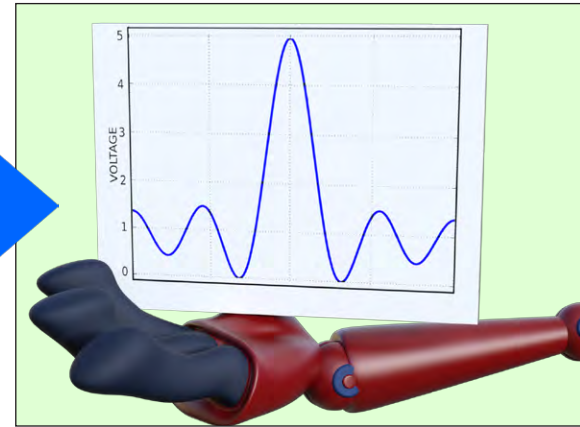
The type of signal is important: it dictates which I/O pins on the Arduino you can use. Digital signals connect to any I/O pin; analog signals can only connect to the pins marked **Analog In**.



CAUTION

When applying a signal to the Arduino Uno, be sure the signal **does not exceed 5 volts!**

Analog signals are continuously variable, from 0 to 5 volts. Inside the Arduino these voltages are translated to numbers ranging from 0 to 1023.



Digital signals have only two states:

0 Volts = LOW, or 0
5 volts = HIGH, or 1

Use one of the bumper switches connected to the Make:Bot to demonstrate reading a digital signal.

DigitalTest

```
#define SWITCH 3           // Switch to pin 3 (left)
int swValue;              // Variable holds value

void setup() {
  pinMode(SWITCH, INPUT);  // Left switch input
  pinMode(4, OUTPUT);      // Ground for switches
  digitalWrite(SWITCH, HIGH); // Set internal pullup
  Serial.begin (9600);     // Start Serial Monitor
}

void loop() {
  swValue = digitalRead(SWITCH); // Read switch value
  Serial.println (swValue);      // Display value
  delay(250);                   // Wait 1/4 sec
}
```

WHAT TO SEE

Open the Serial Monitor (see Page 28) and press the left bumper switch. Observe the values:

| Switch State | Value |
|--------------|----------|
| Open | 1 (HIGH) |
| Closed | 0 (LOW) |

The switch is 1 (HIGH) when **not** activated because it is connected to a pullup resistor inside the Arduino. Closing the switch connects it to ground, so the signal goes 0 (LOW).

Use one of the line following sensors connected to the Make:Bot to demonstrate reading an analog signal.

AnalogTest

```
#define SENSOR A5          // Right line analog pin A5
int sValue = 0;           // Variable holds value

void setup() {
  Serial.begin (9600);    // Start Serial Monitor
}

void loop() {
  sValue = analogRead(SENSOR); // Read sensor value
  Serial.println (sValue);     // Display value
  delay(250);                 // Wait 1/4 second
}
```

WHAT TO SEE

Open the Serial Monitor and wave your hand over the line sensors. Values range from 0 to 1023.

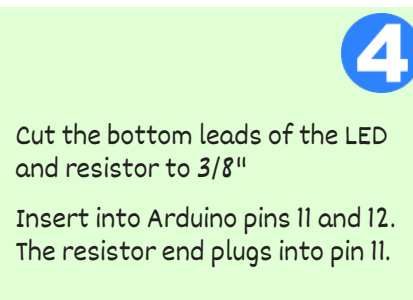
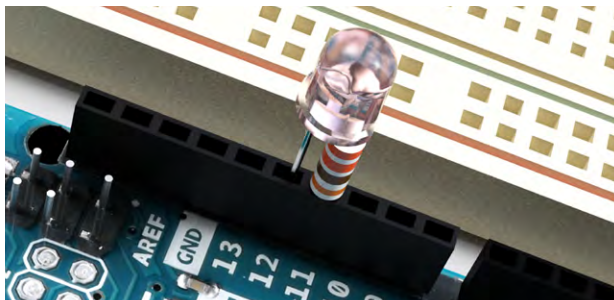
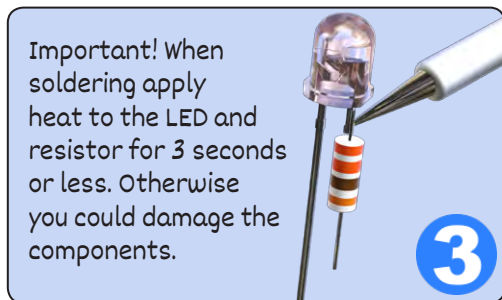
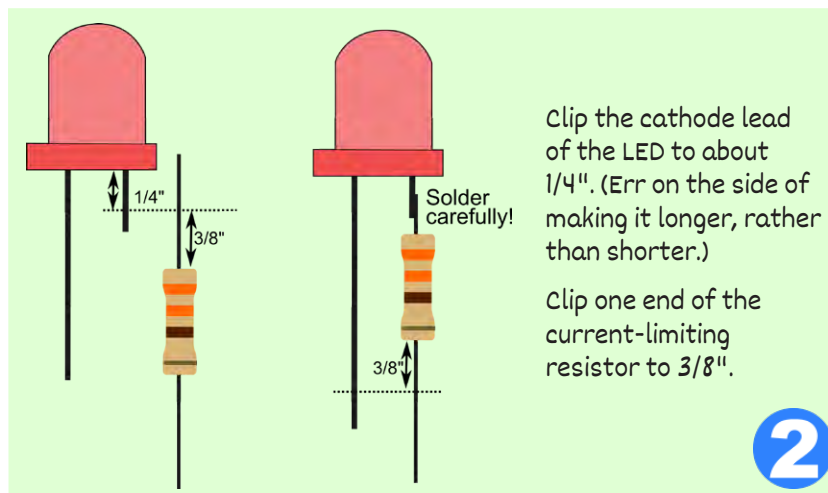
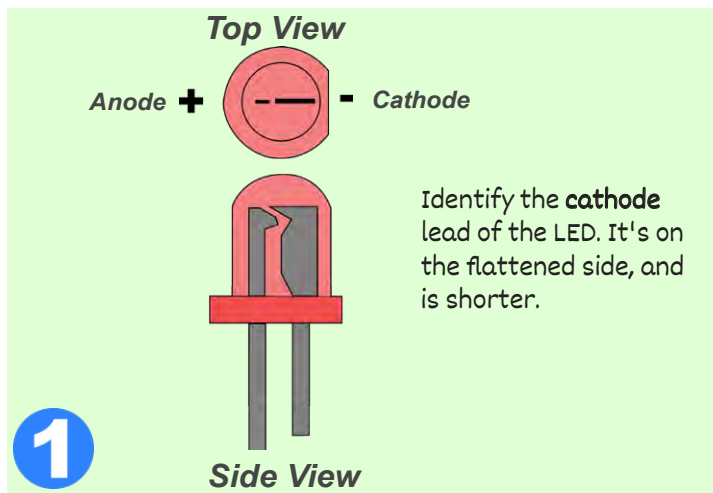
| Value | Voltage |
|--------|------------|
| 0 | 0 |
| 1-1022 | 0.1 to 4.9 |
| 1023 | 5 |

Because of the way the sensors are constructed, higher values = less light reflecting into the sensor.

Project 10

Add Blinky Lights to Your Make:Bot!

Lights are an easy way for your Make:Bot to communicate with you. Lights are cheap and easy to add: just a **light emitting diode (LED)** and a **resistor**. This project shows how to add a single LED that glows or blinks in two colors!

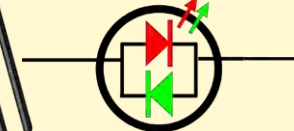
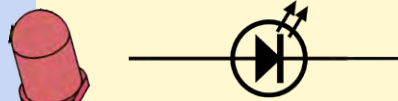


LEDs are a kind of semi-conductor made to emit light.

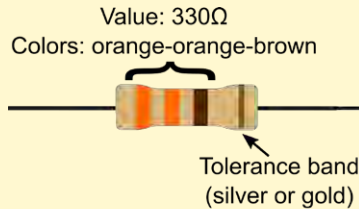
Most LEDs are a single color. The Make:Bot's LED is special: it produces two colors, red or green.

The color you see is determined by the +/- **polarity** of the LED's power leads, and can be controlled by the Arduino.

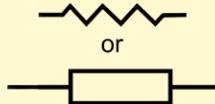
Symbol for one-color LED



Symbol for bi-color LED



Symbol for resistor



LEDs are used with a resistor, which **limits current** flowing through the semiconductor.

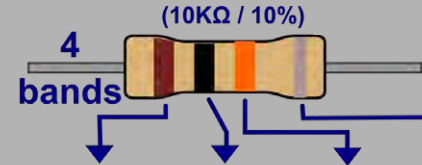
Resistors are **color coded** using bands. The bands indicate the amount of resistance, in **ohms**, provided by the resistor.

A common resistor value for use with LEDs is **330 ohms**, abbreviated as **330Ω**.

CAUTION

NEVER use an LED without a resistor to limit current. Otherwise the LED will burn out in seconds!

Color Codes



| Color | 1st band | 2nd band | Multiplier | Tolerance |
|---------------|----------|----------|-------------|-------------|
| Black | 0 | 0 | 1 | |
| Brown | 1 | 1 | 10 | |
| Red | 2 | 2 | 100 | |
| Orange | 3 | 3 | 1k | |
| Yellow | 4 | 4 | 10k | |
| Green | 5 | 5 | 100k | |
| Blue | 6 | 6 | 1M | |
| Violet | 7 | 7 | 10M | |
| Gray | 8 | 8 | | |
| White | 9 | 9 | | |
| Gold | | | | ±5% |
| Silver | | | | ±10% |

BlinkyTest flashes the LED connected to pins 11 and 12. Because 5V power is applied to the LED **cathode** (pin 11), it blinks green.

BlinkyTest

```
#define LED_GREEN 11
#define LED_RED 12
```

The Arduino has a built-in LED which is permanently connected to pin 13. You can control LEDs on any other pin by specifying the pin number. The #define statements associate the external LED with pins 11 and 12.

```
void setup() {
  pinMode(LED_RED, OUTPUT);
  pinMode(LED_GREEN, OUTPUT);
  digitalWrite(LED_RED, LOW);
  digitalWrite(LED_GREEN, LOW);
}
```

Pins 11 and 12 are made outputs, and set to LOW (0 volts). During the sketch pin 12 (LED_RED) doesn't change, and simply serves as a ground (0 volt) connection.

```
void loop() {
  digitalWrite(LED_GREEN, HIGH);
  delay(250);
  digitalWrite(LED_GREEN, LOW);
  delay(250);
}
```

The loop() function repeatedly toggles the LED_GREEN pin (pin 11) HIGH and LOW, which makes the LED flash green.

BlinkyColors flashes the LED between red and green, by alternating the polarity (LOW and HIGH) of pins 11 and 12.

BlinkyColors

#define and setup() are the same as **BlinkyTest**.

```
void loop() {
  red_blink();
  green_blink();
}
```

The loop() consists of calls to two user-defined functions (below), which make the LED flash green or red.

```
void red_blink() {
  digitalWrite(LED_RED, HIGH);
  delay(250);
  digitalWrite(LED_RED, LOW);
  delay(250);
}

void green_blink() {
  digitalWrite(LED_GREEN, HIGH);
  delay(250);
  digitalWrite(LED_GREEN, LOW);
  delay(250);
}
```

The red or green color is determined by the LED polarity.

| Pin 12 | Pin 11 | Color |
|--------|--------|-------|
| LOW | LOW | -off- |
| HIGH | LOW | Red |
| LOW | HIGH | Green |
| HIGH | HIGH | -off- |

LEDs can help you confirm operation: You know things are working if the LED glows. Here's how you can enhance the **BumperBot** sketch by adding green/red LED signalling whenever the robot goes forward or backward.

BumperBotLights

```
...
boolean started = false;
#define LED_GREEN 11
#define LED_RED 12

void setup() {
  ...
  started = true;
  pinMode(LED_GREEN, OUTPUT);
  pinMode(LED_RED, OUTPUT);
  digitalWrite(LED_GREEN, LOW);
  digitalWrite(LED_RED, LOW);
}
```

Add lines in **red**; existing lines from **BumperBot** sketch are shown in **gray**.

```
void goForward() {
  servoRight.write(0);
  servoLeft.write(180);
  led_green();
}

void goReverse() {
  servoRight.write(180);
  servoLeft.write(0);
  led_red();
}
```

Make the LED green when the robot goes forward; make it red when it goes in reverse.

BumperBotLights, continued...

```
void goRight() {
  servoLeft.write(180);
  servoRight.write(180);
  led_off();
}

void goLeft() {
  servoLeft.write(0);
  servoRight.write(0);
  led_off();
}
```

Turn the LED off for turns.

```
void led_green() {
  digitalWrite(LED_GREEN, HIGH);
  digitalWrite(LED_RED, LOW);
}

void led_red() {
  digitalWrite(LED_GREEN, LOW);
  digitalWrite(LED_RED, HIGH);
}

void led_off() {
  digitalWrite(LED_GREEN, LOW);
  digitalWrite(LED_RED, LOW);
}
```

Set LOW/HIGH polarity for green or red. Set both pins 11 and 12 to LOW to turn LED off.

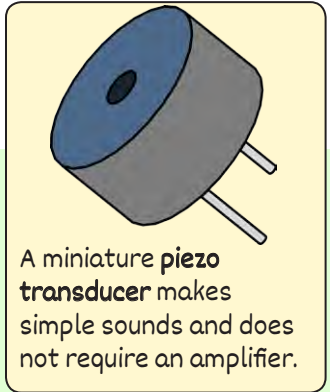
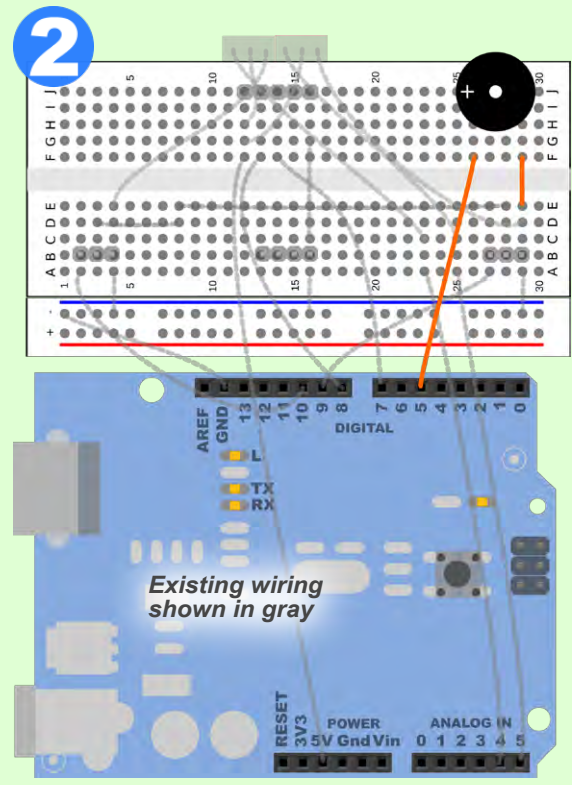
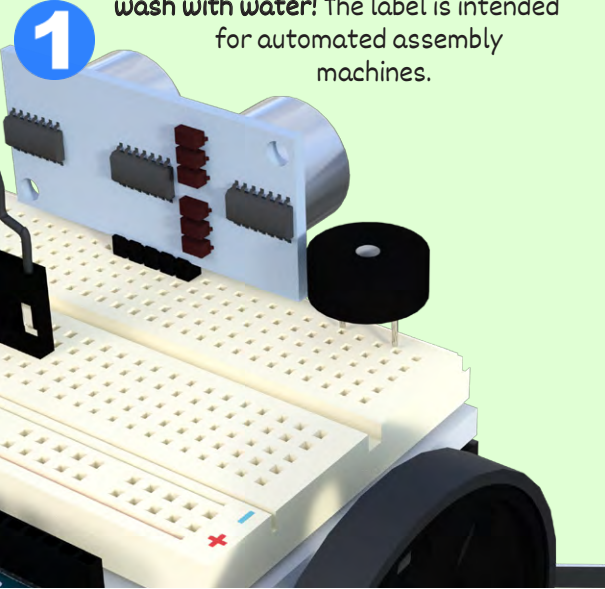
Project 11

Create Sounds with Your Make:Bot!

Add simple robotic sounds to your Make:Bot using an inexpensive **piezo transducer**. The Arduino generates signals that come out as plain buzzing and beeping tones.

Plug the piezo transducer into breadboard sockets J26 and J29. The + pin of the transducer goes into socket J26.

Your transducer may come with a label that says "Remove seal after washing." **Do not wash with water!** The label is intended for automated assembly machines.



A miniature **piezo transducer** makes simple sounds and does not require an amplifier.

Use a pre-stripped solid jumper wire between breadboard sockets E29-F29. This is the ground connection to the transducer.

Use an orange male-male 6" jumper between Arduino digital pin 5 and breadboard socket F26.

SoundTest verifies the piezo transducer is properly connected to the Arduino. It repeats a sequence of three tones.

SoundTest

```
#define SPKR 5
```

Defines Arduino pin 5 connected to the + terminal of the piezo transducer (the - terminal is connected to ground).

```
void setup() {           // setup() is intentionally empty  
}
```

```
void loop() {  
  tone(SPKR, 247, 300);  
  delay(200);
```

Makes a tone through the transducer. The frequency is 247 Hertz, with a duration of 300 milliseconds. The sketch then pauses for 200 milliseconds.

```
  tone(SPKR, 131, 300);  
  delay(200);  
  tone(SPKR, 1175, 300);  
  delay(200);  
}
```

Produces additional tones at different frequencies, also with a duration of 300 ms. The sketch pauses between each tone for 200ms. The tones are repeated.

RemoteBotSound adds tones to the RemoteBot sketch. Different tones play when the left or right switch is actuated.

RemoteBotSound

Added lines in red; existing lines from **RemoteBot** in gray.

```
boolean started = false;  
#define NOTE_B3 247  
#define NOTE_C3 131  
#define NOTE_C4 262  
#define NOTE_D6 1175  
#define SPKR 5
```

Defines "musical" notes and their corresponding frequencies. These #defines are a convenience for coding.

```
if (pbLeft == HIGH) {  
  goReverse();  
  makeSound (NOTE_B3, 200);  
  makeSound (NOTE_C3, 200);  
  makeSound (NOTE_D6, 200);
```

```
  if (pbRight == HIGH) {  
    goReverse();  
    makeSound (NOTE_D6, 200);  
    makeSound (NOTE_C4, 200);
```

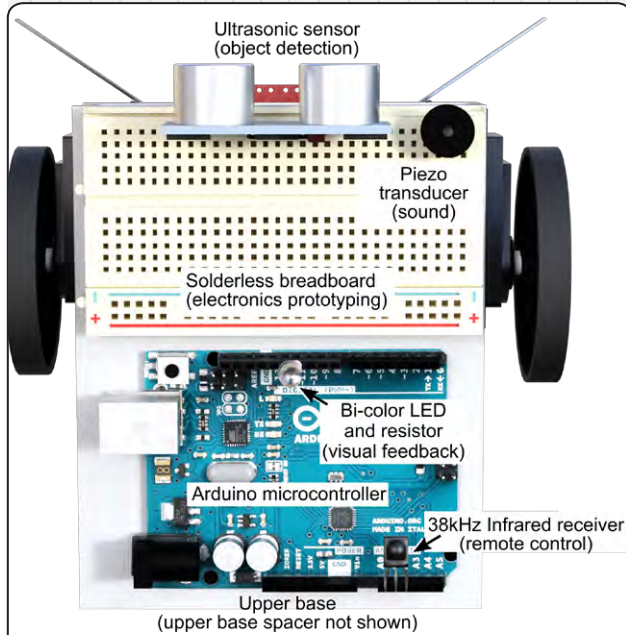
Adds sequence of tones when the switches are hit.

```
void makeSound(int frequency, int duration) {  
  tone(SPKR, frequency, duration);  
  delay(100);  
}
```

Plays sound using the Arduino **tone** statement.

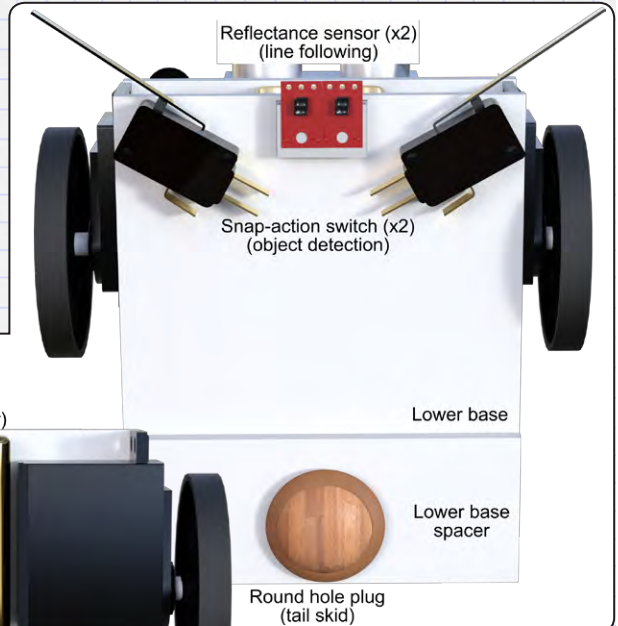
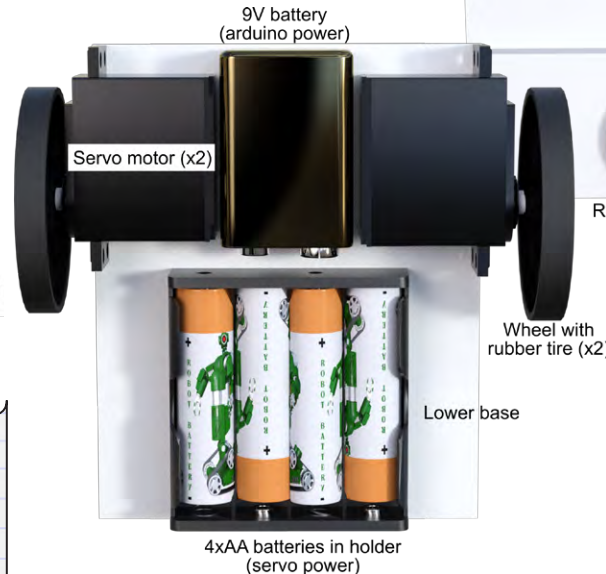
Make:Bot Top to Bottom

Make:Bot is a "two deck" robot, consisting of a lower and upper deck. The decks are separated by battery cells and motors. These views show the major components of the Make:Bot.



TOP VIEW

BETWEEN DECKS VIEW



BOTTOM VIEW

Frequently Asked Questions

Everyone has questions. Here are some common ones, and what to do about them.

If you're new to the Arduino, check out the various help pages on the www.arduino.cc website.

Can I use a non-UNO Arduino board?

It might work but it's not recommended. Arduino boards can differ in their voltage requirements, pin functions, and other features. Substitute only if you already know the issues.

Why do my servos turn when the robot is supposed to stop?

The servos may need to be calibrated. Upload the **Calibrate** sketch from the **MakeBot: Support Site** to your Arduino. Use a small Phillips screwdriver to adjust the knob in the side of the servos. The servos are calibrated when the motors stop.

Can I just use the 9V battery to run the Make:Bot?

No. The 9V battery does not provide enough current to operate the servo motors.

What type of surfaces are best for Make:Bot?

Tabletops, wood floors, tile, and very low nap carpet.

How about using just one battery pack instead of the two?

Not recommended. Arduino requires at least 7 volts, but most servos are made for just 4.8-6V. And, it's not advisable to run servos from the Arduino's 5V regulated supply.

How do I make the IDE recognize my Arduino board?

Try plugging in the USB cable and restarting the IDE. See also the Getting Started page at the www.arduino.cc website for additional troubleshooting ideas.

Why won't my sketches compile and upload?

Most common problem: incorrect programming syntax. Check for typos, proper capitalization, and semi-colons at the end of statement lines. Some Make:Bot sketches require additional third-party libraries. See the text.

Why won't the robot move after uploading a sketch?

Be sure the servo power jumper wire is in the **ON** position. Check if the batteries need to be replaced.

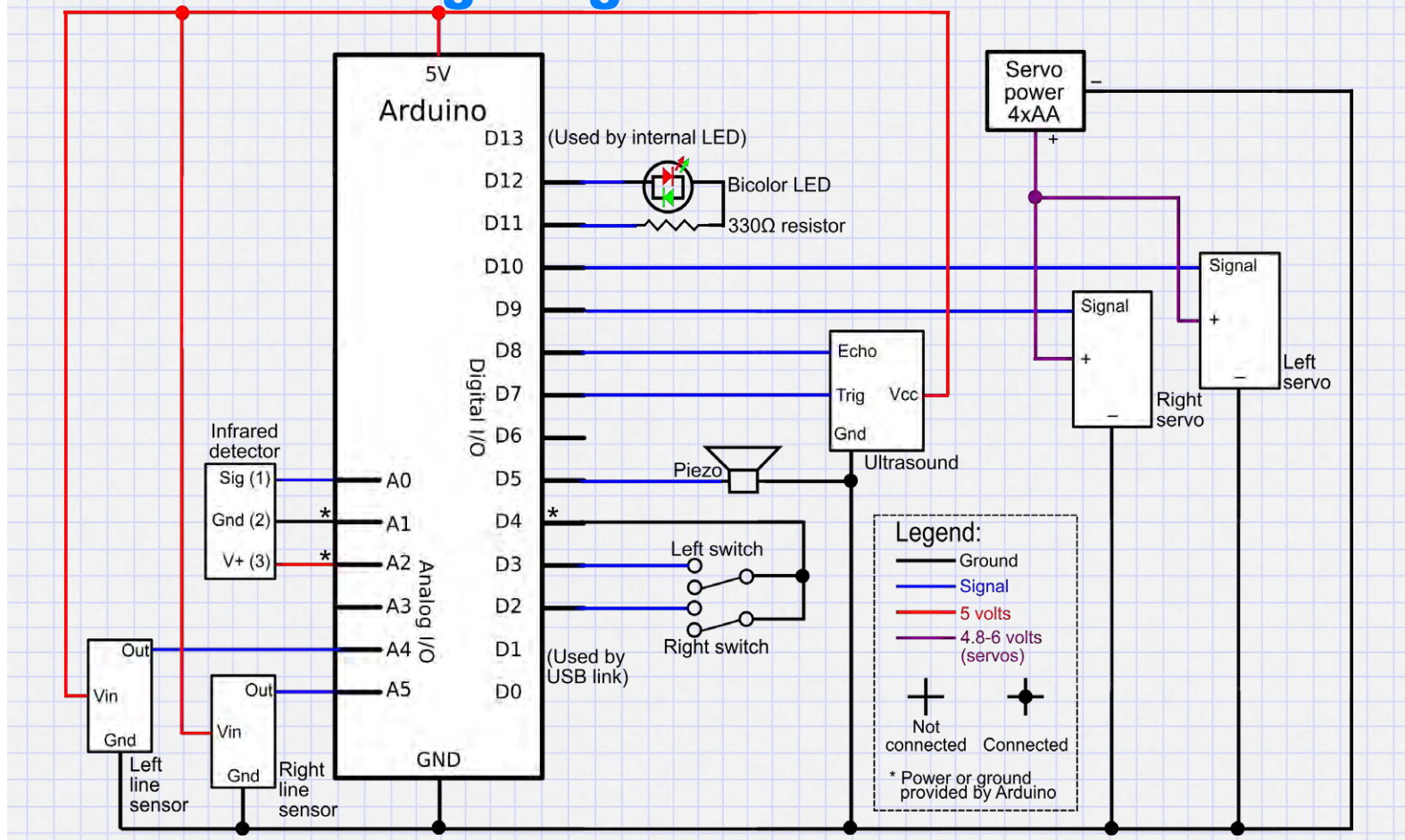
Why is my robot only turning in circles?

Check the servo wiring. Be sure the power, ground, and signal wires are properly placed.

Why does my LED glow red when it should glow green?

The anode and cathode connections of the LED are reversed. Simply turn the LED around in its pins.

Make:Bot Wiring Diagram



Resources for Building Better Bots

Be a robot maker maven with these affordable books and kits. These are among my favs!

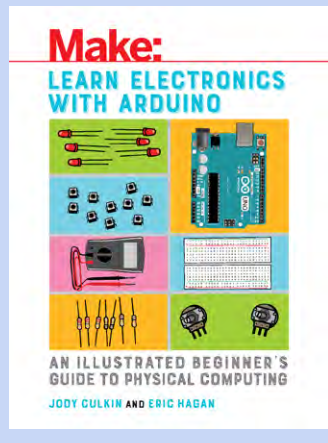
Everything is available at www.makershed.com.

Learn Electronics with Arduino

Culkin & Hagen

No experience with electronics or Arduino? No problem! Covers everything you need to know.

Includes quick hands-on projects. In color, with tons of illustrations.



Soldering Kit

Master the art of soldering with this handy kit—comes with soldering iron and solder.

Also includes tools, cleaning supplies, how-to book, and nifty practice project!



Easy Electronics Kit

Learn electronics without blowing up your brain!

Includes parts and project booklet written by electro-guru Charles Platt.

...AND MORE!

Getting Started With Arduino, 3rd Ed., Banzi & Shiloh

Getting Started with Soldering, de Vinck



Parts List

Find these parts online at www.jameco.com

| Item | Jameco Part | Qty |
|--|-------------|-----|
| Arduino Microcontroller UNO R3 | 2163840 | 1 |
| USB-A Male to USB-B Male Cable* | 222010 | 1 |
| 400-Point Solderless Breadboard | 20601 | 1 |
| 9V Battery Snap with Barrel Plug | 2207056 | 1 |
| 4x AA Battery Holder with Wired Connector | 105858 | 1 |
| Flexible-wire Jumpers Male to Male (40pcs) | 2260738 | 1 |
| Flexible-wire Jumpers Male to Female (40pcs) | 2260754 | 1 |
| 70-piece solid-wire jumper wire kit | 2127718 | 1 |
| Continuous Rotation Servo | 283039 | 2 |
| Boe-Bot/Sumobot Wheel/Tire Set | 2109624 | 2 |
| Piezo Transducer | 2098523 | 1 |
| Snap Action Switch | 187805 | 2 |
| HC-SR05 Ultrasonic Distance Sensor | 2245407 | 1 |
| 38 kHz Infrared Receiver | 2229351 | 1 |
| QTR-1A Reflectance Sensor (2-Pack) | 2229343 | 1 |
| LED Bi-Color Green/Red | 94553 | 10 |
| Resistor 330 Ohm 1/4 Watt | 690742 | 10 |
| 3M Tartan Vinyl Electrical Tape Roll | 285587 | 1 |

*The USB cable you get depends on the USB connectors on your computer and Arduino. See Page 8.

Reflects
minimum
quantity
requirements

Save time! Save money!
Get a kit of all electronics parts from:

makershed.com/products/make-how-to-make-a-robot

Find these materials at a local craft store...

| Item | Qty |
|--|-----|
| 1/4" Foam board, 20" x 30" | 1 |
| Velcro tab, 1", round or square | 5 |
| Double-sided foam tape (1") | 1 |
| White poster board, 22" by 28" | 1 |
| Round hole plug, 3/4" to 1" diameter | 1 |
| 3/8" to 1/2" height (example: Cindoco #F34B) | |

Get these most anywhere...

| | |
|---------------------------------------|---|
| AA batteries, alkaline or NiMH | 4 |
| 9V alkaline battery | 1 |
| Universal remote (Sony TV compatible) | 1 |

Get Code!

All code examples are available online for your convenience.
Get the code from the **Make:Bot Support Site**:

www.robotoid.com/make-bot

1. Click on the code file you wish to download and save to your computer.
2. All examples are compressed in zip file format. Unzip to your Arduino sketches directory.
3. Open the example in the Arduino IDE software program for use with your Arduino.

For more information see the **Getting Started** section at www.arduino.cc.