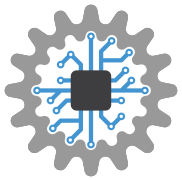


Curtin Robotics Club

TinyBot

Ilke Dincer
ilke@curtinrobotics.org

February 25, 2021

Contents

1 Introduction

There are many different components of a robot; the most important being the microcontroller (the brain), the motors (the legs), and any sensors (how the robot sees the world).
This guide will take you through building a simple robot dubbed TinyBot. It has 2 wheels, a caster wheel, a battery, an arduino, and a breadboard.

2 Assumed Knowledge

The below knowledge is assumed for this project. Feel free to ask other CRoC members for help or explanation of the below concepts.

- Basic circuit knowledge
  - Current, Voltage, Resistance
  - Series and Parallel
- Breadboards
- Basic coding skills

3 Components

Component	Quantity	Price	Sources
Arduino Uno	1	$5-$80	Arduino's are discussed in Section 5. A genuine Arduino will cost about $80, however Arduino clones can be bought online for as little as $5. Ebay is a good starting point for finding an Uno.
Breadboard	1		
N20 Motor	2		
Dual H-Bridge	1	$7 - $16	Altronics stocks both motor drivers and motor controllers, though they can also be found on eBay and sites such as RS components. For this tutorial, only a basic H-bridge driver is necessary (costing about $7), though more expensive motor controllers can be used as well.
Wheels	2	$0	The wheels for this project are 3D printed, and are supplied by the club.
Caster Wheel	1	?	The caster wheel consists of 2 parts, a marble and it's 3D printed casing. The 3D print will be supplied by the club at no charge, however you must source your own marble.

Additional sensors can be bought and integrated with TinyBot, however that is not covered in this project guide.

4 Construction

5 Microcontroller

A microcontroller is a really small microcomputer on a very small chip, see Figure 1. These are used in a variety of devices; including robots, vending machines, phones, computers, etc.

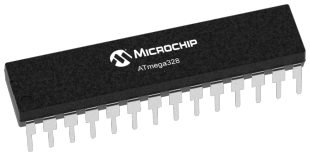


Figure 1: A Microchip

Arduino's are a development board; consisting of an microcontroller, power regulation, and input/output (also known as IO) pins. As microcontrollers are very tiny prototyping with them or using them to build something would be really difficult. The purpose of an arduino is to provide a medium that allows easy development with microcontrollers. There are many different kinds of arduinos, each using a different microchip.

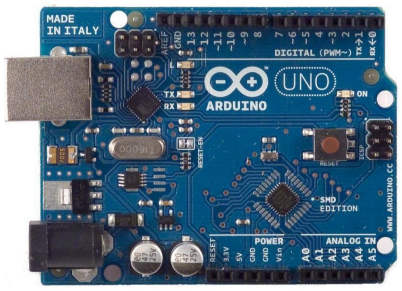


Figure 2: An Arduino Uno

The difference between Arduino Uno, Mega, and Nano is the form factor and the microchip used on them. Figure 2 is what a real Arduino Uno looks like, though the colour and text may be different from brand to brand. Genuine Arduinos are quite expensive, there are many clones available which are much cheaper. Figure 3 shows a stylised view of an Uno, labelling all the different pinouts.

Arduino's and other development boards are used extensively by hobbyists, they are cheap, easy to use, and extremely versatile. Arduino's are used in nearly every CRoC project, and can be used in countless DIY projects.

An Uno has many different ports and pins. Figure 3 shows and labels all the different ports on a standard Uno.

An important distinction to make is between the pins 5V, 3.3V, and VIN. VIN stands for voltage in; and this port is used to supply power to the arduino from batteries. Power can also be supplied through the barrel jack connection, see the black rectangle like block on figure 3.
The 5V and 3.3V pins supply 5 volts or 3.3 volts respectively for powering other commponents, such as LEDs, ICs, or sensors.

!	Never put supply voltage into the 3.3V or 5V pins; this will break the Arduino.

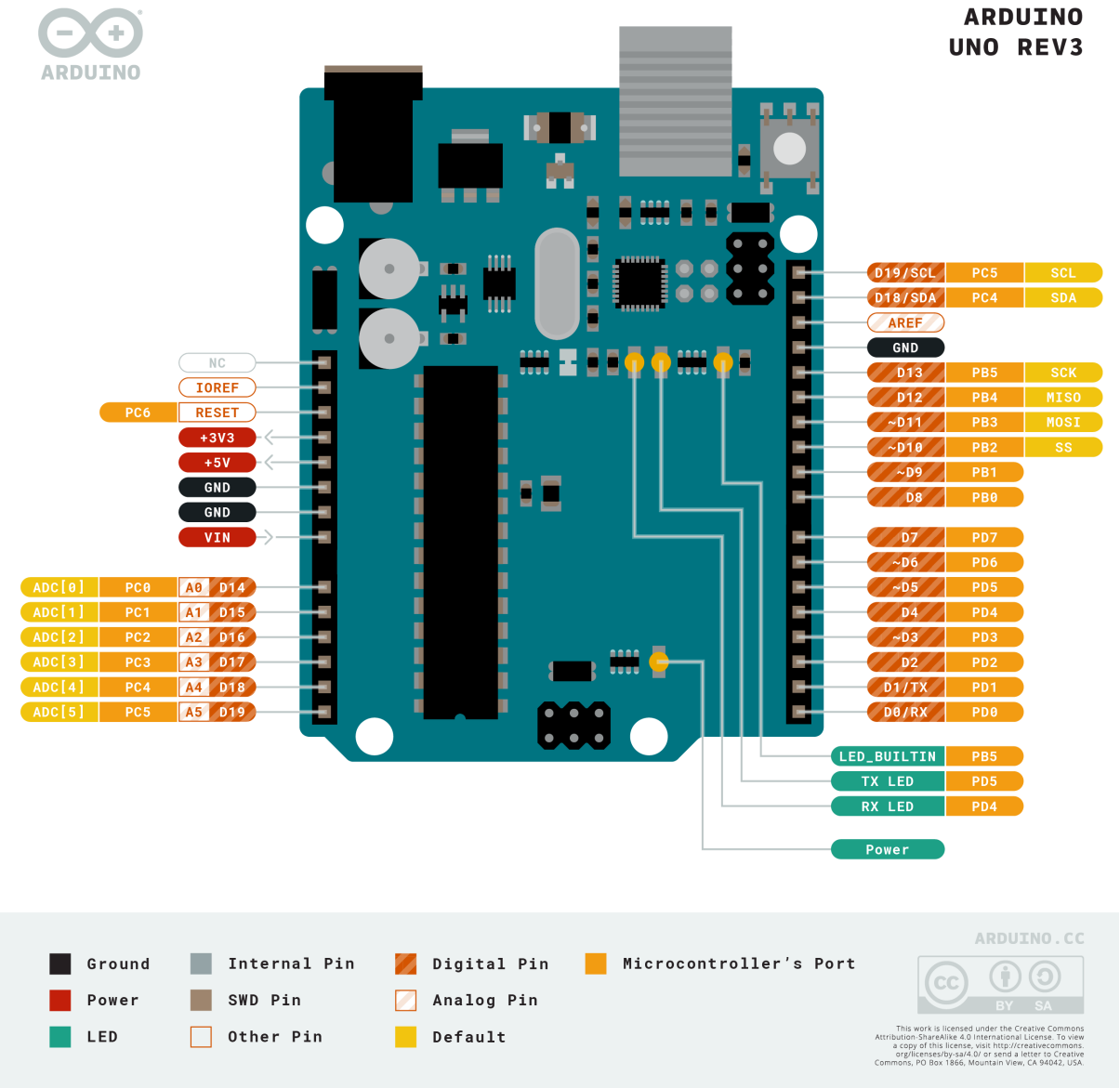


Figure 3: Pinout of Arduino Uno

There are two different kinds of pins on an arduino, digital and analog. Look at the legend to see which pins are digital, The digital pin numbers all start with `D`, just as the analog pins start with `A`. Analog pins can also be used as digital pins, but digital pins cannot be used as analog pins.

Digital pins can be set to `HIGH` or `LOW`, think of it like a button it can be on or off. Setting a pin to `HIGH` turns it on, and `LOW` turns it off. As circuits get more complicated, it is possible that setting a pin to `LOW` will enable a part of the circuit; though for beginners it is best to think of `HIGH` as on and `LOW` as off; especially when working with a H-Bridge.

6 Motor

gearbox, motor
To follow this guide it is not necessary to have an understanding of how motors work, though it may be interesting for you to learn. This link has a good indepth explanation. Motors turn in proportion to the amount of current put through them. More current means a faster motor.
When a motor stalls, it stops rotating. This happens when there is more force acting on the motor shaft than the motor can overcome. The stall torque of a motor is the maximum current drawn when a motor stalls, in other words, applying its maximum torque.

Similarly, free current is the current drawn when the motor is rotating freely, under no load.

Each motor has a certain amount of torque it can provide. Gearboxes can be attached to a motor to increase the amount of torque provided, and change the rotations per minute (RPM) of the motor.

7 Motor Controller

The motors used in this guide, the N20 motors, have a stall current of 1.6A (see section 6 for what stall current means). The digital pins on an Arduino Uno supply at most 40mA. This is not enough to power the motors.

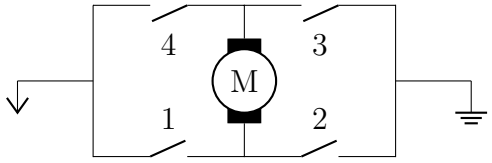
To get around this, the Arduino instead interfaces with a **motor controller**. Motor controllers have a separate power supply that can supply enough current to drive the motor. Motor controllers also have digital inputs that allow control of the motor.

An added benefit of using a motor controller is that it is possible to control the direction and the speed of the motor.

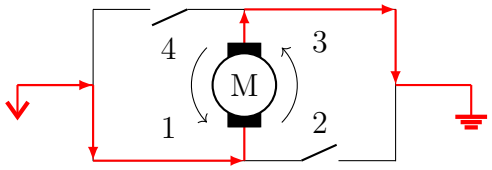
The phrase motor controller is often used as a generic term for any device, circuit, or IC which controls a motor. However, motor controllers are a circuit that consists of a motor driver and some digital harness that acts as an interface to the driver. Motor controllers can be dropped into a circuit and easily controlled, allowing feedback from the motor and more control than a simple driver provides.

7.1 Motor Driver

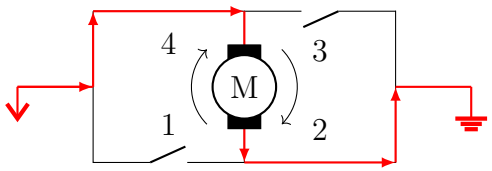
A basic motor-**driver** is a H-bridge. The simplest H-bridge is shown in the below schematics, as well as an explanation of how using a H-bridge allows control over the motors direction.



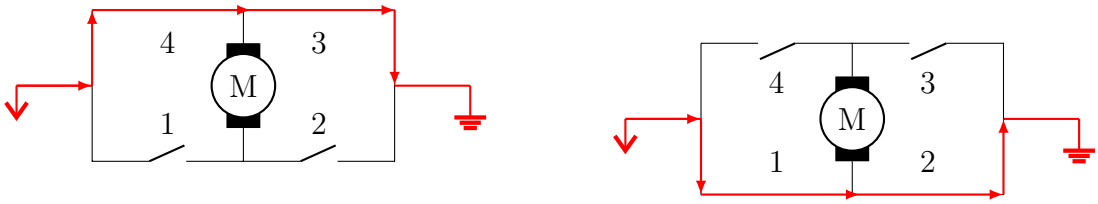
When switches 1 and 3 are closed, the current will flow through the motor making it turn anticlockwise.



In the same vein, closing switches 2 and 4 will cause the motor to turn clockwise.



If pins 4 & 3 or pins 1 & 2 are closed at the same time, a short circuit will be formed and the H-bridge will break.



Breaking a H-bridge is fairly common, especially the cheaper low power ones. Some higher end H-bridges are designed to prevent the H-bridge from shorting if the wrong pins are closed. Most motor-controllers will have this protection built-in, though most motor-drivers do not.
While working on this guide, don't worry if your H-bridge stops working suddenly, it is quite common to short them out.

7.2 Motor Controller

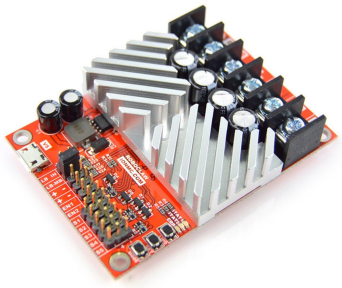


Figure 4: RoboClaw Motor Controller

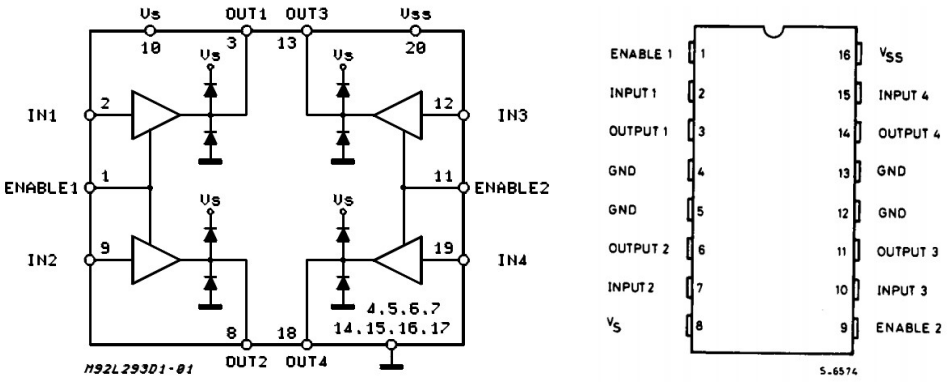
A motor controller has a lot more features than a motor driver. See, for example, the RoboClaw (see Figure 4) which has in built features such as PID tuning, data logging, diagnostic LEDs, and serial control.

The in-built control modes, as well as being capable of serial communication, is present only in motor controllers. Motor drivers are far simpler in comparison.

7.3 L293D H-Bridge IC

The information in this section is included for information's sake, you don't need to understand it to be able to build TinyBot; though it can be useful knowledge. You can skip this section if you would like.

Datasheets hold a lot of useful information about microchips, the data sheet for the L293D H-Bridge can can be found online (linked here).



(a) Block Diagram for 20 Pin Chip (b) Pinout of 16 Pin Chip

Figure 5: Diagrams from the L293D datasheet

Being able to read diagrams such as in Figure 5a really depends on whether or not you know what each symbol means.

The hollow circles along the outside rectangle of the block diagram represent the pin outs on the microchip, they can be matched up with labelled pins in Figure 5b. There is also a 20 pin variant of this H-bridge, which is why some of the pin labels on the block diagram are greater than 16.

The hollow triangles in the block diagram are buffers that isolate the the input signal from the enable line.

The bold black triangle and line combinations represent components called Transistors.

8 Intro to Programming

This section will briefly explain what programming is, if you have programmed before, or feel confident in your programming knowledge, feel free to skip to the next section.
If you have 0 programming experience, and are very confused by this section you may find it worthwhile to google programming guides and tutorials to really help you understand how to code and how code works. There are some links you may find useful at the end of this section.

Programming is how we tell computers what we want them to do. We can program a computer to blink a light, play a noise every time something comes too close, or drive a robot around. The set of instructions we write is called code, and so programming is also called coding. Like with spoken languages, there are many different programming languages. Popular languages include Java, C++, and Python. This guide will be introducing C++.

Each coding language has a specific structure that must be followed, called syntax. For the computer to understand your code, it must conform to the expected syntax exactly. Syntax errors occur when there is a comma somewhere there shouldn't be, a word that is capitalised when it shouldn't be, there's an extra bracket, etc.
The computer will tell you when there's a syntax error, and will often tell you what line the error is on. Sometimes this line number is a bit off, the syntax error might be a few lines above the specified line. When you first start coding, noticing where there is a syntax error is quite difficult; however as with many things, as you get more used to programming you get better at noticing where the syntax error is.

A crucial part of programming is saving data, you want to be able to save and store some data that you can use later. This data is called a variable, and because computers require specific syntax, you need to tell the computer exactly what type of data the computer is being told to remember.

Here we tell the computer to remember an `int`eger variable called `number1` which has the value 4.

```
1 int number1 = 4;
```

The keyword `int` is really important, it tells the computer that the variable `number1` is an integer (a whole number, negative or positive).

Some other basic keywords (known as datatypes) that can be used in place of `int` are:

- `float` - a decimal number such as 0.5, 3.14159, etc.
- `char` - a character, such as 'a' or '2' - note the single quotation marks which are important when declaring (creating) a character variable
- `string` - a word or lots of words e.g. "hello world" or "this is a string"; double quotation marks are essential for strings.

These datatypes are a common concept across many languages, though some languages don't require you to explicitly state what datatype a variable is.

Some more C++ resources:

- https://www.w3schools.com/cpp/default.asp
- https://www.learncpp.com/

9 Intro To Arduinos

To program an Arduino, you will need a USB cable, and a laptop/computer with the Arduino IDE installed, IDE stands for Integrated Development Environment.

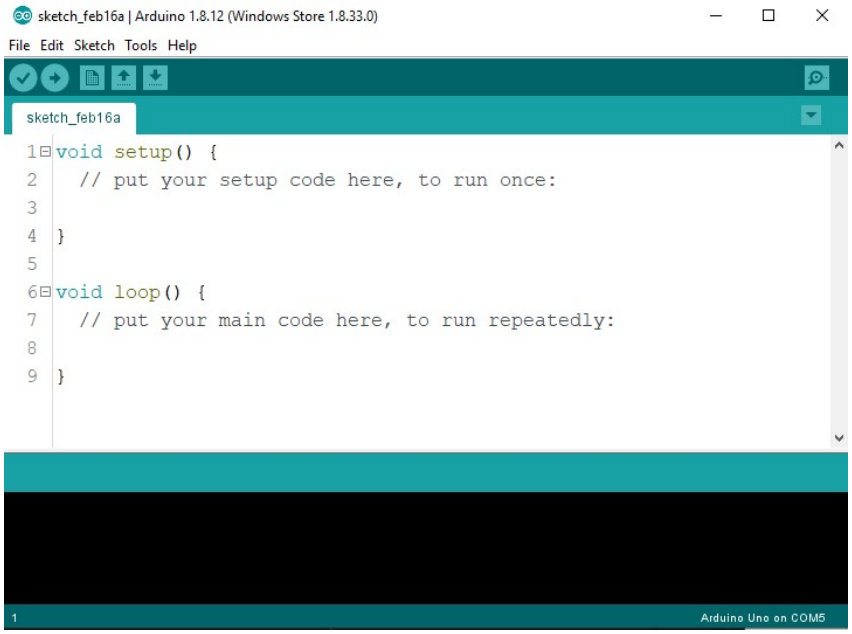


Figure 6: The Arduino IDE

The two round buttons in the top right, the tick and the arrow, are the verify and upload buttons. Verify checks your code, making sure that the syntax (the structure of the code, think of it like a grammar checker) of your code is correct. Upload sends the code you've written to the Arduino board.

However, before you can upload your code there's some setup you need to do. First of all, you need to select the board by going into the Tools menu, as shown in the image below.

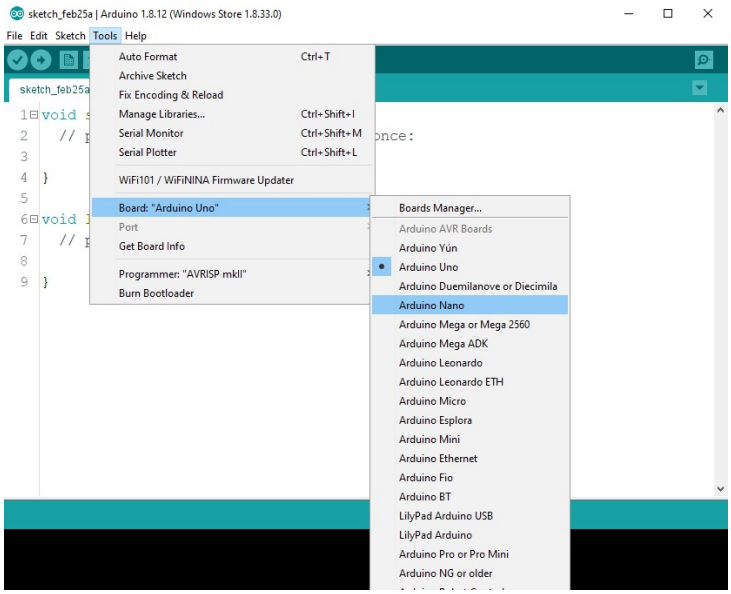


Figure 7: Selecting Board

Next, you need to select the USB port that the Arduino is connected to. This is also done through the Tools menu. The port should appear as COM followed by a number. This number will change depending on which USB port the Arduino is plugged into. If the port option is greyed out for you, try plugging the Arduino into a different USB port.

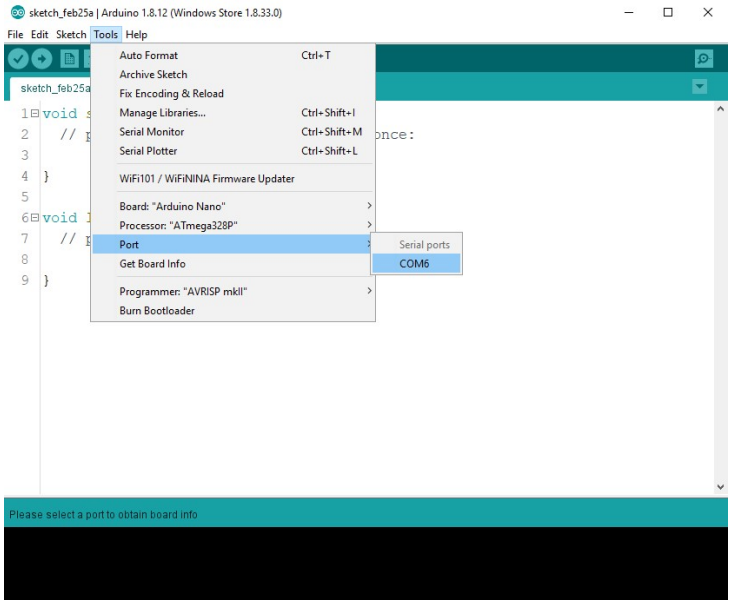


Figure 8: Selecting Port

Arduino's are programmed in the programming language C++; though there are a few differences. The below code section details a few features of coding.

```
// this is a comment, comments are not read by the computer and can
   be anything you want

// variables declared not in a function will be accessible
// in all functions
int global_var = 0;

void setup {
  // everything in this function will run once

  // this code will run when the board is powered on,
  // or when the reset button is pressed
}

void loop {
  // everything in this function will run repeatedly
}
```

A useful feature of the Arduino IDE is all the example code which is provided.

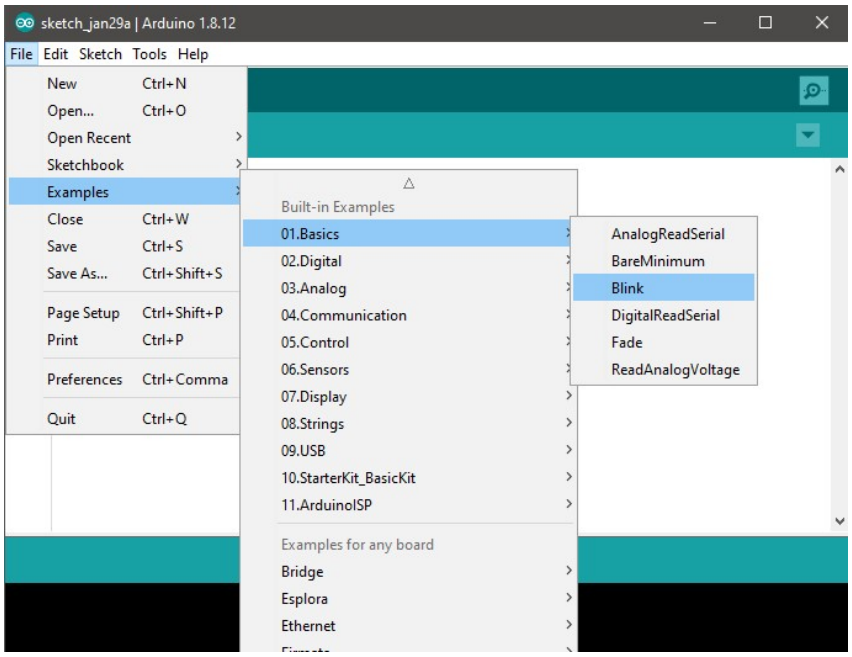


Figure 9: Arduino IDE Example Code

The simplest Arduino example is the Blink code, which turns on and off an onboard LED.

```
1 void setup() {
2   // initialize digital pin LED_BUILTIN as an output.
3   pinMode(LED_BUILTIN, OUTPUT);
4 }
5
6 // the loop function runs over and over again forever
7 void loop() {
8   // turn the LED on (HIGH is the voltage level)
9   digitalWrite(LED_BUILTIN, HIGH);
10
11   delay(1000);       // wait for a second
12
13   // turn the LED off by making the voltage LOW
14   digitalWrite(LED_BUILTIN, LOW);
15
16   delay(1000);       // wait for a second
17 }
```

There are a few common aspects present in the code of nearly every Arduino project, no matter how simple or complicated.

`pinMode(<pin number>, <mode>)` sets a digital pin on the Arduino to be either an `INPUT` or and `OUTPUT`.

`digitalWrite()` is used to set digital pins `HIGH` and `LOW`.

10 Wiring

If you just want to see the wiring schematic, see Figure 10. Continue reading for an explanation of the L293D H-Bridge and of the circuit.

Before starting construction on any project, it is always a good idea to wire up the circuit on a flat breadboard and Arduino; as it is far easier to build the circuit on its own before putting together all the parts.

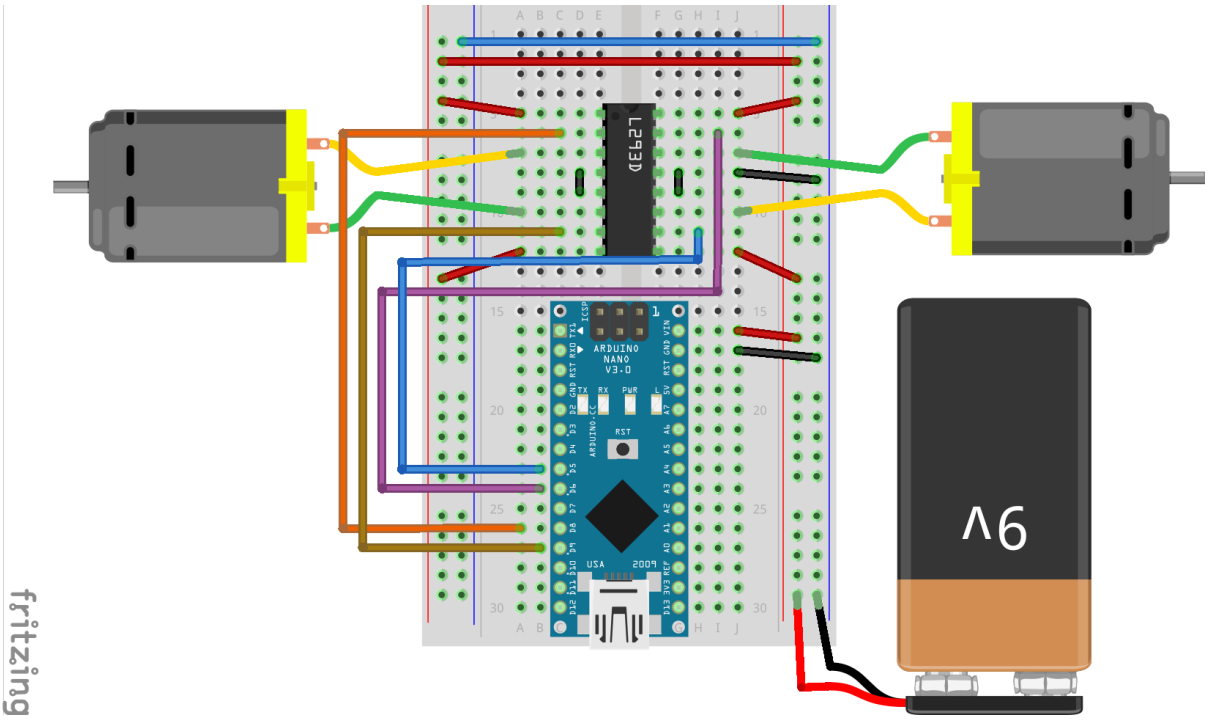


Figure 10: Wiring Schematic for H-Bridge

Make sure to check which wire is the positive wire on the motor you have, it should be written on the back plate of the motor. Plugging the motor in backwards will not break anything, the motor will just spin backwards.
Swapping the motor direction can be done by swapping the green and yellow wires attached to the motor.

11 Code

Thinking back to the H-Bridge, to control the direction of the motor we want to control which switches are closed and which are open. Switches are generally active low - which means that they are off by default. To turn them on, in other words close them, we want to set pins to high.

To make it easier on ourselves, let's assign the pin numbers for each switch to a variable. Let's put these at the very top of our script, so that every function we write later can access the variables.

```
1 //Motor 1
2 const int motorPin1  = 6;  // Pin 15 of L293
3 const int motorPin2  = 5;  // Pin 10 of L293
4 //Motor 2
5 const int motorPin3  = 9; // Pin  7 of L293
6 const int motorPin4  = 8;  // Pin  2 of L293
```

The keyword `const` means that the variable cannot be changed later in the code. The value of the variables can be changed to any digital pin number on the Arduino, though make sure that the variable number matches the physical pin used.

Next we need to set the pin mode of the pins we're using to interact with the H-Bridge.

```
1 void setup(){
2     //Set pins as outputs
3     pinMode(motorPin1, OUTPUT);
4     pinMode(motorPin2, OUTPUT);
5     pinMode(motorPin3, OUTPUT);
6     pinMode(motorPin4, OUTPUT);
7 }
```

To drive forward, we want the motors to turn in the same direction and so we want to enable `IN1` and `IN3`.

```
1 void setup() {
2     // set pinMode as done above
3
4     digitalWrite(motorPin1, HIGH);
5     digitalWrite(motorPin2, LOW);
6     digitalWrite(motorPin3, HIGH);
7     digitalWrite(motorPin4, LOW);
8     delay(2000); // wait for 2 seconds
9 }
```

As this code is in `setup()`, it will run once, i.e. the robot will drive forwards for 2 seconds, then stop. If you want the robot to drive forward continuously, then move the digital writes to `loop()`. Make sure to put the robot on the floor so that it doesn't drive off the desk.
As you can imagine, having to set all 4 motor pins individually when controlling the robot can get quite tedious. A simple solution to this is to create functions for driving in each direction. We're going to assume that the left motor is motor 1, and the right motor is motor 2.

```
1  void driveForward(n) {
2      // motor 1 - left
3      digitalWrite(motorPin1, HIGH);
4      digitalWrite(motorPin2, LOW);
5      // motor 2 - right
6      digitalWrite(motorPin3, HIGH);
7      digitalWrite(motorPin4, LOW);
8      delay(n*1000); // wait for n seconds
9  }
10
11 void driveBackwards(n) {
12     // motor 1 - left
13     digitalWrite(motorPin1, LOW);
14     digitalWrite(motorPin2, HIGH);
15     // motor 2 - right
16     digitalWrite(motorPin3, LOW);
17     digitalWrite(motorPin4, HIGH);
18     delay(n*1000); // wait for n seconds
19 }
20
21 void turnLeft(n) {
22     // turn off right motor, and drive left motor forwards
23     // motor 1 - left
24     digitalWrite(motorPin1, HIGH);
25     digitalWrite(motorPin2, LOW);
26     // motor 2 - right
27     digitalWrite(motorPin3, LOW);
28     digitalWrite(motorPin4, LOW);
29     delay(n*1000); // wait for n seconds
30 }
31
32 void turnRight(n) {
33     // turn of left motor, and drive right motor forwards
34     // motor 1 - left
35     digitalWrite(motorPin1, LOW);
36     digitalWrite(motorPin2, LOW);
37     // motor 2 - right
38     digitalWrite(motorPin3, HIGH);
39     digitalWrite(motorPin4, LOW);
40     delay(n*1000); // wait for n seconds
41 }
42
43 void stop(n) {
44     // motor 1 - left
45     digitalWrite(motorPin1, LOW);
46     digitalWrite(motorPin2, LOW);
47     // motor 2 - right
48     digitalWrite(motorPin3, LOW);
49     digitalWrite(motorPin4, LOW);
50     delay(n*1000); // wait for n seconds
51 }
```

12 Challenges

These challenges can be done using only the knowledge and skills gained in this guide.

- Turn the robot around on the spot
- Drive in a square

These challenges require knowledge that is not covered in this guide.

- Make a line following robot (requires a light sensor)
- Make a robot that bounces of walls or other objects in its way (requires an ultrasonic sensor)

Tinybot has been designed so that sensors and additional functionality can be easily added, adding components to a breadboard is fairly trivial. However, the programming aspect of these extra challenging challenges is quite involved.