

CURTIN ROBOTICS CLUB

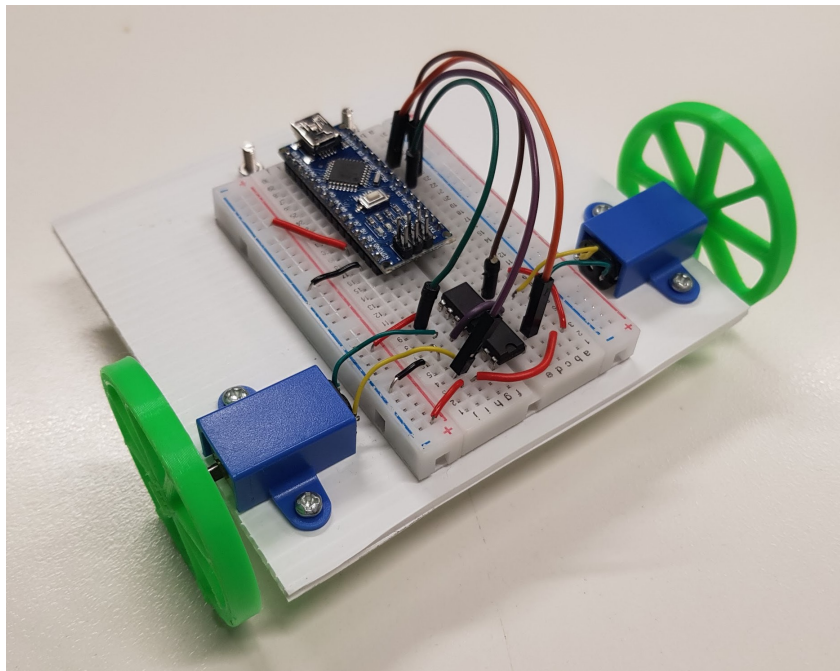
TinyBot

A Beginner's Guide to Robotics

Ilke Dincer

ilke@curtinrobotics.org

March 21, 2021





Contents

1	Introduction	1
2	Assumed Knowledge	1
3	Components	1
4	Microcontroller	2
5	Motor	4
5.1	Gear Boxes	4
6	Motor Controller	6
6.1	Motor Driver	6
6.2	Motor Controller	7
6.3	L293D H-Bridge IC	8
7	Intro to Programming	9
8	Intro To Arduinos	10
9	Wiring	14
10	Code	16
11	Challenges	19



1 Introduction

There are many different components of the average robot: the microcontroller (the brain and nervous system), the actuators (e.g. motors), and any sensors (how the robot senses the world).

This guide will take you through building a simple robot dubbed “TinyBot”. It has two 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, and resistance
 - Series and parallel circuits
- Breadboards
- Basic coding skills

3 Components

Component	Quantity	Price	Sources
Arduino Uno	1	\$5-\$80	Discussed further in 4 , a genuine Arduino will cost about \$80. Fortunately, Arduino clones can be bought online for as little as \$5. Ebay is a good starting point for finding an Uno.
Small breadboard	1	\$8	Can be found at Altronics or Jaycar.
Micro N20 Motor	2	\$17.25	Altronics or Jaycar.
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 (about \$7).
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.

While additional sensors can be bought and integrated with TinyBot, this is not necessary for this tutorial.



4 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.

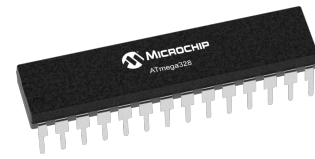


Figure 1: A Microchip

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.

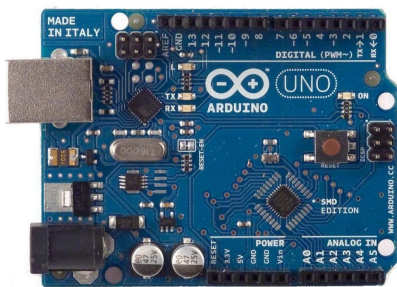


Figure 2: An Arduino Uno

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 C/C++ project, and can be used in countless DIY projects.

An Uno has many different ports and pins, as figure 3 shows.

An important distinction to make is between the pins 5V, 3.3V, and VIN. VIN stands for “voltage in”: this port is used to supply power to the Arduino, such as a battery. 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 components, such as LEDs, ICs, or sensors.



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

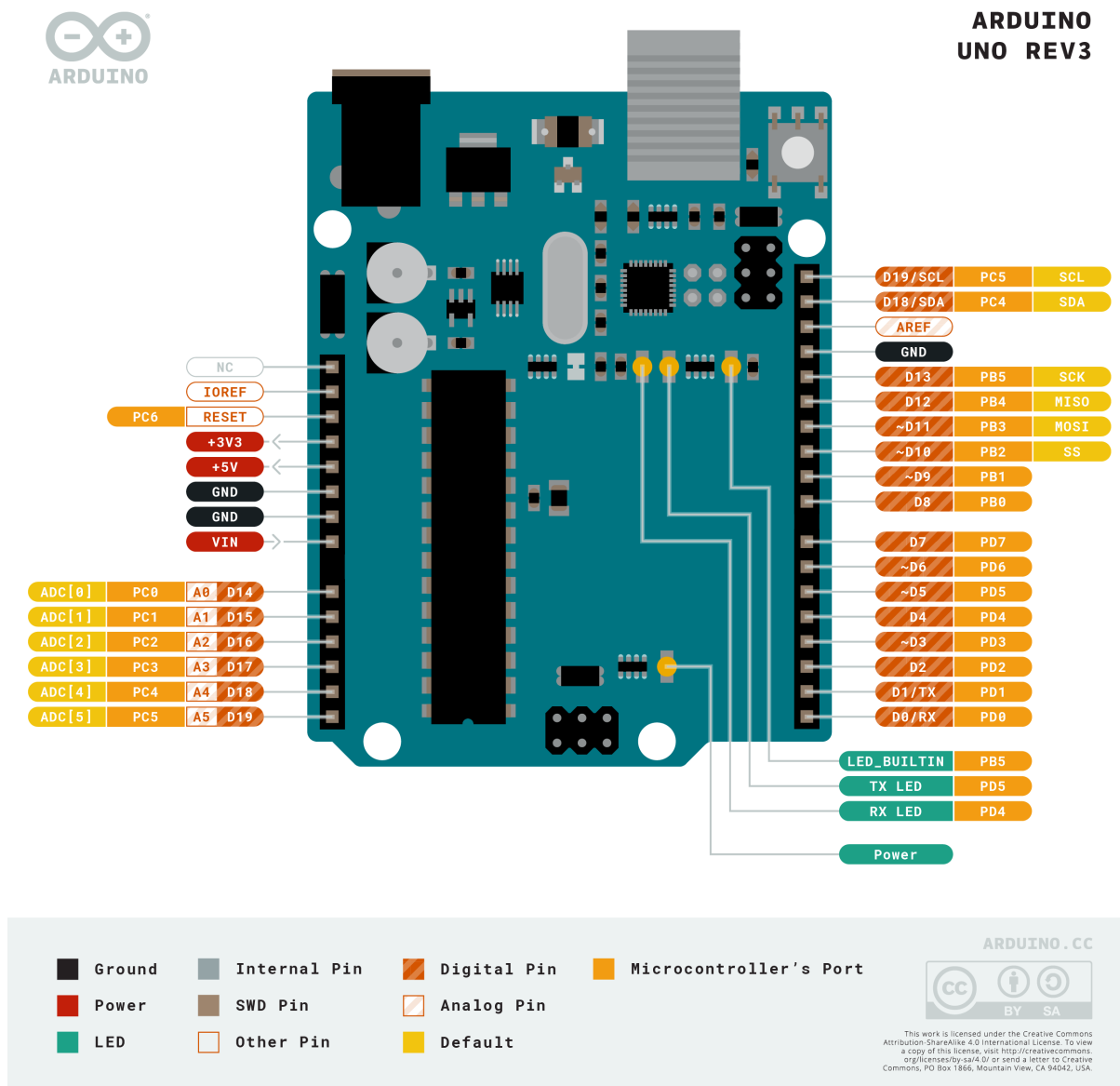


Figure 3: Pinout of Arduino Uno

There are two different kinds of pins on an Arduino: digital and analog. 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.



5 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. The more electrical current passing through the motor, the more torque the motor is outputting through its shaft. The more torque the motor outputs, the more it can accelerate. When a load is applied to the motor shaft, the more current will be needed to continue accelerating. When the load applied to the motor shaft roughly equals the motor's torque, the shaft stops rotating. This is known as "stall torque". Similarly, "free current" is the motor's current draw when the shaft is rotating without a load. Each motor has its own maximum torque and speed it can output, which is heavily reliant on the size of the motor. Attempting to go above this limits can break the motor or overheat the system.

This [link](#) has a good indepth explanation.

5.1 Gear Boxes

Sometimes, we want the motor to output a torque higher than what it can provide. Ideally, the motor should be able to output high torques without high currents, as the higher the current is the more heat is generated and the more power hungry the system is. This can be accomplished with a gearbox.

Gearboxes can increase torque in exchange for speed or vice-versa. Gearboxes have a property called "gear ratio" which dictates this change in torque and speed. By counting the number of teeth of both the driven and driving gears, the output speed and torque can be calculated as below. Notice that ω is the angular speed, D is the diameter, N is the number of teeth, τ is the torque, and η is the efficiency (typically $\sim 95\%$ for a pair of spur gears). Hence, when the output gear has more teeth, the torque increases and the speed decreases, and vice-versa. Also notice that the driving gear will rotate in the opposite direction as the driven gear.

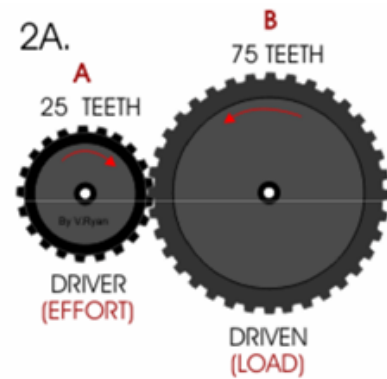


Figure 4: Driver and Driven Gear

$$R = \frac{N_B}{N_A} = \frac{D_B}{D_A} = \frac{\omega_A}{\omega_B}$$

Hence,

$$\omega_B = \frac{N_A}{N_B} \cdot \omega_A$$

and

$$\tau_B = \eta \cdot \frac{N_B}{N_A}$$

When connecting multiple gears, we call this a gear train. There are two types of gear trains: simple and complex. Simple gear trains are much like their name, simple. They



connect next to each other and you only need to do the calculation for the first gear as an input gear and last gear as the output gear. These are useful for rapidly increasing torque or speed across a long distance with a short height. Alternatively, a compound gear train means there can be multiple gears on the same shaft, meaning the gears on the same shaft will rotate at the same speed and torque despite being different sizes. You won't have to deal with compound gear trains for this project, but feel free to investigate it more. To summarise gears, the ratio of your input gear to output gear affects the characteristics of your output performance. If you want a higher torque use a larger output gear. If you want more speed, use a smaller output gear.

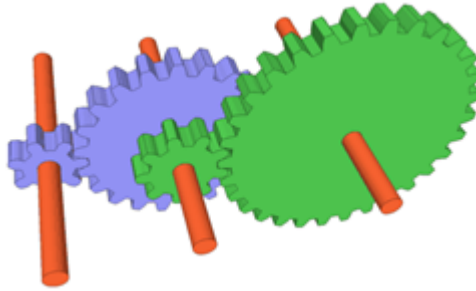


Figure 5: Complex Gear Train



6 Motor Controller

The motors used in this guide, the N20 motors, have a stall current of 1.6 A (see section 5 for what stall current means). The digital pins on an Arduino Uno can supply at most 20 mA without breaking (see the [official webpage](#)). Clearly, this is not enough to power the motors.

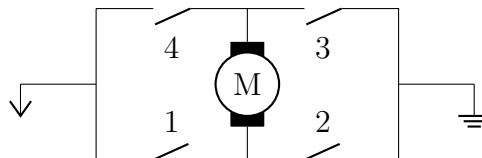
To get around this, the Arduino instead interfaces with a “motor controller”. Motor controllers have a separate power supply that can supply enough current to drive the motor. Motor controllers also have digital inputs that allow control of the motor.

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

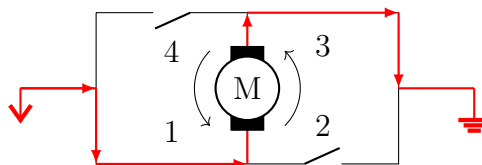
The phrase motor controller is often used as a generic term for any device, circuit, or IC which controls a motor. However, motor controllers are a circuit that consists of a motor driver and some digital harness that acts as an interface to the driver. Motor controllers can be dropped into a circuit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

6.1 Motor Driver

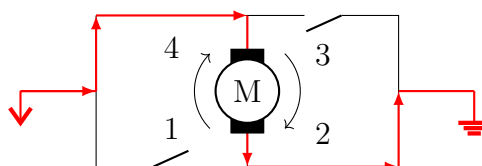
A basic motor-**driver** is a H-bridge. The simplest H-bridge is shown in the below schematics, as well as an explanation of how using a H-bridge allows control over the motors direction.



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

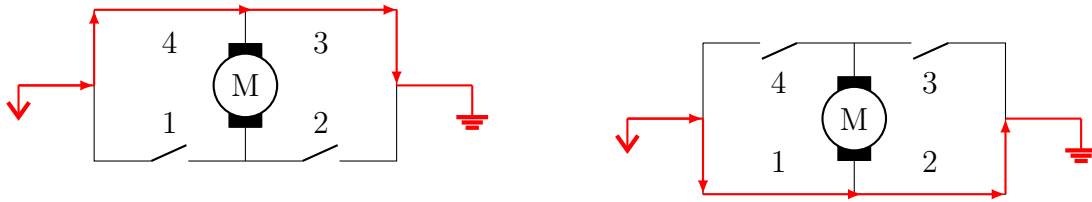


In the same vein, closing switches 2 and 4 will cause the motor to turn clockwise.





If pins 4 and 3 or pins 1 and 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.

6.2 Motor Controller

A motor controller has a lot more features than a motor driver. See, for example, the RoboClaw (see Figure 6) 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.

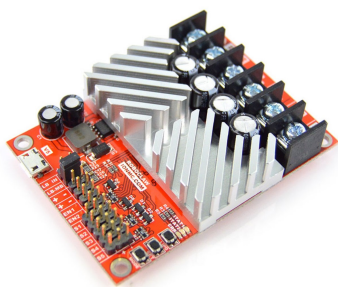


Figure 6: RoboClaw Motor Controller



6.3 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 be found [online](#).

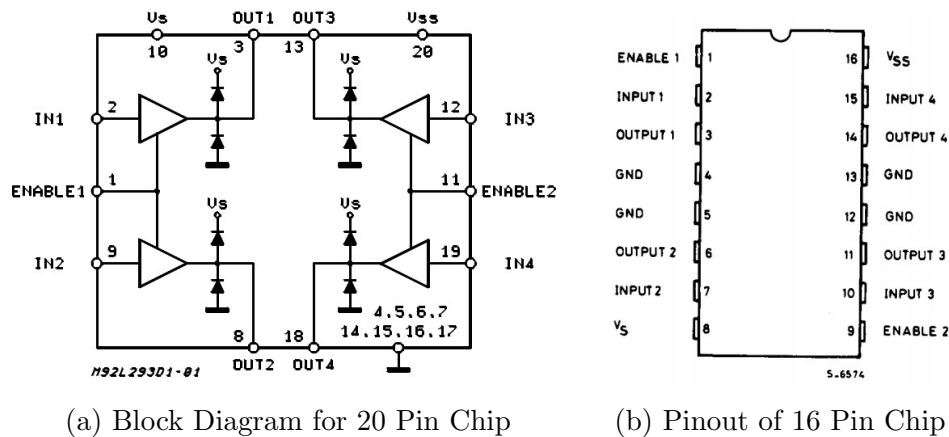


Figure 7: Diagrams from the L293D datasheet

Being able to read diagrams such as in Figure 7a 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 7b. 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. These components essentially act as digital switches, allowing a large current to be activated by sending a small current to the base. Hence, the Arduino can turn on the power to the motors without drwaing too much current.



7 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 the 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, hence why 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 the Arduino implementation of C++.

Each coding language has a set of keywords and special characters that must be used, called “syntax”. For the computer to understand your code, it must conform to the expected syntax exactly. Syntax errors occur when the syntax is not followed, such as there being an unnecessary comma, a letter in the wrong case, 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 due to the complexity of the syntax; 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 and guessing where a syntax error is.

A crucial part of programming is saving data so it can be used later. Variables are the simplest way of saving data. Since C++ is a strong typed language, 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 [integer](#) 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, positive, or 0).

Some other basic datatypes include:

- ▷ [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 sequence of characters e.g. "hello world" or "this is a string"; double quotation marks are essential for strings in C++.

These datatypes are a common concept across many languages, though some languages, like Python, don’t require you to explicitly state what datatype a variable is (known as a weak typed language).



Some more C++ resources:

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

8 Intro To Arduinos

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

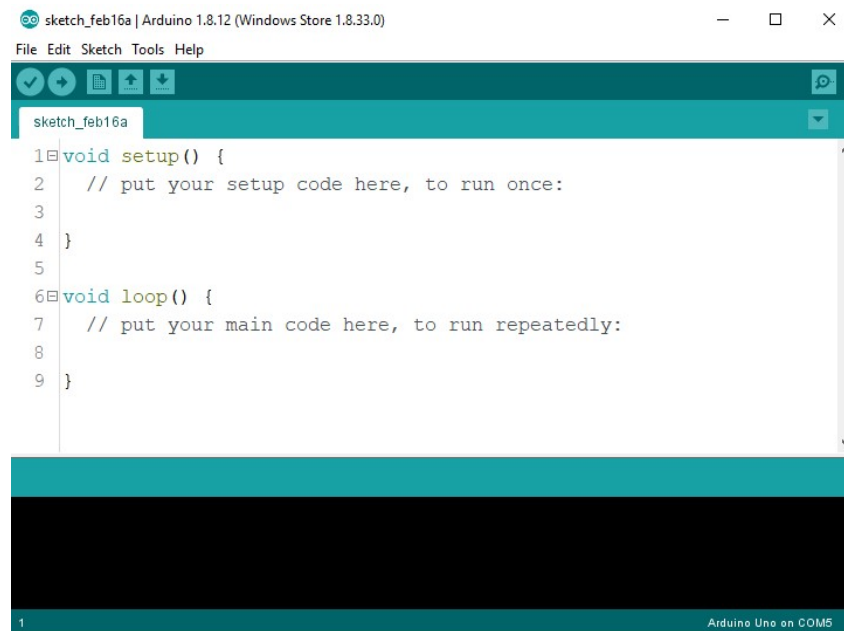


Figure 8: The Arduino IDE

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 (think of it like a grammar checker) of your code is correct. Upload sends the code you’ve written to the Arduino board, if it can be verified.

However, before you can upload your code you need to select the Arduino board being used, as shown in figure 9.

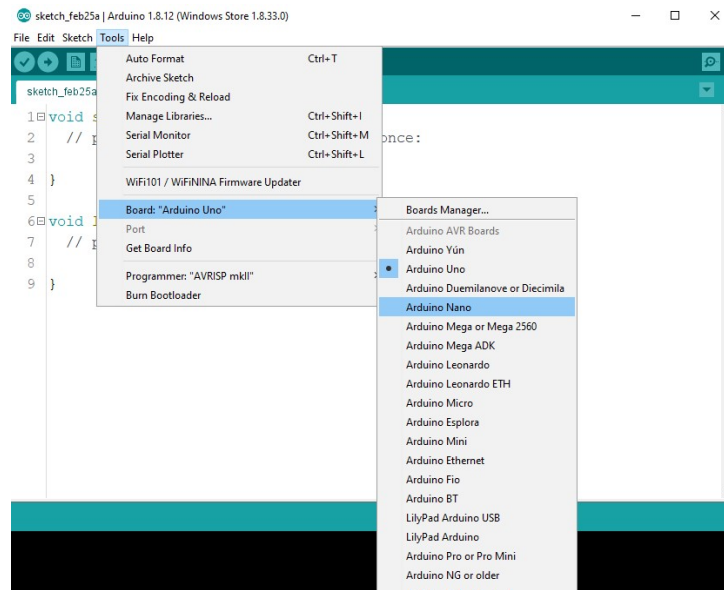


Figure 9: Selecting the Board

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.

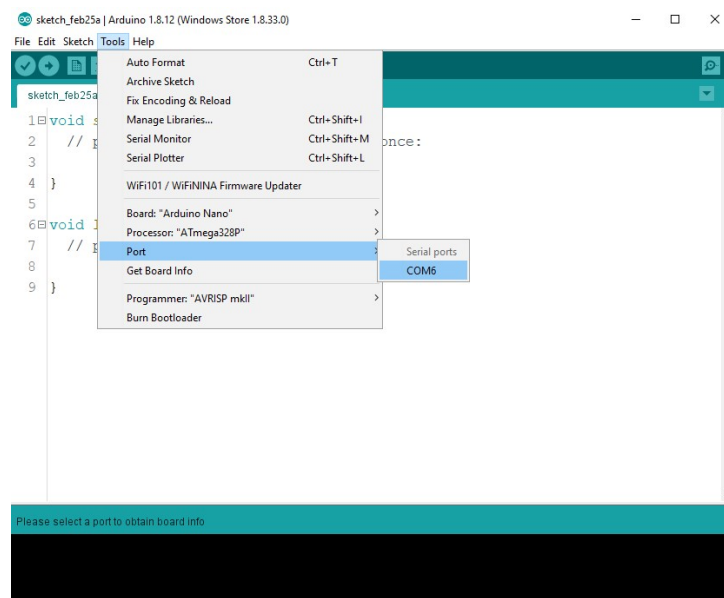


Figure 10: Selecting Port



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.

```
1 // this is a comment, comments are not read by the computer and can
   be anything you want
2
3 // variables declared not in a function will be accessible
4 // in all functions
5 int global_var = 0;
6
7 void setup {
8   // everything in this function will run once
9
10  // this code will run when the board is powered on,
11  // or when the reset button is pressed
12 }
13
14 void loop {
15   // everything in this function will run repeatedly
16 }
```



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

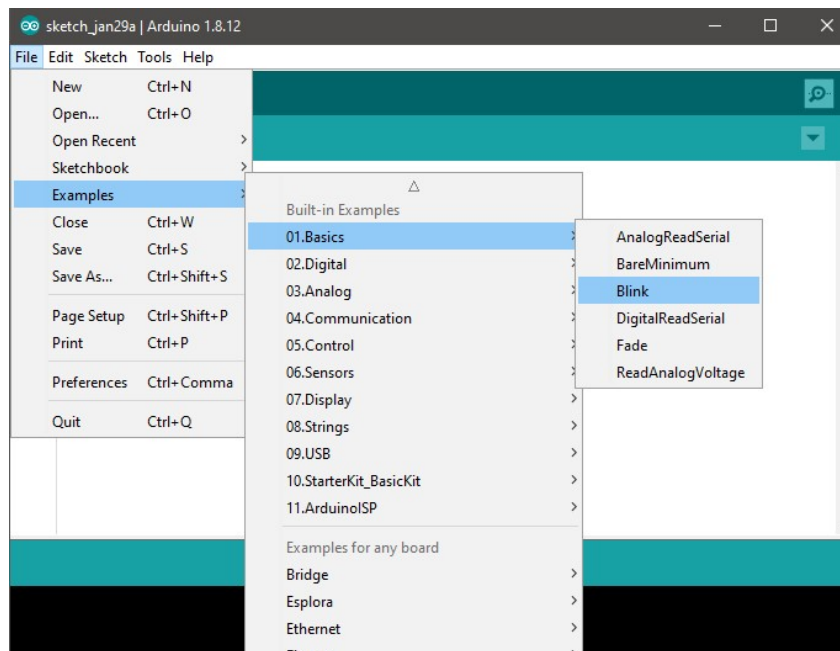


Figure 11: Arduino IDE Example Code

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`.



9 Wiring

If you just want to see the wiring schematic, see Figure 12. 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.

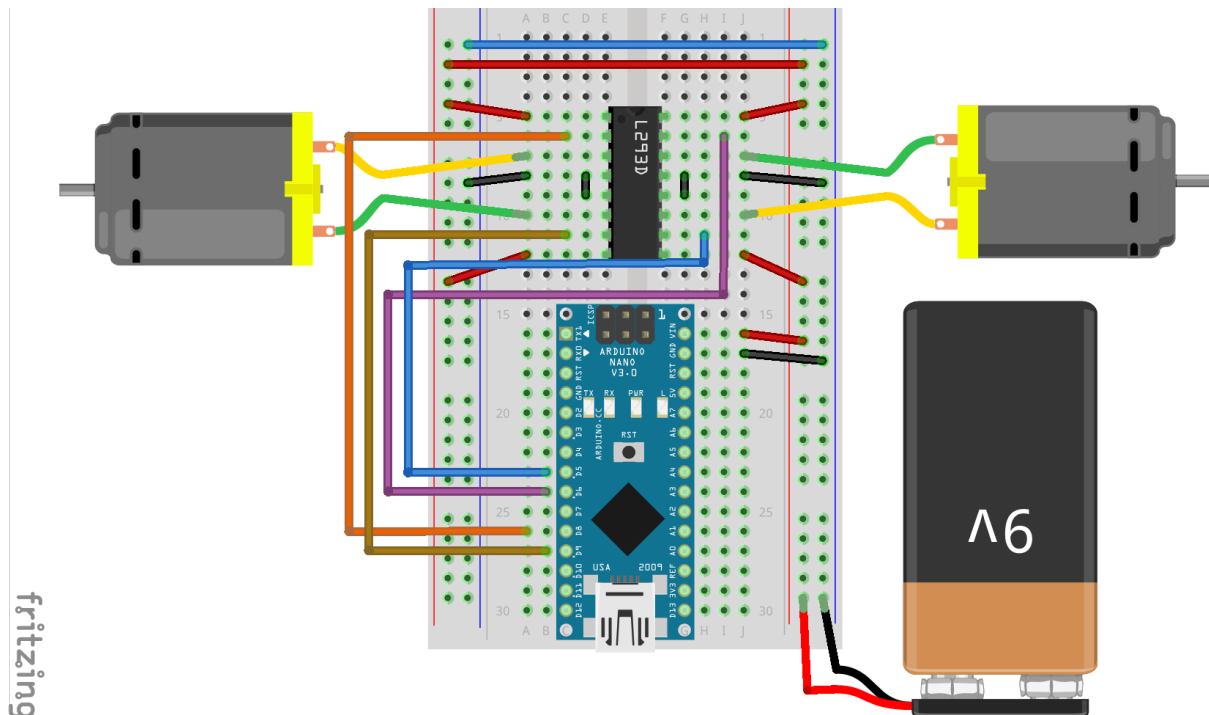


Figure 12: Wiring Schematic for H-Bridge

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.



If you don't have a battery, the circuit is mostly the same, though with some differences with how power is supplied to the circuit. Instead of powering the Arduino and circuit with the 9V battery you power the Arduino through USB and the circuit from the 5V output pin. See figure 13 below for a wiring diagram without the battery.

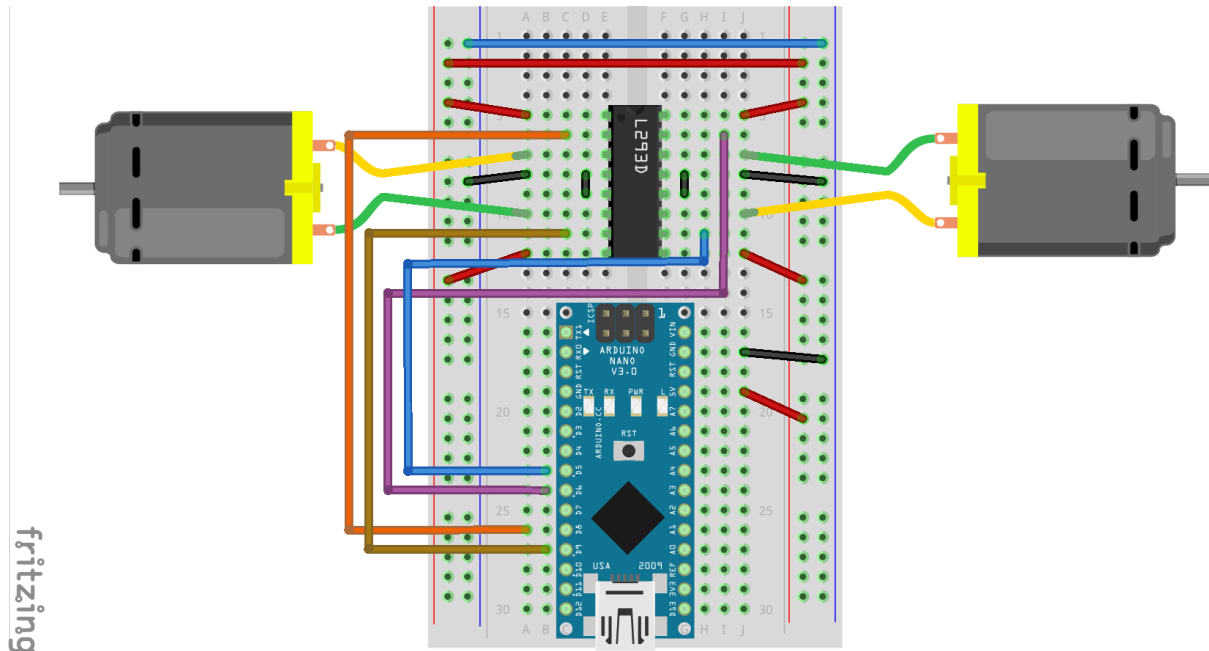


Figure 13: Wiring Schematic for H-Bridge



10 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, or “close them”, we want to set pins to high.

To make it easier on ourselves, let’s name these switches as constants. Let’s put these at the very top of our script so that every function we write later can access these constants.

```
1 //Motor 1
2 #define MOTOR_PIN_1 6    // Pin 15 of L293
3 #define MOTOR_PIN_2 5    // Pin 10 of L293
4 //Motor 2
5 #define MOTOR_PIN_3 9    // Pin 7 of L293
6 #define MOTOR_PIN_4 8    // Pin 2 of L293
```

The keyword `#define` means that the value cannot be changed later in the code. Annoyingly, when we define constants like this, we have to omit the equals symbol and semi-colon.¹ The value of the constants can be changed to any digital pin number on the Arduino, though make sure that the value 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(MOTOR_PIN_1, OUTPUT);
4     pinMode(MOTOR_PIN_2, OUTPUT);
5     pinMode(MOTOR_PIN_3, OUTPUT);
6     pinMode(MOTOR_PIN_4, 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(MOTOR_PIN_1, HIGH);
5     digitalWrite(MOTOR_PIN_2, LOW);
6     digitalWrite(MOTOR_PIN_3, HIGH);
7     digitalWrite(MOTOR_PIN_4, LOW);
8     delay(2000); // wait for 2 seconds
9
10    // Turn motors off
11    digitalWrite(MOTOR_PIN_1, LOW);
12    digitalWrite(MOTOR_PIN_3, LOW);
13 }
```

¹Research the C++ preprocessor in your spare time to find out why



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() {
2     // motor 1 - left
3     digitalWrite(MOTOR_PIN_1, HIGH);
4     digitalWrite(MOTOR_PIN_2, LOW);
5     // motor 2 - right
6     digitalWrite(MOTOR_PIN_3, HIGH);
7     digitalWrite(MOTOR_PIN_4, LOW);
8 }
9
10 void driveBackwards() {
11     // motor 1 - left
12     digitalWrite(MOTOR_PIN_1, LOW);
13     digitalWrite(MOTOR_PIN_2, HIGH);
14     // motor 2 - right
15     digitalWrite(MOTOR_PIN_3, LOW);
16     digitalWrite(MOTOR_PIN_4, HIGH);
17 }
18
19 void turnLeft() {
20     // turn off right motor, and drive left motor forwards
21     // motor 1 - left
22     digitalWrite(MOTOR_PIN_1, HIGH);
23     digitalWrite(MOTOR_PIN_2, LOW);
24     // motor 2 - right
25     digitalWrite(MOTOR_PIN_3, LOW);
26     digitalWrite(MOTOR_PIN_4, LOW);
27 }
28
29 void turnRight() {
30     // turn off left motor, and drive right motor forwards
31     // motor 1 - left
32     digitalWrite(MOTOR_PIN_1, LOW);
33     digitalWrite(MOTOR_PIN_2, LOW);
34     // motor 2 - right
35     digitalWrite(MOTOR_PIN_3, HIGH);
36     digitalWrite(MOTOR_PIN_4, LOW);
37 }
38
39 void stop() {
40     // motor 1 - left
```



```
41     digitalWrite(MOTOR_PIN_1, LOW);
42     digitalWrite(MOTOR_PIN_2, LOW);
43     // motor 2 - right
44     digitalWrite(MOTOR_PIN_3, LOW);
45     digitalWrite(MOTOR_PIN_4, LOW);
46 }
```



11 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.

- Allow the robot to move at different speeds
 - (HINT: research the function `analogWrite()`)
- Make a line following robot
 - (HINT: requires a light sensor)
- Make a robot that bounces of walls or other objects in its way
 - (HINT: 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. Make sure to research existing libraries to see how to use various sensors and actuators.