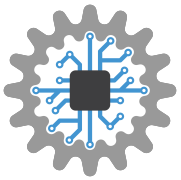
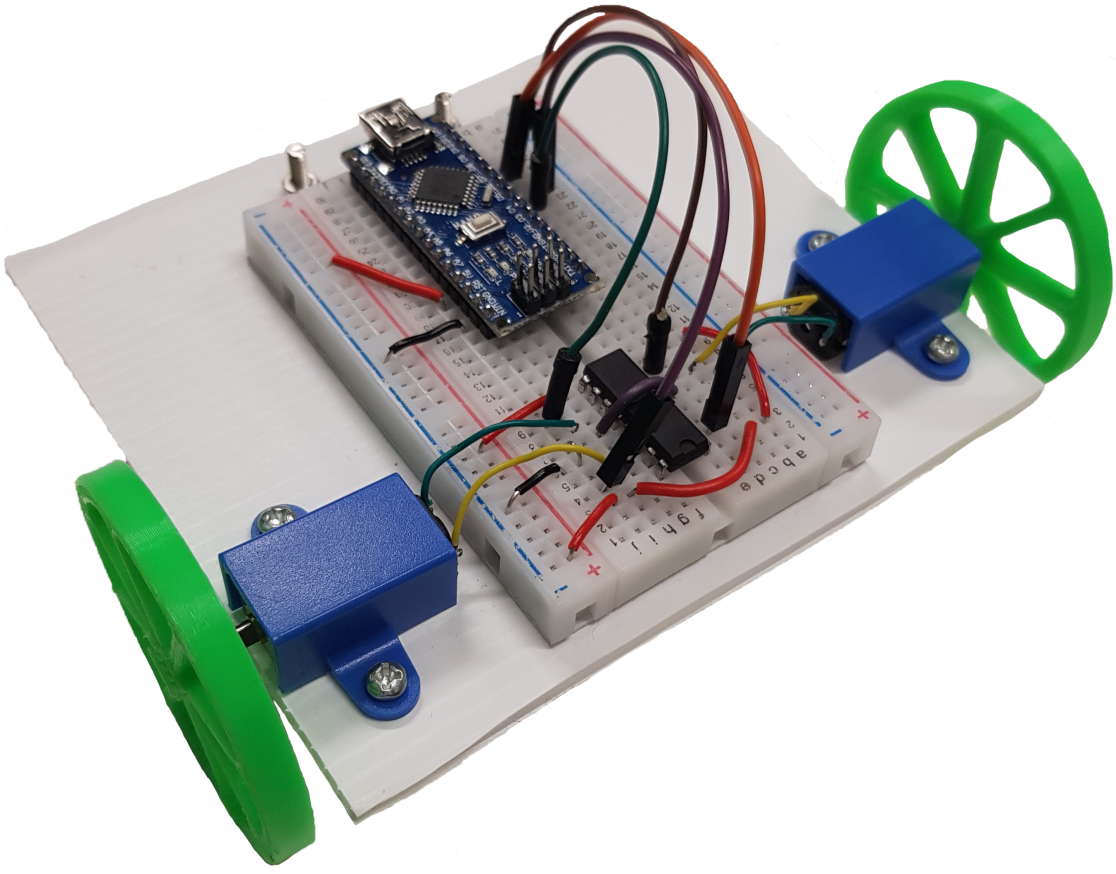


Curtin Robotics Club

TinyBot



ilke@curtinrobotics.org

July 24, 2022

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.

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

3 Components

The table on the following page shows the components used on TinyBot; this table is only important if you wish to purchase the components yourself in order to keep TinyBot upon completion. When purchasing components, ask about a student discount if you're a student. It's a great way to save money when purchasing components.

Additional sensors can be bought and integrated with TinyBot, however that is not covered in this project guide. If you are intending to integrate sensors with TinyBot, it is recommended to use an Arduino Uno or Mega, this change can be made with only minor adjustments to the design in the construction phase.

Component	Num.	Price	Sources
Arduino Nano	1	$5-$80	Arduino's are discussed in Section 4. 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 Arduino.
Breadboard	1	$8	Altronics sells breadboards at about $7 each, which is about the same price as you'll find on eBay. Ali Express has them much cheaper but shipping can take quite a while.

Geared DC Motor	2	$5 - $20	The motors used in this guide are N20 motors (small and strong), however any **geared** DC motor can be use (such as these). There are 3 ways to source N20 motors: • From AliExpress, eBay, or other online stores • From Altronics - though they are significantly more expensive sourced this way
L293D 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. The STL files are provided in this repository if you wish to print them. It is also possible to buy suitable wheels from Altronics and Jaycar
Caster Wheel	1	$0	The caster wheel consists of 2 parts, a marble and it's 3D printed casing. Again, the STL is available in this repository. Marbles will need to be sourced yourself.
Core Flute / Cardboard	-	-	The base of TinyBot can be built out of any material you have on hard given it is stiff enough to support the components. The club uses core flute, but cardboard, wood, hard plastics, etc. can be used instead.
Wires	-	-	Wires can be purchased from Altronics - options include assorted length solid core and standard jumper wires.
9V Battery	1	-	Can be purchased from Coles, Altronics, Bunnings, etc.
9V Battery Snap Clip	1	-	Can be purchased from Altronics

!

There are two different kinds of wires, solid core and multi core (also called stranded wire).

Multicore wire is more common, and is better suited for soldering as the strands in the two wires can mesh together and the solder wraps around all the strands forming a better bond between the wires. Multicore wire is annoying to use with breadboards as individual strands can be bent and not plug in correctly. This can often be fixed by "tinning" the wires, where the stripped ends of the wires are coated in a thin layer of solder to hold all the strands together.

Solid core wire is made of a single thicker strand of wire, making it better for bread boarding as there are no thin strands to be pushed out of shape. However, solid core wire is terrible for soldering as good connections are hard to ensure.

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.

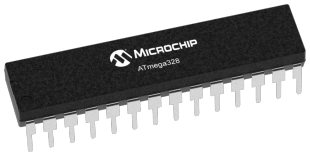


Figure 1: A Microchip

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.

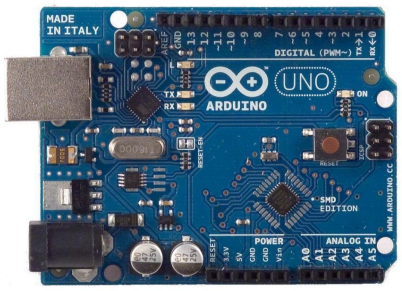


Figure 2: An Arduino Uno

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 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 components, such as LEDs, ICs, or sensors.

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

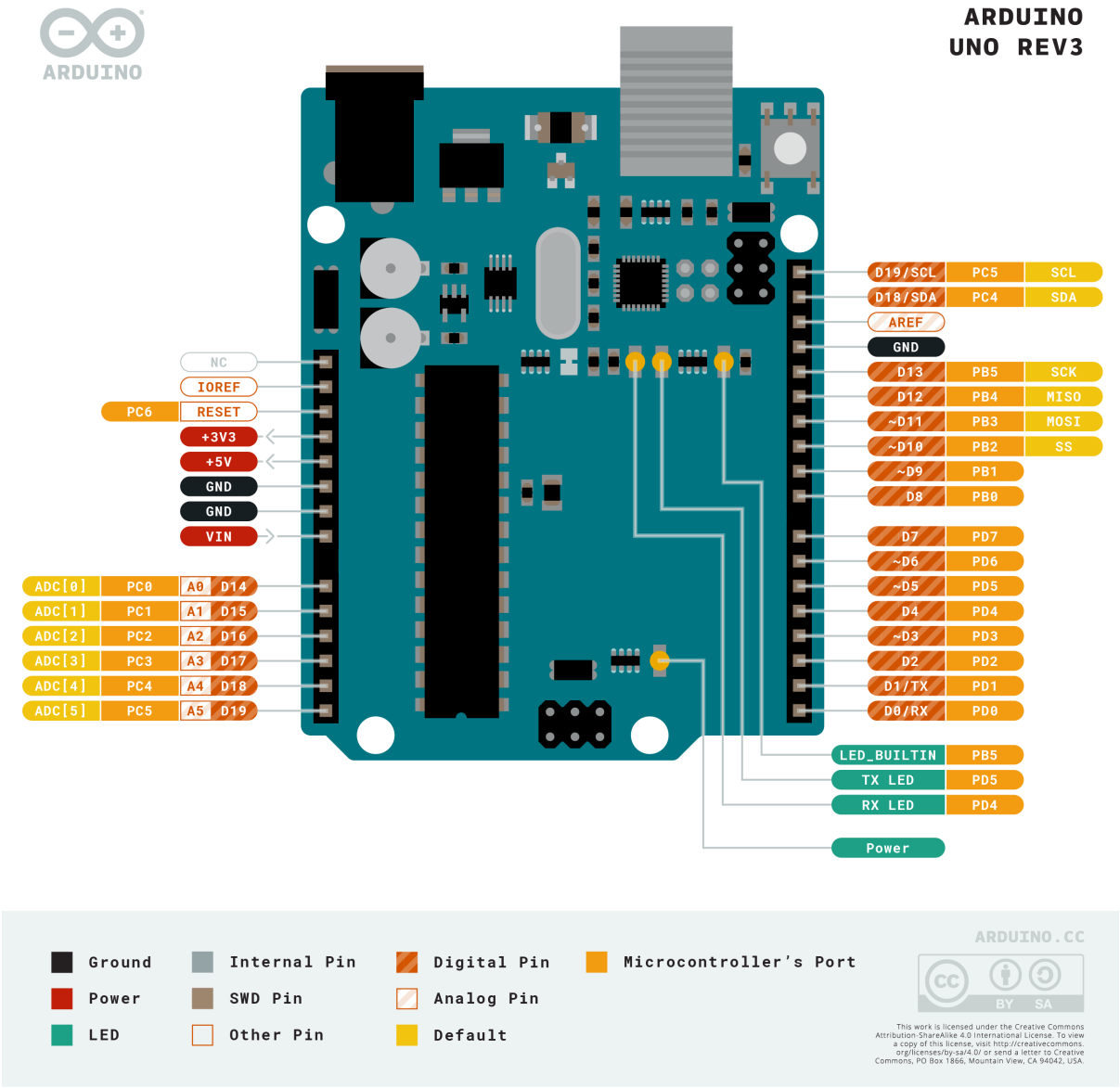


Figure 3: Pinout of Arduino Uno

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

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. This link has a good indepth explanation.

Figure 4: TODO: Picture of a motor with coils and current

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 current of a motor is the maximum current drawn when a motor stalls, in other words, applying its maximum torque - known as stall torque.

Similarly, free current is the current drawn when the motor is rotating freely, under no load - free current can also be thought of as the amount of current that has to be applied for a motor to overcome its internal friction. The stall current is usually much larger than the free current, and keeping a motor in a stalled condition can lead to overheating and damage to the motor due to the high current through it.

Each motor has a certain amount of torque it can provide, which depends greatly on the size and current draw of the motor. The speed of a motor also depends on the current going through the motor. Motor torque and speed are proportional, with torque increasing as speed decreases. This is because the mechanical power of a motor depends on torque $\times$ speed ($P = \tau\omega$), the power of a motor is constant meaning that when τ increases, ω must decrease.

It is possible to control the speed of a motor by controlling the input voltage, however this often results in high speeds and low torques. When we want to use motors to drive a robot around we're going to need relatively high torque from our motors as we need to overcome friction between the robot's wheels and the ground. To increase the torque of a motor, we want to use a gearbox.
Gearboxes chain together different sized gears (gear size is commonly measured by the number of teeth on the gear which leads to a larger diameter), with the ratio of teeth size determining the output speed/torque. The ratio can be calculated as

$$R = \frac{N_1}{N_1} = \frac{D_1}{D_1} = \frac{\omega_2}{\omega_1}$$

In the above equation, ω is the angular velocity, D is the diameter of the gear, and N is the number of teeth. The ratio can be expressed in plain English as a higher number of teeth meaning a smaller angular velocity and a larger diameter gear. This makes sense intuitively, in Figure 5 gear A has 1/3 the number of teeth of gear B. This means that gear A will fully rotate 3 times in the time it takes gear B to rotate once. The gears must be rotating at the same angular velocty as they are mechanically linked via the gear teeth, and to maintain power (as no energy can be destroyed) gear B must have more torque than gear A.

We can take advantage of this to increase the output torque of a gearbox to whatever torque we need by simply changing the gear ratios within a gearbox. It is possible to buy both geared and non-geared motors. It is usually easy to spot when a motor has a gearbox attached TODO: images of geared and non geared motors. For this project, make sure you have a geared motor - non-geared motors will not be able to supply enough torque for your robot to move.

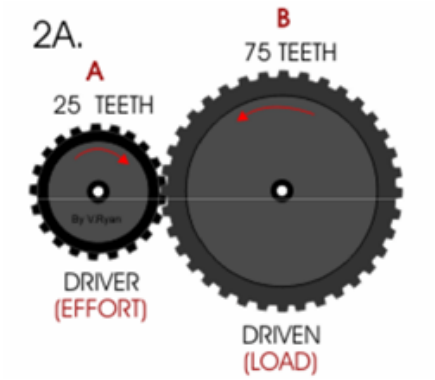


Figure 5: Driver and Driven Gear

6 Motor Controller

The motors used in this guide, the N20 motors, have a stall current of 1.6A (see section 5 for what stall current means). The digital pins on an Arduino Nano supply at most 40mA. This is not enough to power the motors.

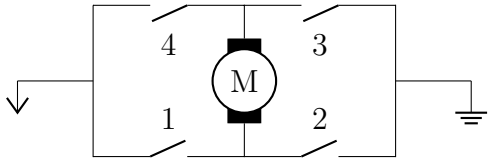
To get around this, the Arduino instead interfaces with a **motor controller**. Motor controllers have a separate power supply that can supply enough current to drive the motor. Motor controllers also have digital inputs that allow control of the motor.

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

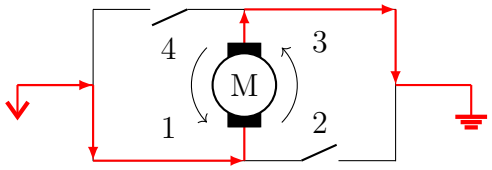
The phrase motor controller is often used as a generic term for any device, circuit, or IC which controls a motor. However, motor controllers are a circuit that consists of a motor driver and some digital harness that acts as an interface to the driver. Motor controllers can be dropped into a circuit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

6.1 Motor Driver

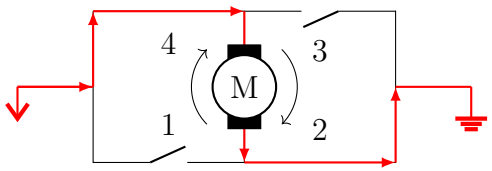
A basic motor-**driver** is a H-bridge. The simplest H-bridge is shown in the below schematics, as well as an explanation of how using a H-bridge allows control over the motors direction.



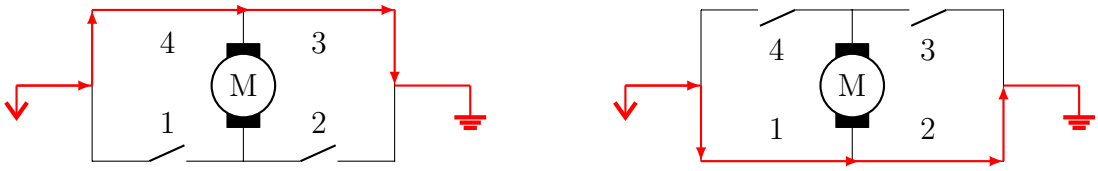
When switches 1 and 3 are closed, the current will flow through the motor making it turn anticlockwise.



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



If pins 4 & 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.

6.2 Motor Controller

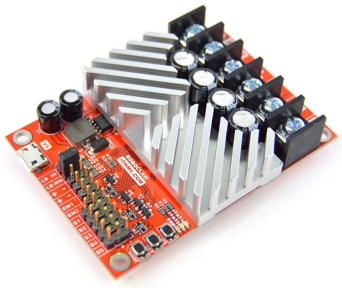


Figure 6: RoboClaw Motor Controller

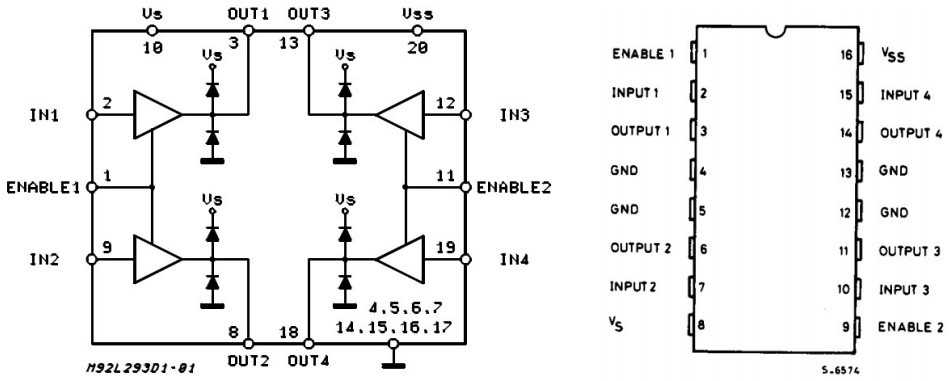
A motor controller has a lot more features than a motor driver. See, for example, the RoboClaw (see Figure 6) which has in built features such as PID tuning, data logging, diagnostic LEDs, and serial control.

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

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 can be found online (linked here). The L293D is a dual H-bridge IC, meaning that it can control two motors at once. Technically, the L293D is a quadruple half H-Bridge driver - which allows the same functionality as a dual H-bridge, but makes the circuit more robust and unlikely to be shorted by opening the wrong pins.



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

Figure 7: Diagrams from the L293D datasheet

Something important to note from the pinout in Figure 7b is the pins labelled enable 1 and enable 2. These pins need to be set to a positive voltage for the motor to be enabled.

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 input signal from the enable line.
The bold black triangle and line combinations represent components called Transistors.

The pinouts in Figure 7b are really important when wiring up the H-Bridge, as discussed later in Section 9.

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 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/

8 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.

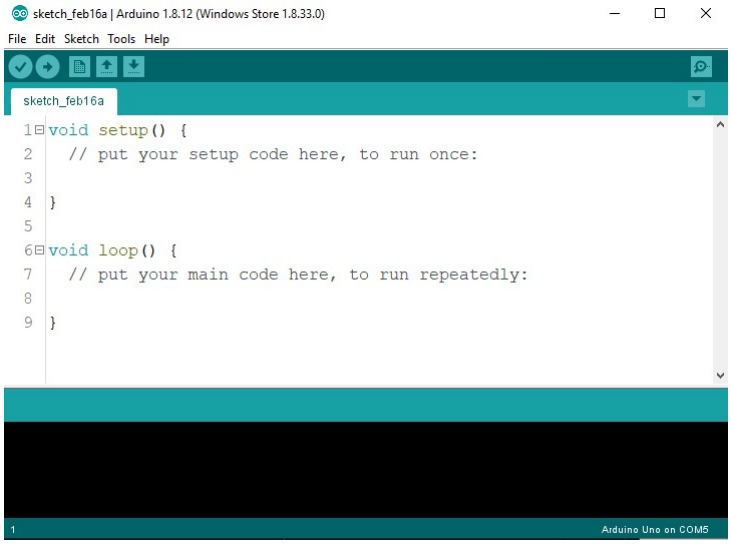


Figure 8: The Arduino IDE

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 (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.

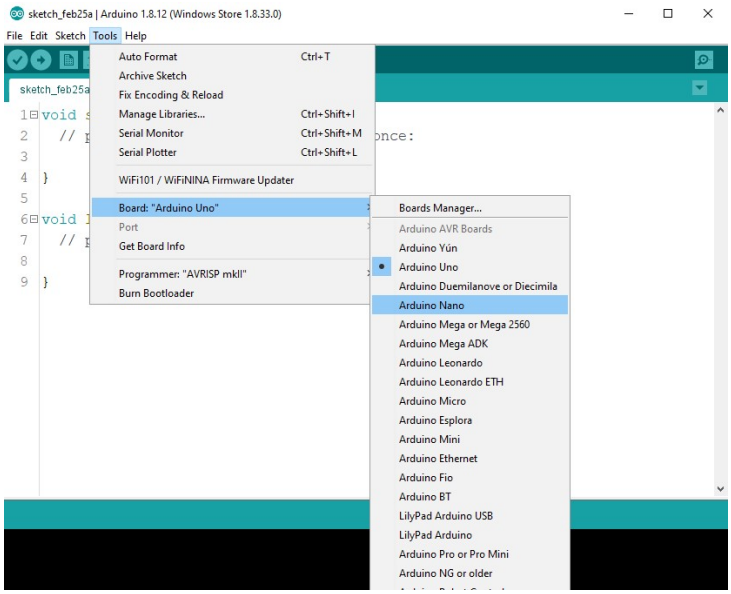


Figure 9: Selecting Board

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.

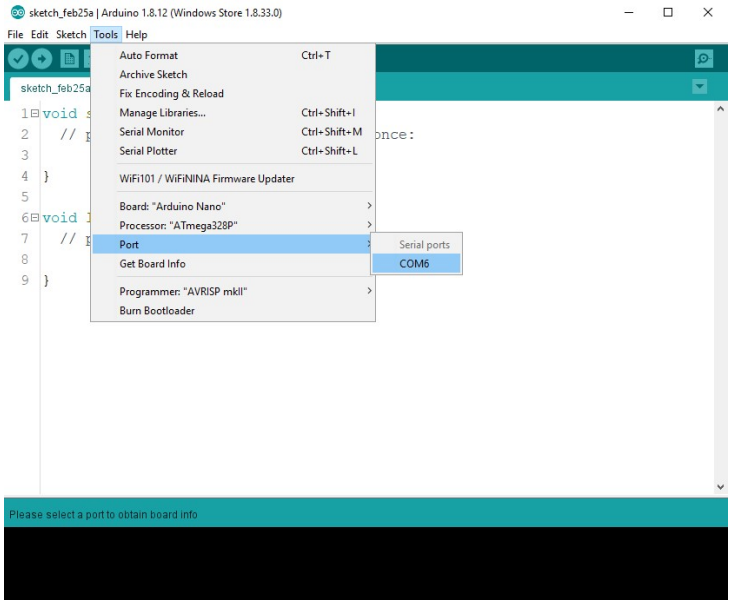


Figure 10: Selecting Port

!

If you cannot select the port because it is greyed out try plugging the Arduino into a different USB port. If this doesn't work, try unplugging and plugging in the Arduino end of the USB cable.

If you still cannot select a port, it may be an issue with the cable or with the Arduino. Use a different cable, or a different Arduino to test if they are the issue.

If you still cannot select the port, even after trying different USB ports, cables, and Arduinos; the issue is likely with your computer, reinstalling the Arduino IDE may help.

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,
```

```
  // 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.

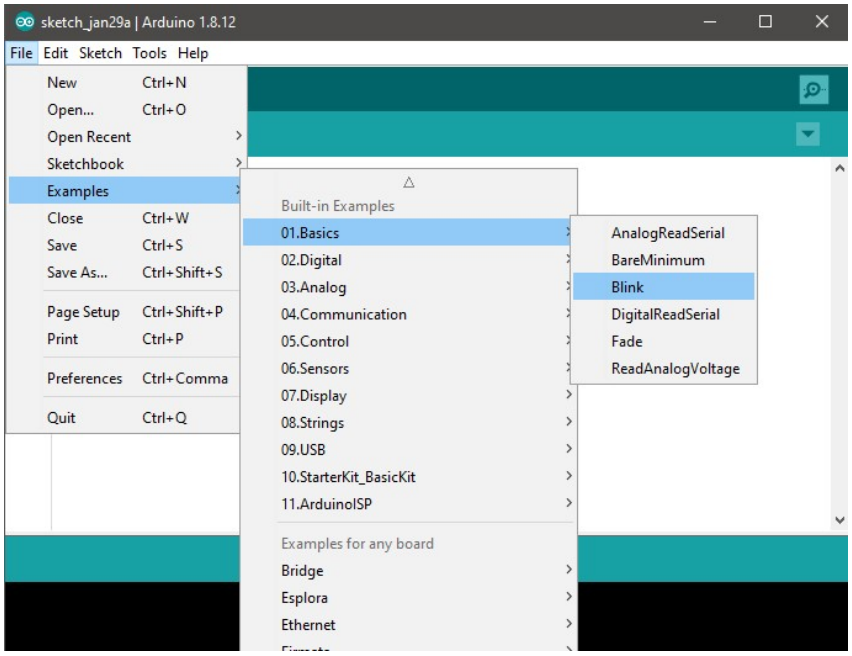


Figure 11: Arduino IDE Example Code

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

```
void setup() {
  // initialize digital pin LED_BUILTIN as an output.
  pinMode(LED_BUILTIN, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {
  // turn the LED on (HIGH is the voltage level)
  digitalWrite(LED_BUILTIN, HIGH);

  delay(1000);       // wait for a second

  // turn the LED off by making the voltage LOW
  digitalWrite(LED_BUILTIN, LOW);

  delay(1000);       // wait for a second
}
```

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`.

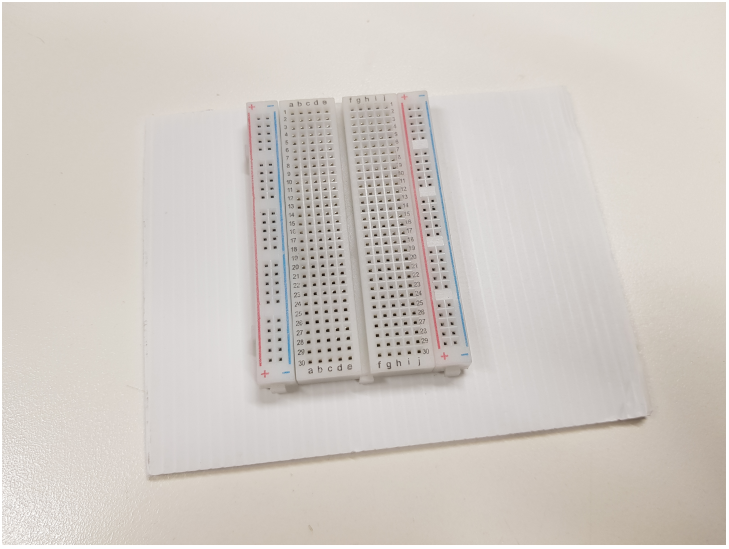
`digitalWrite()` is used to set digital pins `HIGH` and `LOW`. High can be thought of as turning on something and low is turning off something just like with the LED in the above example.
Sometimes this will change, it depends on whether the component is active high (normally low, make it high to turn it on) or active low (normally high, make it low to turn it on).

9 Construction

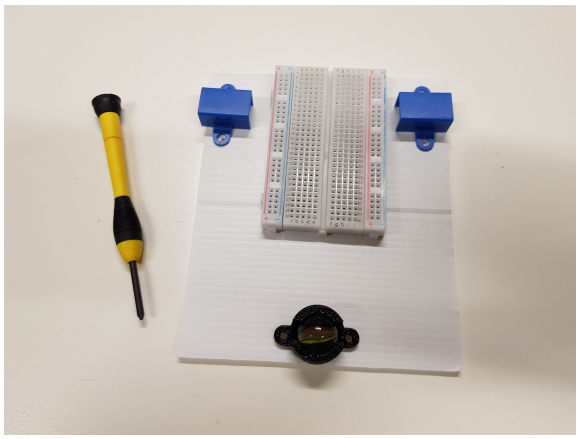
The first step is to cut out your base material. This can be any material stiff enough to support the Arduino, breadboard, and motors. The club provides corflute, though other materials such as wood or plastic from milk bottles can be used if desired. The base can be any shape, though ensure it is big enough to fit the breadboard, the Arduino you are using if not using a Nano, and a battery. It is useful to lay out the components you are using on the base material before cutting.

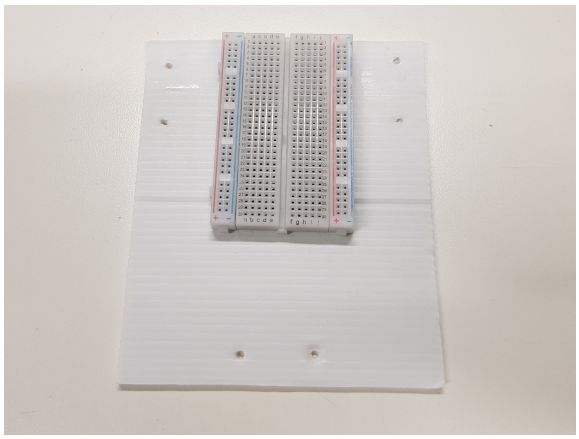
!	Get creative with the shape of your robot base! Make it a diamond, some kind of squiggle, any shape that fits the components will do!

Once you have cut out the base, peel the backing paper from the breadboard and stick the breadboard to the base.

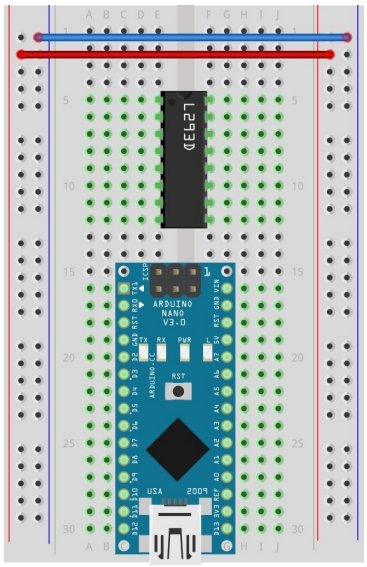


Then place the motor covers and caster wheel on the base and mark out where holes need to be made. The club has plenty of motor covers, though cable ties can be used as an alternative to secure the motors.
If using a material like corflute the holes can be marked/made with a screwdriver. Holes will have to be marked and made differently if using other materials.



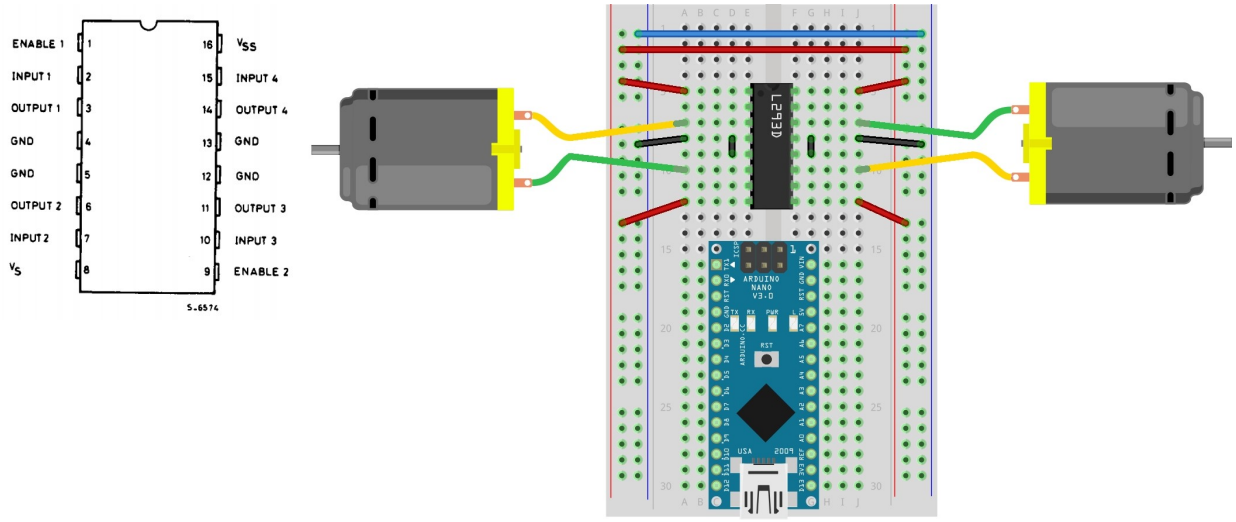


It is easier to wire the circuit after the breadboard is stuck to the base as you don't have to worry about pulling any wires out when sticking on the breadboard. The steps to wire up TinyBot are shown below. Before starting to wire up the circuit, double check your H-Bridge is in fact a H-Bridge; a lot of IC chips look really similar and are difficult to tell apart. On the back of your chip there should be some serial numbers, make sure that one of these numbers is L293D - there are lots of variations of this H-Bridge so numbers like L293DE are also valid H-Bridges.



If you are using an Arduino Nano, push the Nano into the breadboard so that the header pins (the metal "legs") go into the breadboard. Make sure the USB connector is facing out so you can plug the USB cable in later. Also push the H-Bridge in the breadboard, paying attention to where the notch on the IC is. Connect the power and ground rails along each side with a cable. Connecting the rails in this manner allows power and ground to be plugged into either side and still get power. You nearly always want to do this with the ground rail, but sometimes it's useful to have different voltages in the rails on either side and in this case you don't want to connect the power rails.

Next we want to wire up the H-Bridge. This is where the pinout diagram from earlier is very useful.



First, connect all the pins labelled ground. Using the pinout, we can see that the four central pins are all GND and so are connected in the wiring diagram. Next, we wire all the enable and V_S pins to the power rail. Then we can connect the motor leads to the output pins.

Output pins 1 and 2 are for one motor, and output pins 3 and 4 are for the second motor. Which lead gets plugged into which output pin doesn't matter too much, if you get the pins wrong, nothing will break - the motor will just spin backwards. If you need to swap the motor direction later, just swap the green and yellow leads around.

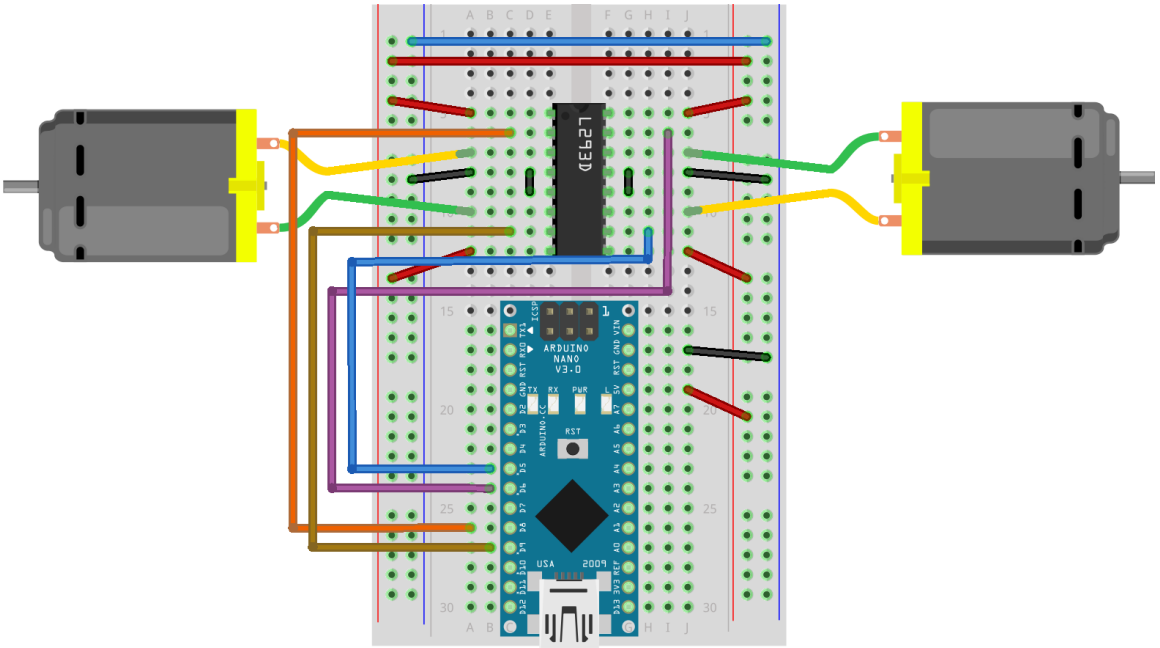


Figure 12: Wiring Schematic for H-Bridge

After wiring up the H-Bridge, we can connect it to the Arduino. We want to connect all

the input pins to the Arduino - these are the pins we'll be setting high and low to control the motor direction.
The pins used in the wiring diagram are:

- Input 4 → `D6`
- Input 3 → `D5`
- Input 2 → `D9`
- Input 1 → `D8`

> **!** It doesn't matter which pins on the Arduino are connected to the H-Bridge inputs, but the pin numbers must match up in code and wiring.

It's also important to connect power and ground the Arduino. Currently, we're assuming that the Arduino is being powered via USB; and then is supplying power from the `5V` pin to the breadboard. This is good for testing, but not very useful when we want to put TinyBot on the floor and let it drive around.

So, we want to power TinyBot with a battery so it doesn't have to be connected via USB cable. Luckily the circuit with a battery is very similar. The ground wire connected the Arduino to the ground rail can stay where it is. The 5V wire needs to be unplugged, then `Vin` (meaning voltage in) needs to be connected to the power rail. Also plug the battery into the power rails - with the black wire from the battery going to the ground rail and the red going to the power rail. This circuit is shown in Figure 13.

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

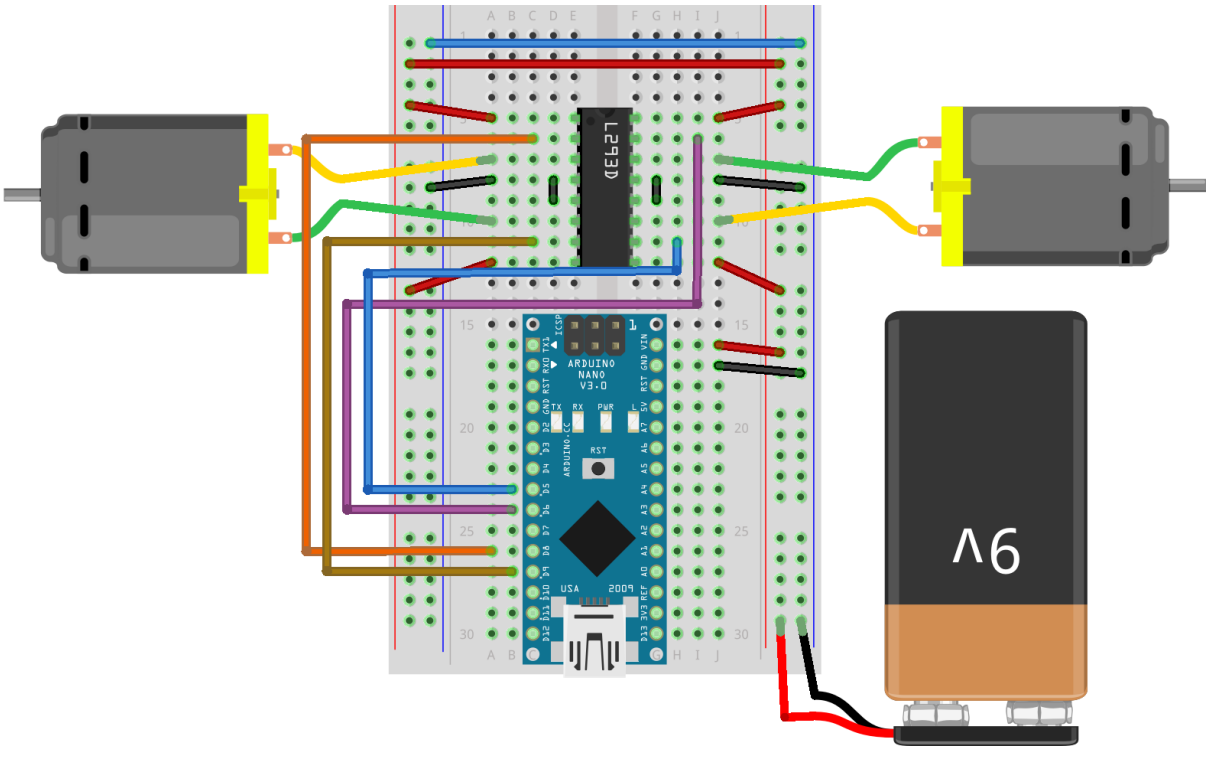
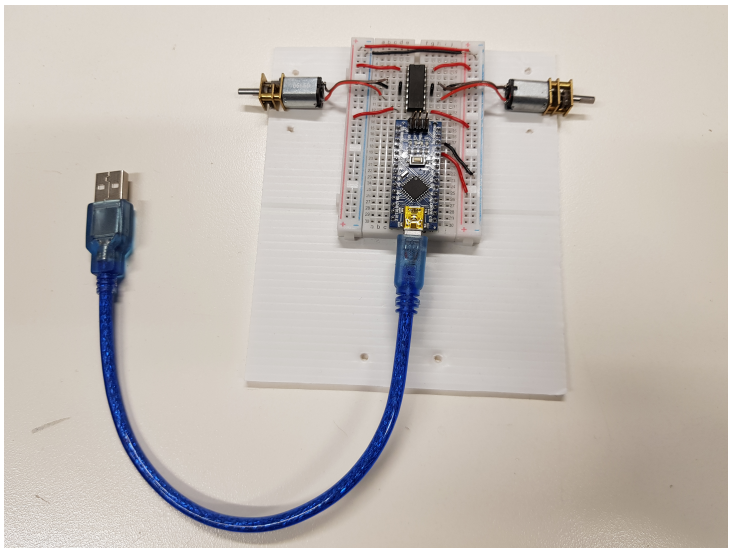
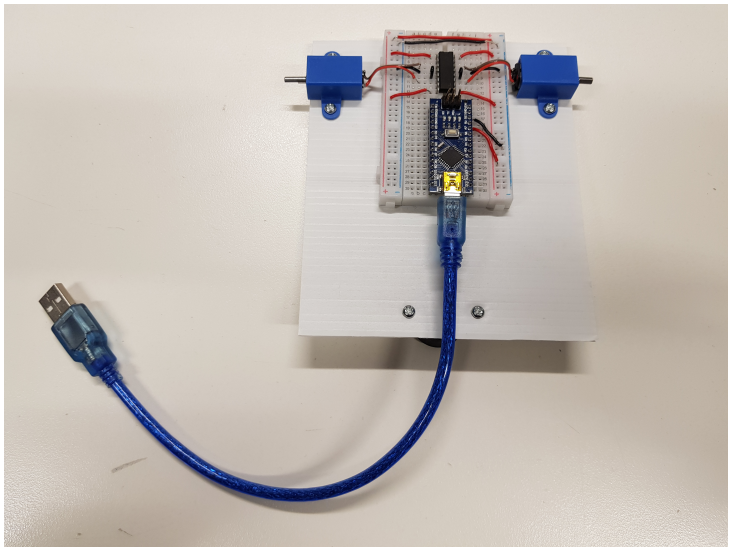


Figure 13: Wiring Schematic for H-Bridge

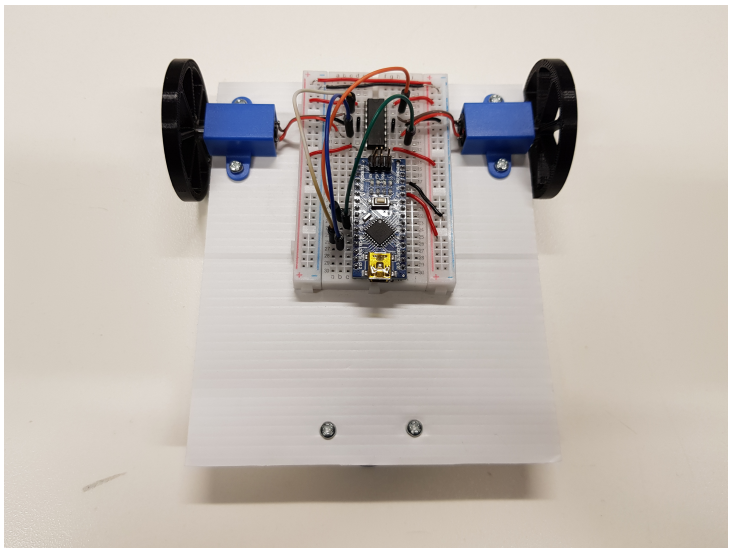
The circuit in real life should look something like this:



Place the motor covers over the motors and attach to the base using the provided screws. The caster wheel is also attached using screws.



Your TinyBot is now fully assembled. (The TinyBot pictured is missing a battery, make sure your TinyBot has a battery)



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, 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 `PIN1` and `PIN3`.

```
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 }
```

11 Challenges

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

- Turn the robot around on the spot
  - think about what way each motor should rotate to turn on the spot
- Drive in a square
  - a square is just a line and a 90 degree turn, repeated 4 times

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 are quite involved.

Bonus Challenge: decorate TinyBot! Glitter, paint, coloured paper, go crazy! The best looking TinyBots get a social media post all to themselves (plus their maker of course)!