# The Engineer's Guide to SumoBots
# Don't Panic!

The SumoBot Execcutive 2014

March 2, 2015

## Contents

## 1 Introduction

Welcome to the McMaster Sumobot competition! This handbook will give you a basic understanding of the components and systems needed to make your first Sumobot.

## 2 What is a Robot?

Autonomous robots are a lot like animals. Both observe the environment, collecting information before processing an appropriate behavioral response. So if you want to build a functional Sumobot, you need three things: a sensory system to collect data, a processing system to convert data into behavior, and a motor output system to perform those behaviors.

We all know how sensory systems work in animals. The eyes, ears, nose, tongue, and skin work together to collect information, just like how a robot's sensors gather data from the environment. But that's all it is: raw data. Your eyes can't tell you what you're looking at; they just absorb light. Your tongue can't tell you what you're eating; it just absorbs chemicals. If you want to know what you're eating — if you want to understand and interpret the information provided to you by your tongue — you need a brain.

A robot's brain is called a microcontroller or microprocessor. A microcontroller receives data from the robot's sensors and interprets them based on a man-made system of logic. The man (or woman), in this case, is *you.* You decide how your microcontroller converts sensory input to motor output. You wire the robot's sensors so the data goes where it needs to. *You* write the code, *you* upload it to the microcontroller, and *you* watch your robot do the exact opposite of what it should, because no one gets it right on their first try. But when you do get it right — when your little robot finally figures out how to push other robots out of the ring — you'll feel as proud as any parent, and deservedly so.

Most amateur robot enthusiasts use the Arduino brand of microcontroller. Specifically, you will use the Arduino Uno. There are many different kinds of Arduino on the market, each with its own purpose, but the Arduino Uno is best for simple robots like Sumobots, which only use a few sensors and motors. Check out Appendix A for a run down of the Arduino.

Your Sumobot will probably be limited to DC motors, which will help it move around the ring. But depending on your design, you might end up using other, more complex devices, like servomotors.

That should cover the basics. If you have any other questions about designing robots, choosing parts, or following Sumobot guidelines, just keep reading, and you'll find what you're looking for. Good luck!

## 3 What is a Sumobot?

A Sumobot is exactly what it sounds like: a sumo-wrestling robot. To be more specific, a Sumobot is an autonomous robot programmed to push other Sumobots out of a ring without getting pushed out itself.

The Sumobot ring is black with a white border. Our competition follows the international regulations for Sumobot size and shape, as outlined in the McMaster Sumobot Rulebook.

## 4 Designing Your Robot

Your design will be divided into three parts

- Mechanical Design
- Electrical Design
- Software Design

All three of these designs are co-dependent. You must have a sound mechanical and electrical system to control it with software. But to begin with, you need to start off with how your robot will look and behave so we'll start there.

## 5 Mechanical Design

You should already be familiar with the mechanical design process, but we'll give you some pointers just in case. Over the course of this project, you'll probably come up with several design ideas for your Sumobot. These designs might be very different from one-another; they might use different parts, differently shaped chassis, etc. Chassis and robot parts are both expensive, and they can be difficult to swap out, so make sure your design is as complete as possible before you start building.

For a complete description of Sumobot size regulations, please refer to the McMaster Sumobot rulebook.

To reiterate, the design stage is the most critical part of your project. If you're not absolutely sure about your current design, then ask! That's why your mentors are there.

### 5.1 Rough Sketches

One of the first tings your group should do is to sit down and sketch possible ideas. You don't need any special sort of software to do brainstorming. Just use lots of paper and quick sketches to describe what your robot will do. Only when you all decide on a design will you move forward to the next step.

### 5.2 CAD implementation

When you've finished your design, you will make a CAD model of the robot's chassis. This will formalize and confirm the design that your team thought of by forcing you to consider where everything goes, and it will allow you to print your chassis should you choose to do that. If you are unfamiliar with CAD or 3D printing, just talk to your mentor, and they will see to it that you learn everything you need to know.

When doing the CAD you must consider where everything goes. Here's a quick checklist to ensure that you've considered everything. This checklist is also available in Appendix B.

### 5.3 The CAD Checklist

- Ensure design is within competition constraints
- Ensure that all sensors have been considered (IR and Distance)
  - Ensure that all sensors can be wired
  - Ensure that sensors are able to see the environment
- Ensure that Arduinos and supporting wires are in a secure place
- Ensure that Batteries are at an easily accessible place
  - Ensure that all components (Sensors, Actuators and Arduinos) can be powered from the battery
- Ensure Motors are at the correct place
  - Ensure that motors can touch the ground and move the robot

## 6 Electrical Design

Once you figure out how your robot looks, you must determine how your robot will feel and act on the environment. In addition to that, you must consider how everything will be powered. That's the electrical design part.

## 6.1 Sensors

As stated earlier, your robot's sensors are its eyes and ears. When choosing sensors for your robot, keep these questions in mind:

1. How will your SumoBot detect the boundaries of the ring?
2. How will your SumoBot detect other robots?

Different sensors detect different things, and they detect these things in different ways. Some sensors can only tell you whether stimuli are "DETECTED "or "NOT DETECTED," while others provide more detailed information, such as your robot's distance from the stimulus. Every sensor has its pros and cons. You may have to use two or more sensors, each one compensating for another's weakness. It's a difficult decision, so to make things easier, we created this chart:

Table 1: Sensor Properties

| SENSOR | MECHANISM OF DETECTION | ADVANTAGES | DISADVANTAGES |
|---|---|---|---|
| Infrared(IR) | Emits and Detects IR Light | Little interference from noise<br><br>Low power requirement<br><br>Easy to use<br><br>Relatively inexpensive<br><br>Unaffected by climate | Target must be in direct line of sight<br><br>Short range only<br><br>Signal can be absorbed by dark surfaces<br><br>Relatively low sensitivity for distance measurements<br><br>Large dead zone |
| Ultrasonic | Emits and detects sound waves greater than 18 kHz | Short and long range<br><br>Very sensitive distance measurements<br><br>Easy to use<br><br>Little-to-no dead zone<br><br>Easy to use | Must consider Doppler effect if your opponent's robot is fast<br><br>Irregular or coarse surfaces deflect the signal<br><br>Relatively expensive<br><br>Lots of High Frequency Noise<br><br>Affected by climate |
| Photo-resistor | Semiconductor resistance decreases in light and increases in the dark | Easy to use<br><br>Relatively inexpensive<br><br>Unaffected by climate | Does not provide distance<br><br>Low sensitivity for low levels of lightl<br><br>Time delay between signal detection and output |
| Optical sensor | IR and optical light Detects light in the IR and optical electromagnetic range, 300-1100 nm (varies between manufacturers) | Similar to IR | Similar to IR |
| *Dead zone refers to the sensor's "blind spot," where data becomes unreliable. | | | |

This table is helpful, but only if you need to make a quick, surface-level reference. For a complete breakdown of sensors and their specificities, you have to READ THE MANUFACTURER'S DATA SHEET. Only then will you know if a sensor does what you need it to. Appendix C has a great resource on hooking up sensors to the Arduino.

## 6.2 Actuators

### 6.2.1 DC Motors

DC motors will drive your robot's wheels, making them a vital component of the motor system. For a basic understanding of how DC motors work, watch this two-minute video

Every wheel and motor is different. They do different things, meet different goals, and satisfy different needs. These are the needs and goals you have to consider:

*Speed vs. Torque*- Do you want your robot to outspeed its opponent, or would you rather plow through the competition? It all comes down to the robot's wheel size and gear ratio. The bigger the wheels, the larger the gear ratio, the more torque you get out of the system, and the slower your robot becomes. Choose carefully.

*Two Wheels vs. Four Wheels* - A robot with two motors is lightweight, inexpensive, and energy-conscious, but it doesn't have a lot of power to fall back on. Robots with four motors are pricier, heavier, and more difficult to manage, but that extra power could make a difference. Again, choose carefully.

*A note about motors* ,no matter how similar a pair of motors may seem, they will always move at slightly different speeds, even when given the same throttle. You may have to adjust each motor's throttle individually to keep your robot from swerving.

### 6.2.2 Servomotors

A servomotor is a device that makes limited but accurate changes in the angular position of the robot's armatures. A single servomotor contains a DC motor and a sensor. The sensor detects the armature's angle, compares it to angle at which the armature should be, and sends feedback to the DC motor, which then corrects the armature's position. But again, servomotors are limited. They typically cannot move farther than 180 degrees, so keep this in mind when designing your robot. You can use servomotors to mount sensors, rotate parts of your robot, or perform other functions. Like almost everything else in this project, it's up to you.

## 6.3 Power Systems

Your robot will have two power systems: a motor power system and a logic power system. Crisscrossing the power systems will burn your microcontroller — it's not meant to handle large voltages and amperages — so keep these systems separate.

### 6.3.1 The Motor Power System

It's time to connect the Arduino to the motors! All you have to do is build an electrical circuit, and you're done, right? Oh, if only it were that simple…

This process puts your Arduino in danger for two reasons. First, a motor is a power generator, so it will probably try to send a current back into the Arduino. Second, motors create something called back electromotive force (or back EMF for short). Back EMF comes from the difference in relative motion between the motor's armature and magnetic field, and it manifests itself as a voltage that pushes against any incoming current. This includes the current from your Arduino, so you'll have to take some precautions.

These precautions come in the form of an H Bridge.

#### 6.3.2 H-Bridges

H Bridges take signals from the microcontroller and increase their power with amplifier circuits. They also prevent current from going back the way it came. To give you a better idea of how an H Bridge works, we stole this diagram from Wikipedia:

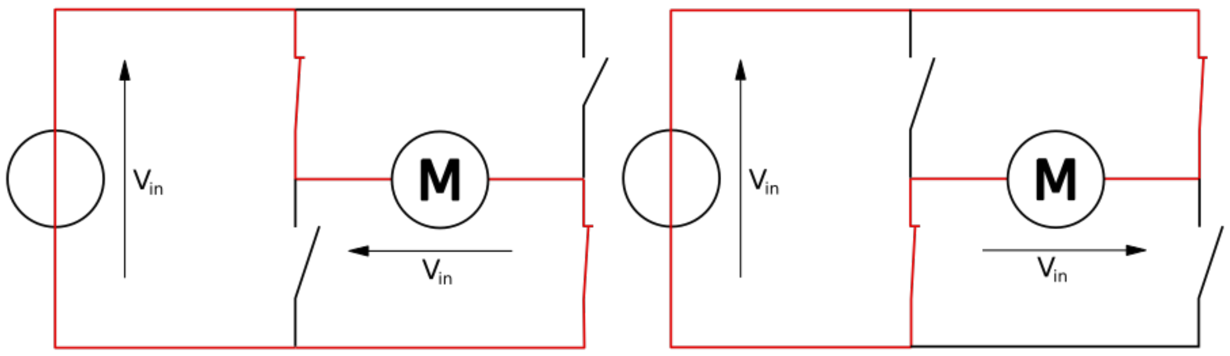

Figure 1: How H-Bridges work

The switches only allow current to move forwards; it cannot go back. S1 and S4 close at the same time to go forwards, while S2 and S3 close simultaneously to go backwards. Hooking up an H Bridge is straightforward enough. Just use the wiring diagram in Appendix C for reference and try not to short anything.

### 6.4 The Logic Power System

The Arduino Uno has a recommended input voltage of 7-12 V. Anything below 7 V, and you won't have enough voltage. Anything above 12 V, and you risk damaging the microcontroller. We recommend using a 9 V battery, but other batteries are fine as long as they stay in the 7-12 V limit.

## 7 Software Design

I know what you're thinking of right now, you want to boot up your computer and start typing some code. *Stop!* Before any code can be written down, you must determine how your robot will behave. One of the handiest tools that can be used to solve this is the Finite State Machine.

### 7.1 Finite State Machines

The finite state machine is a diagram of all of the possible actions a system can do. It outlines how one can get to a state, and where they can go from that state. It's really hard to put down in words, so we'll use an example instead. One of the simplest Finite State Machines is a blinking light. This light turns on for a second and turns off for a second.

As you can see, there are two bubbles; these represent the state of the light. These states are on and off. The arrows are where states can go if conditions are met.

For example, suppose the light is on. The system will remain in that state until one second is passed. When that condition is met, the system will switch its state to where the arrow points.

Your system will not be more complicated; it will have more states to consider. What will your robot do when it sees a white line, what about when it sees your opponent?

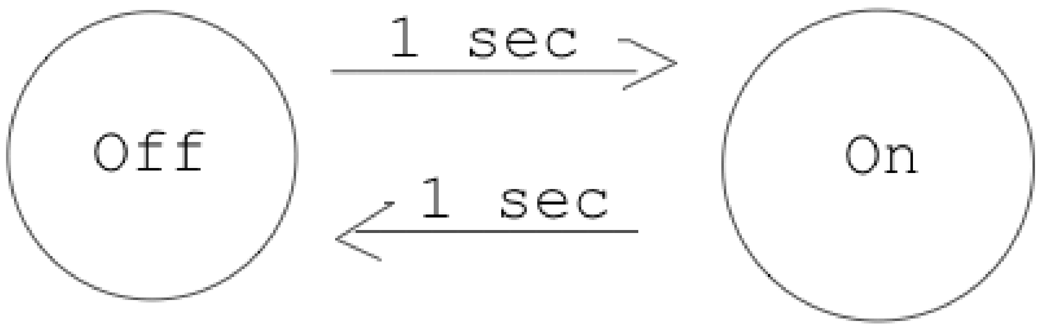

Figure 2: The Blinking Finite State Diagram

It's ok for this part to be difficult; it is one of the most challenging parts of the entire ordeal. But some time spent here will reap rewards in the future.

## 7.2 Coding

The Arduino has its own programming language, one similar to Java or C. If you have experience with these languages, Arduino programming shouldn't be too hard. If not, don't worry! The language is fairly straightforward, and there are tons of online resources to help you out. (This handbook is also pretty helpful, FYI.)

Let's go through the blinking light program discussed in the FSM section. We'll break it down bit by bit to show you how each function contributes to the program in its entirety.

### 7.2.1 Declare variables

The first thing you always do is declare your variables. If you want to use a pin, declare it here with an easy-to-read variable name. Leave a comment with // in case you forget which variables are which.

```
// On most Arduino boards, pin 13 is connected to the LED.
// Give it a name:
int led = 13;
int state = 0;
```

### 7.2.2 Set up your ports

Once you've declared all your variables, it's time to tell the pins what to do. Go to the setup area and declare whether the pins are analog, digital, input, or output.

```
void setup() {
  Serial.begin(9600);
  // Initialize the digital pin as an output for the LED.
  pinMode(led, OUTPUT);
}
```

### 7.2.3 The main program

The Arduino's main program runs in a constant loop, checking its conditions over and over until the machine turns off. It is in this loop that you will execute your code.

A microcontroller system has lots of sensors and actuators to keep track of in real time, and it needs to constantly check and see whether these sensors are working as they should. There are two common types of checking behaviour:

1. **Polling**: Polling is when the microcontroller periodically checks on a sensor or actuator, usually as part of an infinite loop. This would be like if you peeked out the window while waiting for a friend to come to your house.
2. **Interrupts** : When a sensor or actuator sends an alert to the microcontroller, it's called an interrupt. This would be like if your friend rang the doorbell when they arrived.

The Arduino is mainly a polling microcontroller. Sometimes, a poorly-coded polling program gets stuck on a single statement, repeating forever. These statements are always "while" statements, so don't use "while" statements unless you really know what you're doing.

```
// The loop routine runs over and over again:
void loop() {
        switch (state) {
                case 0:
                        // Turn the LED on. HIGH is the voltage level.
                        digitalWrite(led, HIGH);

                        // Wait for one second (1000 milliseconds.)
                        delay(1000);

                        state = 1;    // Switch the state
                        break;

                case 1:
                        // Turn the LED off. LOW is the voltage level.
                        digitalWrite(led, LOW);
                        delay(1000);
                        state = 0;  // Switch the state
                        break;
        }
}
```

### 7.2.4 Writing functions

You will probably have to duplicate certain pieces of code in the loop function, such as code that tells your robot to move forwards or turn left. You may want to write separate modules for these functions to reduce the size and redundancy of your code. See Appendix D for examples.

### 7.2.5 All Together

All together, your code should look like this:

```
/*
 Blink:
 Turns an LED on for one second, then off for one second.
Repeats infinitely.
 This example code is in the public domain.
*/
// Step 1: Declare your variables.
// On most Arduino boards, pin 13 is connected to the LED.
// Give it a name:
int led = 13;
state = 0;

// Step 2: Set up your pins
// the setup routine runs once when you press reset:
void setup() {
 // Initialize the digital pin as an output.
 pinMode(led, OUTPUT);
}
// Step 3: The main program
// The loop routine runs over and over again forever:
void loop() {
  switch (state) {
        case 0:
                 // Turn the LED on. HIGH is the voltage level.
                digitalWrite(led, HIGH);
                // Wait for one second (1000 milliseconds).
                delay(1000);
                state = 1;    // Switch the state
                break;

        case 1:
                // Turn the LED off. LOW is the voltage level.
                digitalWrite(led, LOW);
                delay(1000);     // Wait for one second.
                state = 0;  // Switch the state
                break;
        }
}
```

Once this is complete it's time to upload your code to the Arduino, and see if it works.

## 7.3 Pulse Width Modulation

As stated earlier, sensors and actuators can represent data in digital and analog forms. Digital signals have two possible states, on and off. While analog have an infinite number of states between on and off. The issue arises when trying to control analog systems with digital interfaces.

Thankfully, with a technique called pulse width modulation (PWM), we can represent non-binary information as a binary signal. Most modern sensors, motor drivers, and actuators rely on PWM.

#### 7.3.1 How does it work?

Imagine that you had a simple circuit of a motor, battery and switch connected in series. The motor takes some time to fully ramp up to speed. If you left the switch on forever, the motor would reach its maximum speed.

Imagine that you were able to toggle the switch really quickly. By flipping the switch on and off, you give it a bit of energy, but not enough to allow the motor to reach its maximum speed.

Theoretically by toggling the switch, you would be able ramp the speed of the motor to somewhere in between 0 and the maximum speed of the motor.

This is how it works in a microcontroller, every few milliseconds; the microcontroller sends a 5V “pulse” of electricity to the motors. The greater the required speed, the longer the “pulse” will be. Or in other words, the greater the required speed is, the longer the switch stays on.

Appendix G shows the difference between PWMs and Analog implementation in Arduino code.

## 8 Conclusion

These are the basic concepts that you’ll be using when building your robot. If you encounter any problems, talk to your mentors, or shoot me an email I hope this guide is a useful introduction to SumoBots. Good Luck!

## A Arduino Anatomy

This picture, with all its parts and pins, might seem complicated to someone who's never used a microcontroller before. Don't worry — it's not as hard as it looks. And to prove it, we made a chart for easy reference! Parts are listed clockwise from "External Power Supply" to "Reset Pin."

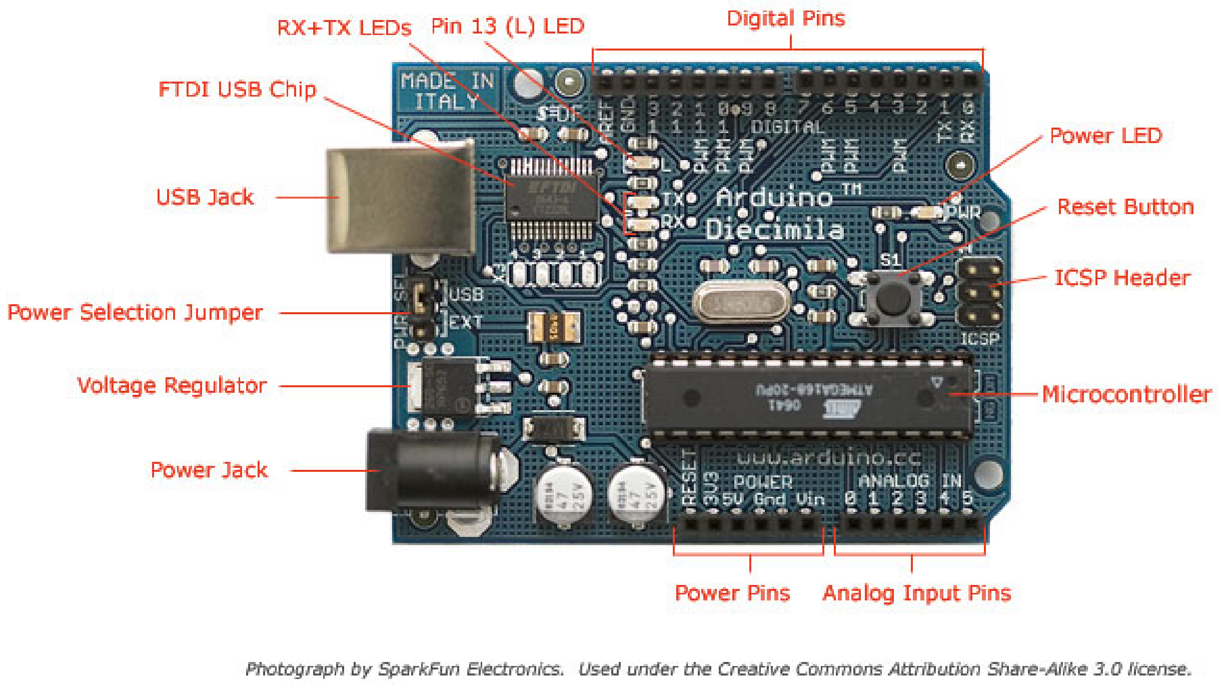

Figure 3: The Arduino

| Port | Function |
|---|---|
| External power supply | Plug for the battery that powers the Arduino and sensors. Use a separate power supply for the motors. |
| Analog reference pin | You probably won't use this pin. Provides a reference voltage for Analog inputs. Default is 5 V, but can be changed. |
| Digital ground | Ground wires for digital inputs and all outputs must terminate here. |
| Digital I/O pins | Digital inputs and all outputs are connected here. 12 pins available for digital inputs and outputs, numbered 2-13. |
| Serial out (TX) | Uses serial communication to send data to a computer or other device |
| Serial in (RX) | Uses serial communication to receive data from said device |
| Reset button | Erases all programming embedded on the Arduino. Don't push it by accident! |
| In-circuit serial programmer | You probably won't use this pin. Allows you to burn sketches onto the Arduino board through an external programmer instead of the bootloader (which loads the OS). |
| ATMega328 microcontroller | This is the processor. Contains 32 KB of memory. |
| Analog in pins | Connection point for sensors that take Analog readings. 6 pins are available for Analog inputs. |
| Voltage in | You probably won't use this pin. An alternative to the Arduino's external power supply. |
| Ground pins | Connection point for ground wires from Analog inputs. There are two ports. It doesn't matter which one you use. |
| 5 Volt power pin | This pin has 5 V (relative to the ground). Sensors and motors that require 5 V or Vcc should be wired to this pin. |
| 3.3 Volt power pin | This pin has 3.3 V (relative to the ground). Sensors and motors that require 3.3 V or Vcc should be wired to this pin. |
| Reset pin | Will reset the Arduino if it receives a positive voltage |

## B The CAD Checklist

- Ensure design is within competition constraints
- Ensure that all sensors have been considered (IR and Distance)
  - Ensure that all sensors can be wired
  - Ensure that sensors are able to see the environment
- Ensure that Arduinos and supporting wires are in a secure place
- Ensure that Batteries are at an easily accessible place
  - Ensure that all components (Sensors, Actuators and Arduinos) can be powered from the battery
- Ensure Motors are at the correct place
  - Ensure that motors can touch the ground and move the robot

## C Hooking Up Sensors and H-Bridges

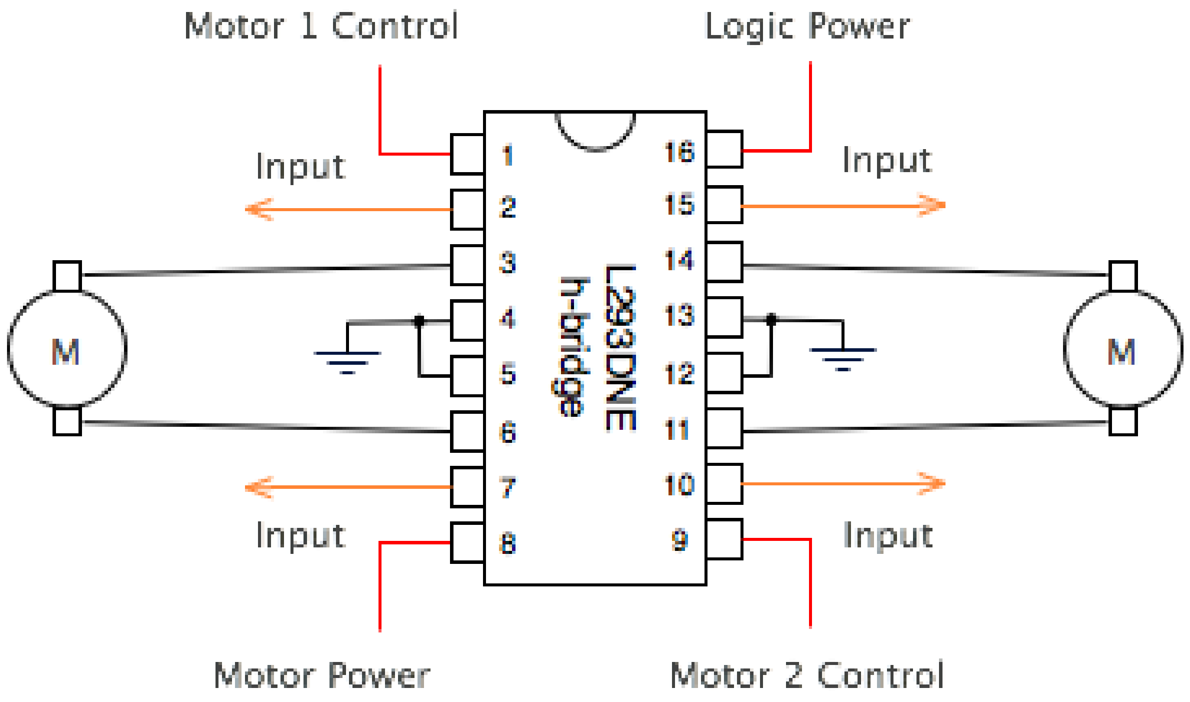

Figure 4: The Arduino

http://bildr.org/2011/03/various-proximity-sensors-arduino/ has a great tutorial on hooking up sensors to the arduino.

## D Sample SumoBot Code

```
//Declare pins for the right motor.
const int motor_pin1 = 5;
const int motor_pin2 = 6;

//Declare pins for the left motor.
const int motor_pin3 = 10;
const int motor_pin4 = 11;

//Declare pins for photoresistors and the IR sensor.
const int IRSens = 0;
const int PHOTO1 = 1;
const int PHOTO2 = 2;
const int PHOTO3 = 3;
const int PHOTO4 = 4;

//Declare photoresistors  cut-off values for black-white detection.
//This is what you came up with after testing:
int InitVal1 = 120;
int InitVal2 = 50;
int InitVal3 = 270;
int InitVal4 = 190;

//Step 2: Set up your ports

void setup() {

    // Data travels through the serial data stream at the 9600 band.
    Serial.begin(9600);

    // Set up motor pins.
    pinMode(motor_pin1, OUTPUT);
        .
    // Set up sensor pins.
    pinMode(PHOTO1, INPUT);
    ...
}

//Step 3: Writing functions
//Helpful as these examples may be,
//you  ll probably have to write other functions on your own.

// This next function is a debugging tool that
//prints sensor readings for your own sake.
//The Arduino doesn t actually use checkSens() to check the sensors.
void checkSens() {
    Serial.print("Distance: ");
    Serial.print(analogRead(0));
    Serial.print("\tPhoto 1: ");
    Serial.print(analogRead(PHOTO1));
    Serial.print("\tPhoto 2: ");
    Serial.print(analogRead(PHOTO2));
    Serial.print("\tPhoto 3: ");
    Serial.print(analogRead(PHOTO3));
    Serial.print("\tPhoto 4: ");
    Serial.println(analogRead(PHOTO4)); }

//This function tells the motors to move forward:
void forward() {
    analogWrite(motor_pin1,0);
    analogWrite(motor_pin2,150);
    analogWrite(motor_pin3,150);
    analogWrite(motor_pin4,0);
}

//This function stops all motors:
void halt () {
    AnalogWrite(motor_pin1,0);
    AnalogWrite(motor_pin2,0);
    AnalogWrite(motor_pin3,0);
    AnalogWrite(motor_pin4,0); }

//Step 4: The main code
//Your main code will be much longer than the provided example,
//but it should give you an idea of where to start:

void loop() {

    checkSens();
    //Get readings from the sensors that can be stored as variables.
    unsigned int read1 = analogRead(PHOTO1);
    unsigned int read2 = analogRead(PHOTO2);
    unsigned int read3 = analogRead(PHOTO3);
    unsigned int read4 = analogRead(PHOTO4);
    unsigned int thing = analogRead(IRSens);

    //Decision-making code:
    if( (read1 > InitVal1) && (read2 > InitVal2)) {
        //If both front sensors detect white:
        halt();
    }
    else {forward;}

}

Put together, the code looks like this:

//Declare pins for the right motor.
const int motor_pin1 = 5;
const int motor_pin2 = 6;
```

```
//Declare pins for left motor
const int motor_pin3 = 10;
const int motor_pin4 = 11;

//Declare pins for photoresistors and the IR sensor.
const int IRSens = 0;
const int PHOTO1 = 1;
const int PHOTO2 = 2;
const int PHOTO3 = 3;
const int PHOTO4 = 4;

//Declare photoresistors  cut-off values for black-white detection.
//This is what you came up with after testing:
int InitVal1 = 120;
int InitVal2 = 50;
int InitVal3 = 270;
int InitVal4 = 190;

//Get readings from the sensors that can be stored as variables:
unsigned int read1;
unsigned int read2;
unsigned int read3;
unsigned int read4;
unsigned int thing;

void setup() {

    // Data travels through the serial data stream at the 9600 band.
    Serial.begin(9600);

    // Set up motor pins.
    pinMode(motor_pin1, OUTPUT);
        .
    // Set up sensor pins.
    pinMode(PHOTO1, INPUT);
    ...
}

// Tells you what the sensors see.
//Doesn t make the Arduino check the sensors.
void checkSens() {
    Serial.print("Distance: ");
    Serial.print(analogRead(0));
    Serial.print("\tPhoto 1: ");
    Serial.print(analogRead(PHOTO1));
    Serial.print("\tPhoto 2: ");
    Serial.print(analogRead(PHOTO2));
    Serial.print("\tPhoto 3: ");
    Serial.print(analogRead(PHOTO3));
    Serial.print("\tPhoto 4: ");
    Serial.println(analogRead(PHOTO4)); }
```

```

//Tells the motors to move forward:
void forward() {
    analogWrite(motor_pin1,0);
    analogWrite(motor_pin2,150);
    analogWrite(motor_pin3,150);
    analogWrite(motor_pin4,0);
}

//Stops all motors:
void halt () {
    AnalogWrite(motor_pin1,0);
    AnalogWrite(motor_pin2,0);
    AnalogWrite(motor_pin3,0);
    AnalogWrite(motor_pin4,0); }

void loop() {

    checkSens();
    //Get readings from the sensors that can be stored as variables:
    read1 = analogRead(PHOTO1);
    read2 = analogRead(PHOTO2);
    read3 = analogRead(PHOTO3);
    read4 = analogRead(PHOTO4);
    thing = analogRead(IRSens);

    //Decision-making code:
    if( (read1 > InitVal1) && (read2 > InitVal2)) {
        //If both front sensors detect white:
        halt();
    }
    else {forward;}

}
```

## E PMWs in Arduinos

### E.1 Analog

```
//First, map the pins.
const int analogInPin = A0;  // Analog input
const int analogOutPin = 9; // Analog output

//Define other variables:
int InputValue = 0;
int OutputValue = 0;

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



void loop() {
        //Tell it how to read the analog input value:
        ImputValue = analogRead(analogInPin);

        // Do all necessary calculations and rescale
        // the values so they fall between 0 and 255.
        OutputValue = map(InputValue, 0, 1023, 0, 255);

        // Change the analog output value.
        analogWrite(analogOutPin, OutputValue);
}
```

### E.2 PWM

```
//First, map the pins.
cons tint digitalInputPin = 10; // Digital Input
const int digitalOutPin =  9;      // Digital Output

//Define other variables:
int InputValue = 0;
int OutputValue = 0;

void setup() {
  Serial.begin(9600);
  pinMode(digitalInputPin, OUTPUT);
  pinMode(digitalOutputPin, INPUT);
}

void loop() {
    //Tell it how to read the digital input value:
  InputValue = digitalRead(digitalInputPin);

  // It isn't necessary to rescale digital values,
  // because they already fall between 0 and
  // 255.
  OutputValue = map(InputValue, 0, 255, 0, 255);

  // Output the necessary digital value.
  digitalWrite(digitalOutputPin, OutputValue)
}
```