# Aberystwyth Robotics Club - Magician Chassis - Programming Instructions

## Motor Instructions

The motors on the Magician Chassis are controlled by a h-bridge motor controller and has 2 channels, which allows the control of two DC motors.

1. **First declare the variables.**

```
byte leftForward = 5;
byte leftReverse = 6;
byte rightForward = 9;
byte rightReverse = 10;
```

2. **Define the motors inside the setup function as output devices.**

```
pinMode(leftForward, OUTPUT); //set leftForward as output
pinMode(leftReverse, OUTPUT); //set leftReverse as output
pinMode(rightForward, OUTPUT); //set rightForward as output
pinMode(rightReverse, OUTPUT); //set rightReverse as output
```

3. **Create the five functions below, to give instructions to the robot.**

   - robotForwards()
   - robotLeft()
   - robotRight()
   - robotReverse()
   - robotStop()

   "A function is to divide up your code into chunks, this will make it easier to create instructions to the robot and prevents code duplication!"

See below the example of creating a function.

digitalWrite means we want to send a digital signal to whatever is inside this function. leftForward is the pin we would like the digitalWrite to act on.

HIGH is what we would like the pin to do. Digital signals work on 1s and 0s, 1 being on and 0 being off. In our case, 1 will be go and 0 will be stop.

```
void robotForwards(){
  digitalWrite(leftForward, HIGH); //send the left motor forwards
  digitalWrite(leftReverse, LOW); //prevent the left motor reversing
  digitalWrite(rightForward, HIGH); //send the right motor forwards
  digitalWrite(rightReverse, LOW); //prevent the right motor reversing
}
void robotLeft(){
  digitalWrite(leftForward, LOW); //prevent the left motor going
      forwards
  digitalWrite(leftReverse, HIGH); //reverse the left motor
  digitalWrite(rightForward, HIGH); //send the right motor forwards
  digitalWrite(rightReverse, LOW); //prevent the right motor reversing
}
```

| Arduino Signal | Motor Action |
|---|---|
| HIGH | GO |
| LOW | STOP |

4. **Try the functions in void loop().**

   Once you create the instructions in the functions you can call the function in void loop(), try all the functions you have created and make the robot move around:

```
void loop(){
  robotForwards();
}
```

5. **Work out how you could use these functions and the delay function to navigate a maze.**

   For example:

```
void loop(){
  robotForwards(); //move robot forwards
  delay(1000) //1000 milliseconds = 1 second
  robotLeft(); //move robot left
  delay(500) //500 milliseconds = 0.5 second
}
```

## Servo

A servo is a small motor which, using a potentiometer, can measure the amount the shaft has turned. In our case we will want to know how far the servo has turned in degrees.

### Hardware Required

- Arduino or Genuino Board
- Servo Motor
- Jumper Wires

### Circuit

Servo motors have three wires: power, ground, and signal.

1. **Connect the power wire to the 5V pin on the Arduino or Genuino board.**

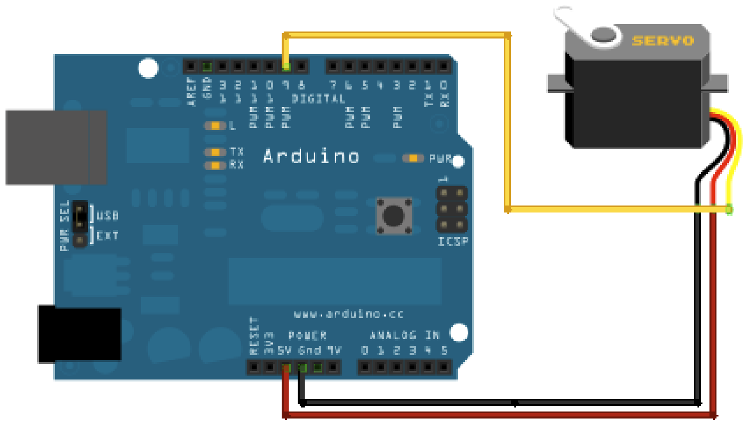
   The power wire is typically red.

2. **Connect the ground wire to a ground pin on the board.**

   The ground wire is typically black or brown.

3. **Connect the signal wire to pin 11 on the board.**

   The signal wire is typically yellow, orange or white.



## Code

1. **Import the Servo library.**

   First we need to import a library for the Arduino, this stores some extra code so we don't have to write as much!

   ```
   #include <Servo.h>
   ```

2. **Create a servo object.**

   The first word calls the Servo library and myservo is what we will call the servo later on in the code:

   ```
   Servo myservo;
   ```

3. **Create a variable to store the position of the servo.**

   We will set the servo start point as 0 for now:

   ```
   int pos = 0;
   ```

4. **In setup, attach myservo to pin 11.**

   ```
   void setup() {
   myservo.attach(11);
   }
   ```

5. **Move the servo.**

   The next step is to move the servo. The first part of the code is the most important part.

   Below is some pseudo code to explain what is happening.

   We start the position of the servo at 0 degrees, if the position is less than or equal to 180 degrees, then add one degree to the position variable.

   Now we have the value 1 in variable 'pos', myservo will move 1 degree and wait 15 milliseconds.

   Once it has moved 1 degree and waited, the code returns to the top and adds an extra degree to the position variable and loops this over and over until the position variable reaches 180.

   Once the servo reaches 180 degrees, it jumps outside the for loop, then moves along to the next for loop where it does exactly the same action but in reverse from 180 to 0.

   This can be done using the code below:

   ```
   void loop() {
     for (pos = 0; pos <= 180; pos += 1) { // goes from 0 degrees to 180
        degrees
       // in steps of 1 degree
       myservo.write(pos); // tell servo to go to position in variable "
          pos"
       delay(15); // waits 15ms for the servo to reach the position
     }
     for (pos = 180; pos >= 0; pos -= 1) { // goes from 180 degrees to 0
        degrees
       myservo.write(pos); // tell servo to go to position in variable "
          pos"
       delay(15); // waits 15ms for the servo to reach the position
     }
   }
   ```

## Light Dependent Resistors Instructions

In order to detect the intensity of light or darkness, we use a sensor called an LDR (Light Dependent Resistor). The LDR is a special type of resistor which allows higher to pass through it (low resistance) whenever there is a high intensity of light, and passes a low voltage (high resistance) whenever it is dark.

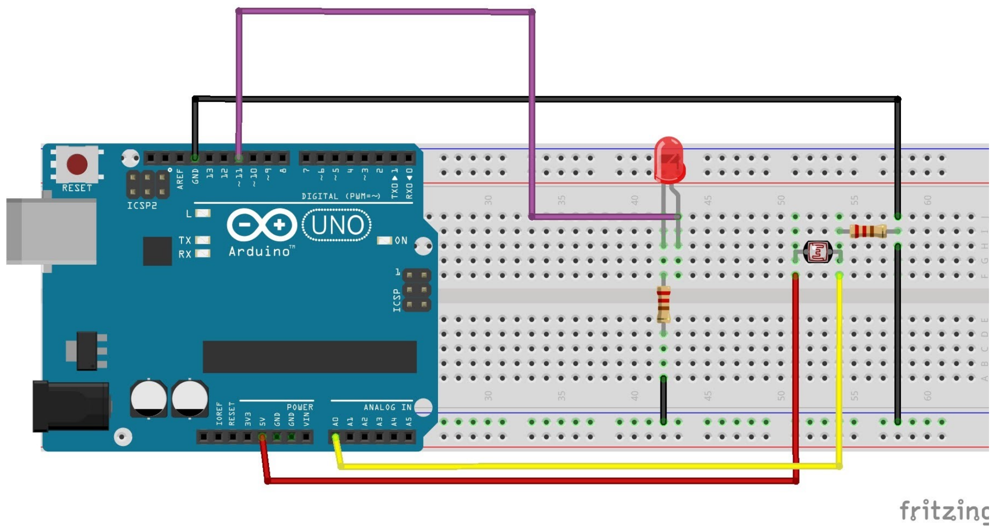
### Hardware Required

- Arduino
- LED
- LDR
- Jumper Wires
- 100K resistor
- 220 ohm resistor
- Breadboard

### Circuit

You have to use a voltage divider configuration to connect the LDR. The connection diagram for the Arduino is given below, using a breadboard.

1. **Connect one leg of the LDR to VCC (5V) on the Arduino.**
2. **Connect the other leg of the LDR to the analogue input pin A0 on the Arduino.**
3. **Connect the 100K resistor to the same leg and ground.**
4. **Connect the power/long leg of the LED straight to pin 13 on the Arduino.**
5. **Connect the ground/short leg of the LED to the 220 ohm resistor.**
6. **Connect the 220 ohm resistor to ground.**



Connecting a LDR sensor and LED to an Arduino via a bread-board

## Code

The first lines of code will let the Arduino know which pins are used in the code.

1. **Assign the pin values.**

```
int LDR = A0; //LDR pin is connected to A0
int LED = 13; //LED pin is connected to 13
```

2. **Create a variable to store data from the LDR called LDRvalue.**

```
int LDRvalue = 0; //a place to store the LDR value
```

3. **In the setup function, begin serial communications, at 9600 bits of data per second, between your board and the computer.**

```
Serial.begin(9600); //set up data connection speed
```

4. **In the setup function, initialise the LDR as an input.**

```
pinMode(LDR, INPUT); //set LDR pin (A0) as input
```

5. **In the setup function, initialise the LED as an output.**

```
pinMode(LED, OUTPUT); //set LED pin 13 as output
```

6. **In the loop function, store values read from the analogue input in the variable LDRvalue.**

```
LDRvalue = analogRead(LDR); //LDRvalue will equal values from the LDR
    pin
```

Once we have the value from the LDR, we can then use it to control the brightness of the LED.

7. **Use the value of the LDR to change the LED's brightness.**

```
analogWrite(LED, LDRvalue; //LDRvalue will determine intensity of LED
    brightness
```

8. **Use the serial line to print the information stored in the LDRvalue variable to the "Serial Monitor".**

```
Serial.print("LDR␣reading␣is:␣"); //print out everything between the
    quotes
Serial.println(LDRvalue); //print out data in LDRvalue
delay(10); //delay the code for 10 milliseconds so we can keep up with
     the readings
```

You should expect to see values between 0 and 1023 on teh computer when you cover and uncover the LDR. The brightness of the LED should also change slightly.

## Example Code

```
int LDR = A0;
int LED = 13;
int LDRvalue = 0;

void setup{
  Serial.begin(9600);
  pinMode(LDR, INPUT);
  pinMode(LED, OUTPUT);
}

void loop() { //the loop function runs over and over again forever
  LDRvalue = analogRead(LDR);
  Serial.print("LDR␣reading␣is:␣");
  Serial.println(LDRvalue);
  delay(10);
}
```

## Behaviours

1. Can you make the robot follow a torch?

   You may consider using “if” statements to do this.

## Ultrasonic Instructions

This example shows you how to find the distance from an obstacle using ultrasonic sensors, for now we will return the values from the ultrasonic to the serial monitor on the Arduino.

## Hardware Required

- Arduino
- HC-SR04 Ultrasonic
- Jumper Wires
- Optional breadboard

### Circuit

Ultrasonics work by sending out pulses of sound and then counting how long it takes for those pulses to come back. So an Ultrasonic sensor actually has two parts: a transmitter and a receiver.

If you look at the sensor it has two round bits, labelled T for transmitter and R for receiver. (In electronics things which both transmit and receive signals are often called "Transceivers" so this is really an Ultrasonic transceiver.)

The Ultrasonic sensor has 4 wires labelled Vcc, Trig, Echo and Gnd.

These are for the voltage, the trigger of the sonar, the echo detection signal, and the ground.

1. **Wire up the the first wire (voltage, or Vcc) to the 5v output of your Arduino.**
2. **Wire up the trigger or Trig to data pin 3 on your Arduino.**
3. **Wire up the echo detection signal to data pin 4 on your Arduino.**

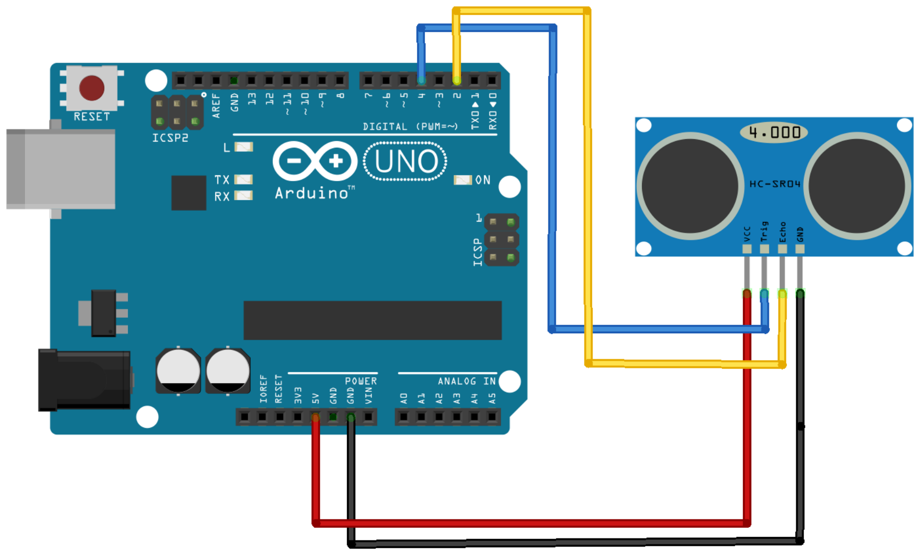


Figure 1: Connecting a Ultrasonic sensor to an Arduino

## Code

Remember that all Arduino programs have two parts: the setup which happens once, and is used to set up anything you need to use in the program. This is where you tell the Arduino which components are wired up to which pin, and stuff like that. There is also a loop, and the loop is repeated again and again until the Arduino is turned off (or runs out of battery power).

In the setup function here we need to tell the Arduino which pin has the Trig connection, and which pin has the Echo connection.

We are going to use the trigger pin to send a message to the sensor from the Arduino (so that is an OUTPUT pin), but we are going to use the Echo pin to read a message from the sensor (so that is an INPUT pin).

1. **Plug in your Arduino board.**
2. **Start the Arduino Software (IDE).**
3. **Declare the variables echo and trigger.**

   ```
   int trigger = 3;
   int echo = 4;
   ```

4. **Enter the code below in the setup function.**

   ```
   Serial.begin(9600);
   pinMode(trigger, OUTPUT);
   pinMode(echo, INPUT);
   ```

   As well as telling the program what's connected to what pin, it also sets up the serial monitor so we are ready to start monitoring stuff.

5. **Enter the following code into the loop function.**

   ```
   long duration;
   digitalWrite(trigger, HIGH);
   delayMicroseconds(10);
   digitalWrite(trigger, LOW);
   duration = pulseIn(echo, HIGH);
   Serial.print(duration);
   Serial.println("␣:␣duration␣";
   ```

   This triggers the sensor to send out a signal, then reads the signal coming back from the sensor.

   There is a lot going on in this program.

   - First we setup a variable called duration to hold the length of time between the sound going out and the sound bouncing back.
   - Then we trigger a quick pulse of sound (by writing HIGH to the trigger pin) - we only do this for a tiny amount of time (10 microseconds) before turning it off again by writing LOW to the trigger pin.
   - We then capture the time from the pulse being sent out to coming back by looking for a HIGH pulseIn on the echo pin.
   - Finally we print the duration out to the serial monitor.

6. **Test the program by clicking the tick.**
7. **Put the program on the Arduino using the arrow button.**
8. **Open the serial monitor.**

   To open the serial monitor, click on the button that looks a bit like a magnifying glass in the top right corner of the Arduino IDE window (it will say "Serial Monitor" when you hover over it). That will open up a new window which contains the stuff you have printed to Serial. You should see something like this:

   ```
   386: duration
   417: duration
   381: duration
   ```

   These are measurements of how long it took (in microseconds) for the sound to bounce off a thing in the world and then back to the ultrasonic sensor.

## Going from a measurement of time to a measurement of distance

1. **Set up a barrier near your computer, and get hold of a ruler.**
2. **Measure the distance from the sensor to the barrier, and fill in a table of measurements from the serial monitor.**
   (a) If the barrier is 10cm from the sensor, how long does it take the sound to travel?
   (b) Take some more measurements. Can you come up with a way of converting “sound-travel-time” into centimetres?
   (c) If you can come up with a useful conversion system, can you then get that into your program - that is, can you work out how to get your program to give an output in centimetres? Give it a go, and then test your ultrasonic ruler against a real ruler.

## Behaviours

1. Can you avoid obstacles using the ultraonic sensor whilst the robot is driving?