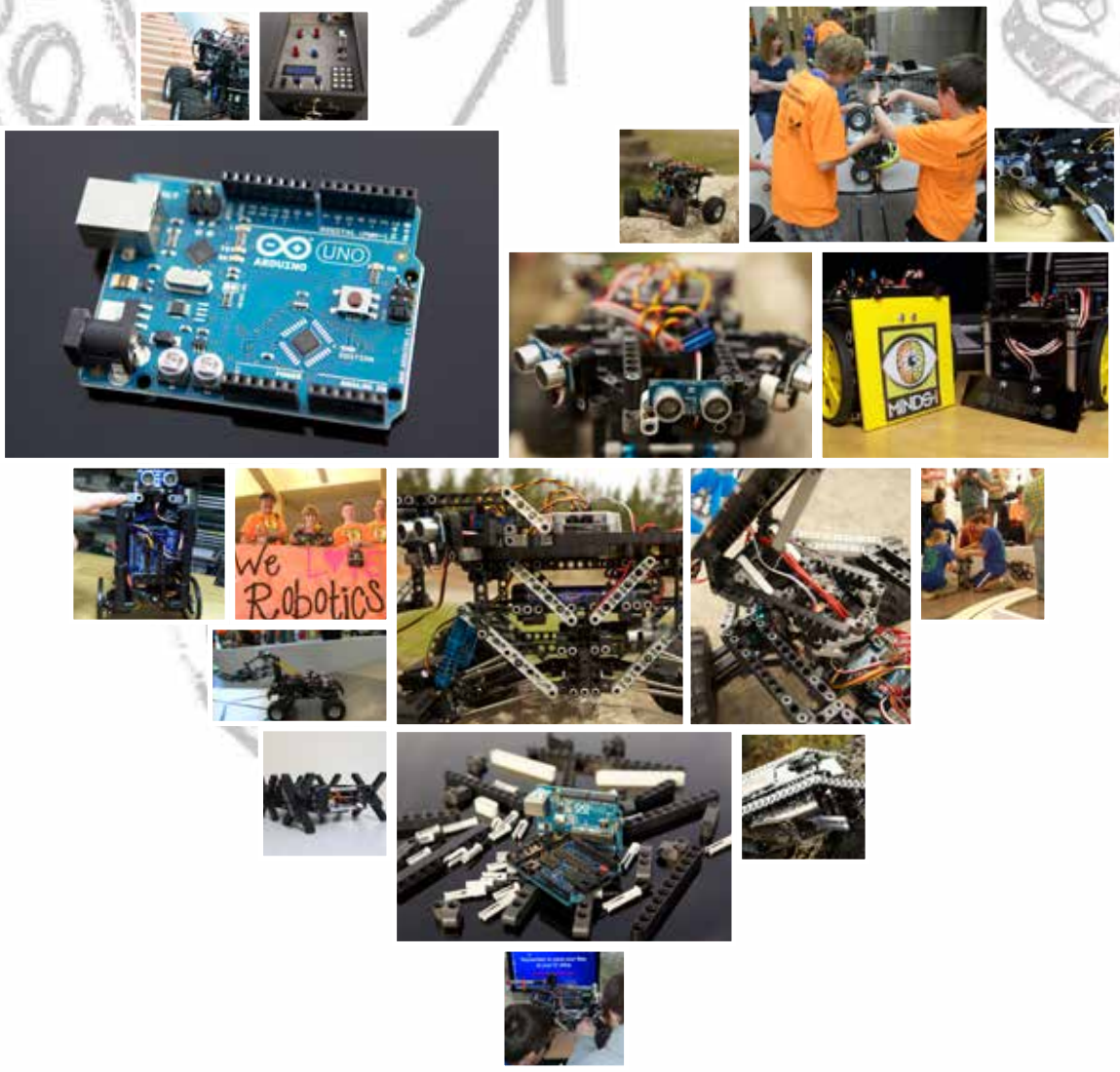
MINDS-i EDUCATION

EMC-MAUG-001

Arduino User Guide



Helpful instructional videos available at: mindsieducation.com

Install the Arduino Software (IDE)

This document explains how to install the Arduino Software (IDE) on Windows machines. Visit Arduino.cc for step by step insructions for other operatating systems like Mac OS X, Linux, and Portable IDE (Windows and Linex).

The Arduino Software (IDE) allows you to write programs and upload them to your board.

Download the Arduino Software (IDE)

Get the latest version of Arduino Software (IDE) on Arduino's website, Arduino.cc, from the download page.

TIP

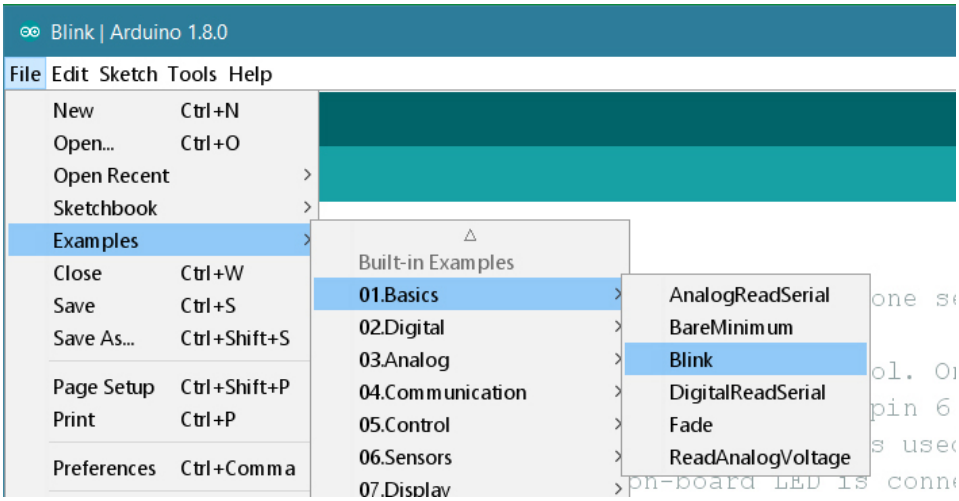
Before installing Arduino contact your school disctrict computer administrator or IT department to ensure the program is correctly installed with admistrator priveledges.

Installing drivers for Leonardo

Drivers should be automatically installed plugging with an USB cable the board to your PC, but with some version of the Windows operative system (like Windows 7, Vista and 10) it can happen that your board won't be recognized and you will get the message Unknown USB device. It is so necessary to manually install them following the guide Manually install Drivers on Windows.

Open the Blink example

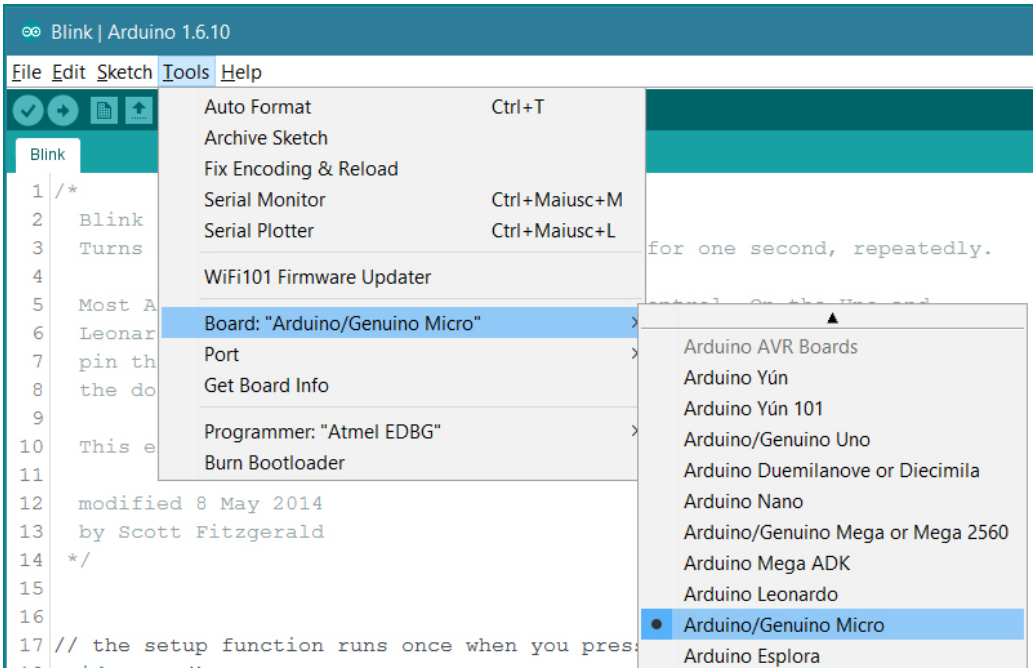
Now that you've set up your online IDE let's make sure your computer can talk to the board, it's time to make sure you can upload a program. To do that let's open the LED blink example sketch: File > Examples > 1.Basics > Blink.



Select your board

You'll need to select your board in the Tools > Board menu:

Arduino Leonardo
according to the board you have.



Select your serial port

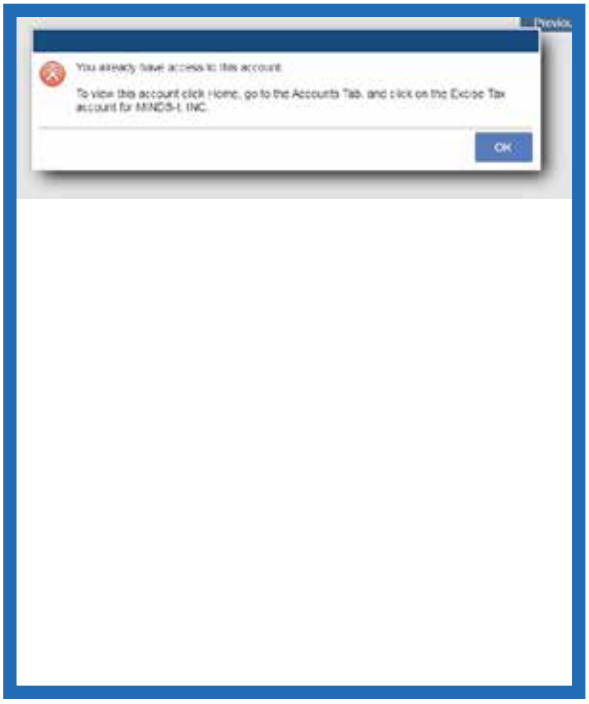
Select the serial device of the board from the Tools > Serial Port menu.

Upload and Run your first Sketch

Click the Upload button in the upper left to load and run the sketch on your board:

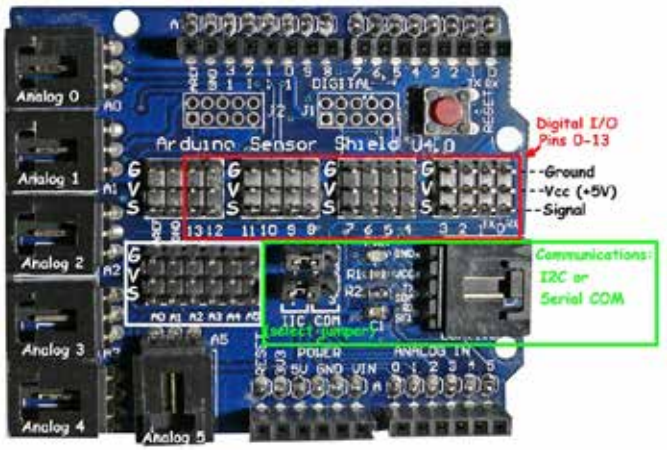
After the compilation and upload process, you should see the message Done Uploading and the built-in LED of the board should start blinking.

Install the MINDS-i Libaries, go to Tools > Manage Libraries, and filter your search for the MINDS-i libaries. Install the latest version of MINDSi by MINDSi corp.



Sensor Shield

The sensor shield plugs on top of an Arduino and makes it easy to connect to all of the Digital Inputs, Outputs and the Analog Inputs. It also makes it easy to connect to relays, LEDs, Servos, etc. The sensor shield breaks out all of the Digital and Analog pins of the Ardunio as well as providing a power and ground pin for each.

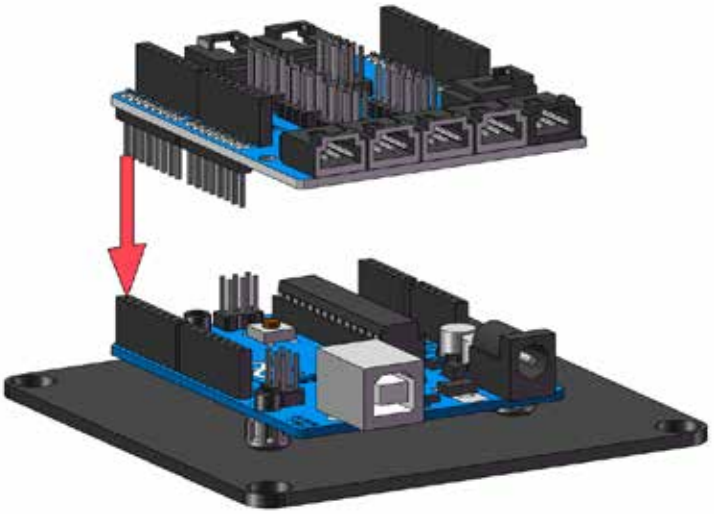


Pin Breakout

The included Arduino has 14 Digital pins (0-13), which can be used as Inputs or Outputs. It has 6 Analog pins (A0-A5) that are Input only. The sensor shield breaks out the serial and I2C port as well.

Assembling the Shield 7 Ardunio

The sensor shield and Arduino can be assemble by lining up the pins sticking out of the bottom of the shield with the headers in the top of the Arduino. The Digital pin 0 on the shield should line up with pin 0 on the Arduino. Push the two together carefully until they are fully seated.



Servo

Servos allow for specific motion control. A standard servo is capable of 180 degrees, 0 to 179. Some allow for continuous rotation. Servos also may be referred to as rotary actuators. Applications may include, steering, arms, legs, wheels and other components on a robot that involve motion.

> **Example Code**
> In the Arduino Program: File > Examples > MINDSi > 1-Calibration > Servo

Code:

This section will cover using writing code to run a servo. To begin, include the servo library:

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

Define a "Servo" object named "myservo":

```
Servo myservo;
```

In the "void setup()" function attach "myservo" to a pin 5 on the Arduino. Pins 0 to 13 and A0 to A5 could have been used instead.

```
void setup( )
{
      myservo.attach(5);
```

Also in the "void setup()" function, set the value to send to the servo. We can use any value from 0 to 179. (See figure 1.1) Neutral or center is usually around 90. You can figure what value centers your servo for yourself, see the "Challenge" section. The "void setup()" function should now be:

```
void setup( )
{
      myservo.attach(5);
      myservo.write(90);
}
```

To check if your code is correct press the verify button: Run by clicking upload:

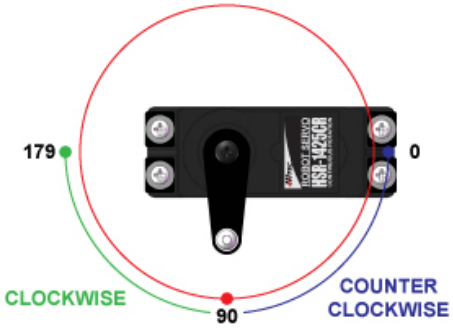
1.2 Challenge Fine Right and Left, Forward and Reverse

The value of right and left or forward and reverse can be different for each servo. It depends on how the servo is mounted on your robot and may even vary due to manufacturer.

Open the Arduino program, Go to File > Examples > 0. Minds-i > 1-Calibration > Servo. Plug your Arduino into your computer via USB and plug your servo into pin 5. Look for this line of code in the void loop section: “myservo.write(90);” Change the value of 90 to 0 or 179 Watch which direction your servo rotates or moves.

Figure 1.1: Servo

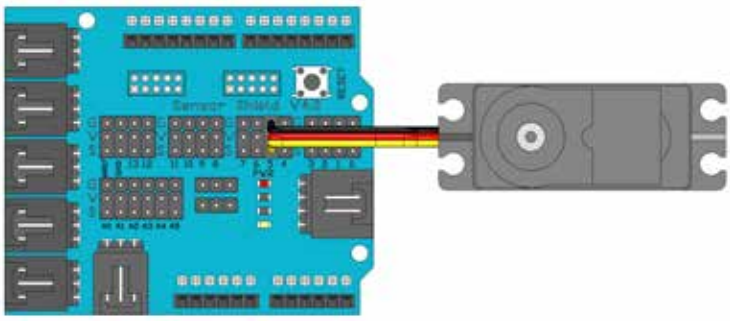
Figure 1.2: Continuous Rotation Servo



Now that you know center, find the end points. Write the following in the “void loop” section:

```
void loop()
{
      myservo.write(0);
      delay(1000);
      myservo.write(90);
      delay(1000);
      myservo.write(180);
      delay(1000);
}
```

Wiring Diagram for the Servo



1.2 Challenge Find Right and Left, Forward and Reverse

The value of right and left or forward and reverse can be different for each servo. It depends on how the servo is mounted on your robot and may even vary due to manufacturer.

Open the Arduino program, Go to File > Examples > 0. Minds-i > 1-Calibration > Servo. Plug your Arduino into your computer via USB and plug your servo into pin 5. Look for this line of code in the void loop section: “myservo.write(90);” Change the value of 90 to 0 or 179 Watch which direction your servo rotates or moves.

ESC and Motor

The Electronic Speed Controller (ESC) is used to regulate voltage and current for a DC motor. It is similar to a continuous rotation servo in that entering values from 0 to 179 changes the direction of the motor. However, the motor and ESC are for higher torque applications and contain an electronic brake making them ideal for drive systems. Other applications are arms, winches, and turntables.

Calibration Code:

This section will cover writing code to run a motor attached to an ESC.

> **Example Code**
> In the Arduino Program: File > Examples > MINDSi > 1-Calibration > ESC

Include the "MINDSi" library and the "Servo" library:

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

A "Servo" object can be used to control an ESC. Create a "Servo" object named "motor":

```
Servo motor;
```

In the "void setup()" function, attach the "motor" object to pin 4 on the Arduino. This will be the pin that the ESC will be plugged into. The ESC could also have been attached to any pin from 0 to 13 or A0 to A5.

```
void setup()
[
      motor.attach(4);
```

Whatever value is sent to the ESC initially from the microcontroller will be set as neutral. Any value in the range from 0 to 179 could be sent; however, 90 is about in the middle of this range. Send a value of 90 initially to the ESC and 90 will be set as neutral. To give the ESC time to arm before other commands are issued, the code is set to wait

for 2 seconds (2000 milliseconds) after the value of 90 is sent to the ESC.

In the "void setup()" function, add the line: "motor.write(90);" to send the initial value to the ESC and add the line: "delay(2000);" to give the ESC time to arm.

```
void setup()
[
      motor.attach(4);
      motor.write(90);
      delay(2000);
}
```

To run the code plug in the Arduino to the computer and click the upload button: There should be audible tone from the ESC as the speed controller arms.

Application Code:

Now that the ESC is setup the next step is to test the system with a robot. This example assumes that the motor is mounted as a drive motor on a robot with the ESC attached to pin 4. If a steering servo is available on the robot then it should be attached to pin 5. The ESC/motor drive assembly can be used to run the robot forwards, backwards and to brake.

Example Code
In the Arduino Program: File > Examples > MINDSi > 1-Applications > ESC

To begin with, include the "MINDSi" library and the "Servo" library:

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

Create the "Servo" objects: "drive" and "steer".

```
Servo drive, steer;
```

In the "void setup()" function, attach the "drive" object to pin 4 and the "steer" object to pin 5. The ESC will be wired to pin 4 on the Arduino and the steering servo, if there is one, to pin 5.

```
void setup()
[
      drive.attach(4);
      steer.attach(5);
```

Set both to 90, this should center the servo (if it is connected) and set the ESC at neutral. As before, a 2 second delay is added to allow the ESC to arm.

```
      drive.write(90);
      steer.write(90);

      delay(2000);
}
```

The robot will be programmed to drive forward for one second; brake; drive backwards for one second and then coast for one second.

In the “void loop()“ function add the line: “drive.write(100)” to tell the robot to drive forward. Since 90 has been set as the neutral value for the ESC, any value from 91 to180 will tell the robot to drive forward. The robot will continue to drive forward at the speed written to the ESC until another command is issued. Values closer to 180 make the robot drive faster.

```
void loop(){

      drive.write(100);
```

The next step is to keep driving forward for 1 second (1000 milliseconds). To do this add the command: “delay(1000)”.

```
void loop(){

      drive.write(100);
      delay(1000);
```

The ESC can be used to apply an electronic brake. If the robot has been driving forward, sending any value below the neutral value will apply the brake. In this case since the neutral value is 90, any value between 0 and 89 will trigger the brake. Add the line: “drive.write(45);” to send the value of 45 to the ESC and thus apply the brake.

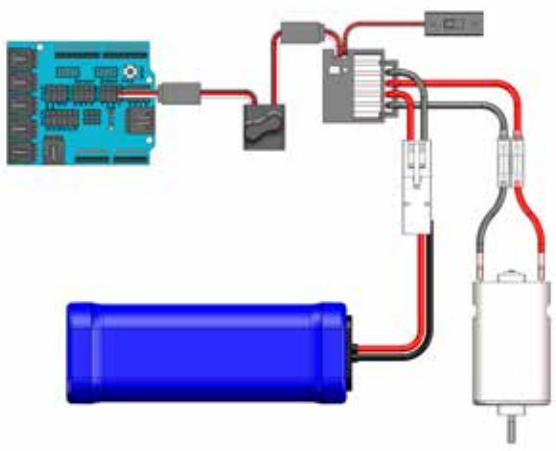
```
void loop(){

      drive.write(100);
      delay(1000);

      drive.write(45);
```

Also add a “delay” instruction to wait for the brake to take effect. Add the line: “delay(1000);”

```
void loop(){

      drive.write(100);
      delay(1000);

      drive.write(45);
      delay(1000);
```

To allow the robot to start moving again the electronic brake must be disengaged. To disengage the brake, send the command to put the ESC in neutral followed by another delay instruction. Add the following lines to the program: “drive.write(90);” and “delay(50);”. 50 milliseconds is enough time for the brake to disengage.

Wiring Diagram for the ESC and Motor



```
void loop(){

      drive.write(100);
      delay(1000);

      drive.write(45);
      delay(1000);

      drive.write(90);
      delay(50);
```

To drive the robot backwards once the ESC has been reset to neutral, any value below neutral can be used. In this case any value from 0 to 89 will put the robot in reverse. In this case, values close to 0 make the robot go FASTER. To drive in reverse slowly send 80 to the speed controller by adding the line: “drive.write(80);”. The robot will drive backwards for 1 second so also add the line: “delay(1000);”.

```
void loop(){

      drive.write(100);
      delay(1000);

      drive.write(45);
      delay(1000);

      drive.write(90);
      delay(50);

      drive.write(80);
      delay(1000);
```

Finally put the robot in neutral at the end of the “void loop()” function so that it is ready to start driving forward again when the sequence restarts. Add the lines: “drive.write(90);” and “delay(1000);” The completed function should look like:

```
void loop(){

      drive.write(100);
      delay(1000);

      drive.write(45);
      delay(1000);

      drive.write(90);
      delay(50);

      drive.write(80);
      delay(1000);

      drive.write(90);
      delay(1000);
}
```

1.2 Challenge: Understanding Forward, Reverse, and the Electronic Brake

WARNING: These robots drive very fast. Keep the values sent to the ESC near 90, for instance, between 80 and 100 until the robot’s speed has been assessed.

Try changing the values in the “drive.write(…)” commands to make the robot go faster. Try changing for how long the robot drives before braking. Try having the robot go forward again after it brakes to a stop instead of going into reverse.

To test the code plug in the Arduino and hit the upload button. The robot should drive forward; come to a stop; drive backwards; stop; and repeat.

Digital Outputs

A digital output is a device that has two states, usually on and off. A common digital output is a LED or Light Emitting Diode. LED's are often used as indicator lights. The LED's supplied with your kit can be connected directly to the Arduino shield. If a higher amperage digital output device is required, it can be controlled through a relay.

Example Code
In the Arduino Program: File > Examples > 01.Basics > Blink

*Replace all references to "LED_BUILTIN" with "12" before running the above code to make the examples in this chapter work.

Test Code:

In the "void setup()" function setup pin 12 to be used as an output pin. Make sure that the LED is plugged into pin 12 on the Arduino shield. Add the line "pinMode(12,OUTPUT);" to the "void setup()" function.

```
void setup()
[
      pinMode(12,OUTPUT);
}
```

Besides pin 12, any pin from 0 to 13 could have been used for digital output.

To see the LED blink, the LED must be turned on and off. This is accomplished by sending instructions that tell the output pin the Arduino to either go to 0 volts, "LOW", or 5 volts, "HIGH". Add the lines "digitalWrite(12,HIGH);", "delay(1000);", "digitalWrite(12,LOW);" and "delay(1000)" to the "void loop()" function:

```
void loop()
{
      digitalWrite(12,HIGH);
      delay(1000);
      digitalWrite(12,LOW);
      delay(1000);
}
```

The "digitalWrite(...)" instructions switch pin 12's state, thus turning the LED on or off. The "delay(1000)" instructions pause the code for one second after each state switch. Since "delay(1000)" pauses the code for 1000 milliseconds or one second, the LED blinks on and off once every 2 seconds.

1.1 Challenge: Test the code

Double check that the LED is plugged correctly into pin 12, the orientation of the plug with respect to the locations of the signal and ground wires must be correct. The LED will not function if the polarity is reversed. If the LED does not work with one orientation, reverse the plug and test it again.

To test the code, attach the Arduino to the computer and hit the upload button. The LED should blink on and off.

Radio Transmitter

A radio transmitter, or radio control (RC) unit, sends a signal from the transmitter, the controller, to the receiver. There are many different types of transmitters ranging in frequency and power. Many high-end modern transmitters use a frequency of 2.4 GHz.

> **RadioTest Example Code**
> In the Arduino Program: File > Examples > MINDSi > 1-Calibration > RadioTest

RadioTest: RadioTest.ino 8.1 Sketch - RadioTest

This code will read values being sent by the radio. The radio receiver should be connected to the Arduino as shown in Figure.11.1: “Wiring Diagram for the Radio Receiver”, and the radio should be paired with the receiver. (See RC instructions for details on how to pair the unit.)

In the Arduino sketch, start by including the “MINDSi” and the “Servo” libraries:

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

Define an integer “int” called “val”. It will be used to store the value retrieved from the radio later.

```
int val;
```

In the “void setup()” function initiate a serial connection that will later send data to the computer.

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

In the "void loop()" function the radio signal from the receiver will be read using the "getRadio(..)" function from the "MINDSi" library. It is assumed that channel 1 on the radio receiver is plugged into pin 2 on the Arduino as in Figure 11.1. The line: "val = getRadio(2)" reads the radio value sent to pin 2 on the Arduino. The next line: "Serial. println(val)" sends this value to whatever is on the serial(USB) connection, which will be the computer.
Plug the Arduino into the computer, make sure the radio receiver is connected to the Arduino as shown in Figure

```
void loop()
{
        val=getRadio(2);
        Serial.println(val);
}
```

11.1.

Upload the code.

After the code finishes uploading, open the serial monitor. Work the controls of the radio and watch how the values change. When the control on the radio transmitter corresponding to channel 1 is rotated through its full range of motion the value should vary between 0 and 180. If the full range is not reached, adjust the trim on the radio.

SimpleRadioDrive Example Code
In the Arduino Program: File > Examples > MINDSi > 1-Calibration > RadioTest

SimpleRadioDrive:

This example is an extremely basic remote-control robot. The example assumes that the robot has one drive motor attached (with its ESC) to pin 4. The robot should also have one steering servo, which should be attached to pin 5. At the end of this example, the robot should follow the commands from the Radio Controller.

Start a new sketch. Include the "MINDSi" and "Servo" libraries and define the objects "drive" and "steer" that will control the ESC/motor and the steering servo respectively.

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

Servo drive, steer;
```

Define two integer variables that will be used later: “driveSig” and “steerSig”.

```
int driveSig, steerSig;
```

In the “void setup()” function attach the ESC to pin 4 and the servo to pin 5. The ESC should have been wired to pin 4 on the Arduino and the steering servo to pin 5.

```
void setup()
{
drive.attach(4);
steer.attach(5);
```

Initialize the ESC and center the steering servo by adding the lines: “drive.write(90);”, “steer.write(90)” and “delay(2000)”. The completed “void setup()” function will be:

```
void setup()
{
drive.attach(4);
steer.attach(5);

drive.write(90);
steer.write(90);

delay(2000);
}
```

In the “void loop()” function, the values from the radio will be read. Connect channel 1 on the radio receiver to pin 2 on the Arduino. Likewise, connect channel 2 on the radio to pin 3 on the Arduino. The lines “driveSig = getRadio(2);” and “steerSig = getRadio(3)” in the “void loop()” function will read these values from channel 1 and channel 2 respectively. The values are stored in the variables defined earlier: “driveSig” and “steerSig”.

```
void loop()
{
driveSig = getRadio(2);
steerSig = getRadio(3);
```

To control the vehicle, write the values retrieved from the radio to the drive (ESC/motor) and steer (steering servo) objects. The complete “void loop()” function will be:

```
void loop()
{
driveSig = getRadio(2);
steerSig = getRadio(3);

drive.write(driveSig);
steer.write(steerSig);
}
```

Make sure the robot is positioned so that its wheels DO NOT touch the ground and the battery is NOT connected. Plug the Arduino into the computer via USB and upload the code . Once the code is uploaded, DISCONNECT the robot from the computer. Turn the receiver on first, then plug in the battery and, if needed, turn the ESC power switch on. The robot should now operate under remote control.

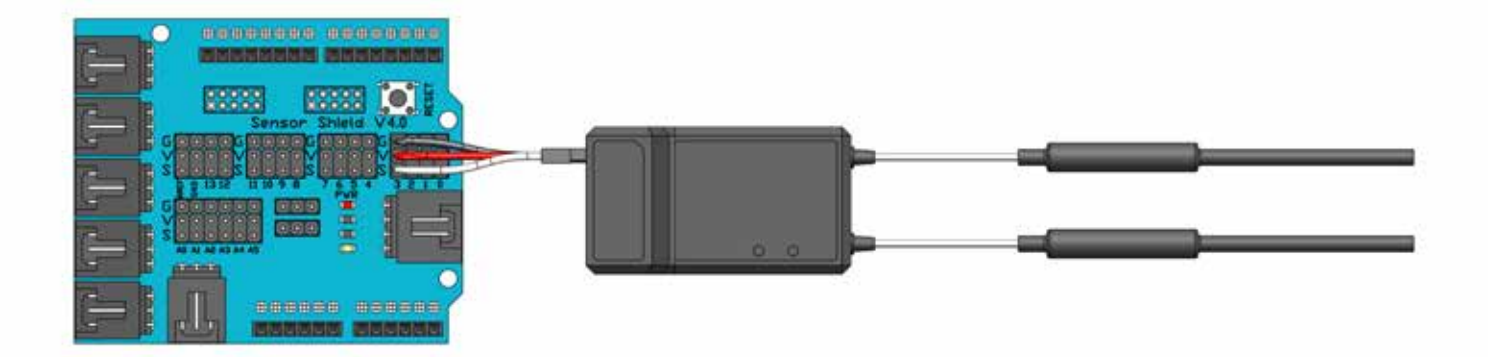
1.2 Troubleshoot

A. If nothing happens
   1. Make sure both batteries are charged
   2. Make sure the ESC is on if it has a switch
   3. Check that all connections are secure.

B. Robot makes some noise but does not drive correctly.
   1. Check that the wiring is correct
   2. Disconnect the battery. Upload the example program: File > Examples > MINDSi > 2-Applications > SimpleRadioDrive to the Arduino and try again.

Still Trouble? See the Troubleshooting guide section.

Fig 11. 1: Wiring Diagram for the Radio Transmitter



Push Button

Push Buttons allow physical interactions to be detected by a micro-controller. Most commonly they are used for human interactions and interfaces. They can also be used as limit switches to detect travel of an object. Push Buttons come in a couple common configurations, normally open, or normally closed.

Example Code: Button
In the Arduino Program: File > Examples > 02.Digital > Button

The Code: Button

In this example a push button will be used to control a LED. To begin with, initialize two "const int" variables: "buttonPin" and "ledPin". These variables will store the Arduino pin numbers to which the button and the led will be connected. The variables are "const" since they will not change.

```
const int buttonPin = 2;
const int ledPin = 13;
```

A variable that will later store the state of the push button is defined for later use:

```
int buttonState=0;
```

In the "void setup()" function the pin connected to the LED will be initialized as an output pin and the pin connected to the push button will be initialized as an input pin. This is done with the lines "pinMode(ledPin, OUTPUT);" and "pinMode(buttonPin, INPUT);".

```
void setup()
{
pinMode(ledPin, OUTPUT);
pinMode(buttonPin, INPUT);
}
```

For this code to work, an LED must be connected to pin 13 on the Arduino and the push button switch to pin 2 on the Arduino.

In the "void loop()" function the state of the push button switch will be read. Since the button has only two states "closed" or "open" it is a digital input. The button state is read with the line "buttonState = digitalRead(buttonPin);" and stored in the variable "buttonState".

```
void loop()
{
      buttonState = digitalRead(buttonPin);
```

The next lines switch the LED based on the button state. The lines "digitalWrite(ledPin, HIGH);" and "digitalWrite(ledPin, LOW);" switch the LED. If the signal wire coming from the button is HIGH (usually +5 volts) then the button's state is "HIGH" and the signal pin connected to the LED is set to "HIGH". Otherwise the signal pin connected the LED is set to the ground (0) voltage.

The full "void loop()" function is:

```
void loop()
{
      buttonState = digitalRead(buttonPin);

      if(buttonState == HIGH){
          digitalWrite(ledPin, HIGH);
      } else {
          digitalWrite(ledPin, LOW);
      }
}
```

Depending on the configuration of the button the button my go either "HIGH" or "LOW" when it is pressed. Whether the LED turns on or off when it is set to "HIGH" or "LOW" also depends on the LED. The LED will either switch on from off when the button is pressed or off from on.

To run the code, upload it: and while leaving the Arduino plugged into the computer, test by pressing the button.

Checking the Button State:

Since buttons can switch to either "HIGH" or "LOW" when pressed, it is useful to see the output from "digitalRead(buttonPin)" displayed on the serial monitor. To set this up, add the line: "Serial.begin(9600);" to the "void setup()" function:

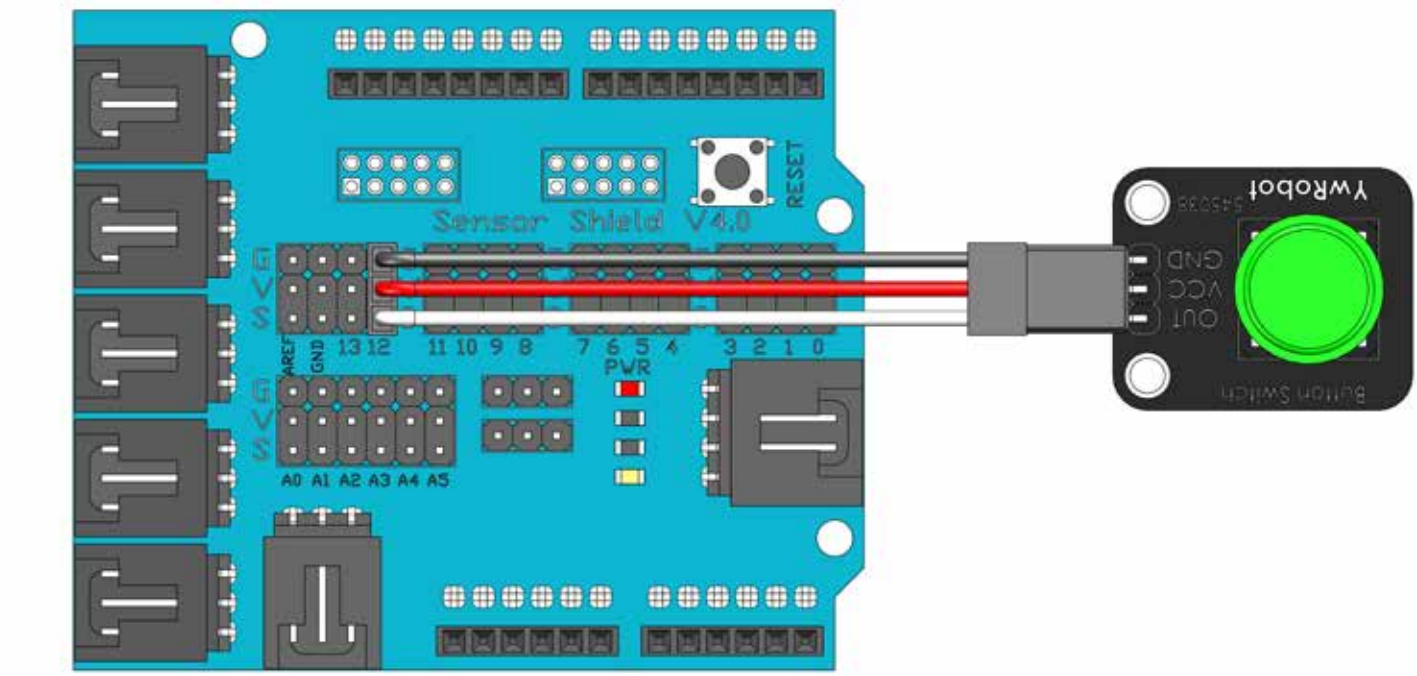
```
void setup()
{
      Serial.begin(9600);

pinMode(ledPin, OUTPUT);
pinMode(buttonPin, INPUT);
}
```

Modify the “void loop()” function by adding the lines “Serial.println(“high”);” and “Serial.println(“low”);” to the “if-else” statement. The final function will be:

```
void loop()
{
       buttonState = digitalRead(buttonPin);

       if(buttonState == HIGH){
            Serial.println(“high”);
            digitalWrite(ledPin, HIGH);
       } else {
            Serial.println(“low”);
            digitalWrite(ledPin, LOW);
       }
}
```

To run the code, upload it: Leave the Arduino plugged into the computer.
Open the serial monitor:
…and test by pressing the button.

Figure 12.1: Wiring Diagram for the Push Button



Ultrasound Sensor

Parallax's PING)))™ ultrasonic sensor is an easy method of distance measurement and autonomous obstacle detection and avoidance. The Ping sensor measures distance using sonar; an ultrasonic (well above human hearing) pulse is transmitted from the unit and distance-to-target is determined by measuring the time required for the echo's return. The Ping sensor provides precise, non-contact distance measurements within a 2 cm to 3 m range. The ping sensor can be connected to the Arduino using a servo extension cable. The LED indicator flashes when the ping sensor is active.

> **Example Code:**
> In the Arduino Program: File > Examples > MINDSi > 1-Calibration > PingSensor

Reading the Ping Sensor

This chapter will cover how to write simple test program that reads the value returned from the ping sensor and displays it in the serial monitor.

Begin a new sketch by including the "MINDSi" and "Servo" libraries.

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

In the "void setup()" function initiate a serial connection that will later send data to the computer.

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

A function (command) from the "MINDSi" library, "getPing(..)", is used to retrieve the output of the ping sensor. In this case, the ping sensor will be plugged into pin 10 on the Arduino; therefor, the command that retrieves the value from the ping sensor on pin 10 is: "getPing(10)". The value retrieved by "getPing(10)" is printed to the serial monitor. The line: "Serial.println(getPing(10));" retrieves the value from the ping sensor and prints it.

```
void loop()
{
      Serial.println( getPing(10) );
}
```

The ping sensor requires a slight delay between readings. The line "delay(50);" is added to pause the program for 50 milliseconds before the next reading from the ping sensor is taken.

The complete "void loop()" function should look like:

```
void loop()
{
      Serial.println( getPing(10) );

      delay(50);
}
```

Plug the Arduino into the computer via the USB cable and upload the code.

After the code finishes uploading, open the serial monitor and view the output. There should be a stream of numbers on the screen. If there is no output, check that the serial monitor is set to 9600 in the bottom left of the window.

The numbers returned by "getPing(..)" indicate roughly how long in microseconds it took for the echo to return to the sensor. As a reference, it takes sound about 900 microseconds to travel one foot.

14.1 Challenge 1

Set the sensor above a level surface so that the path in front of it is not obstructed. The sensor should display a fairly high value (>5000) in the serial monitor. Then place a solid object in front of the sensor. Note how the value changes. Using a yardstick measure the distance of the object from the sensor. Make a graph of the values returned by the ping sensor for different object distances.

Each sensor may read a little differently so make sure you calibrate each sensor individually using this process.

Example Code:
In the Arduino Program: File > Examples > MINDSi > 1-Applications > PingSensor

Obstacle Detection with the Ping Sensor

This chapter will cover programming a simple robot to avoid obstacles. The ping sensor will be used to stop the robot before it drives into an obstacle. The robot should be configured such that it has one drive motor attached to an ESC and one steering servo. The ESC should be wired to pin 4 on the Arduino and the steering servo to pin 5. The ping sensor should be mounted on the front of the robot facing outward and wired to pin 10 on the Arduino. The steps to wire and mount the ping sensor are covered in the "Arduino Autonomous Upgrade Module" document.

Begin a new sketch, and include the "MINDSi" and "Servo" libraries.

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

Define two "Servo" objects to control the drive motor and the steering servo. These will be names "drive" and "steer" respectively. Add the following line to the program, underneath the include statements:

```
Servo drive, steer;
```

In the "void setup()" function attach the "drive" object which controls the ESC/motor to pin 4; and the "steer" object which controls the steering servo to pin 5. Add the lines: "drive.attach(4);" and "steer.attach(5);":

```
void setup()
{
      drive.attach(4);
      steer.attach(5);
```

Initialize both the servo and the ESC to 90. This should bring the steering servo close to its centered value and set 90 as the neutral value for the ESC. Add the lines: "drive.write(90);" and "steer.write(90);" to the "void setup()" function. Also add a 2 second delay to allow the ESC to arm: "delay(2000);".

The "void setup()" function should now be:

```
void setup()
{
      drive.attach(4);
      steer.attach(5);

      drive.write(90);
      steer.write(90);

      delay(2000);
```

Finally turn the front wheels. This will make the robot drive in circles until it stops. Add the line "steer.write(120);". The complete "void setup()" function will be:

```
void setup()
{
	drive.attach(4);
	steer.attach(5);

	drive.write(90);
	steer.write(90);

	delay(2000);

	steer.write(120);
}
```

The code in the "void loop()" function will run the robot forward (in a circle) unless the ping sensor indicates an obstruction. This conditional response to the ping sensor value is implemented with an if/else statement. The "if" statement checks if the value returned by the ping sensor is less than 1500. Since the value of the ping sensor decreases the closer the obstruction is to the sensor, the robot is brought to stop if it is too close to an obstruction otherwise the robot continues to drive forward in circles.

Add the line "if(getPing(10) < 1500)" to the "void loop()" function. This is the beginning if the "if" statement. The command, or function: "getPing(10)" returns the value of the ping sensor connected to pin 10.

```
void loop()
{
	if(getPing(10) < 1500 )
```

If the value returned by "getPing(10)" is less than 1500, then section following the "if" statement executes. The robot should stop if the ping value is too low. The command: "drive.write(90);" brings the robot to a stop. Add this command to the section following the "if" statement. The result will be:

```
void loop()
{
	if(getPing(10) < 1500 )
	{
		drive.write(90);
	}
```

An "else" statement can be used to create a section of code that executes if the "if" condition is not true. Add an "else" statement to the end of the "if" section:

```
void loop()
{
	if(getPing(10) < 1500 )
	{
		drive.write(90);
	}
	else
```

If there is nothing close to the sensor (the value is greater than 1500) then the robot should continue to drive forward. Since the "else" section will execute when "getPing(10)" returns a value greater than or equal to 1500, the command to drive forward can be put in this section.

The command "drive.write(100);" drives the robot forward slowly. Add the "else" section with this command in it:

```
void loop()
{
	if(getPing(10) < 1500 )
	{
		drive.write(90);
	}
	else
	{
		drive.write(100);
	}
```

As before a delay of 50 milliseconds is added to allow the ping sensors to recover. And the final "void loop()" function is:

```
void loop()
{
	if(getPing(10) < 1500 )
	{
		drive.write(90);
	}
	else
	{
		drive.write(100);
	}

	delay(50);
}
```

Check that the wiring matches the wiring diagram in Figure 14.1.1 and connect the Arduino to the computer with the USB cable. DISCONNECT the vehicles battery. Position the vehicle such that its wheels DO NOT touch the table or other surface. Upload the code to the Arduino: Once the upload is completed, reconnect the battery and set the robot on the floor. The robot should start driving in circles, stopping when something obstructs the ping sensor.

Troubleshooting

Upload the example sketch: File > Examples > MINDSi > 1-Applications > PingSensor to the robot. and try again. If the robot still does not work…

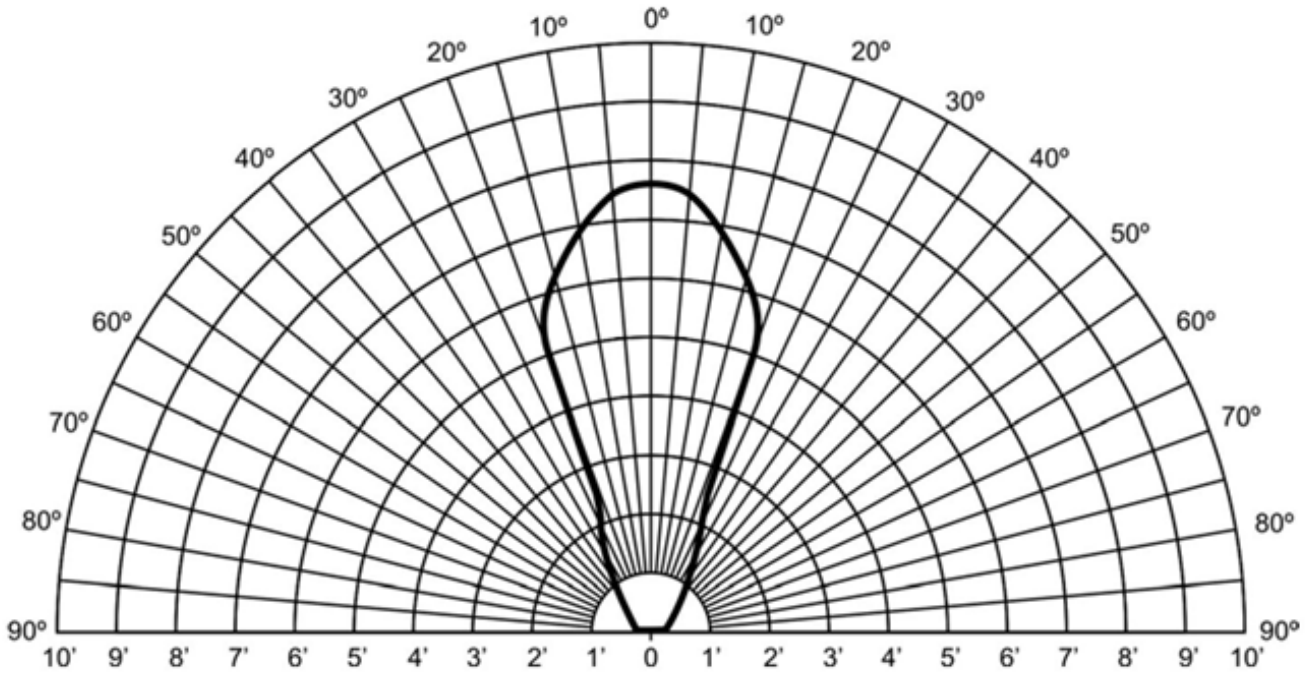
A. The robot does not make a tone
   1. Check that the battery is charged

B. The robot makes a tone, but the wheels don't turn
   1. Check the wiring

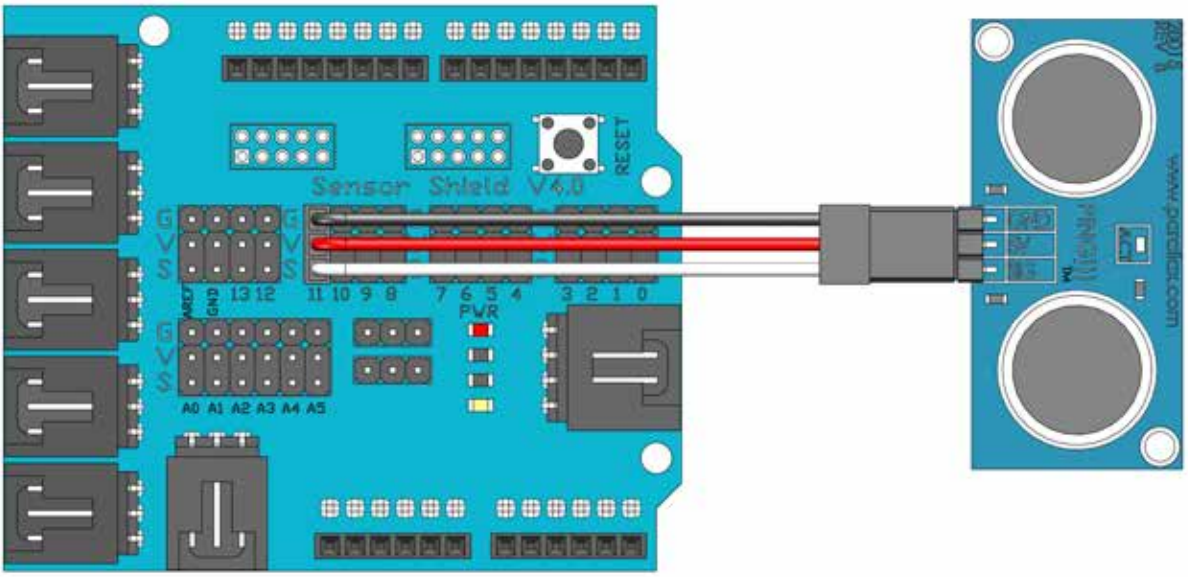
C. The robot makes a tone, and turns the wheels, but does not move.
   1. Open the example sketch. Make a copy. Modify the copy to change the speed from 100 to 110 by changing the line "drive.write(100);" in the "else" brackets. Try again.
   2. If the robot still does not move, change the speed to 120.

Still Trouble? See the troubleshooting guide at the end of this booklet.

Figure 1.1: Ping Sensor Range



Wiring Diagram for the Ping Ultrasound Sensor

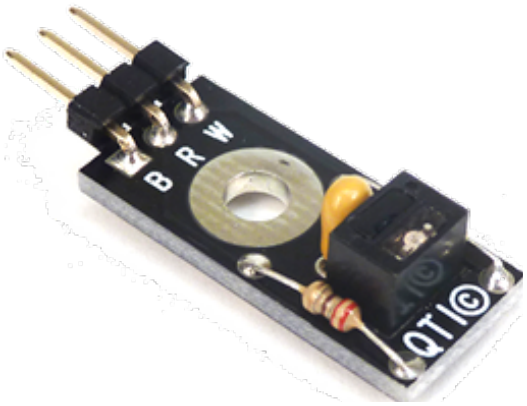


1.2 Challenge: Ultrasound Object Detection

Experiment with stopping the robot at different ranges from the nearest obstruction. To do this change the ping threshold value that stops the robot. This value is the “1500” in the line: “if(getPing(10) < 1500)”. Change “1500” to “1000” or “500” for instance.

Experiment with the robot’s speed. Change the value sent to ESC in the line: “drive. write(100);” in the “else” section. Change “100” to “110” or “120” for instance.

If the robot starts to hit obstacles, modify the code to avoid this by changing either the speed or the ping threshold value.



QTI Sensor

The QTI sensor is an infrared emitter/receiver that is able to differentiate between a dark surface (with low IR reflectivity) and a light surface (with high IR reflectivity). These sensors are used for line following, maze navigation, or sensing the outer rim of a SumoBot ring. We use them as an analog sensor to detect different shades of gray. A daylight filter is built into the sensor.

> **Example Code:**
> In the Arduino Program: File > Examples > MINDSi > 1-Calibration > QTI

Reading the QTI Sensor

This chapter will cover how to write a simple program that reads the value returned by the QTI sensor and displays it in the serial monitor.

Begin a new sketch by including the "MINDSi" and "Servo" libraries.

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

Define an integer "int" called "val". This will be used later.

```
int val;
```

In the "void setup()" function, initiate a serial connection that will later send data to the computer.

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

The QTI sensor sends an Analog output; therefor it should be plugged into one of the Analog ports on the Arduino. In this example the sensor will be plugged into the Analog pin "A0". It could have been plugged into any of the pins from "A0" to "A5".

The "MINDSi" library function, "QTI(..)", will be used to interface with the QTI sensor. The "QTI" function returns the value from the QTI sensor connected to the Analog pin passed to it. The line: "val = QTI(A0);" reads the value from the QTI sensor attached to pin A0 and puts in "val". Add this line to the "void loop()" function:

```
void loop()
{
      val = QTI(A0);
```

Then add the line "Serial.println(val);" to print the value returned by the QTI sensor to the serial monitor. The complete "void loop()" function will be:

```
void loop()
{
      val = QTI(A0);

      Serial.println(val);
}
```

Connect the Arduino to the computer via the USB cable and upload the code.

Once the code finishes uploading, open the serial monitor. The scrolling number on screen should indicate the values currently being read in by the QTI sensor.

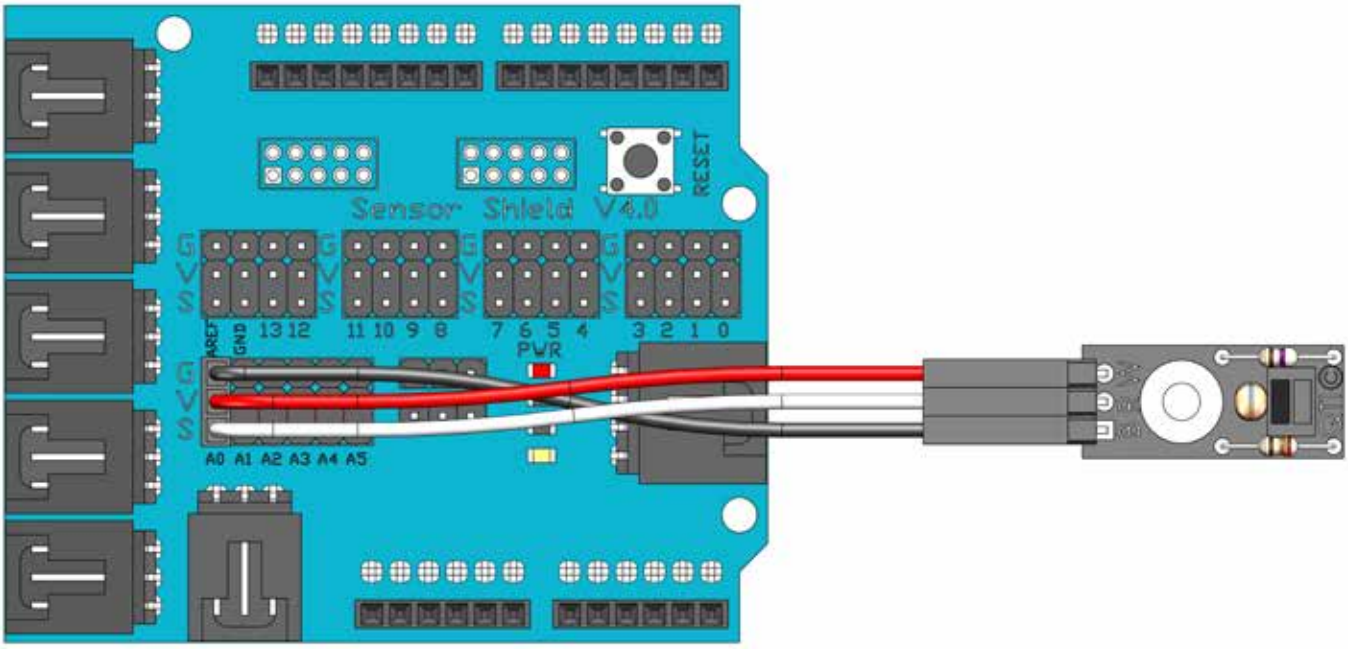
Challenge 15.1.1: Calibrate Your QTI Sensor

Run the code as discussed above. With the serial monitor open, pass the QTI sensor over the "gray scale": Figure 15.1.1. The value returned by the QTI sensor should vary as it is passed over lighter and darker surfaces. Make a note of the values returned for white and likewise the values returned for black.

Hint: You will need to keep the sensor about half an inch away from the paper to get a consistent reading. Mount it on a stand or chassis to keep it at a constant distance from the paper.

Figure 15.1.1: Gray scale

Wiring Diagram for the QTI Sensor



Analog Sensors

Analog Sensors return a varied voltage usually from 0 to 5 volts (sometimes 0 to 3 volts). The voltage returned by the sensor is directly proportional to the resistance in the sensor itself. The main advantage is the fine definition of the analog signal, which has the potential for an infinite amount of signal resolution. Most Micro-controllers measure analog signals as a number between 0 and 1024.

> **Example Code:**
> In the Arduino Program: File > Examples > MINDSi > 1-Calibration > AnalogSensor

Test Code
This chapter will cover reading an analog sensor and displaying its output in the serial monitor.

Begin a new sketch by including the "MINDSi" and "Servo" libraries.

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

Define an integer "int" called "val". This will be used later.

```
int val;
```

In the "void setup()" function start a serial connect at 9600 baud (9600 bit/second) that will be used later to communicate with the computer.

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

An analog sensor can be plugged into any of the pins A0 through A5 on the Arduino. In this case, plug the sensor into pin A0. In the "void loop()" function add the line "val = analogRead(A0);" to read the value from the analog sensor plugged into pin A0. The value read is stored in the variable "val" defined previously.

```
void loop()
{
      val = analogRead(A0);
```

Next add the line "Serial.println(val);". This line will later print the value, "val", to the serial monitor on the computer.

```
void loop()
{
      val = analogRead(A0);
      Serial.println(val);
}
```

Connect the Arduino to the computer via the USB cable and upload the code.

Once the code finishes uploading, open the serial monitor. The scrolling number on screen should indicate the values currently being read in by the analog sensor.

Digital Sensors

A digital sensor is an electronic or electrochemical sensor, where data conversion and data transmission are done digitally. Digital sensors use a varied on / off pulse to send information back to a Micro-controller. The time on vs. the time off is measured and corresponds to the reading taken.

> **Example Code:**
> In the Arduino Program: File > Examples > MINDSi > 1-Calibration > DigitalSensor

Test Code

This chapter will cover reading a digital sensor and displaying its output in the serial monitor.

Begin a new sketch by including the "MINDSi" and "Servo" libraries.

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

Define an integer "int" called "val". This will be used later.

```
int val;
```

In the "void setup()" function start a serial connect at 9600 baud (9600 bit/second) that will be used later to communicate with the computer.

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

A digital sensor can be plugged into any of the pins 0 through 13 or A0 through A5 on the Arduino. In this case, plug the sensor into pin 13. In the "void loop()" function add the line "val = digitalRead(13);" to read the value from the digital sensor plugged into pin 13. The value read is stored in the variable "val" defined previously.

```
void loop()
{
      val = digitalRead(13);
```

Next add the line "Serial.println(val);". This line will later print the value, "val", to the serial monitor on the computer.

```
void loop()
{
      val = digitalRead(13);
      Serial.println(val);
}
```

Connect the Arduino to the computer via the USB cable and upload the code.

Once the code finishes uploading, open the serial monitor. The scrolling number on screen should indicate the values currently being read in by the digital sensor.

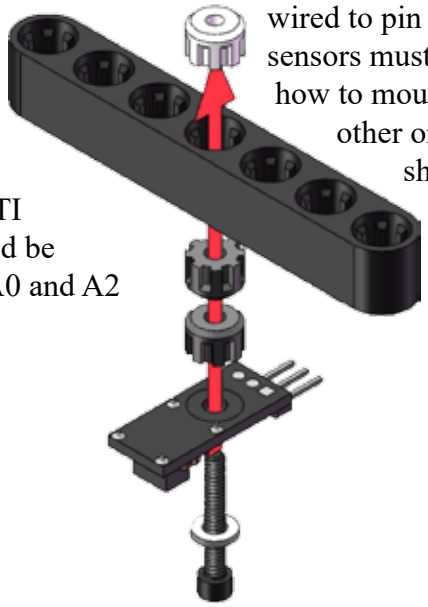
4x4 Line Follower

Autonomous line following can be added to a 4X4 robot with the QTI IR Line Following sensors.Utilizing three downward facing QTI sensors to detect differences in the shade of light bouncing off a surface allows the robot to follow lines.

> **Example Code:**
> In the Arduino Program: File > Examples > MINDSi > 1-Applications > QTI

Mounting the QTI Sensors

This chapter uses the calibration value for white found in: “Challenge 15.1.1: Calibrate your QTI Sensor”. This chapter will cover programming a simple robot to follow a line. The QTI sensors will be used to keep the robot positioned on the line. The robot should be configured with one drive motor attached to an ESC and one steering servo. The ESC should be wired to pin 4 on the Arduino and the steering servo to pin 5. To run this program three QTI sensors must be used mounted to the front bottom of the robot. The diagram below shows how to mount a QTI sensor on a beam. Three sensors should be mounted next to each other on the same beam leaving a one hole gap between the sensor mounts. The beam should be placed in the front center of the robot such that the sensing side of the QTI sensors have about a half inch clearance above the ground. The middle QTI should be plugged into A1 on the Arduino. The left and right QTI’s should be plugged into A0 and A2 respectively.



The Code

To begin with the code, open a new sketch and add the libraries:

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

Define two “Servo” objects to control the drive motor and the steering servo. These will be names “drive” and “steer” respectively. Add the following line to the program, underneath the include statements:

```
Servo steer, drive;
```

Define an “int” variable “threshold” and set it to 300. Since this variable will not change during the program it is defined as a “const”. Add the following:

```
const int threshold = 300;
```

In the “void setup()” function attach the “drive” object which controls the ESC/motor to pin 4; and the “steer” object which controls the steering servo to pin 5. Add the lines: “drive.attach(4);” and “steer.attach(5);”:

```
void setup()
{
      drive.attach(4);
      steer.attach(5);
```

Initialize both the servo and the ESC to 90. This should bring the steering servo close to its centered value and set 90 as the neutral value for the ESC. Add the lines: “drive.write(90);” and “steer.write(90);” to the “void setup()” function. Also add a 2 second delay to allow the ESC to arm: “delay(2000);”. The “void setup()” function should now be:

```
void setup()
{
      drive.attach(4);
      steer.attach(5);

      drive.write(90);
      steer.write(90);

      delay(2000);
```

Instruct the robot to start driving forward slowly by adding the line: “drive.write(100);” The complete “void setup()” function should be:

```
void setup()
{
      drive.attach(4);
      steer.attach(5);

      drive.write(90);
      steer.write(90);

      delay(2000);

      drive.write(100);
}
```

In the “void loop()” function the robot’s steering will be controlled. The robot will be programed to drive forward if the value beneath the center QTI is black; to drive left if the value under the left QTI is black and right if the value under the right QTI is black. In this way every time the robot drives off the line it will steer to drive back on to the line. To execute this a series of conditional “if/else” statements will be used.

The first "if" statement checks if the value returned by the left QTI sensor is less than the threshold value. Values less than the threshold value are assumed to indicate that the QTI sensor is above the black line. Add the line "if(QTI(A0) < threshold)" to the "void loop()" function:

```
void loop()
{
      if(QTI(A0) < threshold)
```

Then add the code that is executed if the value returned by the A0 QTI sensor is below the threshold. If this is true then the robot should steer left. The line "steer.write(45);" turns the robot to the left. Add this line and the brackets to the "void loop()" function:

```
void loop()
{
      if(QTI(A0) < threshold)
      {
            steer.write(45);
      }
```

If the black line is not under the left sensor, then the code will check if the line is under the right(A2) sensor. The "else if" condition executes the code in its section if the previous if statement is false AND the new "if" condition is true. The new if condition is "QTI(A2) < threshold".

Add the line: "else if (QTI(A2) < threshold)".

```
void loop()
{
      if(QTI(A0) < threshold)
      {
            steer.write(45);
      }
      else if(QTI(A2) < threshold)
```

The section under this "else if" statement will execute if "QTI(A0)" is greater than or equal to the threshold and "QTI(A2)" is less than the threshold.

If the line is under the right sensor and not under the left sensor, the robot should turn right. The line "steer.write(135);" turns the robot to the right. Add this line to the section under the last "else if" statement:

```
void loop()
{
      if(QTI(A0) < threshold)
      {
            steer.write(45);
      }
      else if(QTI(A2) < threshold)
      {
            steer.write(135);
      }
```

Finally, if the line is not under the right or the left QTI sensors the code checks if it is under the center(A1) QTI sensor. If it is then the robot centers the steering. The line: "else if(QTI(A1) < threshold)" checks if the line is under the center sensor. Add this line to the code and add the line to center the steering: "steer.write(90);" to the bracketed section under the "else if" statement.

The final "void loop()" function should be:

```
void loop()
{
	if(QTI(A0) < threshold)
	{
		steer.write(45);
	}
	else if(QTI(A2) < threshold)
	{
		steer.write(135);
	}
	else if(QTI(A1) < threshold)
	{
		steer.write(90);
	}
}
```

If the line is not found the robot will just continue driving with its wheels turned to the last position in which they were set.

Before uploading the code reset the "threshold" value. Check which values the QTI sensor read for the white and black extremes of Figure 15.1.1 a.k.a. the answers to "Challenge 15.1.1: Calibrate your QTI Sensor". Select a value halfway between these two numbers. This will be the new threshold value. Reset "const int threshold"; change the value of 300 in the line "const int threshold = 300;" to whatever new threshold value is.

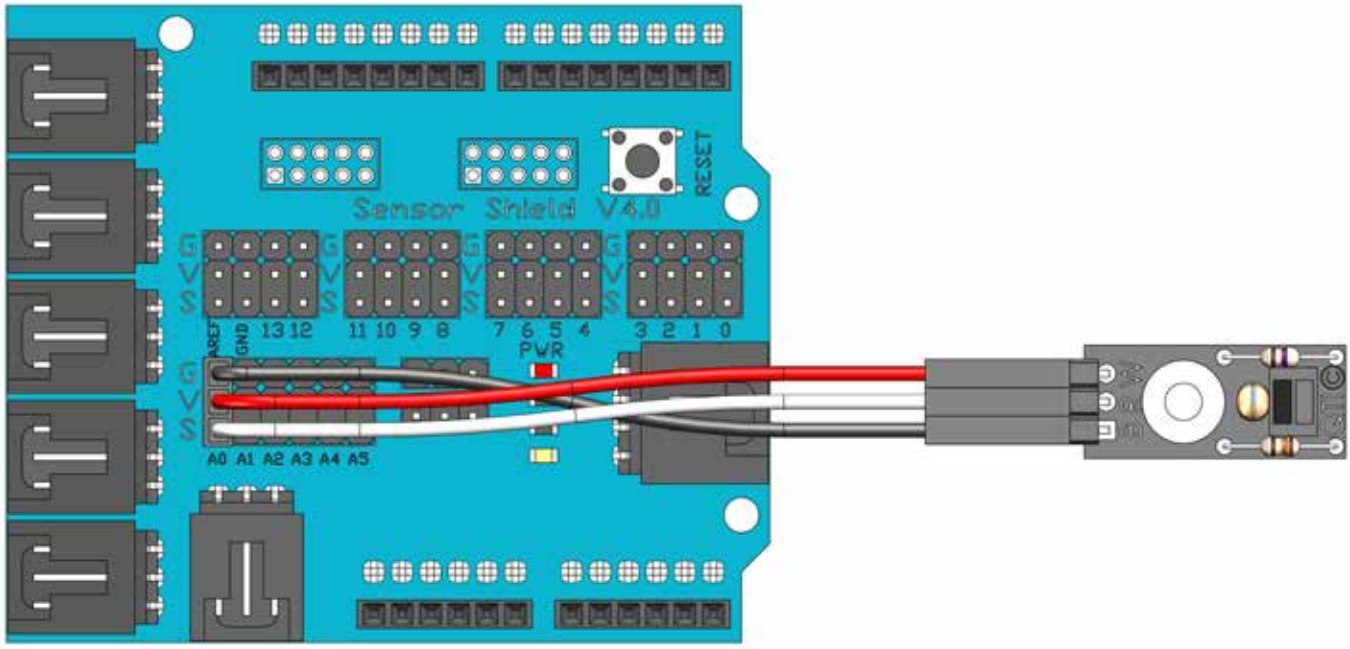
DISCONNECT the vehicles battery. Position the vehicle such that its wheels DO NOT touch the table or other surface and connect the Arduino to the computer with the USB cable. Prepare a surface with a black line down the middle. Preferable the line should be around a half inch thick, or slightly more.

Upload the code to the Arduino:

Once the upload is completed, reconnect the battery and set the robot on the line with the QTI sensor centered over it. The robot should start driving along the line. If the robot drives off the line, try reversing the connections for the "A0" and "A2" QTI sensors. Depending on the orientation of the steering servo, right and left could be reversed.

Challenge 15.1.2: Follow a Line
Test how well the robot follows lines at different speeds. To adjust the robot's speed change the line "drive.write(100);" in the "void setup()" function. Try changing "100" to "110" or "120" for instance. If the robot is leaving the line it will be necessary to slow down.
Mechanical Hint: Generally, the darker the line is, or the greater the contrast between the line and the background, the better the QTI sensor is at detecting the line. The value the sensor gets also changes depending on how fast the rover is moving.

Wiring Diagram for the QTI Sensor



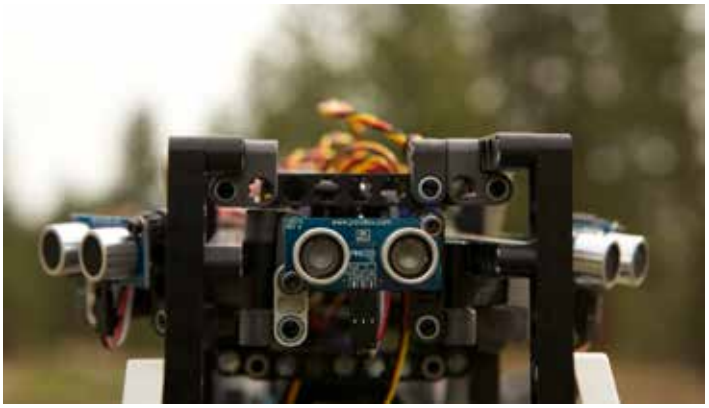
Troubleshooting

Upload the example sketch: File > Examples > MINDSi > 1-Applications > QTI to the robot. and try again. If the robot still does not work…

A. The robot does not make a tone
   1. Check that the battery is charged

B. The robot makes a tone, but doesn't move
   1. Check the battery charge
   2. Check the wiring
   3. Open the example sketch. Make a copy. Modify the copy to change the speed from 100 to 110 by changing the line "drive.write(100);" in the "else" brackets. Try again.
   4. If the robot still does not move, change the speed to 120.

Still Trouble? See the troubleshooting guide at the end of this booklet.



3 Ping Obstacle Avoidance

4x4 and 6x6

Add autonomous obstacle avoidance to your MINDS- i creation with the AUV Module (Autonomous Ultrasound Vehicle). Utilizing three forward facing Ultrasound sensors to detect objects as they approach it will steer away from or stop and back away to avoid a collision.

> **Example Code:**
> In the Arduino Program: File > Examples > MINDSi > 3-Projects > AUV

Programming Collison Avoidance

In this example the robot will be programed to steer around obstacles.

Begin a new sketch by including the "MINDSi" and "Servo" libraries.

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

Define a series of constants that will be used later:

```
const bool IR_ENABLED     = false;
const int  CENTER         = 90;
const int  TURN           = 45;
const int  FWDSPEED       = 115;
const int  REVSPEED       = 70;
const int  BACKUP_TIME    = 2000;
const int  SIDE_RANGE     = 4000;
const int  HAZARD_DIST[]  = {   800,   3000,   800};
```

Create the "Servo" objects "drive", "frontsteer" and "backsteer". "drive" will be used to control the ESC/motor. "frontsteer" and "backsteer" will be used to control the front and back steering servos respectively.

```
Servo drive, fronsteer, backsteer;
```

In the "void setup()" function setup the digital pins 12 & 13 for input:

```
void setup()
{
	pinMode(13, INPUT);
	pinMode(12, INPUT);
```

The ESC/motor should be connected to pin 4, the front steering servo to pin 5 and the back steering servo to pin 6. In the code, attach the "drive" object to pin 4, the "frontsteer" object to pin 5, and the "backsteer" object to pin 6. In the "void setup()" function add the lines:

```
drive.attach(4);
	frontsteer.attach(5);
	backsteer.attach(6);
```

Add some lines to set the ESC and the servos to their initial values. The "steer(...)" function will be defined later.

```
steer(CENTER);
	drive.write(90);
```

Finally add the line "delay(2000);" to give the ESC time to arm. The final "void setup()" function will be:

```
void setup()
{
	pinMode(13, INPUT);
	pinMode(12, INPUT);

	drive.attach(4);
	frontsteer.attach(5);
	backsteer.attach(6);

	steer(CENTER);
	drive.write(90);
	delay(2000);
}
```

Now create a new function. This function will be called "steer". It will not return a value so it is prefixed by the term "void". This function will accept an integer, "int", value called "out".

Begin the function definition with the line "void steer(int out)" followed by a bracket "{".

```
void steer(int out)
{
```

The function "void steer(int out)" will take whatever value "out" is passed to it and write this value to the "frontsteer" "Servo" object. This will cause the front servo to turn to the corresponding value. This function will also write the value of "180 – out" to the "backsteer" object. This causes the back servo to turn in the opposite direction from the front servo. Fill in the body of the "void steer(int out)" function with the lines "frontsteer.write(out);" and "backsteer.write(180 – out);". The complete function will be:

```
void steer(int out)
{
      frontsteer.write(out);
      backsteer.write(180 - out);
}
```

This is the function called by the line "steer(CENTER)" in the "void setup()" function. The line "steer(CENTER)" centers the front and back servos using the "void steer(int out)" function. The next step is to code the "void loop()" function. At the beginning of this function the values from the right, left, and front/center ping sensors. After each ping value is retrieved a delay of 10 milliseconds, "delay(10);" is programed. Add the following lines to the "void loop()" function:

```
void loop()
{
      int right = getPing(9);
      delay(10);
      int left = getPing(11);
      delay(10);
      int front = getPing(10);
```

The right ping sensor should be plugged into pin 9 on the Arduino; the left into pin 11 and the center into pin 10.

The next part of the code checks if any object is close enough to the ping sensors to be considered an obstruction, or "hazard". "HAZARD_DIST" is an array. Array elements are retrieved using square brackets: "[]". "HAZARD_DIST[0]" is 800; "HAZARD_DIST[1]" is 3000; and "HAZARD_DIST[2]" is 800. The instructions: "right < HAZARD_DIST[0]", "front < HAZARD_DIST[1]", and "left < HAZARD_DIST[2]", therefor check if the right ping value is less than 800; the front ping value is less than 3000; and the left ping value is less than 300. The notation "||" means "or". If any of the ping values are less than their corresponding "HAZARD_DIST", then the line: "left < HAZARD_DIST[0] || front < HAZARD_DIST[1] || right < HAZARD_DIST[2]" returns true.

Add the following lines to the "void loop()" function under the previous lines:

```
if( left < HAZARD_DIST[0] ||
          front < HAZARD_DIST[1] ||
          right < HAZARD_DIST[2] )
      {
```

If a hazard has been encountered, that is if one of the ping sensors returns a value less than its "hazard" distance, the condition inside the brackets "{}" of the above "if" statement will be executed.

The first response to encountering the hazard is to stop. This is accomplished by the line "drive.write(90);". The line after this one "delay(2000)" gives the vehicle two seconds to come to a stop. Add these lines in the body of the "if" statement:

```
if( left < HAZARD_DIST[0] ||
          front < HAZARD_DIST[1] ||
          right < HAZARD_DIST[2] )
      {
            drive.write(90);
            delay(2000);
```

The next step is to back up. The robot will first turn its wheels so that when it drives in REVERSE it turns away from the obstacle. Then the ESC must be setup to send the motor into reverse. Finally the robot should drive in reverse for a time interval "BACKUP_TIME".

Add these lines to the "if" statement below "delay(2000);":

```
            if(left > right){
                  steer(CENTER +TURN);
            } else {
                  steer(CENTER - TURN);
}
```

These lines instruct the robot to turn its wheels in preparation for driving in reverse. If the left ping "hazard" distance is greater than the right ping "hazard" distance, then the "hazard" is more towards the right side of the robot. If the "hazard" is towards the RIGHT side, then the robot must turn its front wheels RIGHT so that when it drives in reverse it turns away from the "hazard". Likewise, if the "hazard" is towards the LEFT then the robot must turn LEFT to turn away from it when it drives in reverse. If the "hazard" is in the center it doesn't matter which way the robot turns. "steer(CENTER+TURN)" turns the robot's servos such that it is configured to drive to the right; likewise, "steer(CENTER-TURN)" turns the robot's servos such that it is configured to drive to the left.

The next lines prepare the ESC for the motor to be run in reverse. Add these lines also to the body of the "if" statement:

```
drive.write(85);
            delay(50);
            drive.write(90);
            delay(50);
```

Add the line that tells the robot to go in reverse:

```
drive.write(REVSPEED);
```

The next line: "uint32_t endTime = millis() + BACKUP_TIME;", defines a an integer variable "endTime" which holds a time a duration "BACKUP_TIME" in milliseconds into the future. "millis()" is a function that retrieves the current times. Therefore "millis() + BACKUP_TIME" is a time "BACKUP_TIME" or 2000 milliseconds into the future.

Add this line to the code underneath the rest:

```
uint32_t endTime = millis() + BACKUP_TIME;
```

The "while" loop: "while(millis() < endTime){ ... }", causes the code to loop over the contents of the "while" loop until the current time matches or exceeds the "endTime". The line "millis() < endTime" causes the current time to be retrieved and compared against "endTime". Therefore the "while" loop here delays the code for an interval of 2000 milliseconds as set by "BACKUP_TIME". During these 2000 milliseconds the robot continues in reverse.

Underneath what has been added before, add the lines:

```
      while(millis() < endTime){
      if( IR_ENABLED && (!digitalRead(13) | !digitalRead(12) )  ) break;
            }
```

Since "IR_ENABLED" has been set to "false" the line "if (IR_ENABLED && (!digitalRead(13) | !digitalRead(12))) break;" does nothing. If there were IR sensors enabled on this robot this line would stop the robot if one of the sensors detected an obstacle.
On robots without IR sensors, the line: "uint32_t endTime = millis() + BACKUP_TIME;", and the following "while" loop: "while(millis() < endTime) { .... }", could be replaced by a delay statement: "delay(BACKUP_TIME);".

When the robot finishes backing up for two seconds it is set to coast to a stop with its wheels centered. The line: "drive.write(90)", tells the ESC to stop the motor. The line: "steer(CENTER)", causes the robot to center its wheels. The line: "delay(1000)", allows the robot time to stop. Add these lines to the "if" statement and close it with a bracket: "}"

```
            drive.write(90);
            steer(CENTER);
            delay(1000);
      }
```

This "if" statement will now be transformed into an "if/else" statement. The first part of this "if/else" statement, the "if" part, covered what the robot does if it is within a "hazard" distance of an obstacle. The second part of this "if/else" statement, the "else" part, will cover what the robot does if no obstacle, or "hazard", is near.

Add the beginning of the "else" section below the bottom of the "if" statement already completed:

```
else {
```

The code in the "else" section will be executed if "(left < HAZARD_DIST[0] || front < HAZARD_DIST[1] || right < HAZARD_DIST[2])" is false. The first lines are:

```
else {
            left = constrain(left, 0, SIDE_RANGE);
            right = constrain(right, 0, SIDE_RANGE);
```

These lines take the values "left" and "right" returned by the ping sensor and constrain them to the range between 0 and "SIDE_RANGE", or 4000. The "constrain" function, in this context, takes the first value it is passed, in this case "left" or "right" and returns this value if it is between 0 and "SIDE_RANGE". Therefore if "left" or "right" is greater than 0 and less than or equal to "SIDE_RANGE", or 4000, these lines do not change its value. If the value for or "right" is less than zero, then these lines set it to zero. If the value for "left" or "right" is greater than "SIDE_RANGE", then "constrain" sets it to "SIDE_RANGE" or 4000. Add the next line:

```
long steerValue = map( (left - right),
    -SIDE_RANGE, SIDE_RANGE,
    -TURN,              TURN);
```

This line takes the difference between the values returned by the "left" and "right" ping sensors. It scales this difference by the value "SIDE_RANGE" and uses the scaled value to determine how far to turn the wheels. The closer the absolute value of "(left – right)" is to "SIDE_RANGE" the closer the "steerValue" is to "TURN". The smaller the absolute value of "(left – right)" is the closer "steerValue" is to zero. If "left" is greater than "right", then the "steerValue" is positive. Otherwise, it is negative.

To turn the wheels add the next line:

```
steer(CENTER - steerValue);
```

If an obstacle is closer to the right side of the robot than the left side, then “right” is smaller than “left”. Therefor “left – right” is positive and “steerValue” is positive. If the function: “steer(...)”, is passed a value less than “CENTER” then the robot turns left. When “steerValue” is positive, “CENTER – steerValue” is less than “CENTER”. Therefor if an obstacle is closer to the right side of the robot the robot will turn left. It can be worked out that when an obstacle is closer to the left side of the robot, the robot turns right.

Add the final line of the “else” block and close the block with a bracket: “}”

```
            drive.write(FWDSPEED);
      }
```

This last line instructs the robot to drive forward at “FWDSPEED”. This is the end of the “void loop()” function. Add the final bracket: “}”.
The full “void loop()” function is:

```
void loop()
{
      int right = getPing(9);
      delay(10);
      int left = getPing(11);
      delay(10);
      int front = getPing(10);
if( left < HAZARD_DIST[0] ||
           front < HAZARD_DIST[1] ||
           right < HAZARD_DIST[2] )
      {
            drive.write(90);
            delay(2000);

if(left > right){
                  steer(CENTER +TURN);
            } else {
                  steer(CENTER - TURN);
}

drive.write(85);
            delay(50);
            drive.write(90);
            delay(50);


drive.write(REVSPEED);
uint32_t endTime = millis() + BACKUP_TIME;
      while(millis() < endTime){
      if( IR_ENABLED && (!digitalRead(13) | !digitalRead(12) )  ) break;
            }

            drive.write(90);
            steer(CENTER);
            delay(1000);
      }
else {
```

```
            left = constrain(left, 0, SIDE_RANGE);
            right = constrain(right, 0, SIDE_RANGE);
long steerValue = map( (left - right),
    -SIDE_RANGE, SIDE_RANGE,
    -TURN,              TURN);

steer(CENTER - steerValue);
            drive.write(FWDSPEED);
      }
```

Completed, the "void loop()" function drives and steers the robot while reacting to obstacles. If an obstacle is found very close, the robot stops and backs up with its wheels turned such that it rotates away from the obstacle. If there is an obstacle close but not too close, the robot drives forward while steering away from it. If there is no obstacle within the predefined range, "SIDE_RANGE", the robot continues to drive straight forwards.

To test the code, DISCONNECT the battery; turn the ESC switch OFF; and place the robot on a stand so that its wheels DO NOT touch the ground. Then connect the Arduino via USB connection to the computer.
Upload the code:

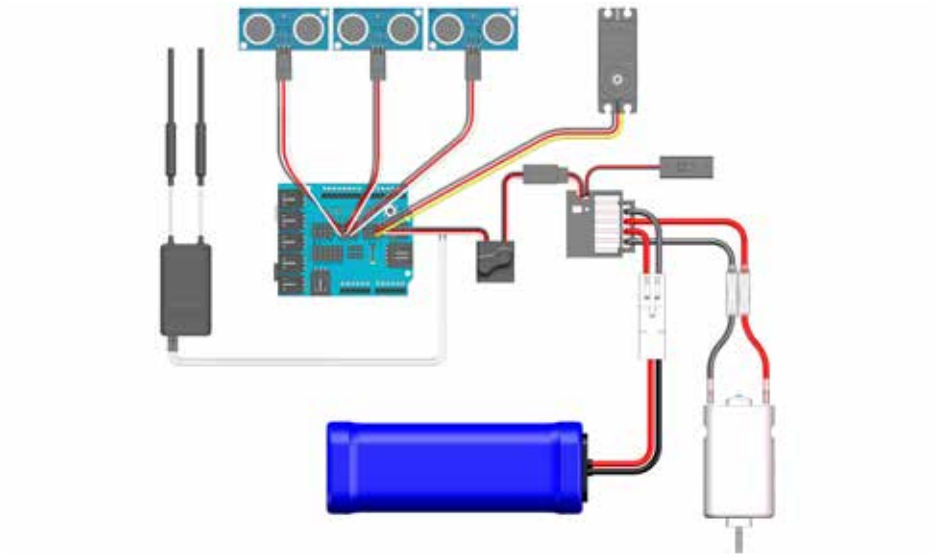
When the upload is completed, DISCONNECT from the computer; place the robot on the floor; turn ON the ESC switch; and reconnect the battery. The robot should drive forward avoiding obstacles that are detectable by its ping sensors. Some obstacles may be either above or below the ping sensors. The robot may hit these obstacles - so watch it.

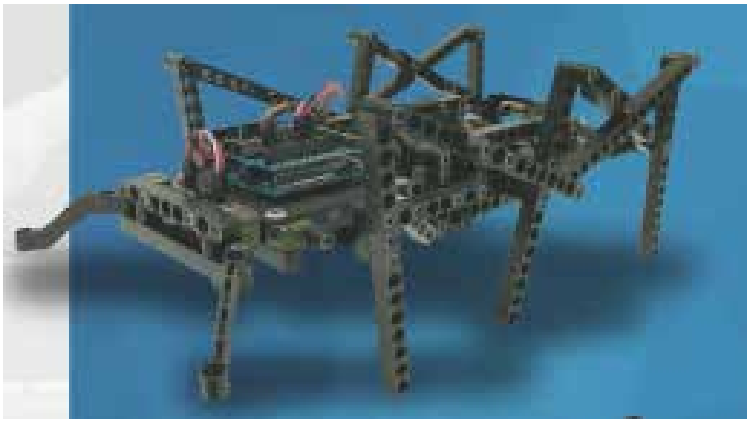
1.1 Challenge:

Try changing how fast the robot drives. Change the value of "FWDSPEED" from 115 to 120. Also try changing some of the other values such as "TURN", "BACKUP_TIME", "SIDE_RANGE" and the values in the array "HAZARD_DIST". See how these changes affect the way the robot drives.

Note: "IR_ENABLED" should be left at "false" and "CENTER" should be left at 90

Wiring Diagram for the AUV Module





HexaBot

The HexaBot utilizes two continuous rotation servos to opperate the cam driven legs for motion. The push buttons get activated by the “antenna” triggering the code to back up and turn away from an obstacle.

Example Code
In the Arduino Program: File > Examples > MINDSi > 3-Projects > HexaBot

Code:

This section will use written code to run the HexaBot. To begin, include the servo and MINDSi libraries:

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

Define our “Servo” objects, “leftServo, rightServo”:

```
Servo leftServo, rightServo;
```

In the “void setup()” function we will tell the arduino that we are attaching “leftServo” to pin 4 and “rightServo” to pin 5 on the Arduino. We will also declare pins 11 and 12 as INPUTs.

```
void setup( )
{
      leftServo.attach(4);
      righServo.attach(5);
      pinMode(12, INPUT);
      pinMode(11, INPUT);
```

The first thing we will do in the “void loop()” function will be to tell the servos to rotate. We will tell the left one to go to the value of 45 and the right to go to 135. This will cause one servo to rotate clockwise and the other to rotate counter clockwise. This is necissary as the servos are mounted as mirrors to eachother causing one to need to be driven the opposite way.

```
void loop()
{
      leftServo.write(45);
      rightServo.write(135);
```

With the servos now driving the HexaBot forward we will need to set two if statements up for when the antenna are activated. The first if statement will start by looking at the push button plugged into pin 12. If that value becomes not true (signified by the ! in front of digitalRead (12)) the code will run through the process of stopping, backing up for two seconds and then rotating to face away from the object that triggered the button.

```
  if  (!digitalRead (12)) {
      leftServo.write(90);
      rightServo.write(90);
      delay(250);
      leftServo.write(135);
      rightServo.write(45);
      delay(2000);
      leftServo.write(45);
      rightServo.write(45);
      delay(2000);
  }
```

The second if statement will start by looking at the push button plugged into pin 11. If that value becomes not true (signified by the ! in front of digitalRead (11)) the code will run through the process of stopping, backing up for two seconds and then rotating to face away from the object that triggered the button.

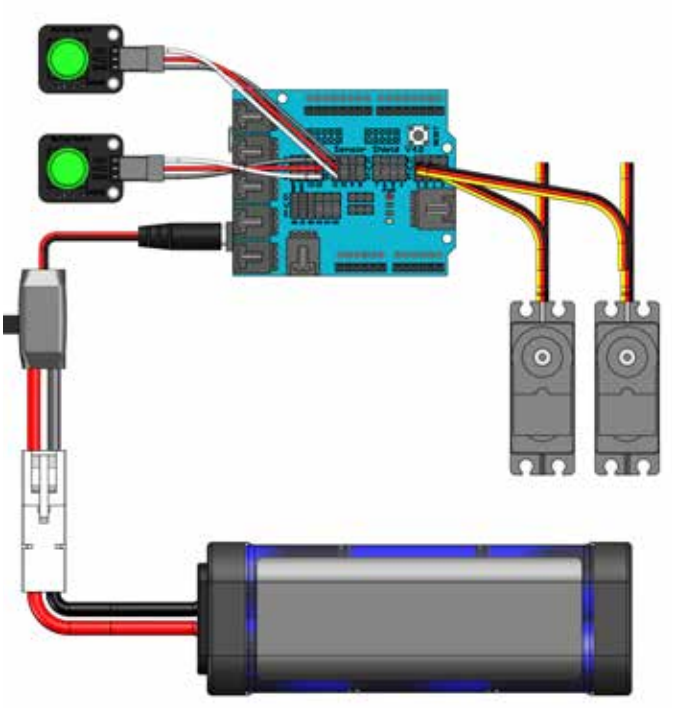
```
  if  (!digitalRead (11)) {
      leftServo.write(90);
      rightServo.write(90);
      delay(250);
      leftServo.write(135);
      rightServo.write(45);
      delay(2000);
      leftServo.write(135);
      rightServo.write(135);
      delay(2000);
  }
}
```

1.1 Challenge Set up a Course for the HexaBot

Set up a small maze for the HexaBot to navigate through. You can include various dead ends and T's in the path to add to the complexity of the maze.

Open the Arduino program, Go to File > Examples > 0. Minds-i > 3-Projects > HexaBot.

Wiring Diagram for the HexaBot



LineFollowBot_Line

This version of the LineFollowBot utilizes two continuous rotation servos to opperate the servo wheels for motion and the three QTI (infared sensors) will be used to measure and compare the intensity of light reflecting off of a surface with the goal of following a line on that surface.

> **Example Code**
> In the Arduino Program: File > Examples > MINDSi > 3-Projects > LineFollowBot_Line

Code:

This section will use written code to run the LineFollowBot. To begin, include the servo and MINDSi libraries:

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

Define our "Servo" objects, "left, right":

```
Servo left, right;
```

Define our "float" objects, "Lout, Rout":
"Lset. Rset"

```
float Rout, Rout;
float Rset, Rset;
```

Define our "const" objects, "rxtime, neutral, leftCenter, rightCenter, outsideSpeed, insideSpeed and forwardSpeed":

```
const double rxtime      = 0.25;
const int neutral        = 90;
const int leftCenter     = 4;
const int rightCenter    = 5;
const int outsideSpeed   = 45;
const int insideSpeed    = 12;
const int frowardSpeed   = 35;
```

In the “void setup()” function we will tell the Arduino that we are attaching “left” to pin 5 and “right” to pin 4 on the Arduino. We will also set both of left and right to the value we defined for neutral.

```
void setup( )
{
      left.attach(5);
      righ.attach(4);

      left.write(neutral);
      righ.write(neutral);

      Rout = neutral;
      Lout = neutral;
```

The first thing we will do in the “void loop()” function will be to name and check all three QTI sensors. We will name them left, middle and right. Then we check to see if they are above the value of 120. A value of 120 is in-between white and black. This allows us to distinguish between the two surfaces.

```
void loop()
{
      bool left   = QTI(A0) > 120;
      bool middle = QTI(A1) > 120;
      bool right  = QTI(A2) > 120;
```

With the QTI sensors now named and the values checked we will set up our three tiered if statement. If the left sensor is above 120 we will set the Lset to neutral + insideSpeed and the Rset to neutral + outsideSpeed. If the left sensor isnt above 120 we will continue to checking the right sensor. If the right sensor is above 120 we will set the Lset to neutral - outsideSpeed and the Rset to neutral - insideSpeed. If the right sensor isnt above 120 we will continue to checking the middle sensor. If the middle sensor is above 120 we will set the Lset to neutral - forwardSpeed and the Rset to neutral + forwardSpeed. Finaly set all of the above as output();.

```
  if  (left) {
      Lset = neutral + insideSpeed;
      Rset = neutral + outsideSpeed;
  }
  else if (right) {
      Lset = neutral - outsideSpeed;
      Rset = neutral - insideSpeed;
  }
  else if (middle) {
      Lset = neutral - forwardSpeed;
      Rset = neutral + forwardSpeed;
  }

  output();
}
```

With this code there is a special section at the end that we use to regulate the time between checking the sensors. The amount of light reflecting back from the surface will vary the time it takes for the sensor to get a reading. This last section we call void output.

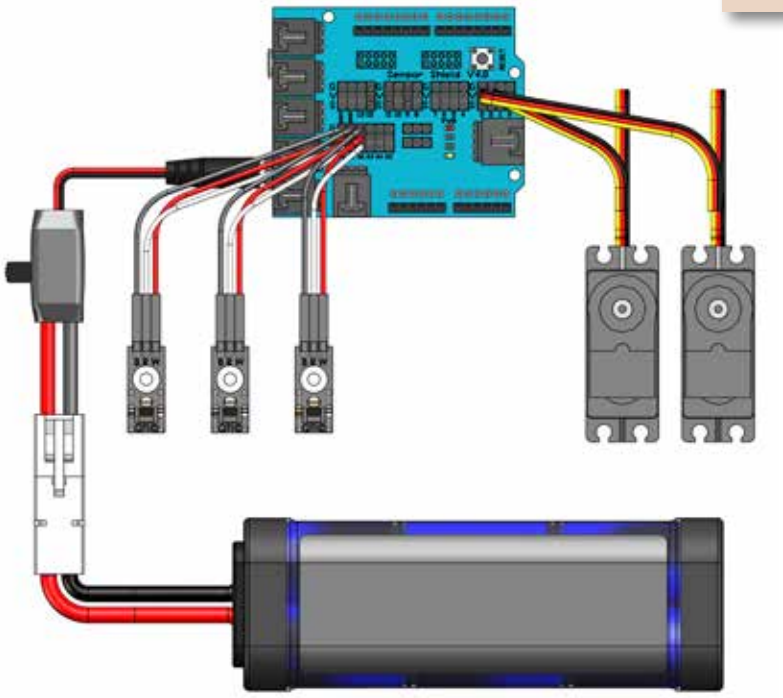
```
void output(){
      static unit32_t time = millis ();

      double dT = ( double(millis() - time) / 100;
      time = millis();

      double adj = pow(rxtime, dt);

      Rout = ( Rout * adj ) + ( Rset * (1 - adj) );
      Lout = ( Lout * adj ) + ( Lset * (1 - adj) );

      Left.write (Lout + leftCenter);
      right.write (Rout + rightCenter);
}
```

1.1 Challenge Set up a Course for Line Following

Set up a small course for the LineFollowBot to navigate through. You can include various dead ends and T's in the path to add to the complexity of the line following course.

Open the Arduino program, Go to File > Examples > 0. Minds-i > 3-Projects > LineFollowBot_Line.

Wiring Diagram for the LineFollowBot



LineFollowBot_Ping

This version of the LineFollowBot utilizes two continuous rotation servos to opperate the servo wheels for motion and one Ping ultrasound sensor to avoid running into any obstacles.

> **Example Code**
> In the Arduino Program: File > Examples > MINDSi > 3-Projects > LineFollowBot_Ping

Code:

This section will use written code to run the LineFollowBot. To begin, include the servo and MINDSi libraries:

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

Define our "Servo" objects, "left, right":

```
Servo left, right;
```

Define our "const int" objects, "center and hazardDist":

```
const int center        = 90;
const int hazardDist    = 750;
```

In the “void setup()” function we will tell the Arduino that we are attaching “left” to pin 5 and “right” to pin 4 on the Arduino. We will also set the forward directions of both of left and right.

```
void setup( ){
      left.attach(5);
      righ.attach(4);

      //driving straight
      left.write(0);
      righ.write(180);
}
```

In the “void loop()” function we will set up an if/else statement. We begin with the if statement by defining the sensor and pin we are wanting to test. In this case we are looking at ping sensor plugged into pin 9. The condition we are testing is to see if the sensor is returning a value less than the hazardDist (750). If the value is less than the hazardDist we will start by telling the servos to go to center (90).

```
void loop()
{
  if  (getPing(9) < hazardDist) {
      //liberal coasting to prevent brownouts
      left.write(center);
      right.write(center);
      delay(500);
```

We will continue with setting the servo values to make it turn for one second, roughly 90º. We will then set both wheels to center.

```
      left.write(180);
      right.write(center);
      delay(1000);

      left.write(center);
      right.write(center);
      delay(250);
```

Lastly we will finish our if/else statement with the else section telling it to drive straight forward.

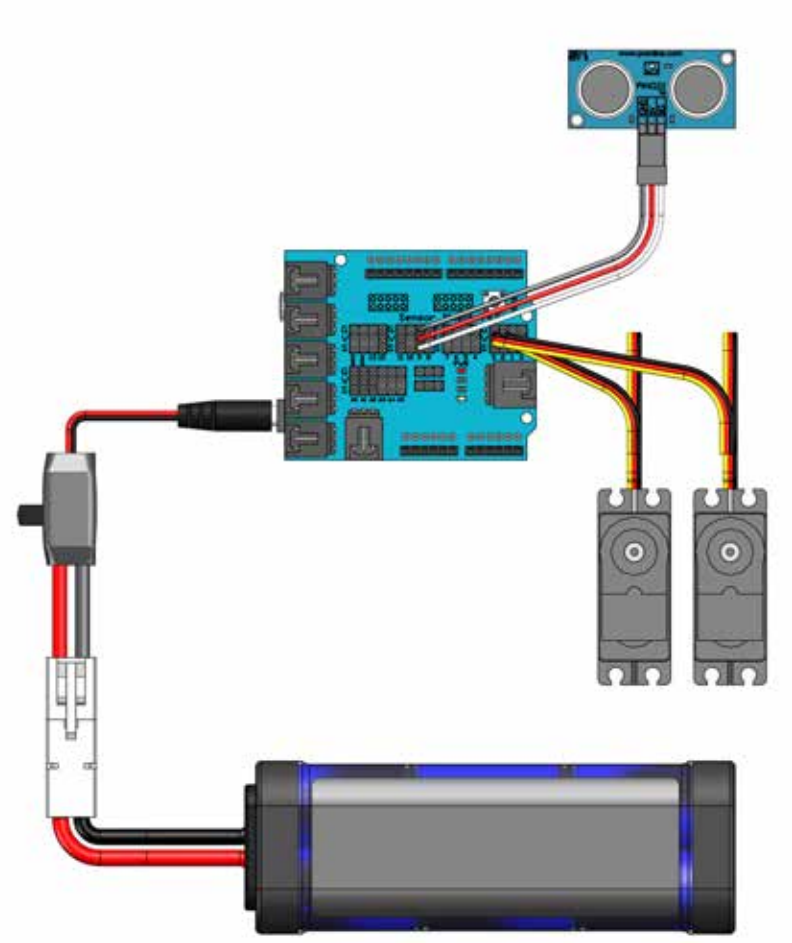
```
  } else if {
      //resume driving straight
      left.write(c0);
      right.write(180);
  }
}
```

1.1 Challenge Avoiding Obstacles

Point the Bot toward an obstacle and watch as it approaches, stops and turns away. The direction it turns is predefined but can be changed by swapping the values in the code

Open the Arduino program, Go to File > Examples > 0. Minds-i > 3-Projects > LineFollowBot_Ping.

Wiring Diagram for the LineFollowBot



TROUBLESHOOTING GUIDE

PROBLEM	CAUSE	SOLUTION
Wheels rotate backwards when forwards throttle is applied.	1) One or more differnetials are installed upside down.	1) Check the assembly instructions to make sure the differentials were installed correctly.
	2) The motor is spinning the wrong direction.	2) Check that the wires from the ESC to the motor are connected properly.
	3) The motor case was installed with the motor facing opposite of what the instructions show.	3) Check the assembly instructions to make sure the motor case was installed correctly.
	4) The “throttle” switch on remote control may be switched to reverse.	4)Flip the switch to normal.
Front wheels rotate opposite of rear wheels when driving.	1) Differentials are installed upside down.	1) Check the assembly instructions for proper installation of the differnetials
Vehicle doesn’t drive in a straight line with the steering channel centered.	1) Steering trim is not centered.	1) Center steering trim.
	2) Servo horn was not installed on the servo while it was centered.	2) Remove the screw that retains the servo horn, then remove the servo horn. Turn the remote control on, center the trim on the remote, turn the vehicle on. Then reinstall the servo horn as the instructions show.
	3) The linkages that connect your steering bar to the front wheel knuckles are not the proper length or do not match each other.	3) Remove the linkages and check that they are the length indicated in your instructions.
	4) The “Y” harness for four wheel steering is not centered.	4) Follow step 2 for front and rear servo, making sure when you center the trim on the remote to also center the trim on the “Y” harness.
Servo does not operate when steering is applied.	1) Servo cable is plugged in upside down.	1) Check that the cable is plugged in according to the wiring polarity guide.
Servo steers only one direction or farter one way than the other.	1) Servo horn is not centered on the servo.	1) Remove the servo screw, then remove the servo horn. Turn the remote control on, center the trim on the remote, turn the vehicle on. Then reinstall the servo horn as the instructions show.
Vehicle has become under powered or sluggish.	1) Charge level of vehicle batter has dropped below useable level.	1) Remove battery from vehicle and charge with the recommended charger.
Vehicle stops unexpectedly or doesn’t respond to remote.	1) Charge level of remote control batteris have dropped below useable level.	1) Check the remote control battery indicator, if necessary replace or rechare batteries.
Vehicle does not move forward or reverse when throttle is applied but audible tone comes from motor.	1) Something has become bound up in the drivetrain.	1) Check the drivetrain from motor to wheels for foreign objects and debris. Removing drivelines that connect the motor case to the differentials then trying to spin the wheels will allow you to more easily locate the issues.

PROBLEM	CAUSE	SOLUTION
Motor spins freely when throttle is applies but vehicle doesn't move.	1) Gear mesh in motor case was not properly set.	1) Check that the gear mesh, Gears should be about the thinkness of this sheet of paper apart for proper mesh.
	2) Pinion set screw was not properly tightened.	2) Check that the pinioin set screw was tightened according to the instructions.
Vehicle is not behaving as programmed.	1) Charge level of vehicle battery has dropped below usable level.	1) Charge battery with recommended charger.
	2) Inputs (sensors) or Outputs (servos, ESC, etc) are wired improperly.	2) Check that each device connected to the Arduino Sensor shield is
	3) Sensor is malfunctioning.	3) Connect Arduino to computer using USB cable, run the calibration program for each sensor on the vehicle. If the sensor
Serial port "COM_" not found.	1) The Arduino is not connected to the computer.	1)Connect the USB cable and try again.
	2) The proper serial port was not selected in the Arduino Program.	2) Disconnect, re-connect the USB cable then go to Tools > Serial Port then select the correct board.
Light on Ping))) sensor is not cycling.	1) The Ping))) sensor is not wired properly.	1) Check the wiring polarity guide for proper installation.
	2) The Ping))) sensor is not connected to the the proper port.	2) Check that the sensor is plugged into the port stated in the code.
QTI sensors not detecting line.	1) The QTI sensor is not wired properly.	1) Check the wiring polarity guid for proper installation.
	2) The QTI sensor is not connected to the proper port.	1) Check that the sensor is plugged into the correct port.

PROBLEM	CAUSE	SOLUTION
Power light on Arduino Sensor Shield doesn't turn on when ESC is powered on.	1) One of the connections to the Sensor Shield is shorted.	1) Check that each device connected to the Arduino Sensor Shield is plugged in properly. Both in polarity and into the proper port the program states. If the problem persists disconnect one sensor at a time till the faulty connection (sensor, servo, ESC etc.) is found, then replace.
	2) No power source is connected to the Arduino.	2) Check to make sure that it is either plugged into your computer with the USB cable or that you are using a charged battery and the ESC cable is properly connected.

Why I can't upload my programs to the Arduino board?

There are many pieces involved in getting a program onto your Arduino board, and if any of them aren't right, the upload can fail. They include: the drivers for the board, the board and serial port selections in the Arduino software, access to the serial port, the physical connection to the board, the firmware on the 8U2 (on the Uno and Mega 2560), the bootloader on the main microcontroller on the board, the microcontroller's fuse settings, and more. Here are some specific suggestions for troubleshooting each of the pieces.

Arduino Software

Make sure you have the right item selected in the Tools > Board menu. If you have an Arduino Uno, you'll need to choose it.

Then, check that the proper port is selected in the Tools > Serial Port menu (if your port doesn't appear, try restarting the IDE with the board connected to the computer). On the Mac, the serial port should be something like /dev/tty.usbmodem621 (for the Uno or Mega 2560) or /dev/tty.usbserial-A02f8e (for older, FTDI-based boards). On Linux, it should be /dev/ttyACM0 or similar (for the Uno or Mega 2560) or /dev/ttyUSB0 or similar (for older boards). On Windows, it will be a COM port but you'll need to check in the Device Manager (under Ports) to see which one. If you don't seem to have a serial port for your Arduino board, see the following information about drivers.

Drivers

Drivers provide a way for software on your computer (i.e. the Arduino software) to talk to hardware you connect to your computer (the Arduino board). In the case of Arduino, the drivers work by providing a virtual serial port (or virtual COM port). The Arduino Uno and Mega 2560 use standard drivers (USB CDC) provided by the operating system to communicate with the ATmega8U2 on the board. Other Arduino boards use FTDI drivers to communicate with the FTDI chip on the board (or in the USB-serial convertor).

The easiest way to check if the drivers for your board are installed correctly is by opening the Tools > Serial Port menu in the Arduino software with the Arduino board connected to your computer. Additional menu items should appear relative to when you open the menu without the Arduino connected to your computer. Note that it shouldn't matter what name the Arduino board's serial port gets assigned as long as that's the one you pick from the menu.

Access to the Serial Port

On Windows, if the software is slow to start or crashes on launch, or the Tools menu is slow to open, you may need to disable Bluetooth serial ports or other networked COM ports in the Device Manager. The Arduino software scans all the serial (COM) ports on your computer when it starts and when you open the Tools menu, and these networked ports can sometimes cause large delays or crashes.
Check that you're not running any programs that scan all serial ports, like USB Cellular Wifi Dongle software (e.g. from Sprint or Verizon), PDA sync applications, Bluetooth-USB drivers (e.g. BlueSoleil), virtual daemon tools, etc.
Make sure you don't have firewall software that blocks access to the serial port (e.g. ZoneAlarm).
You may need to quit Processing, PD, vvvv, etc. if you're using them to read data over the USB or serial connection to the Arduino board.

Physical Connection

First make sure your board is on (the green LED is on) and connected to the computer.
The Arduino Uno and Mega 2560 may have trouble connecting to a Mac through a USB hub. If nothing appears in your "Tools > Serial Port" menu, try plugging the board directly to your computer and restarting the Arduino IDE.
Disconnect digital pins 0 and 1 while uploading as they are shared with serial communication with the computer (they can connected and used after the code has been uploaded).
Try uploading with nothing connected to the board (apart from the USB cable, of course).
Make sure the board isn't touching anything metallic or conductive.
Try a different USB cable; sometimes they don't work.

Forum Support

If it still doesn't work, you can ask for help in the Adruino.cc forum. Please include the following information:

- Your operating system.
- What kind of board you have.
- Whether or not you were ever able to upload to the board. If so, what were you doing with the board before / when it stopped working, and what software have you recently added or removed from your computer?
- The messages displayed when you try to upload with verbose output enabled. To enable verbose output check the box next to File > Preferences > Show verbose output during: > upload.
- Click on Copy error messages button on the right side of the box. When submitting in the forum please use code tags (</> button on the forum website toolbar) to post the output so that it will be correctly formatted.

For technical questions or to place an order:

Phone: (509) 252 - 5767
Fax: (509) 924 - 2219

Email us at:
info@mymindsi.com

Write to:
ATTN: MINDS-i Inc.
22819 East Appleway Avenue
Liberty Lake, Washington 99019

For the latest from MINDS-i visit:
mindsieducation.com

For technical questions with programming email:
code@mymindsi.com