Arduino Programming Laboratory Manual

Introduction to Product Development and Management for Engineers
GNG 2101
Faculty of Engineering
University of Ottawa
Winter 2019

Checks required by TA:

Group Member	Part A	Part B	Part C	Part D

Dr. Hanan Anis
Dr. David Knox
Dr. Patrick Dumond

Objective

To utilize the Arduino IDE Software platform to write simple control logic commands. This control code will be used to make effective use of a variety of inputs and outputs connected to an Arduino Uno microcontroller. This lab will also expose students to more complicated control and communication codes, and online resources available for the Arduino software.

Background

In conjunction with different models of microcontrollers, the Arduino company provides the open-source Arduino Integrated Development Environment (IDE). This programming environment is written in Java and has been designed to introduce programming to an audience that is less familiar with coding. Programs written in this environment are called ‘sketches’, and the language used is based on the Processing language with support for C and C++ programming languages.

A basic Arduino sketch consists of at least two functions: a ‘setup’ function and a ‘loop’ function. The setup function performs any actions that are initially required to run the rest of the program, such as initializing any peripheral components and setting the communication frequency between the Arduino board and PC. The loop function acts as the programs driver; it runs in a continuous loop, and specifies the order of operations that the microcontroller will perform.

Arduino control boards as well as many Arduino-compatible peripherals contain hardware for serial communication. This form of communication transmits data one bit at a time over the serial connection. This allows the components of the system (or a PC) to communicate larger and more complex sequences of data much more easily, without physical input or output pin limitations.

Because of Arduino’s open-source nature, third party companies, as well as end users are free to take the software and tailor it to their individual needs. This means that for almost any application, sketches are available for download online, and companies making peripherals (such as a motor shield or sensors) for use with Arduino boards often have example sketches and sketch libraries available for specific use for their products. These resources can be used to create a custom sketch that can use many different peripherals at once, by combining aspects of many different example sketches.

Apparatus and Equipment Overview

The equipment that will be used in this lab includes:

- 1 x Arduino Uno R3 Microcontroller
- 1 x USB 2.0 Type A Plug to USB 2.0 Type B Plug Cable
- 1 x Lab or Personal Computer (running Windows, Macintosh, or Linux OS)
- 1 x USB flash drive
- 1 x Ultrasonic ranging module (HC - SR04)
- 1 x Adafruit Motor Shield V1 (L293D chipset)
- 2 x DC Motor in Micro Servo Body
- Arduino 1.6.9 IDE Software
- Corresponding Arduino IDE Libraries (outlined below)

- 22 Ga. Wire

Pre-Lab Preparation

Before arriving to the lab, students should review the lab manual and familiarize themselves with the lab setup and procedures. Students may use their own computer for this lab if they wish, provided that they have downloaded and installed the Arduino IDE, as well as the required libraries outlined in downloads section. Reviewing the Arduino IDE syntax, as well as trying to write some of the programs beforehand is also encouraged. A useful list of commands used in the Arduino IDE can be found here: https://www.arduino.cc/en/Reference/HomePage.

Downloads

The Arduino IDE can be downloaded from here: https://www.arduino.cc/en/Main/Software

The libraries used in this lab are:

- AFMotor.h (download from https://learn.adafruit.com/adafruit-motor-shield/library-install)
- NewPing.h (download from http://playground.arduino.cc/Code/NewPing)
- Adafruit Bluefruit LE Master library (downloaded from https://learn.adafruit.com/introducingthe-adafruit-bluefruit-le-uart-friend/software)

To install Arduino libraries, go to Sketch>Add .ZIP Library… and browse to and select the folder that contains the library or libraries that need to be added. For a more integrated (and permanent) option, move the folder containing the library to the Arduino libraries folder in the system files. For most windows users the default pathway will be Program Files (x86)>Arduino>libraries, but this can change with different operating systems, different versions of windows, or if the Arduino IDE was installed to a different location on the computer. For both methods the Arduino IDE should be restarted for the libraries to become useable. Adafruit.com has a useful tutorial on installing libraries that can be found here: h$ps://learn.adafruit.com/adafruit-all-about-arduino-libraries-install-use

Pre-Lab Questions

What is the role of a microcontroller?

____

____

Which microcontroller are we using in this lab?

____

____

How can the ultrasonic sensor measure distance?

____

____

How will the motors be controlled to rotate in both directions?

____

____

Which wireless communication method are we using?

____

____

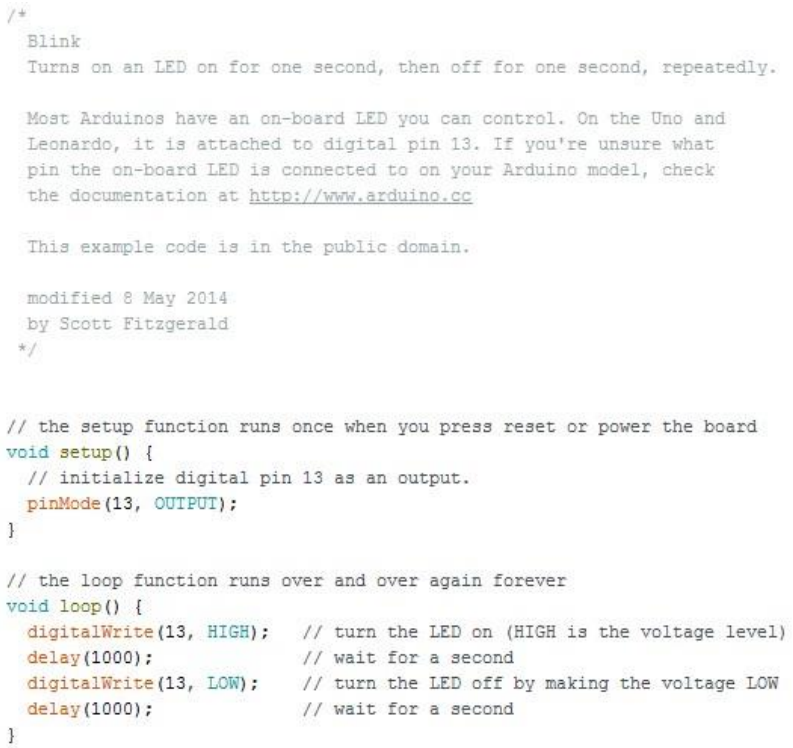
Procedure

Sensitive electrical components like the control boards and sensors used in this lab can be damaged or destroyed from static discharge from mishandling during assembly and use. Before setting up this lab, students should ensure that their workstation is statically prepared (non-metal workstation, no carpet/other fibrous materials), and they have grounded themselves by touching a large metal object before handling the components. There are accessories to improve the assembly workstation for static resistance, such as an antistatic mat and static wrist/ankle straps, which should be used if available.

Part A – LED Blink Program

This program will go over the basic method to connecting and programming an Arduino microcontroller. It utilizes a sample sketch included in one of the Arduino IDE's default libraries, and is primarily used for verifying that the microcontroller and the connection are both functioning properly.

1. Connect the Arduino board to an open USB port on the computer with the USB cable. The Arduino board should light up, as it draws power through the USB connection.
2. Launch the Arduino IDE Software, and open Tools>Board>Arduino/Genuino Uno from the dropdown menu. Back in the 'Tools' tab, select 'Port' and choose the serial port that the Arduino is connected to. If the correct serial port is not apparent, unplug the Arduino, and see which port disappears. Re-connect the board, and select that port.
3. Open File>Examples>01.Basics>Blink. This will open a sketch that looks like the one below.



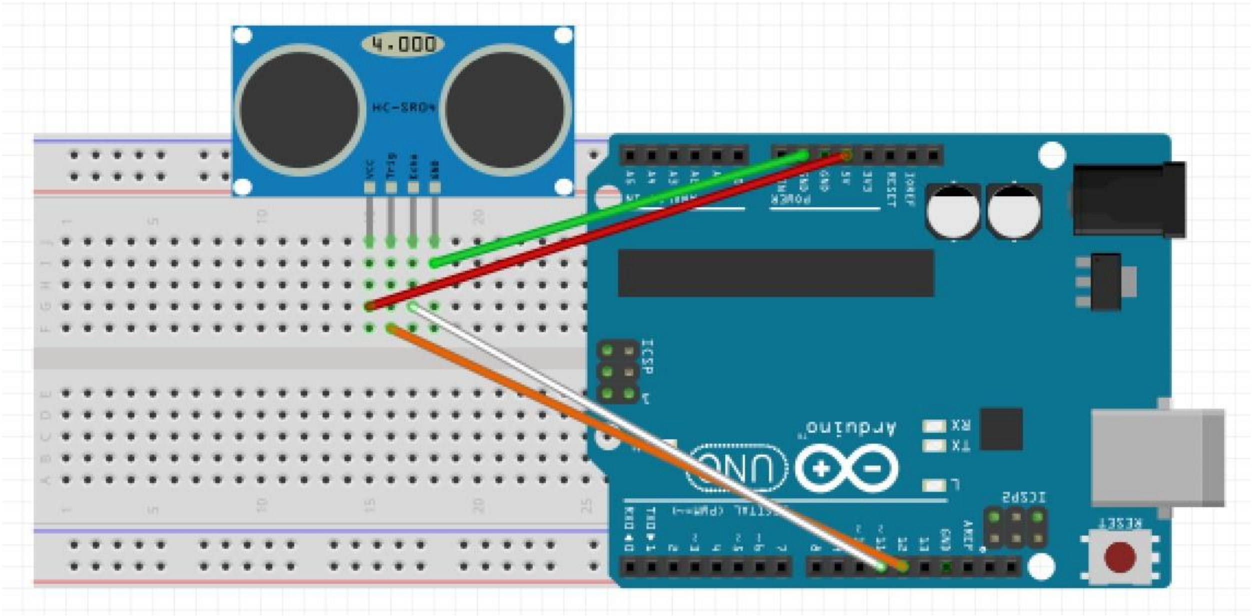
```
/*
  Blink
  Turns on an LED on for one second, then off for one second, repeatedly.

  Most Arduinos have an on-board LED you can control. On the Uno and
  Leonardo, it is attached to digital pin 13. If you're unsure what
  pin the on-board LED is connected to on your Arduino model, check
  the documentation at http://www.arduino.cc

  This example code is in the public domain.

  modified 8 May 2014
  by Scott Fitzgerald
 */


// the setup function runs once when you press reset or power the board
void setup() {
  // initialize digital pin 13 as an output.
  pinMode(13, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {
  digitalWrite(13, HIGH);   // turn the LED on (HIGH is the voltage level)
  delay(1000);              // wait for a second
  digitalWrite(13, LOW);    // turn the LED off by making the voltage LOW
  delay(1000);              // wait for a second
}
```

4. Now verify and upload the sketch to the Arduino board. To verify, click the button in the upper left hand corner. The Arduino IDE will try to compile the sketch (without uploading to the board), and warn of any syntax errors in the programming. Once this is complete, press the upload button. The sketch will re-compile, and upload to the board.
5. Once the upload is complete, the program will automatically run on the microcontroller. In this case, the LED labelled ‘L’ on the Arduino should flash slowly. Show the TA your work.

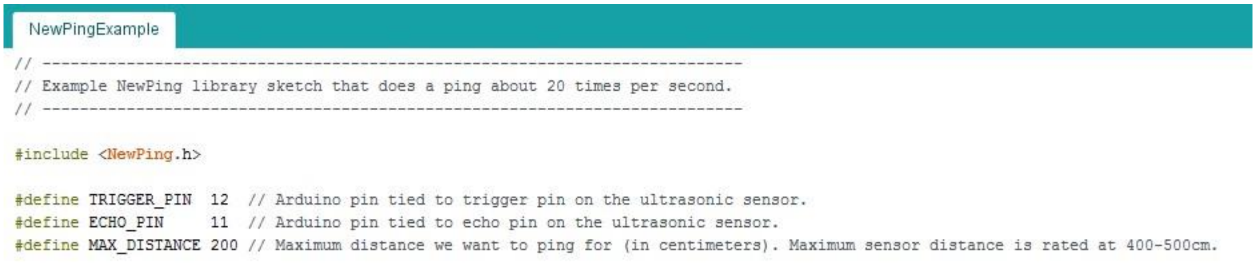
Part B – Sensor Reading

This program will automatically poll an ultrasonic sensor, causing the sensor to emit an ultrasonic sound pulse. The program will read the elapsed time between sending the pulse and receiving an echo, convert the time to distance using the speed of sound, and print the distance to the computer screen via the serial port connection.

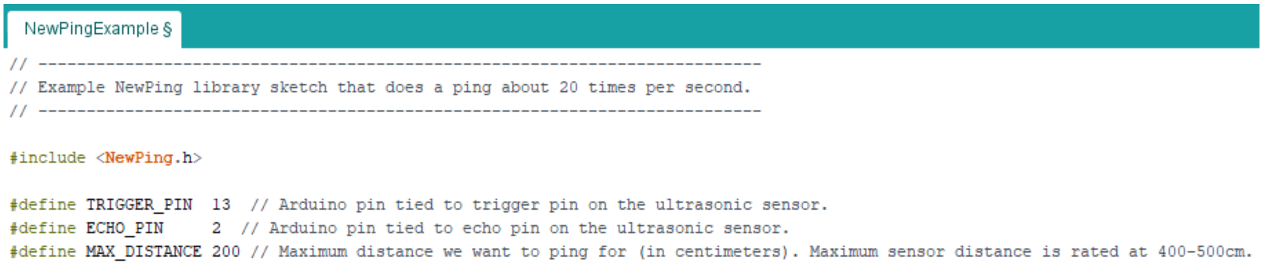
1. Connect an ultrasonic sensor to the Arduino board by using the wiring diagram below. The Arduino board should still be connected to the computer via USB cable.



2. Open a new example sketch in the Arduino IDE. The specific example will be included in the "NewPing" library that should have been added to the Arduino IDE. Go to File>Exmples>NewPing>NewPingExample.
3. Compile and upload this example to the Arduino board. Be sure to verify that the correct board is selected (Arduino/Genuino Uno), and the proper comm port is being utilized.
4. Once the example has been uploaded, open the serial monitor to see the sensor data. Click the icon in the top right corner of the IDE, or go to Tools>Serial Monitor.
5. Set the baud rate of the serial monitor to correspond to the baud rate specified in the example sketch. Select the correct baud rate by utilizing the drop down menu in the lower left hand corner of the serial monitor window. In this case the baud rate should be 115200. After a few moments distance measurements should start to appear in the monitor window. Show the TA your work.
6. After verifying the sensor is operating correctly, change the pins on the Arduino that are used to send and receive data to the sensor. Alter the two lines in the beginning of the sketch that define the echo and trigger pins (the default pins are 11 and 12 respectively) to send a trigger through pin 13 and receive an echo through pin 2 (see below).

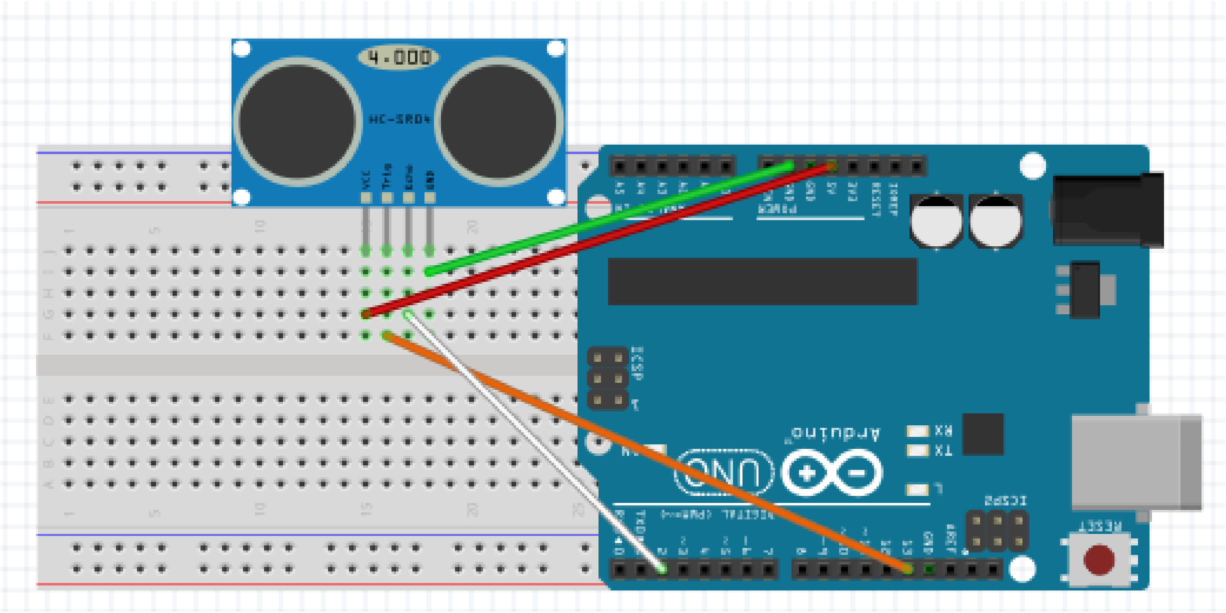


```
// ---------------------------------------------------------------------------
// Example NewPing library sketch that does a ping about 20 times per second.
// ---------------------------------------------------------------------------

#include <NewPing.h>

#define TRIGGER_PIN  12  // Arduino pin tied to trigger pin on the ultrasonic sensor.
#define ECHO_PIN     11  // Arduino pin tied to echo pin on the ultrasonic sensor.
#define MAX_DISTANCE 200 // Maximum distance we want to ping for (in centimeters). Maximum sensor distance is rated at 400-500cm.
```



```
// ---------------------------------------------------------------------
// Example NewPing library sketch that does a ping about 20 times per second.
// ---------------------------------------------------------------------

#include <NewPing.h>

#define TRIGGER_PIN  13  // Arduino pin tied to trigger pin on the ultrasonic sensor.
#define ECHO_PIN     2  // Arduino pin tied to echo pin on the ultrasonic sensor.
#define MAX_DISTANCE 200 // Maximum distance we want to ping for (in centimeters). Maximum sensor distance is rated at 400-500cm.
```

7. Rewire the sensor to the Arduino to reflect the pinout changes that were just made.

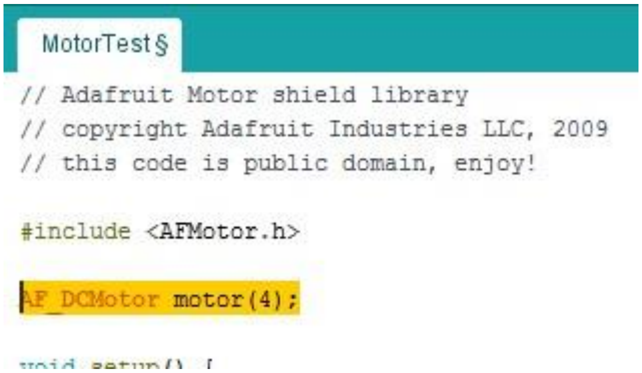


8. Compile and re-upload the new sketch. Open the serial monitor the same was as before, and verify the sensor is using the new pins correctly. Show the TA your work.

Part C – DC Motor Control

This program will test and control two DC motors using a pre-set sequence of commands.

1. Disconnect the sensor from the Arduino. Insert the motor shield board onto the Arduino headers and connect a DC motor to port M1 on the motor shield.
2. Open the example found in Files>Examples>DC Motor Shield Library>Motor Test. This example sketch runs a DC motor connected to port M1 on the motor shield.
3. The motor test example controls a motor connected to port M4 by default. Change this port to port M1 by changing line shown below. The new line should read “AF_DCMotor motor(1);”.

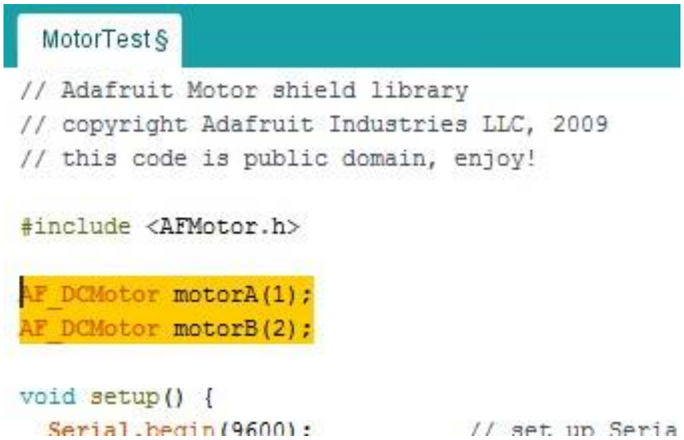


```
MotorTest §
// Adafruit Motor shield library
// copyright Adafruit Industries LLC, 2009
// this code is public domain, enjoy!

#include <AFMotor.h>

AF_DCMotor motor(4);

void setup() {
```

4. Compile and upload this sketch to the Arduino board. Verify that the DC motor spins in conjunction with the programming. Show your work to the TA.

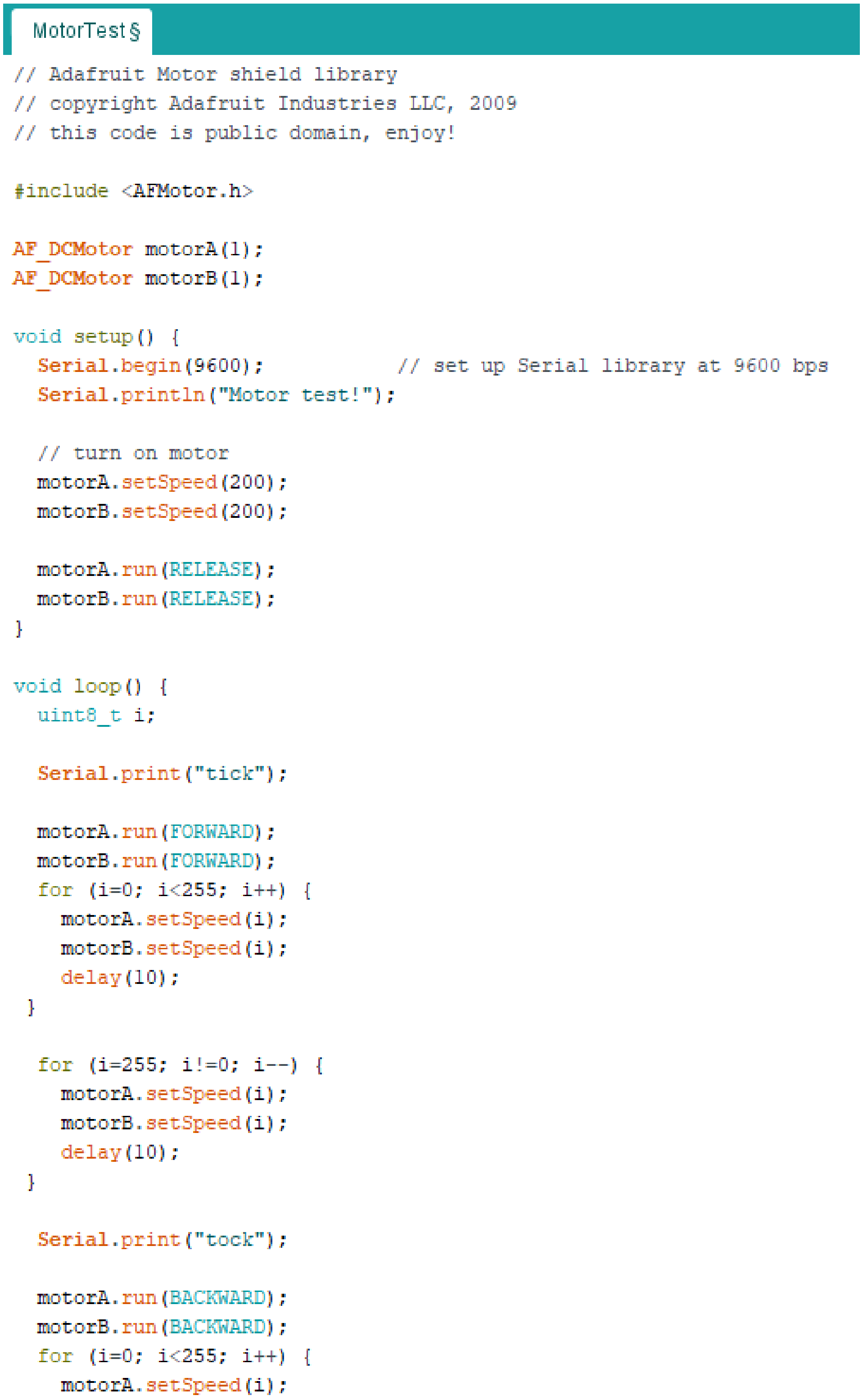
5. Now add commands to control a second DC motor in conjunction with the first motor. Connect a DC motor to port M2 on the motor shield (see below).

6. Add command lines to the sketch to control the second motor. Start by initializing the new motor and the port on the motor shield being used. Write an additional AF_DCMotor line below the one that was altered in Step 3. Both AF_DCMotor statements should name their corresponding motor a unique name (see below).



```
MotorTest §
// Adafruit Motor shield library
// copyright Adafruit Industries LLC, 2009
// this code is public domain, enjoy!

#include <AFMotor.h>

AF_DCMotor motorA(1);
AF_DCMotor motorB(2);

void setup() {
  Serial.begin(9600);           // set up Seria
```

7. Each motor statement within the sketch will now have to be modified to reflect the new motor name. Add a second motor statement below every existing motor statement (not just the ones shown) in order to control the second motor (see below).



```
// Adafruit Motor shield library
// copyright Adafruit Industries LLC, 2009
// this code is public domain, enjoy!

#include <AFMotor.h>

AF_DCMotor motorA(1);
AF_DCMotor motorB(1);

void setup() {
  Serial.begin(9600);           // set up Serial library at 9600 bps
  Serial.println("Motor test!");

  // turn on motor
  motorA.setSpeed(200);
  motorB.setSpeed(200);

  motorA.run(RELEASE);
  motorB.run(RELEASE);
}

void loop() {
  uint8_t i;

  Serial.print("tick");

  motorA.run(FORWARD);
  motorB.run(FORWARD);
  for (i=0; i<255; i++) {
    motorA.setSpeed(i);
    motorB.setSpeed(i);
    delay(10);
 }

  for (i=255; i!=0; i--) {
    motorA.setSpeed(i);
    motorB.setSpeed(i);
    delay(10);
 }

  Serial.print("tock");

  motorA.run(BACKWARD);
  motorB.run(BACKWARD);
  for (i=0; i<255; i++) {
    motorA.setSpeed(i);
```

8. Compile and upload the sketch. Verify that both motors now spin in unison. Show the TA your work.
    a Sometimes the computer isn’t capable of giving enough power for both motors, if this is the case ask the TA for AA batteries.

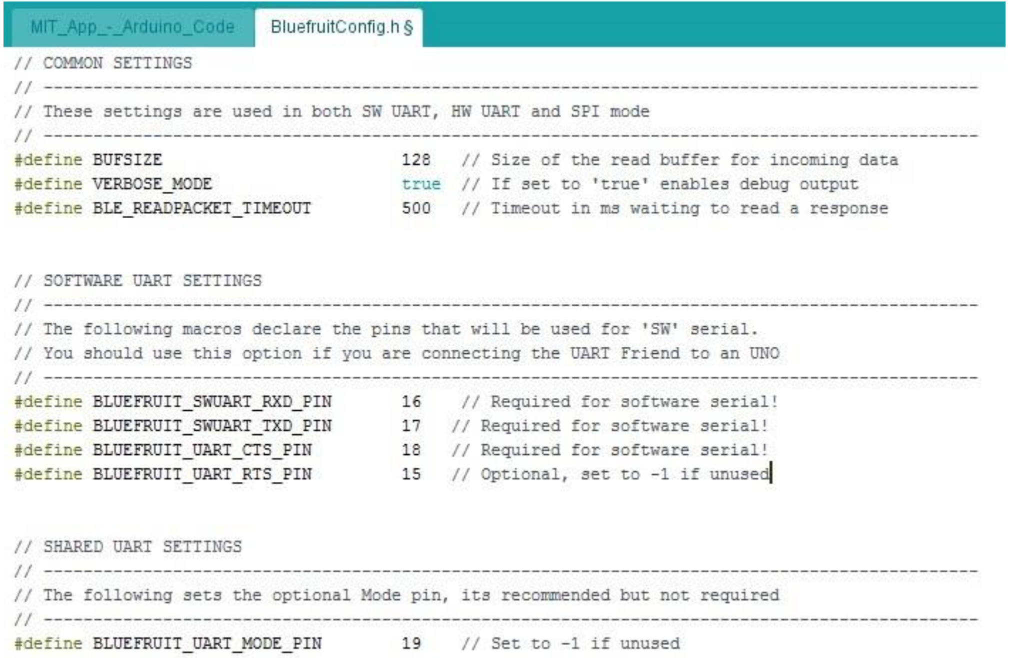
Part D – Bluetooth Motor Control

PLEASE identify if your device uses BLE (Bluetooth 4.0 Low Energy) or the regular Bluetooth (ask a TA if not sure). You should be using a BLE device.

Bluetooth Motor Control For iPHONES (4s and newer) AND ANDROID (LE only)

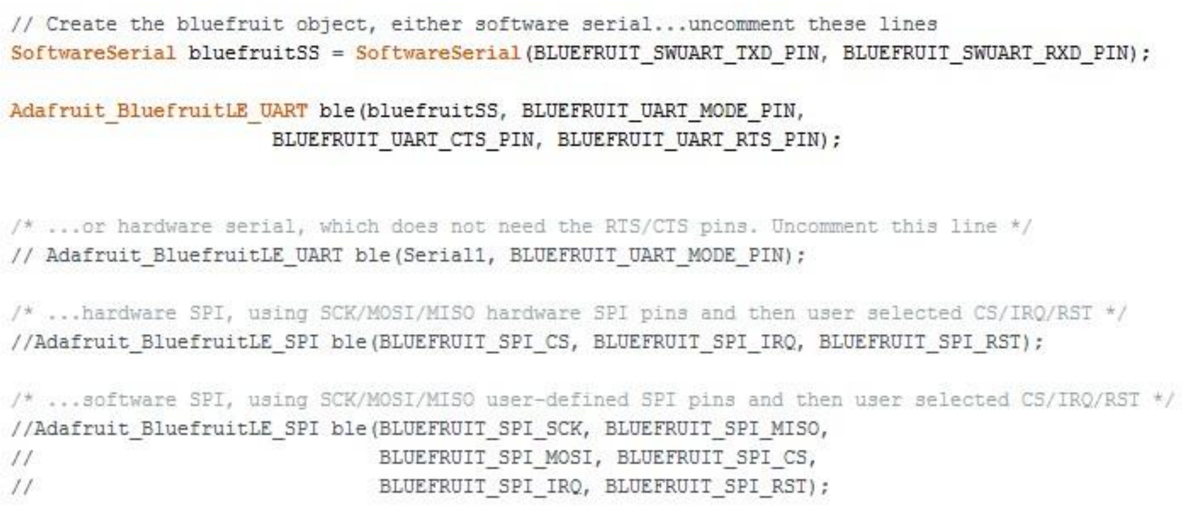
This program will use incoming data from a BluetoothLE compatible device to control the motors with the Bluetooth chip bluefruitBLE. The sensor will act as a check that will limit when the motors can be used. This will be achieved by modifying an open-source sketch from Adafruit Industries.

1. Making sure the correct Arduino libraries are installed, navigate to File>Examples>Adafruit BluefruitLE nRF51 and open the bleuart_cmdmode example sketch. This will open a sketch with two available tabs at the top; bleuart_cmdmode and Bluefruit_Config.h.
2. Change Bluefruit_Config.h tab to utilize a software UART Connection. This is a type of serial connection (similar to the communications used in a USB device) used to communicate between the BluetoothLE module on the car and the Arduino Uno board. The sections of code required to setup a software UART connection are the common settings, the software UART settings, and the common UART settings. All of the other sections can be deleted. The pins used for the connection must also be changed to reflect which pins the module is connected to on the Arduino board. Use the photo below to set the correct pins and ensure the Bluefruit_Config.h file is correctly set up.



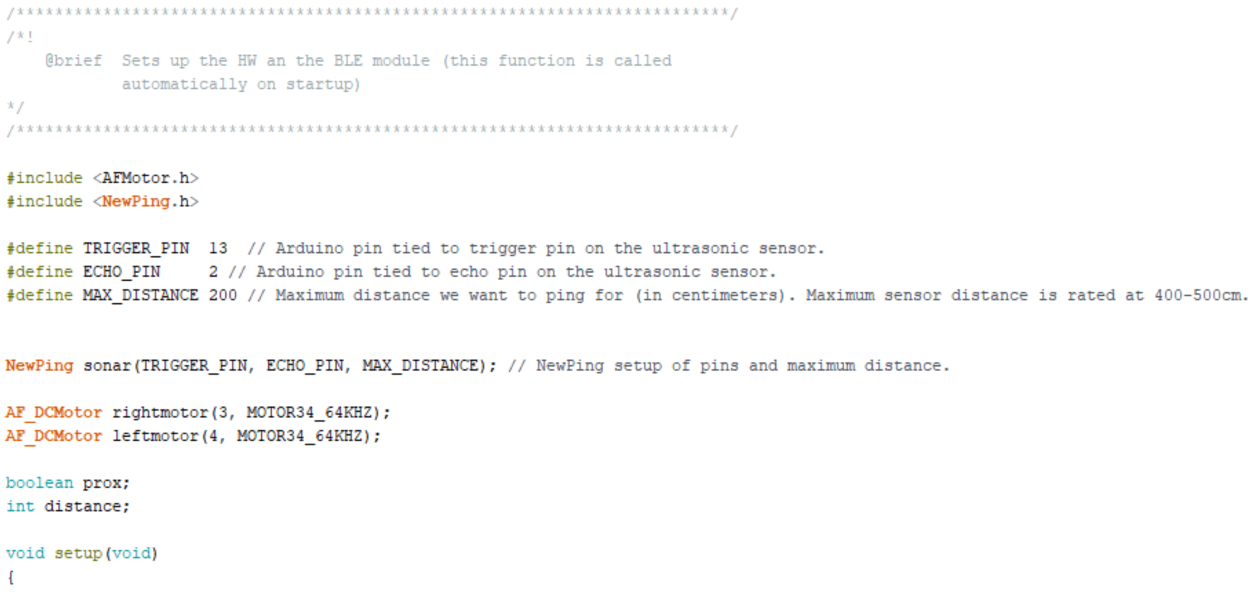
```
// COMMON SETTINGS
// -----------------------------------------------------------------------------------------------
// These settings are used in both SW UART, HW UART and SPI mode
// -----------------------------------------------------------------------------------------------
#define BUFSIZE                        128   // Size of the read buffer for incoming data
#define VERBOSE_MODE                   true  // If set to 'true' enables debug output
#define BLE_READPACKET_TIMEOUT         500   // Timeout in ms waiting to read a response


// SOFTWARE UART SETTINGS
// -----------------------------------------------------------------------------------------------
// The following macros declare the pins that will be used for 'SW' serial.
// You should use this option if you are connecting the UART Friend to an UNO
// -----------------------------------------------------------------------------------------------
#define BLUEFRUIT_SWUART_RXD_PIN       16    // Required for software serial!
#define BLUEFRUIT_SWUART_TXD_PIN       17   // Required for software serial!
#define BLUEFRUIT_UART_CTS_PIN         18   // Required for software serial!
#define BLUEFRUIT_UART_RTS_PIN         15   // Optional, set to -1 if unused


// SHARED UART SETTINGS
// -----------------------------------------------------------------------------------------------
// The following sets the optional Mode pin, its recommended but not required
// -----------------------------------------------------------------------------------------------
#define BLUEFRUIT_UART_MODE_PIN        19    // Set to -1 if unused
```

3. Now alter the bleuart_cmdmode tab. The software UART setting will have to be uncommented (delete the "/*" characters before the lines and "*/" after the lines), and the hardware UART lines must be removed (or commented out by adding "//" before the line).



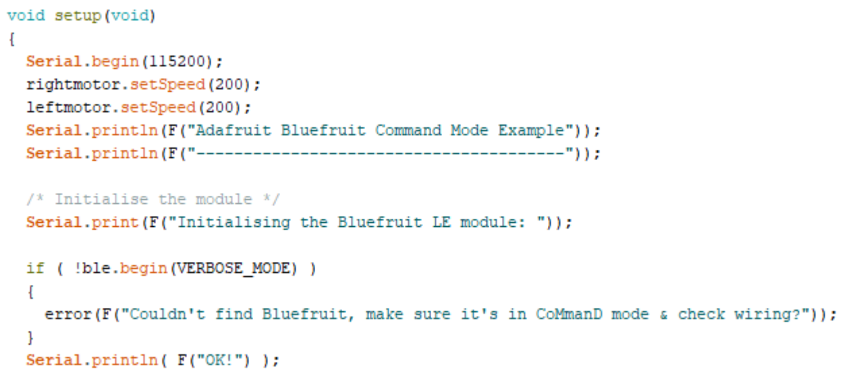
```
// Create the bluefruit object, either software serial...uncomment these lines
SoftwareSerial bluefruitSS = SoftwareSerial(BLUEFRUIT_SWUART_TXD_PIN, BLUEFRUIT_SWUART_RXD_PIN);

Adafruit_BluefruitLE_UART ble(bluefruitSS, BLUEFRUIT_UART_MODE_PIN,
                      BLUEFRUIT_UART_CTS_PIN, BLUEFRUIT_UART_RTS_PIN);


/* ...or hardware serial, which does not need the RTS/CTS pins. Uncomment this line */
// Adafruit_BluefruitLE_UART ble(Serial1, BLUEFRUIT_UART_MODE_PIN);

/* ...hardware SPI, using SCK/MOSI/MISO hardware SPI pins and then user selected CS/IRQ/RST */
//Adafruit_BluefruitLE_SPI ble(BLUEFRUIT_SPI_CS, BLUEFRUIT_SPI_IRQ, BLUEFRUIT_SPI_RST);

/* ...software SPI, using SCK/MOSI/MISO user-defined SPI pins and then user selected CS/IRQ/RST */
//Adafruit_BluefruitLE_SPI ble(BLUEFRUIT_SPI_SCK, BLUEFRUIT_SPI_MISO,
//                             BLUEFRUIT_SPI_MOSI, BLUEFRUIT_SPI_CS,
//                             BLUEFRUIT_SPI_IRQ, BLUEFRUIT_SPI_RST);
```

Before the setup section, include all the libraries, and initialize all the components and variables required to run the motors and sensors. These will include the AFMotor.h and NewPing.h Libraries, defining the sensor pins (trigger and echo) and maximum measuring distance, and initializing the sensor, both motors, and a Boolean variable and a long variable. See the section of code below as a reference.



```
/**************************************************************************/
/*!
    @brief  Sets up the HW an the BLE module (this function is called
            automatically on startup)
*/
/**************************************************************************/

#include <AFMotor.h>
#include <NewPing.h>

#define TRIGGER_PIN  13  // Arduino pin tied to trigger pin on the ultrasonic sensor.
#define ECHO_PIN     2 // Arduino pin tied to echo pin on the ultrasonic sensor.
#define MAX_DISTANCE 200 // Maximum distance we want to ping for (in centimeters). Maximum sensor distance is rated at 400-500cm.


NewPing sonar(TRIGGER_PIN, ECHO_PIN, MAX_DISTANCE); // NewPing setup of pins and maximum distance.

AF_DCMotor rightmotor(3, MOTOR34_64KHZ);
AF_DCMotor leftmotor(4, MOTOR34_64KHZ);

boolean prox;
int distance;

void setup(void)
{
```

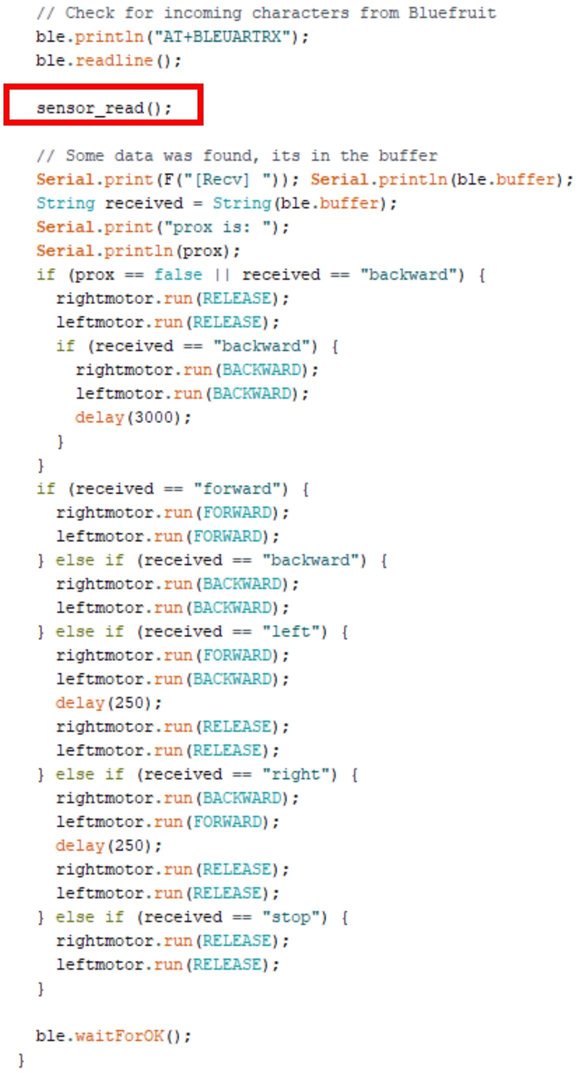
4. Alter the setup section of the sketch. Remove the while statement and the delay statement that occur before the Serial.begin(115200) statement. Insert statements to set the motor speed immediately after the Serial.begin statement (use the “rightmotor.setSpeed(200)” command).



```
void setup(void)
{
  Serial.begin(115200);
  rightmotor.setSpeed(200);
  leftmotor.setSpeed(200);
  Serial.println(F("Adafruit Bluefruit Command Mode Example"));
  Serial.println(F("---------------------------------------"));

  /* Initialise the module */
  Serial.print(F("Initialising the Bluefruit LE module: "));

  if ( !ble.begin(VERBOSE_MODE) )
  {
    error(F("Couldn't find Bluefruit, make sure it's in CoMmanD mode & check wiring?"));
  }
  Serial.println( F("OK!") );
```

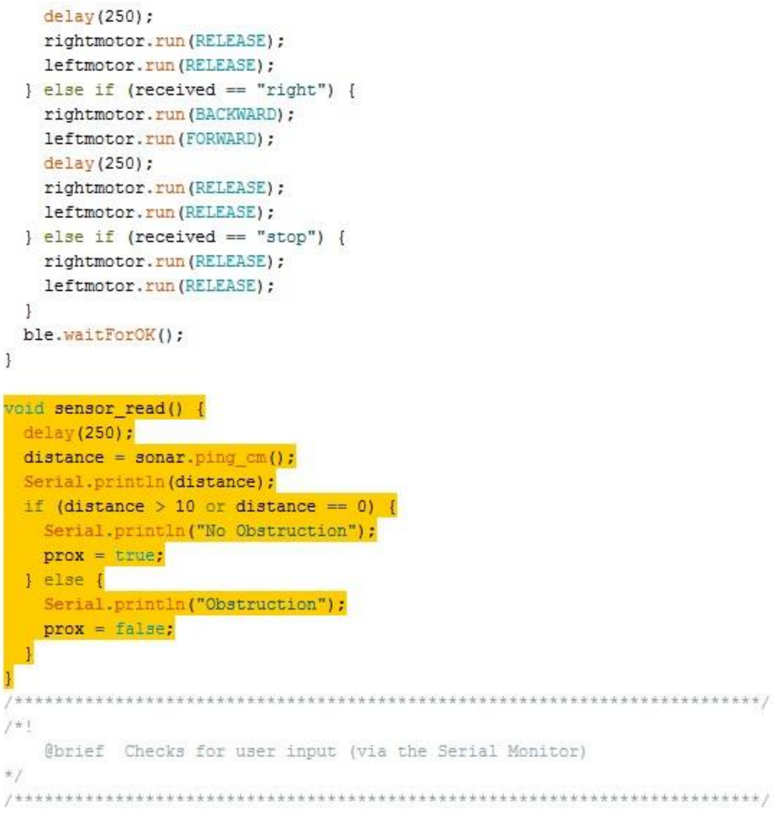
5. After the statement “ble.readline()” call the function “sensor_read()” which will be constructed in the next step. Next delete the following if statement that contains “strcmp”. After the statement “Serial.print(F(“[Recv] ”))” add the statement “String received = String(ble.buffer);”. Now you can add statements to control the motors and read the sensor within the loop section of the sketch. Immediately after the statement “String received = String(ble.buffer);” (near the end of the loop section) add if and if else structure statements to control motor direction based on the data received via Bluetooth. See the sketch section below as a reference.

Don’t forget this!



```
  // Check for incoming characters from Bluefruit
  ble.println("AT+BLEUARTRX");
  ble.readline();

  sensor_read();

  // Some data was found, its in the buffer
  Serial.print(F("[Recv] ")); Serial.println(ble.buffer);
  String received = String(ble.buffer);
  Serial.print("prox is: ");
  Serial.println(prox);
  if (prox == false || received == "backward") {
    rightmotor.run(RELEASE);
    leftmotor.run(RELEASE);
    if (received == "backward") {
      rightmotor.run(BACKWARD);
      leftmotor.run(BACKWARD);
      delay(3000);
    }
  }
  if (received == "forward") {
    rightmotor.run(FORWARD);
    leftmotor.run(FORWARD);
  } else if (received == "backward") {
    rightmotor.run(BACKWARD);
    leftmotor.run(BACKWARD);
  } else if (received == "left") {
    rightmotor.run(FORWARD);
    leftmotor.run(BACKWARD);
    delay(250);
    rightmotor.run(RELEASE);
    leftmotor.run(RELEASE);
  } else if (received == "right") {
    rightmotor.run(BACKWARD);
    leftmotor.run(FORWARD);
    delay(250);
    rightmotor.run(RELEASE);
    leftmotor.run(RELEASE);
  } else if (received == "stop") {
    rightmotor.run(RELEASE);
    leftmotor.run(RELEASE);
  }

  ble.waitForOK();
}
```

6. Now add statements to control the sensor. Create a new function and call it “sensor_read()”. Have the function ping the sensor, print the distance to the serial port, and set the Boolean variable initialized earlier to “true” if the read distance is greater than 10. Remember that a read value of “0” means the distance is out of range. If there is something less than 10 cm away from the sensor, set the Boolean variable to “false”. See the example below for reference.



```
    delay(250);
    rightmotor.run(RELEASE);
    leftmotor.run(RELEASE);
  } else if (received == "right") {
    rightmotor.run(BACKWARD);
    leftmotor.run(FORWARD);
    delay(250);
    rightmotor.run(RELEASE);
    leftmotor.run(RELEASE);
  } else if (received == "stop") {
    rightmotor.run(RELEASE);
    leftmotor.run(RELEASE);
  }
  ble.waitForOK();
}

void sensor_read() {
  delay(250);
  distance = sonar.ping_cm();
  Serial.println(distance);
  if (distance > 10 or distance == 0) {
    Serial.println("No Obstruction");
    prox = true;
  } else {
    Serial.println("Obstruction");
    prox = false;
  }
}
/******************************************************************************/
/*!
    @brief  Checks for user input (via the Serial Monitor)
*/
/******************************************************************************/
```

7. Finally upload the sketch to the Arduino, and test by using the App which will be developed in a later lab. Show the TA your work.

*NOTE: This lab utilizes Arduino IDE libraries that do not come standard with the Arduino IDE environment. In order for these programs to function properly, these have to be downloaded and inserted into the libraries folder within the system files. A list of the required libraries can be found in the downloads section. These will be installed on the lab computers, but will have to be installed if a student is using their personal computer.

Additional resources

- Arduino is open source which means there is lots of information out there. To start you can browse https://www.arduino.cc/en/Tutorial/HomePage to find tutorials and extra information.
- Adafruit also has lots of tutorials on many different types of Arduinos and sensors, https://learn.adafruit.com/.