Fundamentals of Applied Microcontrollers Laboratory Manual

Seth McNeill

Edition Spring 2023 (v0.5)
2023 September 25

Contents

Chapter 1

Introduction

This book is the accompanying lab manual to a class introducing microcontrollers to upper division, non-electrical engineering undergraduate students who have taken some C programming.

If you find this useful, please let me know. If you find any errors, areas that need improvement, or have any improvements to add please let me know.

The class textbook/manual is here.

1.1 License

Chapter 2

Arduino Startup

2.1 Installing the IDE

We want to try installing the IDE at least two different ways. First, on the lab computer, then on your personal computers if you have them. Both lab partners should try installing the software on the lab computer, each with their own login.

2.1.1 Lab Computer

This method may also work on your personal computers.

1. On the search bar, type in Microsoft Store.
2. Click on Microsoft Store
3. In the store search, type arduino
4. If you have the option to choose between version 1.x and 2.x, use 1.x.
5. Arduino IDE App should appear, click on it
6. Click on Get
7. You do not need to sign into Microsoft to make this install work even if prompted
8. Once it has installed, run the Arduino IDE app.

9. It should load up with a window that looks like Figure 2.1.

10. Try rebooting the computer, logging in and seeing if the Arduino app is still present.

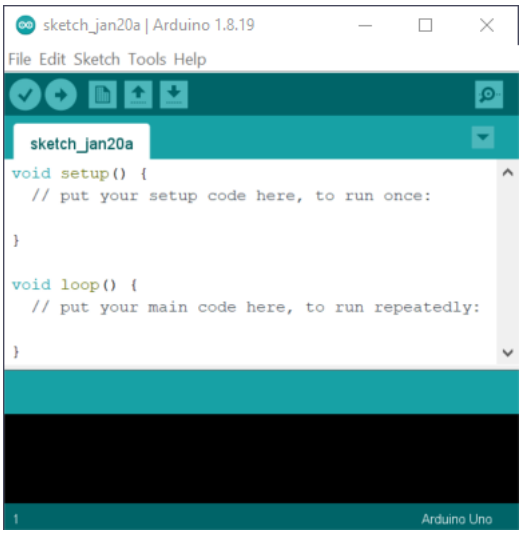

Figure 2.1: This is what the Arduino IDE should load up to.

2.1.2 Personal Computer

1. Go to software download page: https://www.arduino.cc/en/software

2. Scroll down to Legacy IDE (1.8.x)

3. Download the Windows ZIP file (not the first link or the app) for 1.8.19

4. Open the zip file and copy the folder inside (arduino-1.8.19 as of this writing) into your One Drive folder. This may take a while. If you are on your own computer, you can use any of the programs.

5. Once that transfer finishes, go into the folder and run arduino.exe. Windows will try to save you, but if you click More Info you can click Run Anyway.

6. Windows Defender Firewall will also complain. Uncheck the box that is checked and/or click Cancel.

7. It should load up with a window that looks like Figure 2.1.

2.2 Testing the Setup

2.2.1 Installing the Board Drivers

1. In order to get it to connect correctly to your board, you need to install the Arduino Nano Connect RP2040 board.

   (a) Navigate to Tools→Board: “Arduino Uno" (or similar)→Boards Manager

   (b) It should load as shown in Figure 2.2.

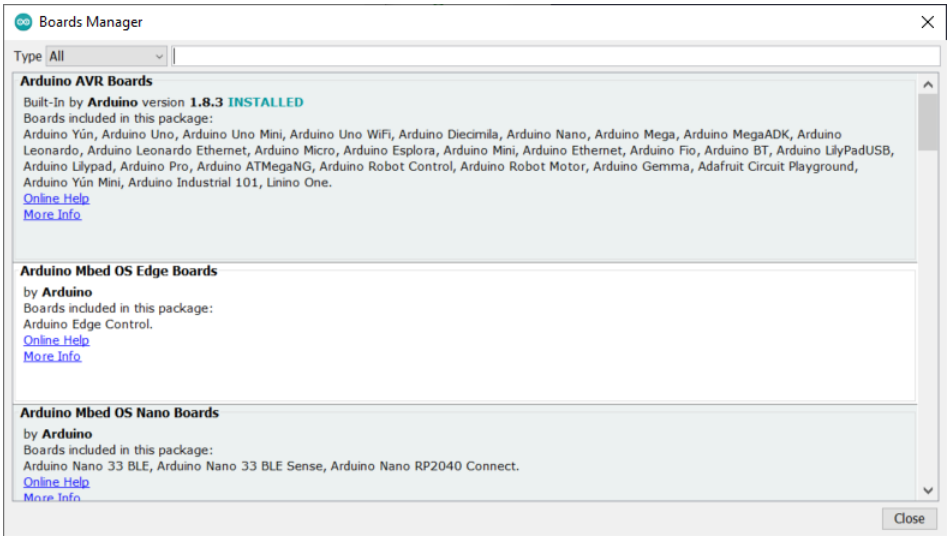

Figure 2.2: This what the Boards Manager loads up to.

   (c) In the search bar, type “arduino nano connect" (without the quotes)

   (d) The first item should be Arduino Mbed OS Nano Boards and should list the Arduino Nano RP2040 Connect.

   (e) Move your cursor over it and it should show an Install button. Click it to install the board library.

(f) Wait for it to finish.

(g) While you are waiting, plug your Nano Connect into your computer and let it install it.

(h) As it finished, I received a User Account Control warning asking if I wanted to let dpinst-amd64.exe make changes to my device. I said yes.

(i) Next it asked me if I wanted to install Arduino Universal Serial Bus devices. Again, click to Install.

(j) It popped up again and I clicked Install again. Now it should say that the Arduino Mbed OS Nano Boards has been installed.

(k) Close the Boards Manager.

2.2.1.1 Compiling, Uploading, and Running

This section can be done on either (or both) computers. The results from one run (between both lab partners) is all that needs to be turned in.

1. Now go to Files→Examples→01.Basics→Blink.
2. This will open another window with the Blink program.
3. Go to Tools→Board→Arduino Mbed OS Nano Boards and select the Arduino Nano RP2040 Connect
4. Go to Tools→Port and select the COM that isn't COM1 (mine showed up as COM5)
5. Click the right arrow under the word edit in the menu to Upload the sketch to the Arduino board.
6. It should say "Compiling sketch..." in the lower left and show a progress bar on the lower right.
7. Then it should switch to Uploading... and finally Done Uploading.
8. An orange light near the USB port on your board should be blinking.
9. Congratulations! You have programmed your board!

10. Now look in the program for the two delay statements. Try changing the values inside the parentheses and re-uploading it. Does the blinking change?

11. In order to save files and have it portable, you need to change the directory where the Arduino IDE stores it's sketchbooks

    (a) Go to File→Preferences

    (b) Change the Sketchbook location to your OneDrive and a folder named arduino (lowercase is good)

    (c) My OneDrive was in C:\Users\mcneils2\OneDrive - Embry-Riddle Aeronautical University\arduino

12. Now try saving the blink sketch with your changed values.

13. Demonstrate your working blink and it's storage location to your instructor/TA

14. Here are some other Examples to test:

    (a) Basics → fade: change the variable `led` to have the value `LED_BUILTIN`, watch the red/orange LED pulse

    (b) Digital → DigitalInputPullup: Change the first pinMode call to use A0 instead of 2. The same for the digitalRead command (2→A0). Press the right button (SW1) and see the LED blink. Note that this program isn't written as well as the others since you have to change a number in two places. Could you rewrite it better?

15. Finally, create a sketch called `getIDs` using the code at https://github.com/semcneil/CEC325Examples/blob/main/getIDs/getIDs.ino

16. You will need to install two libraries to make this script run.

    (a) Go to Sketch → Include Library → Manage Libraries

    (b) Install the ArduinoECCX08 and OneWireNG (only version 0.12.2 or earlier) libraries

17. Run `getIDs` and submit the results in the end of lab Canvas quiz. The results will be in the Serial Monitor which can be accessed through Tools→Serial Monitor or the button in the top right of the IDE. Don't forget that `getIDs` requires two libraries:
    (a) ArduinoECCX08
    (b) OneWireNg version 0.12.x. Version 0.13.x doesn't work

2.3 Turn In

1. Make sure that the TA or instructor has signed off on your modified blink sketch.
2. Submit a PDF of your edited blink.ino and the .ino of your blink sketch. The easiest way to create a PDF is to print to PDF. Only one submission per group is required but make sure that both your names are on the sketch.
3. Fill out the end of lab quiz prior to leaving. Note that it includes asking you for the output of the `getIDs` sketch. Both group members should complete this quiz.

Chapter 3

Multiplexer, LED Display, Binary, HEX

3.1 Purpose

The goal of this lab is to use binary and hexadecimal numbers in an applied setting. This is achieved by having you (the student) use the Arduino Nano Connect RP2040 to setup the PCA9535 I^2C Input/Output (IO) port chip to drive a 4-digit LED display.

3.2 Resources

1. PCA9535 datasheet
2. LED display datasheet
3. PCB Schematic and Layout - see class manual in the Arduino Startup → Schematics and PCB section

3.3 Procedure

3.3.1 Add the PCA95x5 library

In Sketch → Include Library → Manage Libraries search for PCA9535 and install the latest version of the one by hideakitai.

3.3.2 Turn on some LEDs

An example script is shown in Listing 3.1. Use this as a starting point for your code. Be sure to change the header information to include all lab partner names and the correct date. This script displays an 8. in the first digit and zeros in the rest. Note the clearing lines to make it so that there is not ghosting of numbers to the right of where they are displayed. Remember that we are using the PCA9535 chip which requires the PCA95x5 library. When using a library the following steps must be followed or else the microcontroller will probably crash or at least the part you are trying to use will not work. Note the steps in Listing 3.1.

1. Include the library: `#include <PCA95x5.h>`
2. Create an object: `PCA9535 LEDmux;`
3. Initialize the object: usually a `.begin()` method (PCA9535 requires more)
4. Use the object: `LEDmux.write(0xFF0E);`

Now to begin showing some numbers on the 7-segment display.

1. Run the script in Listing 3.1 to make sure it runs. Beware of copying multiple lines from a PDF to the Arduino IDE since it often adds in unwanted characters. Also, underscores (_) and some other symbols sometimes fail to transfer correctly. The is also available for download from the class Canvas site.
2. Change the script so that it displays four consecutive numbers such as 1234. You can use any 4 consecutive, single digit numbers.
3. Demonstrate your sketch working to a TA or instructor.

```
/* 20220907LEDsClass1.ino
*
* Demo of LED display in class.
*
* Seth McNeill
* 2022 September 07
```

```
 */

#include <PCA95x5.h>  // include library

PCA9535 LEDmux;  // create instance (object) of
library

void setup() {
  Serial.begin(115200);
  while(!Serial);
  Serial.println("Starting...");

  // initialize object
  Wire.begin();  // this must be done before LEDmux
.attach
  LEDmux.attach(Wire, 0x21);  // 0x21 is the I2C
address for the PCA9535 attached to the LEDs
  LEDmux.polarity(PCA95x5::Polarity::ORIGINAL_ALL);
  uint16_t mux_direction = 0x0010;  // mostly
outputs (0), setting to 1 designates input
  LEDmux.direction(mux_direction);

  Serial.println("Everything setup");
}


void loop() {
  int delayTime = 0;
  // use object
  LEDmux.write(0x000F);  // required to remove
ghosting on other digits
  LEDmux.write(0xFF0E);
  delay(delayTime);
  LEDmux.write(0x000F);  // required to remove
ghosting on other digits
  LEDmux.write(0x3F0D);
  delay(delayTime);
  LEDmux.write(0x000F);  // required to remove
```

```
  ghosting on other digits
    LEDmux.write(0x3F0B);
    delay(delayTime);
    LEDmux.write(0x000F);  // required to remove
  ghosting on other digits
    LEDmux.write(0x3F07);
    delay(delayTime);
  }
```

Listing 3.1: This listing is a starting point for driving the LED display. This sketch may also be available on Canvas. Beware of copying out of PDFs since some characters (underscore for instance) come through garbled.

Note the anti-ghosting lines in Listing 3.1. This is because the two bytes on the PCA9535 do not change simultaneously. One will change before the other giving a slight showing of the previous number before showing the new number. The fix is to turn off all four segments before writing the new number.

3.3.3 Count

Make a new sketch, based off your first sketch (meaning copy and paste it into the new sketch). This new sketch should count up from 0 to 9 (or 0000 to 0009) incrementing once per second. Demonstrate this counting to a TA/instructor and get it signed off.

3.3.4 Extra credit: 2 pts

Have your counting system count in increments of 1 using more than one digit (i.e. count to 99 if two digits).

3.3.5 Extra Credit Hints

1. Use the `millis()` function to update count every second. The following `if` statement shows an example of how to do this:
   `if(millis() - lastUpdate > updateInterval)`
2. I used an array for the segments required for each number something like
   `nums[] = {0xAB,0x1C,...}` // values not correct

3. I also used an array to specify which digit is showing, something like `digs[[] = {0x12, 0x0E,...}` // values not correct

4. These can be combined since `nums[ii]` is the upper byte and `digs[jj]` is the lower byte of the number that needs to be written to the PCA9535. `nums[ii]` needs to be shifted to be the upper byte using the << operator. For example:
`(nums[ii] << 8) | digs[jj]`

3.4 Turn In

Turn in the following:

1. Make sure that you have been signed off for both consecutive numbers and counting.

2. A PDF of your first sketch that displays 4 consecutive numbers.

3. A PDF of your second sketch that counts.

4. .ino versions of both sketches.

5. Fill out the end of lab quiz prior to leaving. Note that it includes asking you for the output of the `getIDs` sketch.

Chapter 4

Buttons and Serial (UART)

4.1 Purpose

The goal of this lab is to gain a better understanding of the serial interface between the board and the computer, get the buttons all working, and add a few other fun interfaces.

Note that the most time consuming part of this lab in 2022 was getting the non-SW1 switches working.

4.1.1 Serial Library

4.1.1.1 Initialization

So far we have used the Serial library without much explanation of how to use it best. In the sketches we have used we just have used the two lines shown in Listing 4.1 to start Serial.

```
Serial.begin(115200);
while(!Serial);
```

Listing 4.1: If serial is required for a sketch this method of starting Serial blocks until a serial monitor is started.

However, if a serial connection to the computer is not required, the code in Listing 4.1 prevents the rest of the sketch from executing. The Serial library does take some time to start so a simple solution is to just delay for a few seconds after calling `Serial.begin()` as shown in Listing 4.2.

```
Serial.begin(115200);
delay(3000);
```

Listing 4.2: If serial is not required for a sketch this method of starting Serial waits a bit in hopes it connects.

If you want to have the program quit waiting as soon as the Serial connects but not delay forever like in Listing 4.1, you can keep checking for a serial connection a fixed number of times as shown in Listing 4.2.

```
const int maxNoSerial = 300;
int noSerialCount = 0;
while(!Serial && noSerialCount < maxNoSerial) {
  delay(10);
  noSerialCount++;
}
```

Listing 4.3: This snippet tries connecting to Serial a fixed number of times so that it will delay less than Listing 4.2 if a serial connection exists.

4.1.1.2 print vs println

If you want to print a string with an endline, use `Serial.println()`. This is handy for statements like `Serial.println("Starting...")`. However, if you want to print out the value of variables you often don't want newline at the end of the print. For this you use `Serial.print()`. Some examples are shown in Listing 4.4.

```
Serial.print("Number of Names: ");
Serial.println(nNames);
...
Serial.print(F("You're connected to the network,
IP = "));
Serial.println(WiFi.localIP());
```

Listing 4.4: This snippet shows using print and println

4.1.1.3 Reading data from the Serial

The serial connection goes both ways. Information can be sent from the computer to your board. Listing 4.5 is an example of using input from the serial

port. The call to `Serial.available()` tells the sketch whether or not any characters have been received by the serial port. If characters have been received by the serial port, `Serial.read()` reads them in one character at a time. The `switch` function allows different responses depending on what character is received.

Note the layout of the sketch. It is important to define the pins for accessories so that they can be used later.

```
 /*  CEC325-SerialRcv.ino
 *
 *  Demonstrates receiving and reacting to serial
input.
 *  Uses the ERAU CEC325 board v0.3 which has an
 *  Arduino Nano RP2040 Connect on board.
 *
 *  Seth McNeill
 *  2022 February 09
 *  2022 September 20 modified for v0.5 board
 *
 *  This code in the public domain.
 */

#include <WiFiNINA.h>  // for RGB LED

// pin definitions:
#define RIGHT_BUTTON_PIN  A0
#define BUZZ_PIN          2 // buzzer


void setup() {
  // initialize serial:
  Serial.begin(115200);
  while(!Serial) delay(100); // Require serial
  Serial.println("Starting...");

  // make the pins outputs:
  pinMode(LEDR, OUTPUT);  // WiFiNINA RGB LED red
  pinMode(LEDG, OUTPUT);  // WiFiNINA RGB LED green
```

```
  pinMode(LEDB, OUTPUT);  // WiFiNINA RGB LED blue
  pinMode(RIGHT_BUTTON_PIN, INPUT);
  pinMode(LED_BUILTIN, OUTPUT);
  pinMode(BUZZ_PIN, OUTPUT);  // buzzer pin is
output
}

void loop() {
  while(Serial.available() > 0) {
    // characters have been received
    char inChar = Serial.read();
    switch(inChar) {
      case 'a': digitalWrite(LED_BUILTIN,HIGH);
break;
      case 'A': digitalWrite(LED_BUILTIN,LOW);
break;
      case 'b': digitalWrite(LEDB, HIGH); break;
      case 'r': digitalWrite(LEDR, HIGH); break;
      case 'g': digitalWrite(LEDG, HIGH); break;
      case '+': add(); break;
      case 'o': ledsOff(); break;
      case 'z': tone(BUZZ_PIN, 1000, 100); break;
      case '\n': break;
      default: Serial.print("Unknown character: ");
 Serial.println(inChar);
    }
  }
}

// turns off all 4 LEDs
void ledsOff() {
  digitalWrite(LED_BUILTIN,LOW);
  digitalWrite(LEDR,LOW);
  digitalWrite(LEDG,LOW);
  digitalWrite(LEDB,LOW);
}

// Adds characters received subsequent to +
```

```
void add() {
  Serial.println("Adding single digit numbers");
  int a = Serial.read() - 48; // subtract off value
 of ASCII 0
  int b = Serial.read() - 48; // subtract off value
 of ASCII 0
  Serial.print(a);
  Serial.print('+');
  Serial.print(b);
  Serial.print('=');
  Serial.println(a+b);
}
```

Listing 4.5: This sketch shows controlling parts of the board using input from the serial port. Remember, don't copy and paste from a PDF since that process garbles some of the characters.

4.1.2 Buttons

4.1.2.1 I/O Setup

To use general I/O pins on a processor in Arduino the pin has to be defined as an input or an output. This is done in the `setup()` function using the `pinMode()` function. It has the form of `pinMode(pinNumber, direction)`. The pinNumber is typically the number defined in the Arduino specifications as D1 or A0. If it is a D1 type number, then just pass the number to pinMode. If it is an A0 type number, you have to specify both the letter and the number. So for the righthand button (SW1) on the v0.5 board that is attached to the A0 pin you would use `pinMode(A0, INPUT)` since it is an input. The buzzer is attached to pin D2 so we would set it up as `pinMode(2, OUTPUT)` since it is an output.

4.1.2.2 Reading and Writing Pins

Once the pinMode has been setup, you can read the value from an input using the `digitalRead(pinNum)` function which takes a pin number as an argument and returns a 1 or 0 (LOW or HIGH) depending on the read value.

To write a value to an output pin, use the `digitalWrite(pinNum, value)` function. It also takes a pin number as an input along with whether you want the output LOW or HIGH.

Since you will be using the pin number in multiple places, it is best to define it with a name at the top of the program so that you can change it at all places in your program by changing it once.

Examples of writing pins after setting their mode can be seen in Listing 4.5.

4.1.2.3 Other buttons

The v0.5 board has 4 buttons. SW1 is attached to A0 and can be read directly using `digitalRead`. The other three buttons (SW2, FRONT_SW, and BACK_SW) are connected to the PCA9535 I2C to GPIO chip with an address of 0x20 and have to be accessed using the PCA95x5 library.

Remember to install the PCA95x5 library as follows: In Sketch → Include Library → Manage Libraries search for PCA9535 and install the latest version of the one by hideakitai.

1. Include the PCA95x5 library
2. Create a global variable of type PCA9535 named something like `muxU31` (for the chip listed as U31 on the schematic)
3. Initialize the variable/object inside the `setup()` function:
   (a) Start the wire library: `Wire.begin()`
   (b) Attach to the mux using the attach method and correct address: `muxU31.attach(Wire, 0x20)`
   (c) Set the polarity: `muxU31.polarity(PCA95x5::Polarity::ORIGINAL_ALL)`
   (d) Set the direction: `muxU31.direction(0x????)` remembering that a 1 is an input and a 0 is an output. Set non-connected pins as outputs. Change the question marks appropriately by looking at the schematics in the Class Manual linked to in the Resources Section. Note that the only outputs on U31 are Prox1ALRT and SHT31_RST. Set pins that are not connected as outputs too.

4. Once the object is properly setup, read values with the `read()` method. This returns a 16-bit number representing the 16 inputs on the chip.

5. A quick way to check if a bit is high is to bitwise AND (&) it with a number where only the bit of interest is 1: `mux31.read() & 0x0004` reads the third input.

Note that switches SW1 and SW2 are are HIGH when not pressed and LOW when pressed while FRONT_SW and BACK_SW are LOW when not pressed and HIGH when pressed.

```
uint16_t muxVal = muxU31.read();
if(muxVal & 0x0001) {
    // Do something with left button
}
if(muxVal & 0x0002) {
    // Do something with another button
}
```

Listing 4.6: The buttons attached to the PCA9535 can be accessed as shown in this code snippet.

4.1.3 Making Noise (buzzer)

The board for lab has a buzzer attached to pin D2. Arduino has a handy function called `tone(pin, frequency, duration)`. This is an easy way to make your board beep. The duration is in milliseconds and the function blocks until the duration expires. Note that there is another version of the function with the form `tone(pin, frequency)` that does not specify a duration. This is a non-blocking way to make tones. It will continue to make the tone until a call to `noTone(pinNum)` is called.

NOTE: If the tone function is all that is in a loop, it needs a short delay (10 ms will do) after it to keep the Nano Connect RP2040 from crashing.

4.1.4 NeoPixels

The board has 18 multicolor LEDs around its periphery. These are called NeoPixels by the Adafruit company. They get called other things by other companies. They are all based on the WS2812 type chip. We will use the

Adafruit NeoPixel library. Be sure to install it in the usual manner in the Library Manager. It requires the usual:

1. #include the library
2. Initialize an object
3. Setup the library
4. Use the library

A minimal sketch to do all these is shown in Listing 4.7

```
/* NeoPixelSetup.ino
*
* A minimum sketch to setup NeoPixels.
*
* Seth McNeill
* 2022 September 20
*/

#include <Adafruit_NeoPixel.h> // NeoPixel library

#define NEO_PIN           17  // WARNING! THIS IS
GPIO NOT D NUMBER for NeoPixels
#define NEO_COUNT         18  // number of
NeoPixels

// Declare our NeoPixel strip object:
Adafruit_NeoPixel strip(NEO_COUNT, NEO_PIN, NEO_GRB
 + NEO_KHZ800);

void setup() {
  Serial.begin(115200);

  // Setup NeoPixels
  strip.begin();
  strip.clear();  // Set all pixel values to zero
  strip.show();   // Write values to the pixels
}
```

```
void loop() {
  for(int ii = 0; ii < NEO_COUNT; ii++) {
    // set all the pixels to purple
    strip.setPixelColor(ii, strip.Color(255,0,255))
;
  }
  strip.show();
  delay(1000);
  // turn off all the LEDs
  strip.clear();
  strip.show();
  delay(1000);
}
```

Listing 4.7: This snippet shows how to setup and run the NeoPixels on the board.

4.2 Resources

1. Serial Library
2. digitalRead
3. tone
4. notone
5. Adafruit NeoPixel Library
6. PCA9535 datasheet
7. PCA9535 library (McNeill version)
8. PCB Schematic and Layout - see class manual in the Arduino Startup → Schematics and PCB section

4.3 Procedure

Write a sketch with the following functionality:

1. Reacts to all 4 buttons on the board in some way such as sound, light, or Serial. THIS IS THE HARDEST PART. START HERE.
2. Reads in values from the serial port and does something in response.
3. Writes information to the serial port in response to some stimulus/stimuli.
4. Creates a tone in response to some stimulus.
5. Make the NeoPixels do something.

4.4 Debugging

Here are some pointers to help if your program gives errors:

1. Missing `Wire.h`: Make sure you are including the PCA95x5 library
2. Upload just gives a series of dots and says upload failed.
   (a) Check to make sure the correct COM port is selected in Tools menu
   (b) Try pressing the reset button twice to put the RP2040 in upload mode
   (c) If nothing works, bring your board to the professor/TA to have a hard reset performed
3. "Out of scope" errors. Count and match curly brackets ({}) to make sure all your code is inside `setup()` or `loop()`. The only code that can be outside of a function is a variable declaration (e.g. `int x`)

4.5 Turn In

Turn in the following:

1. Make sure that the TA/Instructor signs off on your sketch demonstration.
2. A PDF of your sketch.
3. .ino versions of your sketch.
4. Fill out the end of lab quiz prior to leaving. Note that it includes asking you for the output of the `getIDs` sketch.

Chapter 5

Displays

5.1 Purpose

The goal of this lab is to learn to use the TFT display. Debouncing the buttons so that an `if` statement only executes once per press takes the most time.

```
int lastButtonVal = 1;

void loop() {
    int curButtonVal = digitalRead(RIGHT_BUTTON_PIN);
    if((lastButtonVal != curButtonVal) && !
   curButtonVal) {
        // do something when button pressed
    }
    lastButtonVal = curButtonVal; // very important
   this is outside the if and inside loop()
}
```

Listing 5.1: This is example code for debouncing a button.

There is code on Canvas that demonstrates some display capabilities, but do NOT use the `rButtonWait()` function.

5.2 Procedure

5.2.1 Main Requirements

Write a sketch with the following functionality:

1. Display a custom message for 3 seconds indicating the start of the program when your program starts
2. Choose a favorite character and have it move left 10 pixels for each press of the left button and right 10 pixels for each press of the right button.
   (a) Choose the Y value to be one that looks good to you
   (b) If 10 pixels seems to not be a good value feel free to change it, just note that you changed the value
   (c) This builds on last week's making all the buttons function
   (d) Debounce the buttons so that each press only moves the character once
   (e) Add logic so that the character doesn't go far off the edge of the screen on either side.
3. Draw some (at least 3, but they don't all have to be different) of the drawing primitives (line, circle, rectangle, etc.)

5.2.2 Extra Credit

The following are extra credit options with increasing value:

1. Make the front and back switches move your character up and down along with the left and right from the Main Requirements. (1 point)
2. Make an animation of some kind that lasts at least two seconds. (1 points)
3. Make a game that uses the buttons and the screen (2 points)

5.3 Turn In

Turn in the following:

1. Make sure that the TA/Instructor signs off on your sketch demonstration.
2. A PDF of your sketch.
3. .ino versions of your sketch.
4. Fill out the end of lab quiz prior to leaving. Note that it includes asking you for the output of the `getIDs` sketch.

5.4 Resources

1. Adafruit 1.14" 240x135 Color TFT Display + MicroSD Card Breakout - ST7789
2. Adafruit ST7789 library
3. Adafruit GFX library
4. See the chapter in the class manual about displays
5. PCB Schematic and Layout - see class manual in the Arduino Startup → Schematics and PCB section

Chapter 6

Environmental Sensing

6.1 Purpose

The goal of this lab is to learn to collect data with an ADC and other sensors.

6.2 Procedure

6.2.1 Main Requirements

Write a sketch with the following functionality:

Collect the following data and display it once every 5 seconds via serial and on the display with the appropriate labels and units if you can make them fit.

1. The output from `millis()` (ms)
2. Both light intensities as a number between 0 and 1023 (unitless)
3. Potentiometer value as a voltage between 0 and 3.3 V
4. Temperature in Fahrenheit from TEMP0 (U27 attached to input 0 of U37)
5. Get SHT31 reporting temperature in F and humidity via Serial
6. Add SHT31 data to display
7. Get compass (QMC5883L) reporting azimuth via Serial

8. Add azimuth data to the display
9. Get IMU (LSM6DSOX) reporting accelerations and gyroscope data via Serial
10. Add accelerometer and gyroscope data to display
11. Get 1-Wire (DS18B20) sensor(s) reporting temperature(s) via Serial
12. Display 1-Wire temperature(s) on display

6.2.2 Suggestions

Use a String object to accumulate your display string and then call `display.println(yourString)` to display it. Note a few things:

1. The String type starts with a capital S.
2. You can add to the String object using += or just +, but with only the plus operator, all arguments have to be of the same type.
3. The String object also allows you to limit the number of decimal places for `float` types. `String(tF,1)` displays tF to 1 decimal place.

An example is shown in Listing 6.1.

```
String dispStr;
dispStr = "T(F): ";
dispStr += String(tF,1);
dispStr += "\n";
// Control the display
tft.fillScreen(ST77XX_BLACK);  // clear display
tft.setTextColor(ST77XX_YELLOW);  // set text color
tft.setTextSize(1);  // Normal 1:1 pixel scale
tft.setCursor(0,0);  // Start at top-left corner
tft.println(dispStr);
```

Listing 6.1: This is an example of using a String object to display text and float variables. The floats are limited to 1 decimal place such that 7.123 would be displayed as 7.1.

However, if you want to have different portions of the text in different colors you will need to call `tft.setTextColor` between each `tft.println`.

6.2.3 ADS7142 - TEMP0, POT, LIGHT1, LIGHT2

The light sensors, potentiometer, and temperature sensor are connected to ADS7142 analog-to-digital (ADC) sensors. There is a library in the normal library adding method. Look for the library for the ADS7142 by Seth McNeill. After you install it, there should now be examples under Anitracks ADS7142. Follow the read2Ch example combined with your previous work to complete this lab.

The light sensors are CdS light reactive resistors setup as resistor dividers. They are located in the bottom left and right corners of the circuit board and look like roundish gray with red lines on them. We tried putting mirrors over some of them to try using them as distance sensors, but that didn't work well so many of the mirrors have broken off leaving some hot glue residue.

For converting to voltage, note that the ADS7142 is a 12-bit ADC but returns values as 16-bit which means that the maxADCValue is 65535. The reference voltage is 3.3 V.

As a contrast, if you read the voltage from A6 and A7 and use analogRead, the maxADCValue is 1023 since they return 10-bit values and the reference voltage is 1.1 V.

6.2.4 SHT31 Temperature and Humidity Sensor

Use Adafruit's SHT31 library. Base your code on their example. Be sure to make sure that the heater is off. For this class, you do not need to turn on the heater.

Also, the SHT31 needs to be reset each time the program starts. It's reset pin is connected to a PCA9535. An example of how to do this is shown in Listing 6.2.

```
Wire.begin();
muxU31.attach(Wire, 0x20);
muxU31.polarity(PCA95x5::Polarity::ORIGINAL_ALL);
muxU31.direction(0x1CFF);  // 1 is input, see
   schematic to get upper and lower bytes
muxU31.write(PCA95x5::Port::P10, PCA95x5::Level::L);
   // disable SHT31
delay(100);
muxU31.write(PCA95x5::Port::P10, PCA95x5::Level::H);
   // enable SHT31
```

```
delay(100);

Serial.println("SHT31 test");
if (! sht31.begin(0x44)) {   // Set to 0x45 for
   alternate i2c addr
    Serial.println("Couldn't find SHT31");
    while (1) delay(1);
}
```

Listing 6.2: This listing shows how to reset the SHT31.

6.2.5 QMC5883L Compass

Use the library by MPrograms for the QMC5883L compass. The azimuth example should provide what you need to make it go.

6.2.6 LSM6DSOX IMU

Install the Arduino library for the LSM6DSOX IMU. Your program will need to use a combination of the SimpleAccelerometer and SimpleGyroscope examples. NOTE: This library returns acceleration in g's not m/s^2.

6.2.7 1-Wire Sensors

As mentioned in class, the only library I have found to work with the Nano Connect RP2040 is the OneWireNg library. This library has to be of a version less than 0.13 as of 2023 September 21. Unfortunately, the library examples are very complex. There is a simpler solution by combining the examples from the OneWire library with the capabilities of the OneWireNg library. It has to be done carefully though. Here is how to do it:

1. In the Arduino IDE install both the OneWire and OneWireNg libraries
2. Open the OneWire example named `DS18x20_Temperature`
3. Make the first `#include` in your file be:
   `#include "OneWireNg_CurrentPlatform.h"`
4. This example should now compile and run just fine

6.2.7.1 NOTE

This example reads the results from one (1) sensor per pass through the `loop()` function. For your program, you will want to collect all three sensors' data in a single pass through the `loop()` function.

Start by just reading one DS1820 then after you have everything else working, come back and add in the rest of the DS1820 sensors.

6.3 Main Requirements

Write a sketch with the following functionality:

Collect the following data and display it once every 5 seconds via serial and on the display with the appropriate labels and units if you can make them fit.

1. The output from `millis()` (ms)
2. Both light intensities as a number between 0 and 1023 (unitless)
3. Potentiometer value as a voltage between 0 and 3.3 V
4. Temperature in Fahrenheit from TEMP0 (U27 attached to input 0 of U37)
5. The temperature in Fahrenheit and humidity from the SHT31 sensor
6. The compass azimuth angle
7. The 3-axis accelerometer and gyroscope data (6 data points)
8. The temperatures from at least one 1-Wire DS18B20 sensor (can take about seconds for acquisition for 3 sensors). Do this part last.

6.4 Turn In

Turn in the following:

1. Have either the TA or the instructor sign-off on your lab
2. A PDF of your sketch.

3. .ino versions of your sketch.
4. Fill out the end of lab quiz prior to leaving. Note that it includes asking you for the output of the `getIDs` sketch.

6.5 Resources

1. Adafruit 1.14" 240x135 Color TFT Display + MicroSD Card Breakout - ST7789
2. Adafruit ST7789 library
3. Adafruit GFX library
4. The SHT31 temperature and humidity sensor datasheet
5. The QMC5883L compass datasheet
6. The LSM6DSOX IMU datasheet
7. See the chapter in the class manual about displays
8. The DS18B20 temperature sensor datasheet
9. PCB Schematic and Layout - see class manual in the Arduino Startup → Schematics and PCB section

Chapter 7

IMU

7.1 Purpose

The goal of this lab is to learn to use the IMU data and the SD card.

7.1.1 IMU Angle Measurement

The user manual has the equations for calculating the current angle of the IMU on one axis using a complementary filter to combine data from the gyroscope and the accelerometers. The equation is copied here as Equation 7.1.

$$\theta_{mixed}[t] = \alpha\left(\theta_{mixed}[t-1] + \omega_{gyro}\Delta t\right) + (1-\alpha)\operatorname{atan}2(a_x, a_z) \tag{7.1}$$

7.2 Main Requirements

Write two sketches with the following functionality:

7.2.1 IMU Screen Control

Use the angle measurement to move a character on the screen as described in the Procedure section.

7.2.2 Plotting IMU Data

Save the IMU data as specified in the Procedure to the SD card, then plot the saved data in Matlab or your other favorite plotting program.

7.3 Procedure

It is best to follow this outline to finish this lab most efficiently.

1. Start by making sure the `IMU-CompFilterEx.ino` example on Canvas works as it should for you.
2. Make it so that the angle controls the movement of a character on the screen. You can use the ideas from the programs you and your partner made for Lab 5.
   (a) -180 degrees puts the symbol all the way to the right
   (b) 0 puts the symbol in the middle of the screen
   (c) +180 degrees puts the symbol all the way to the left
3. As the second part of the lab, load the example named `SDReadWrite`
4. Run it to make sure it works as advertised.
5. Change it to save `t` (from curTime), `ax`, `az`, `theta_g`, `theta_a`, and `theta` to the SD card
6. Notice what `dt` is now.
7. Make of plot of the data in Matlab, Python, or your other favorite program for making good plots.

7.4 Extra Credit: 2 pts

After you have completed all the lab requirements, make it so that the character on the screen moves up and down as well as left and right based on the other axis of the PCB rotation.

7.5 Turn In

Turn in the following:

1. Have either the TA or the instructor sign-off on your lab
2. PDFs of your sketches and your plots of the IMU data.
3. .ino versions of your sketch.
4. The script/file you used to plot the data.
5. Fill out the end of lab quiz prior to leaving. Note that it includes asking you for the output of the `getIDs` sketch.

7.6 Resources

1. The LSM6DSOX IMU datasheet
2. PCB Schematic and Layout - see class manual in the Arduino Startup → Schematics and PCB section

Chapter 8

Distance, Motor, Servo

8.1 Purpose

The goal of this lab is to learn the distance sensor, driving DC motors, and driving servo motors.

8.2 Main Requirements

8.2.1 Distance Calibration

Using an example sketch for the distance sensor, create a calibration curve for the distance sensor out to it's maximum sensing distance using the lid of the shoebox as a target. The calibration curve should use at least 5 points and have the readout (0-255ish) on the x-axis and actual, measured distance on the y-axis. It is fastest to just print the distances out and then record them onto another document rather than trying to save them off to the SD card. Plot the calibration curve in your favorite data analysis software (Python, Matlab, Excel, etc.). Use the Pololu library for the VL6180 sensor and set SCALING to 1 for this part of the assignment. Set SCALING to 3 for everything else.

8.2.2 DC Motors

Write a sketch that drives your robot forward until the distance sensor senses something (you choose a reasonable threshold), and then turns to avoid it

and continues on. Remember that to drive the motors at different speeds, you can just use `analogWrite` on one pin and set the other pin to 0 (`analogWrite(0)`). In order for the motor to actually spin, the number passed to `analogWrite` needs to be greater than 50. The motor pins are listed as `AIN1, AIN2, BIN1, BIN2` which map to 0, 1, 8, and A2, respectively, on the schematic. Do not mix `analogWrite` and `digitalWrite` on the same pin in the same program.

Also, make sure that the motor jumpers are set to bat, not +5V. This makes it so that the big power switch has to be in the ON position for the motors to run. Be sure to switch it on when you want the motors running.

Lastly, be sure to do this with SCALING set to 3 so that the detection distance can be longer.

8.2.3 Servo

Write a sketch where the distance is measured every 10 degrees of servo movement. Plot the data on the robot's TFT display. The plot can either be cartesian, with the x-axis as angle from 0 to 180 and the y-axis the distance measured at the angle, or it can be a polar plot with the angle being theta, and the distance as the radius. The `Servo` library is already installed and not far down in the Examples. Look at the Sweep example to get started.

8.3 Turn In

Turn in the following:

1. Have either the TA or the instructor sign-off on your lab
2. A PDF of your sketches.
3. .ino versions of your sketches.
4. The script/file you used to plot the data.
5. Fill out the end of lab quiz prior to leaving. Note that it includes asking you for the output of the `getIDs` sketch.

8.4 Resources

1. VL6180 Distance sensor
2. TMI8837 Motor controller
3. PCB Schematic and Layout - see class manual in the Arduino Startup → Schematics and PCB section

Chapter 9

Peak Detection

9.1 Purpose

The goal of this lab is to experiment with extrema detection by creating a sketch that has the following two parts:

1. Detect peaks/valleys in data
2. Do something when extrema are detected

9.2 Procedure

1. Install the PeakDetection library from:
   https://github.com/semcneil/AnitracksPeakDetection
2. The example uses input from `A0`, which is attached to SW1 on your board. This should work for initial testing that the code runs.
3. There is an example posted on Canvas that uses the potentiometer on your board, download and make sure this example runs too.
4. Choose which sensor you plan to use (not SW1 or the potentiometer)
5. Choose how you want the board to react to positive and negative extrema
   (a) NOTE: Your board must respond to a peak/valley (extrema), not just a threshold situation.

   (b) For example, if you are using a distance sensor, it must react to something coming closer and then getting further away. It cannot just react to something getting closer like we did in the Distance, Motor, Servo lab.

6. Implement your choices of sensor and reactions
7. Complete what is required to turn in

9.2.1 Possible Sensors

1. IMU (accelerometer and/or gyroscope)
   (a) React to waving board
   (b) React to tipping board
   (c) React to impacts
2. Proximity (look for peak, not threshold situation)
3. Light
4. Temperature
5. Humidity
6. Compass

9.2.2 Example Reactions

You can probably come up with more ideas for reactions.

1. Run buzzer with different `tones`
2. Move servo
3. Change motor speed/direction
4. Change NeoPixels
5. Change LED display
6. Display relevant data on the TFT display

9.3 Turn In

Turn in the following:

1. Have either the TA or the instructor sign-off on your lab
2. A PDF of your sketch.
3. .ino versions of your sketch.
4. Fill out the end of lab quiz prior to leaving. Note that it includes asking you for the output of the `getIDs` sketch.

9.4 Resources

1. StackOverflow post about this algorithm
2. PCB Schematic and Layout - see class manual in the Arduino Startup → Schematics and PCB section

Chapter 10

Machine Learning

10.1 Purpose

The Arduino Nano Connect RP2040 has the LSM6DSOX IMU made by ST Microelectronics. It has a Machine Learning Core inside. It also has some preset systems inside for detecting the following:

1. 6D orientation (up, down, left, right, front, back)
2. Free fall detection
3. Pedometer (counting steps)
4. Tap detection, both single and double tap
5. Tilt detection (for orienting screens as users rotate the device)
6. Wake up detection to let the device (microcontroller) know the device has moved

Today's lab will run each of the built-in systems plus a pretrained ML model.

10.2 Laboratory

10.2.1 Download Examples

There is a zip file on Canvas named LSM6DSOX_Examples.zip. Download and extract the zip file. Move the resulting folder into your sketch directory

(usually named arduino or Arduino inside your Documents folder). Now try opening one of the examples from within the Arduino IDE. If you can find it in your File → Sketchbook menu, you have put them in the right place.

10.2.2 Install Library

Install the STM32duino_LSM6DSOX library in the Arduino IDE since these examples rely on it for the interface to the IMU.

10.2.3 Running Examples

Run each of the following examples noting that the outputs will all be via the serial port:

10.2.3.1 6D Orientation

Open the example named LSM6DSOX_6DOrientation. Compile and upload it to the board. When it runs, it should output a drawing indicating the orientation as you rotate the board.

10.2.3.2 Free Fall Detection

Don't drop or throw the board! Open the example named LSM6DSOX_FreeFallDetection. Compile and upload it to your board. When it is running, raise and lower the board while holding it and it should send an output when it thinks it is in free fall.

10.2.3.3 Pedometer

Open the example named LSM6DSOX_Pedometer. Compile and upload it to the board. Once it is running it should print out the number of steps periodically. Shake the board up and down to imitate walking and it should increment the counter.

10.2.3.4 Tap Detection

Open the example named LSM6DSOX_TapDetect. Compile and upload it to the board. Once it is running try tapping the side of the board/module

(not front, back, or top). The serial should output single tap and double tap as it thinks it is tapped.

10.2.3.5 Tilt Detection

Open the example named LSM6DSOX_TiltDetection. Compile and upload it to the board. Once it is running try tilting the board after letting it sit in one orientation for a bit. It should trigger a serial output when you move the board.

10.2.3.6 Wake Up Detection

Open the example named LSM6DSOX_WakeUpDetection. Compile and upload it to the board. Once it is running try moving the board after letting it sit still for a bit. It should trigger a serial output when you move the board.

10.2.3.7 Machine Learning Example

Open the example named LSM6DSOX_MLC. Compile and upload it to the board. Once it is running try moving the board to simulate walking, running, biking, driving, unknown, and staying stationary. Walking and jogging are the easiest to get with vertical motion (Z-axis). Biking seems to be lateral motion (X- or Y-axis). I have seen driving once but don't remember how I got it. I have never seen unknown.

10.2.4 Using the Examples

Take one or more of the examples and do something with them. It could be as simple as adding sound or screen output to the example(s) or something more.

Have fun!

10.2.5 Extra Credit

Use the two light sensors at the bottom of the board to detect location of a hand over the board. There should be four (4) categories:

1. No hand present

2. Hand on left side
3. Hand on right side
4. Hand in the middle (between the sensors)

You get more extra credit for creating Gaussian (normal) models of each class and use them to find the optimal thresholds between the classes. You also need to plot the data to show the thresholds.

10.3 Turn In

Turn in the following:

1. Have either the TA or the instructor sign-off on your lab
2. A PDF of your sketch.
3. .ino versions of your sketch.
4. Fill out the end of lab quiz prior to leaving. Note that it includes asking you for the output of the `getIDs` sketch.

10.4 Resources

1. LSM6DSOX Library Header
2. LSM6DSOX Datasheet
3. PCB Schematic and Layout - see class manual in the Arduino Startup → Schematics and PCB section

Chapter 11

Controls

11.1 Purpose

The goal of this lab is to play with PID control to get a feel for how it works on a real system.

11.2 Laboratory

11.2.1 Getting started

Example code is posted on Canvas as `pid_dist_v0.5.ino`. Download and try running it. After pressing the right button it should try to maintain a distance from an object in front of the robot. The NeoPixels will be red when it is in PID mode, and green when it is in HALT mode. Turning the potentiometer changes the target distance. In order to have the serial port work at the same time as the motors run do the following:

1. Make sure both motor jumpers are set to BAT
2. Plug in the serial port
3. Turn on the battery switch.

Make sure this code works as expected.

11.2.2 Proportional Control

Set the gain for the integral (KI) and derivative (KD) terms to zero. Try different values for the proportional control. Can you turn proportional control into bang-bang control? How well does it work?

11.2.3 Integral Control

Set the gain for proportional (KP) and derivative (KD) control to zero. Try different values for integral control. How well does it run? Can you see the windup? What happens for large values of KI? What happens for small values of KI?

11.2.4 Derivative Control

Set the gain for proportional (KP) and integral (KI) control to zero. Try different values for KD. What happens with large values? How about for small values?

11.2.5 PID Calibration

Based on the tests you have done so far. Try to find gain values for the full PID controller that work better than the defaults in the original file. How can you tell if your gains are better than the original ones?

11.2.6 IMU PID Control

You have been supplied with a ruler to use as a ramp. Copy the PID file to a new one with IMU in the title. Change it to try to level itself by backing up the ramp. You will need to use your code for measuring angles from a previous lab.

Some other changes that need to be made:

1. The distance PID controller is based on `int` values. The angle is a float, so the PID function and values need to be updated accordingly.

2. The error also needs to be changed to a float.

3. Depending on the angle, the robot may not be able to get onto the ramp so you may need to start it on the ramp. Tip the ramp up and down to provide perturbances to the system.

4. The axes that the code measures around need to be changed to measure the front-to-back angle rather than the side-to-side angle.

Be sure to turn off the battery switch prior to putting the robot away.

11.3 Turn In

Turn in the following:

1. Have either the TA or the instructor sign-off on your lab

2. A short writeup with the answers to the questions in the distance PID section. One per group is fine.

3. A PDF of your sketch.

4. .ino versions of your sketch.

5. Fill out the end of lab quiz prior to leaving. Note that it includes asking you for the output of the `getIDs` sketch.

11.4 Resources

1. LSM6DSOX Library Header

2. LSM6DSOX Datasheet

3. PCB Schematic and Layout - see class manual in the Arduino Startup → Schematics and PCB section

Chapter 12

Wireless

12.1 Purpose

The goal of this lab is to play with WiFi, Bluetooth, and time. Some of the example sketches are in a .zip file on Canvas. Download the zip file and unzip it into the Arduino directory (typically Documents/Arduino). You should see directories with your other sketches inside the Arduino folder.

12.2 Laboratory

12.2.1 Preparation

Before you start this lab, be sure you have updated the following:

1. The Arduino IDE to the latest in the 1.x series (not the 2.x IDE)
2. WiFiNINA module firmware
3. The main libraries used for this lab
   (a) WiFiNINA
   (b) ArduinoBLE
   (c) stm32duino's ST25DV library

12.2.2 WiFi startup

Follow the WiFiNINA No Encryption example to connect your board to the network. Be sure to change the `SECRET_SSID` to have `EagleNet` between the double quotes the `arduino_secrets.h` file.

12.2.3 NFC

The ST25DV library is the one in the Arduino Library Manager written by stm32duino. The library also shows as INCOMPATIBLE even though it works fine. Make sure the example (NFC_Demo.ino) works as it is supposed to. It should cycle different things (website, call, text, etc.) each time you press the button. Put the top of your phone near the coil that says NFC on the bottom left of the robot.

Change the example to show one of your email addresses and the IP address of your Nano RP2040 Connect. Demonstrate this to your instructor/TA.

In your code, after WiFi has connected, convert the IPAddress to a `String` with the following:

```
IPAddress ip = WiFi.localIP();
String ipString = String(ip[0]) + "." + ip[1] + "." + ip
[2] + "." + ip[3];
```

Now that you have the IP address as a String, you can update `NFC_messages` to have the IP address String in the correct position. Also update `NFC_protocols` to use http NOT https.

12.2.4 UDP Between Boards

Run the examples `WiFiUdpSend` and `WiFiUdpReceiveSend`, one on each of 2 boards. Be sure to update `remoteIp` in the `WiFiUdpSend` sketch to be the IP address of the other board. Once running, the right button on the board should turn the LED on the other board on and off.

Modify this to do something else (buzzer, NeoPixels, motor, etc.). Demo for the instructor/TA.

12.2.5 Bluetooth Low Energy

Be sure to generate your own UUIDs for your particular project. If you use the UUIDs from the examples, you will not be able to tell if you are connecting to your board or someone else's who is using the same UUIDs. Use the UUID Generator to create random UUIDs. Use UUIDs in the form returned by the website. If you create custom ones with a different format, the Bluetooth examples fail. Be sure to set the device (`BLE.setDeviceName`) and local names (`BLE.setLocalName`) on all boards to something unique to your group so that you can find it when scanning.

12.2.5.1 Phone to Robot Interface

Use the example code (`BLE_ButtonLED`) (being sure to change UUIDs, device name and local name) to turn an LED on and off on the robot from your phone. Change the code to do something else (buzzer, NeoPixel, motor (carefully), servo) when you change the value via the Bluetooth interface. Also, show the button on the board changing the value read on your phone using the Notify property.

12.2.5.2 Inter-Robot Interface

For this part of the lab you will need to work with another group. Using the example of a peripheral (`BLE-ButtonLED.ino`) and central (`BLE-LED-Central.ino`) , use one robot to control something on the other robot. Note that you need a different UUID for each of the peripheral, LED, and Button characteristics but the UUIDs have the be the same between the two sketches. Also, the LocalName needs to agree between the sketches.

12.2.6 AJAX

There are 3 files on Canvas:

1. AJAX-Robot.ino
2. arduino_secrets.h
3. index.h

Open the .ino file and then put the other files in the same directory. Make sure to add them to the sketch (Sketch→Add File) so that they show up as separate tabs. Upload the file to the robot. Once it has uploaded and the LEDs are green, put your phone next to the NFC and go to the URL listed (you must be on the EagleNet WiFi for it to load). Once you are on the site you can control the builtin LED, the servo position, and the NeoPixel color as well as read the distance measurement.

12.3 Shutdown

Be sure to turn off the battery switch prior to putting the robot away.

12.4 Turn In

Turn in the following:

1. Have either the TA or the instructor sign-off on your lab
2. A PDF of your NFC sketch.
3. .ino versions of your NFC sketch.
4. Fill out the end of lab quiz prior to leaving. Note that it includes asking you for the output of the `getIDs` sketch.

12.5 Resources

1. UUID Generator
2. PCB Schematic and Layout - see class manual in the Arduino Startup → Schematics and PCB section

This explains some of the code you are using.

12.5.0.1 Peripheral Setup

1. Include Arduino's BLE library
   `#include <ArduinoBLE.h>`

2. Create a BLEService. This is container for characteristics. You can have multiple services and characteristics per service.
   `BLEService ledService(String UUID);`

3. Create a variable for each characteristic. Characteristics have properties. The most used properties are:

   (a) BLERead - This allows a connected device to read the value of this variable

   (b) BLEWrite - This allows a connected device to change the value of the variable

   (c) BLENotify - This allows a connected device to be notified if the value changes

   ```
   BLEByteCharacteristic LEDCharacteristic(String UUID
   , BLERead | BLEWrite);
   BLEByteCharacteristic buttonCharacteristic(String UUID
   , BLERead | BLENotify);
   ```

4. The next setup is done in the `setup()` function

5. Start the BLE module
   `BLE.begin()`

6. Set the device name. This is the externally advertised name
   `BLE.setDeviceName("BUTTON_LED");`

7. Set the local name. This can be checked by a device once connected
   `BLE.setLocalName("BUTTON_LED_MCNEILL");`

8. Set the service as advertised
   `BLE.setAdvertisedService(ledService);`

9. Add the characteristics to the service
   `ledService.addCharacteristic(LEDCharacteristic);`

10. Add the service to the BLE object
    `BLE.addService(ledService);`

11. Set initial values for each characteristic
    `LEDCharacteristic.writeValue(0);`

12. Finish by starting your BLE advertising its existence
    `BLE.advertise();`

13. In the `loop()` my preferred method is to create central device and check to see if a device has connected. Once it has, update the values of the output characteristics and check if the writeable characteristics have been written to.

12.5.0.2 Central Setup

1. Scan for a device, typically by service UUID.
   `BLE.scanForUuid(BLE_UUID_PERIPHERAL);`

2. Create a peripheral object
   `BLEDevice peripheral = BLE.available();`

3. If the peripheral has been discovered, check its local name
   `if (peripheral.localName() != "BUTTON_LED")`

4. Stop scanning
   `BLE.stopScan();`

5. Connect to the peripheral
   `peripheral.connect()`

6. Read the peripheral's attributes
   `peripheral.discoverAttributes()`

7. Retrieve each characteristic and check to make sure each has the attributes (Read/Write/Notify) expected

```
if (!buttonCharacteristic) {
    Serial.println("Peripheral does not have
   button characteristic!");
    peripheral.disconnect();
    return;
```

```
} else if (!buttonCharacteristic.canRead()) {
    Serial.println("Peripheral does not have a
   readable button characteristic!");
    peripheral.disconnect();
    return;
} else if (!buttonCharacteristic.canSubscribe()) {
    Serial.println("Peripheral does not allow
   button subscriptions (notify)");
} else if(!buttonCharacteristic.subscribe()) {
    Serial.println("Subscription failed!");
} else {
    Serial.println("Connected to button
   characteristic");
}
```

8. Read/write the peripheral's characteristics as desired

```
if(buttonCharacteristic && buttonCharacteristic.
   canRead() && buttonCharacteristic.valueUpdated
   ()) {
    byte peripheralButtonState;
    buttonCharacteristic.readValue(
   peripheralButtonState);
    if(!peripheralButtonState) {
        digitalWrite(ledPin, HIGH);
    } else {
        digitalWrite(ledPin, LOW);
    }
}
```