IESTI-01

TinyML - Machine Learning for Embedding Devices

TinyML Kit Overview - HW and SW Installation & Test

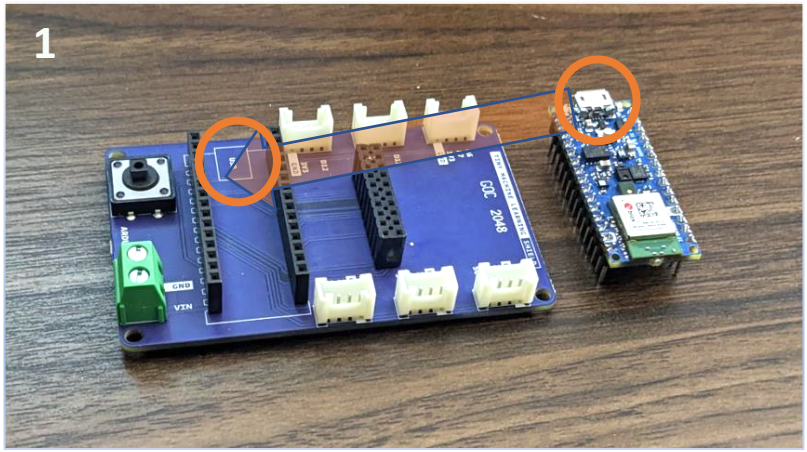
UNIFEI

May 2022

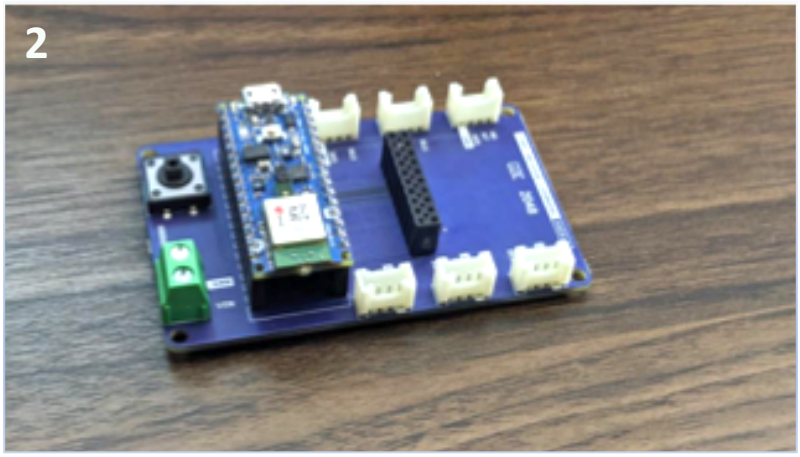
Setting up your Hardware (TinyML Kit)

We outline the required steps below:

Slot the Nano 33 BLE Sense board into the Tiny Machine Learning Shield



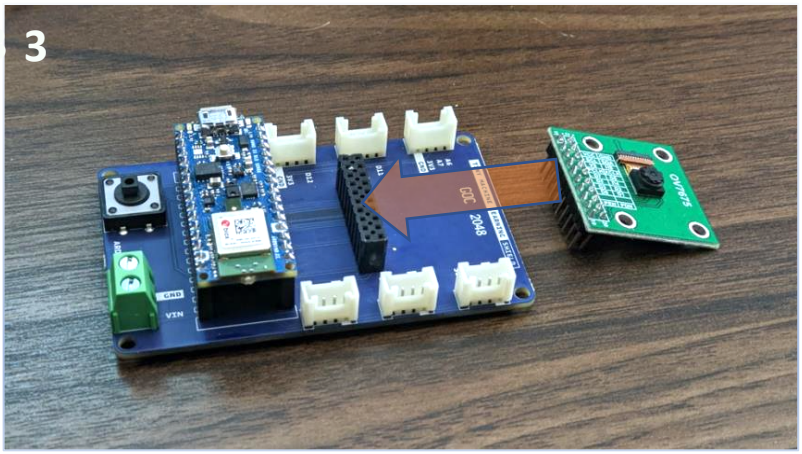
You'll want to target the pair of spatially separated 1x15 female headers. Carefully align the pins of the microcontroller board with the headers below and then gently push down until the board is seated flush against the top of each header. The downward-facing pins should no longer be visible. As best you can, avoid touching the components atop the board to prevent inadvertently damaging the surface mount devices. Pay attention to the orientation of the board so that the indication of the USB port on the PCB silkscreen matches the physical port on the board itself.



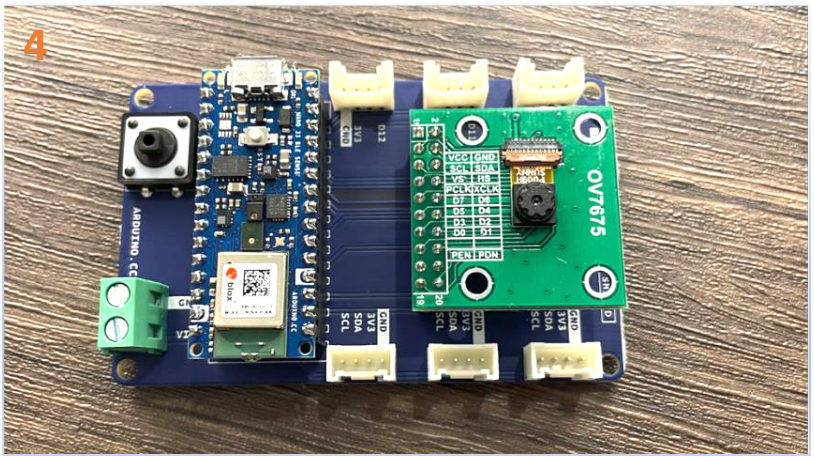
The OV7675 camera module into the shield using the same technique

You'll want to target the 2x10 female header. Carefully align the pins of the camera module with the headers and then gently push down until the board is seated flush against the top of each

header. The downward-facing pins should no longer be visible. As best you can, avoid touching the camera module atop the board to prevent inadvertent damage.



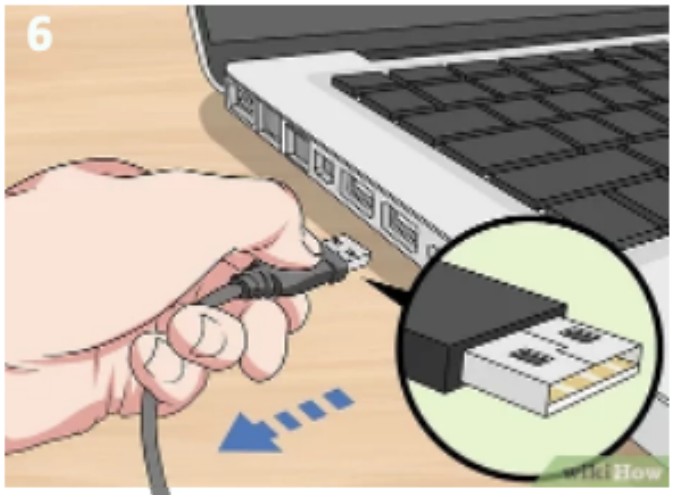
Pay attention to the orientation of the camera module so that the camera sensor is to the right of the header array (as shown), further from the microcontroller board than the header array.

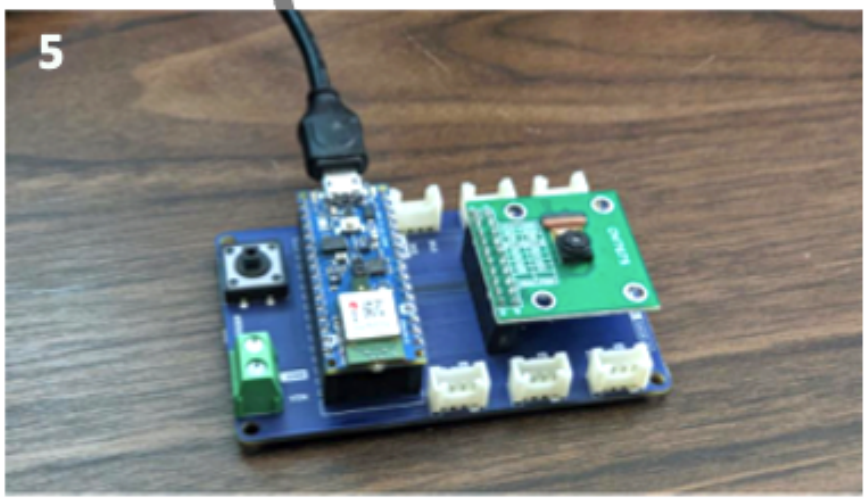


Use the USB cable (type-A to microB) to connect the Nano 33 BLE Sense to your machine.

If your PC only features type-C USB ports, you will need to obtain an adaptor.

Note that if you are *only* calling upon hardware found on the Nano 33 BLE Sense development board (say the MCU and IMU), you could forgo connecting it to the Tiny Machine Learning Shield. If you need to remove either the Nano board or the camera module from the shield, grip each side of whichever board and pull back with a *gentle* rocking motion (back and forth) to work the pins out from the headers below.


6
wikiHow


5

Setting up your Software

Here we will walk through setting up the software you'll need in the projects, the Arduino integrated development environment (IDE). We will be using the Arduino IDE to program your microcontroller.

Introduction

What is Arduino?

Arduino describes itself as "an open-source electronics platform based on easy-to-use hardware and software." In large part, this description is fitting, as the company designs and sells a collection of microcontroller development boards that simplify deployment of embedded hardware alongside a software framework that abstracts away all but the most relevant considerations for your application. Perhaps what's under-represented by this description is the role that the surrounding community plays in enabling many plug-and-play experiences given the number and quality of auxiliary hardware modules, support libraries, and tutorials that have and continue to be produced within Arduino's ecosystem.

One thing we'd like to acknowledge here is that Arduino's mission of creating easy-to-use hardware and software has the necessary tradeoff of limiting the feature set of its development environment to the essentials.

What is an Integrated Development Environment (IDE)?

As is true for all forms of programming, an integrated development environment, or IDE, is an application with a feature set that facilitates software development generally or within a particular niche. The Arduino Desktop IDE is highly specific in the sense that it is intended to facilitate software development for a specific set of microcontroller boards, in C++.

Within the niche of IDEs for embedded software, there is a noticeable dichotomy between light-weight applications, (like Arduino's IDE) that minimize functionality and abstract away details in the name of simplicity and full-featured IDEs (like Visual Studio Code) that exist to support industry professionals.

If you're interested in finding a happy medium to use in future embedded projects, we suggest VS Code with the hardware-specific extension PlatformIO, that together enable nice-to-have features like line completion, reference tracking, et cetera, without all of the complexity that more advanced IDEs introduce (note that the VS Code + PlatformIO will not be necessary, or used in this course.

What is the Arduino IDE?

As you might expect given the description above, the Arduino IDE is a lightweight development environment with features that permit you to very quickly manipulate microcontroller development boards. While there is a cloud-based offering (the Arduino Create Web Editor) as

well as a so-called Arduino IDE 2.0, we are going to use the standard Arduino Desktop IDE in this course.

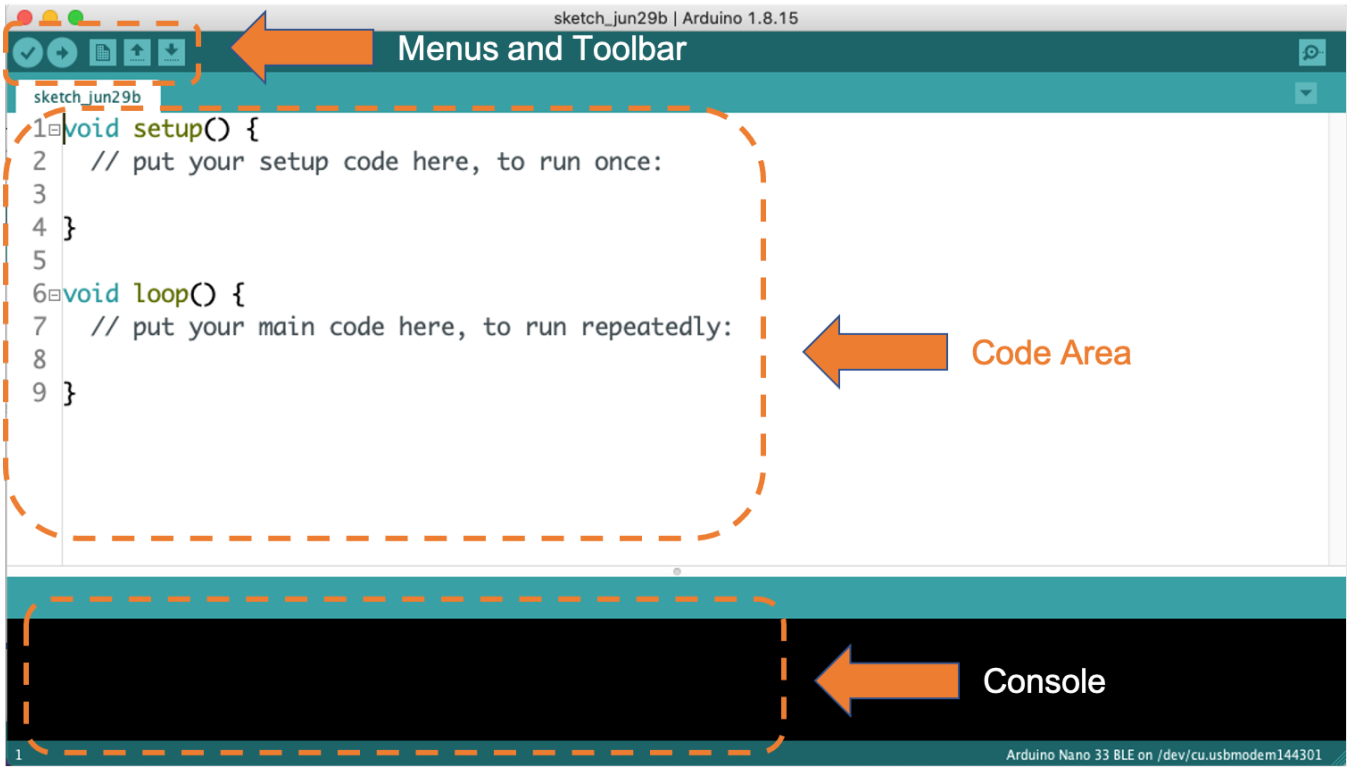
Downloading and Installing the Arduino IDE

1. Navigate to arduino.cc and at the top of the page select Software and click Downloads
2. Click on the download link appropriate for your machine
3. There is no obligation to contribute, you can click 'Just Download' to proceed

Operating specific installation instructions vary, so if what follows isn't self-evident, you can find guidance, by OS, at the following links (Download the latest version, as today: **Version 1.8.16**):

- Windows
- MacOSX
- Linux

A Quick Tour of the IDE



When you first open the IDE you will see a screen that looks something like the above screenshot.

Up at the top of the IDE you will find the menus and the toolbar which we will explore as we work through the rest of this document and when we run the tests later in this section.

Below that you will find the file tabs. For now there will only be one file open that is most likely named sketch_<date>, however later in the course when we work through more complicated examples you will see all of the various files that make up the project across the file tab area. In “Arduino speak,” a sketch is a simple project/application.

In the middle of the screen you will see the large code area. Each sketch can consist of many files and use many libraries but at its core each sketch is made of two main functions, “setup” and “loop”, which you can see pre-populated in the code area. We’ll explore what those functions are used for through the Blink example in a future video. For now, let's continue orienting ourselves with and setting up the IDE.

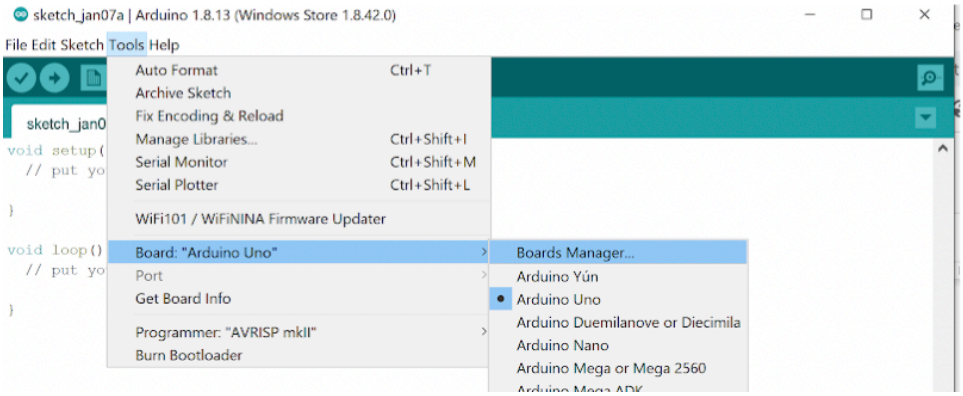
Finally, at the bottom of the screen you’ll find the console. This is where you will see debug and error messages that result from compiling your C++ sketch and uploading it to your Arduino.

Installing the Board Files for the Nano 33 BLE Sense

One of the primary advantages that the Arduino ecosystem affords is the portability of code you write for one or another board within their line-up or even in porting code to affiliate boards. This is made possible by the support files organized in the Boards Manager, which coordinates a download and installation of files that detail the Arduino functions (sometimes ‘core’) that are defined for that particular board (which is how hardware differences between boards are abstracted) as well as compiler or linker details specific to the given board.

To install the board files you will need for your Arduino Nano 33 BLE Sense please do the following:

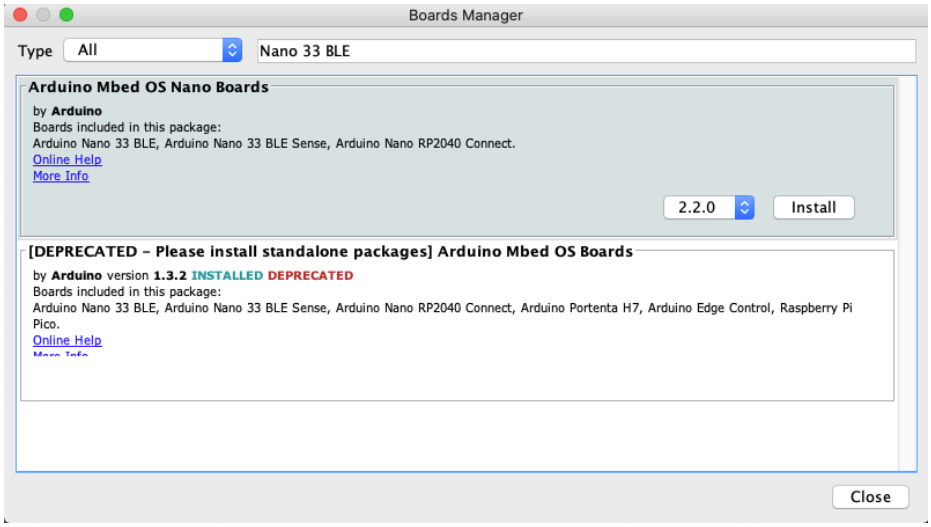
1. Open the Boards Manager, which you can find via the Tools drop-down menu. Navigate, as follows: Tools → Board → Boards Manager. Note that the Board may be set to “Arduino UNO” by default

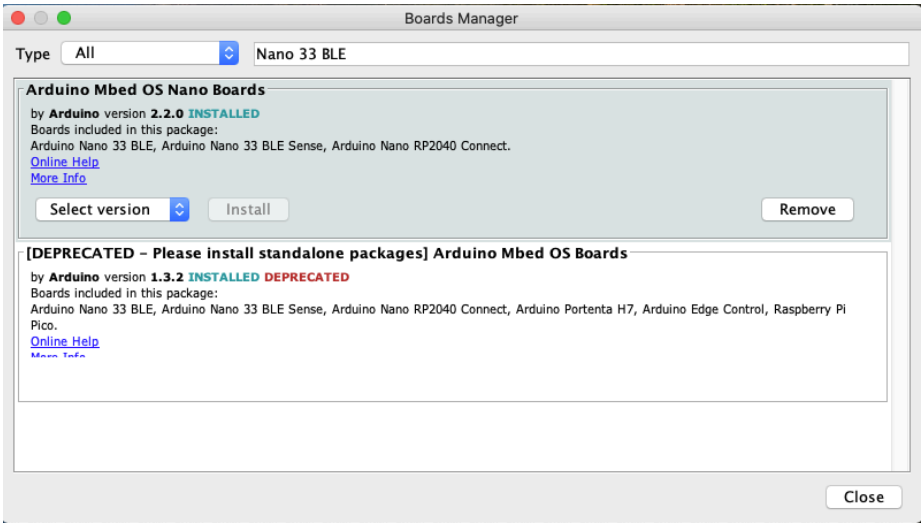


2. In the Boards Manager dialog box, use the search bar at the top right to search for “Nano 33 BLE,” which should bring up two results. We’re interested in the first result (as shown), named “Arduino Mbed OS Nano Boards,” (without the **DEPRECATED** tag). Make sure the newest version is selected (This document was done with **Version 2.2.0,** but it is already deprecated. The latest one is **Version 2.7.2)** and then click “Install.” As

the install process progresses you will see a blue completion bar work its way across the bottom of the Board Manager window. Be patient, you may need to install USB drivers which requires you to approve an administrator privileges popup which can take a couple minutes to appear.

After you have successfully installed the board if you exit and re-open the Board Manager and search again for “Nano 33 BLE” you will now see a green **INSTALLED** next to the library and the option to “Remove” the library or install a different version.



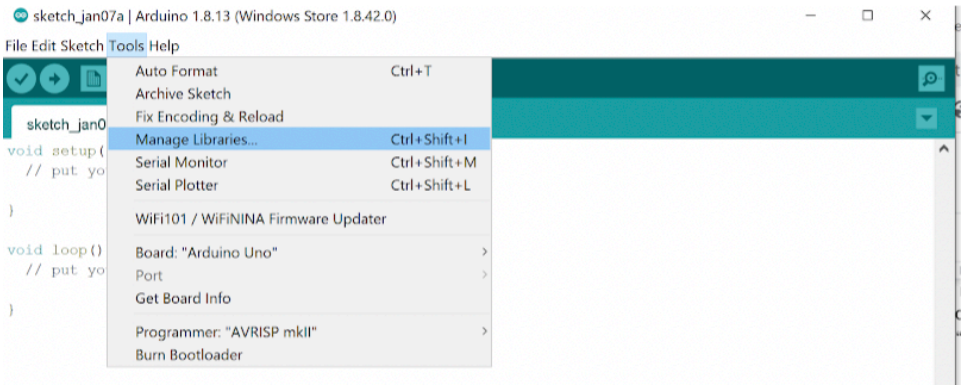


Installing the Libraries needed

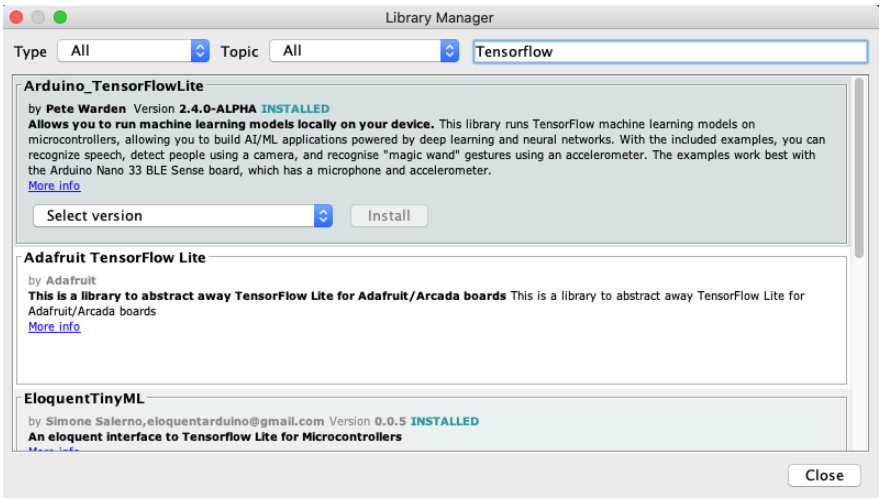
Another advantage of the Arduino ecosystem is the availability of a wide array of libraries for performing various tasks, such as interfacing with a sensor module or manipulating data using common algorithms. There are many libraries that can be accessed from within the Library Manager in the Arduino IDE as described below. Check here for a complete list.

For the basic projects, we are going to need four libraries (in fact 3, the BLE is optional). To install the libraries please do the following and **make sure to install the version specified in the reading below or the tinyML applications will not work**:

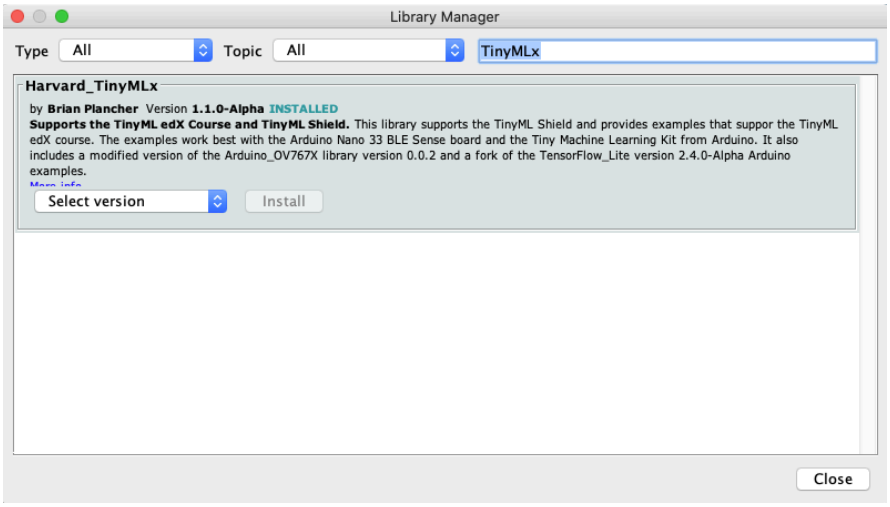
1. Open the Library Manager, which you can find via the Tools drop-down menu. Navigate, as follows: Tools → Manage Libraries.



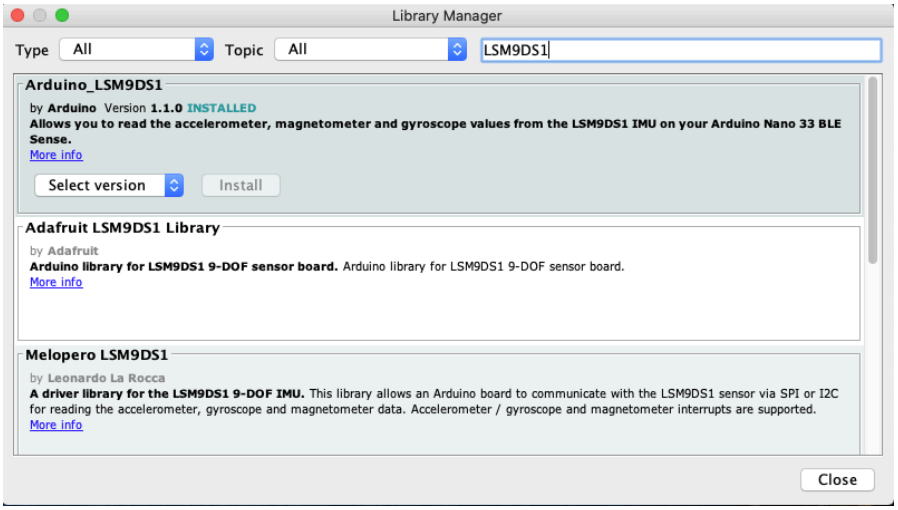
2. Then, much like for the Boards Manager, in the Library Manager dialog box, use the search bar at the top right to search for the following libraries, one at a time. Note that like with the Board manager a blue completion bar will appear across the bottom of the Library Manager window.
   a. The Tensorflow Lite Micro Library
      i. Search Term: Tensorflow
      ii. Library Name: Arduino_TensorFlowLite
      iii. Version: 2.4.0-ALPHA



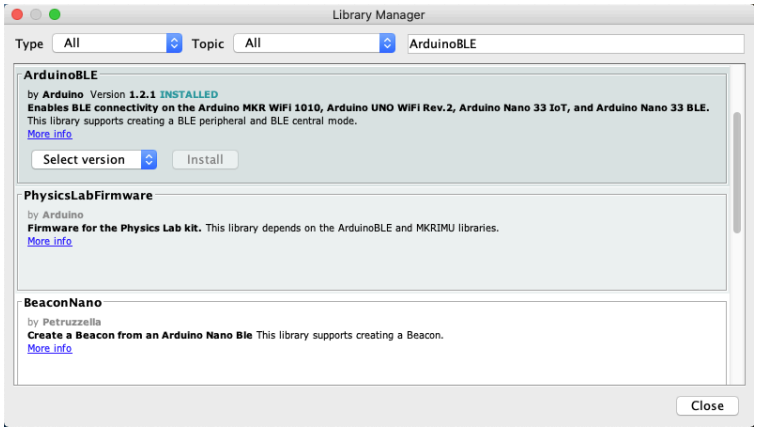
b. The Harvard_TinyMLx Library (includes the camera OV767X library)
   i. Search Term: TinyMLx
   ii. Library Name: Havrard_TinyMLx
   iii. Version: 1.1.0



c. The library that supports the IMU (accelerometer, magnetometer, and gyroscope) on the Nano 33 BLE sense
   i. Search Term: LSM9DS1
   ii. Library Name: Arduino_LSM9DS1
   iii. Version: 1.1.0



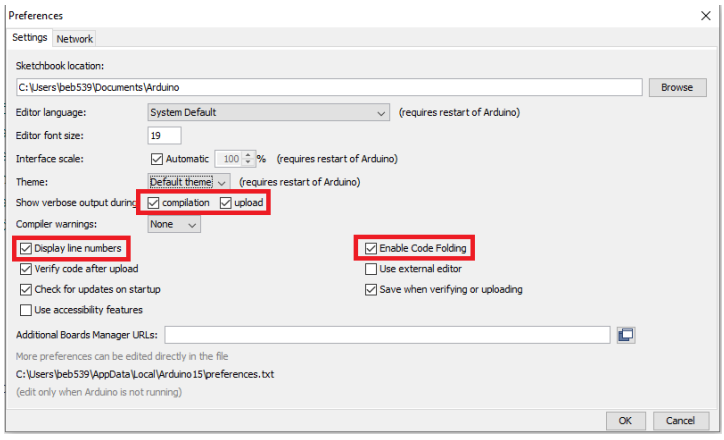
d. ArduinoBLE (Optional)
    i. Search Term: ArduinoBLE
    ii. Library Name: ArduinoBLE
    iii. Version: 1.2.1



Setting your Preferences

You can adjust the preferences set for the Arduino Desktop IDE via the File drop-down menu, Arduino → Preferences. There are a few preferences that we recommend enabling to make the Arduino IDE a little easier to use, namely:

1. Show verbose output during: compilation and upload
2. Enable code folding
3. Display line numbers



Of final note if you don't like the default theme for the Arduino Desktop IDE there is a nice tutorial for a dark theme you can find here. Also if you would like to learn more about the IDE, check out Arduino's documentation.

And that’s it! Your Arduino IDE should be all configured for this course. Now that you have all of the necessary board files and libraries installed it’s time to explore more of the features of the IDE available under the “Tools” menu and start to test out your Arduino by deploying the Blink example!

The Arduino Blink Example

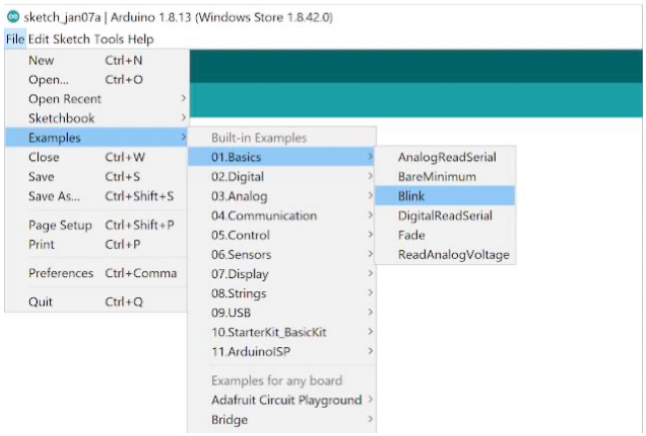
Now, we will deploy the Arduino Blink example to make sure everything is working properly and to give you your first experience deploying code to your Arduino!

Preparing for Deployment

1. Use a USB cable to connect the Arduino Nano 33 BLE Sense to your machine. You should see a green LED power indicator to come on when the board first receives power.

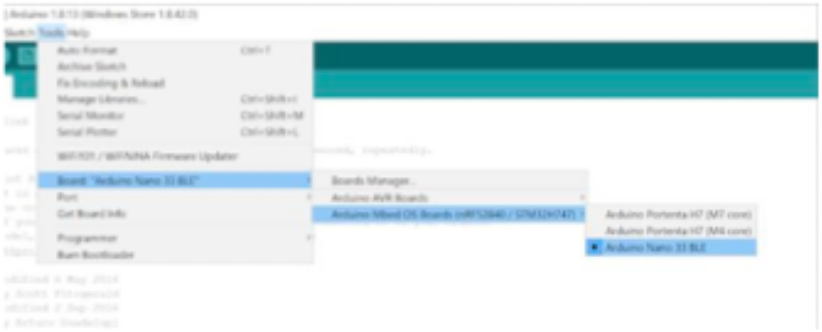
2. Open the *Blink.ino* sketch, which you can find via the File drop-down menu. Navigate, as follows: File → Examples → 01.Basics → Blink.

You'll notice that Arduino has provided a wealth of examples to choose from should you like to explore the board more on your own. There is great documentation about those examples on the Arduino website.



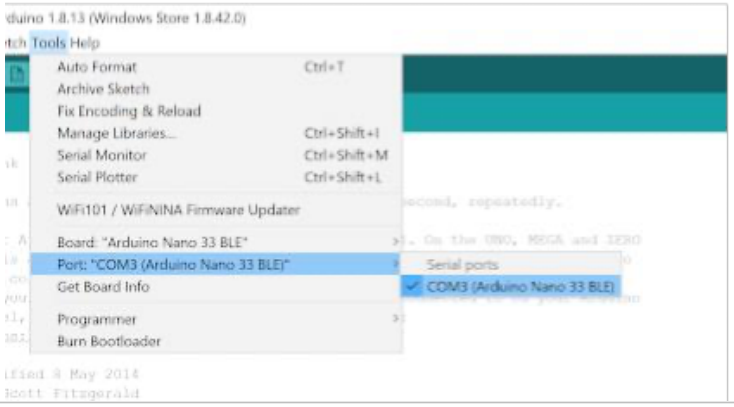
3. Use the Tools drop down menu to select appropriate Port and Board. This is important as it is telling the IDE which board files to use and on which serial connection it should send the code. In some cases, this may happen automatically.

a. Select the Arduino Nano 33 BLE as the board by going to Tools → Board: <Current Board Name> → Arduino Mbed OS Boards (nRF52840) → Arduino Nano 33 BLE. Note that on different operating systems the exact name of the board may vary but/and it should include the word Nano at a minimum. If you do not see that as an option then please go back to Setting up the Software and make sure you have installed the necessary board files.

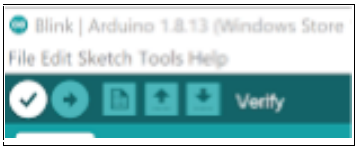


b. Then select the USB Port associated with your board. This will appear differently on Windows, macOS, Linux but will likely indicate ‘Arduino Nano 33 BLE” in parenthesis. You can select this by going to Tools → Port: <Current Port (Board on Port)> → <TBD Based on OS> (Arduino Nano 33 BLE). Where <TBD Based on OS> is most likely to come from the list below where <#> indicates some integer number

i. Windows → COM<#>
ii. macOS → /dev/cu.usbmodem<#>
iii. Linux → ttyUSB<#> or ttyACM<#>



4. Finally, use the checkmark button at the top left of the UI to verify that the code within the example sketch is valid.



Verification will compile the code, so take note of the status and results indicated in the black console at the bottom of the IDE. The level of detail presented here will depend on whether or not you have enabled ‘verbose output during compilation in Preferences.

You should most likely at the end see a final output indicating how much memory the sketch will take on the Arduino once it is uploaded. Something like: “Sketch uses 86568 bytes (8%) of

program storage space. Maximum is 983040 bytes. Global variables use 44696 bytes (17%) of dynamic memory, leaving 217448 bytes for local variables. Maximum is 262144 bytes."

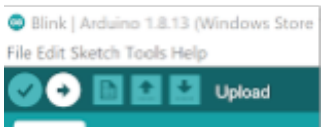
As you can see this is a very simple example and does not take up much space. While, as you'll see in a moment, uploading your code will also verify your code automatically, it is often helpful to verify your code first as you can iron out any compilation errors without having any hardware on hand.

Deploying (Uploading) the Sketch

Once we know that the code at hand is valid, we can 'flash' it to the MCU (This is a colloquial reference to the type of program memory MCUs call on flash memory):

Use the rightward arrow next to the 'compile' checkmark to upload / flash the code.
Note that pragmatically, this step will re-compile the sketch before flashing the code, so that in the future if you intend to sequentially compile and flash a program, you need only press the 'upload' arrow.

As before, take note of the status and results indicated in the black console at the bottom of the IDE. The level of detail presented here will depend on whether or not you have enabled 'verbose output during compilation' in Preferences, accessible via the File drop-down menu in the IDE.

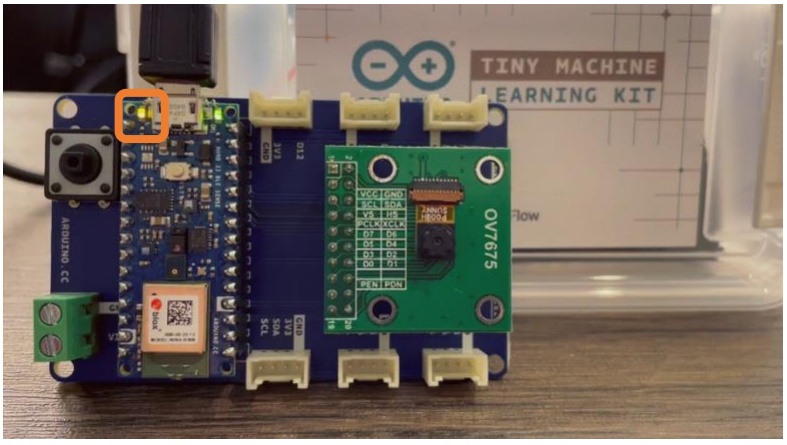


You'll know the upload is complete when you red text in the console at the bottom of the IDE that shows 100% upload of the code and a statement that says something like "Done in <#.#> seconds."

Again if this is the first time you are uploading a sketch to an Arduino the upload may hang for a little while until you get another administrator approval popup and approve it. Don't worry, this is just a one time thing.

If you receive an error you will see an orange error bar appear and a red error message in the console (as shown below). Don't worry -- there are many common reasons this may have occurred. To help you debug please check out our FAQ appendix with answers to the most common errors!

At this point, you'll want to look at the board itself. The orange LED opposite the green LED power indicator about the USB port should now be blinking!



Understanding the Code in the Blink Example

Now that you have gotten the blink example deployed to your microcontroller let's explore the code as shown below.

You'll notice that it consists of two functions: setup, and loop. As we mentioned before, this is the standard setup for an Arduino sketch. This is because when the Arduino turns on it runs the setup() function ONCE to initialize (aka setup) the sketch. Then it runs the loop() function infinitely many times (aka it runs as an infinite loop) to execute the sketch. This works well for most tinyML applications as they are designed to respond to continuous sensor input. You can imagine in the case of Keyword spotting that we need to initialize the neural network and the microphone and then in a loop, we want to listen to audio and trigger (or not) depending upon the output of the neural network!

```
// the setup function runs once when you press reset or power the board

void setup() {
// initialize digital pin LED_BUILTIN as an output. pinMode(LED_BUILTIN, OUTPUT);

}
// the loop function runs over and over again forever

void loop() {
digitalWrite(LED_BUILTIN, HIGH); // turn the LED on (HIGH is the voltage level) delay(1000); // wait for a second
digitalWrite(LED_BUILTIN, LOW); // turn the LED off by making the voltage LOW delay(1000); // wait for a second

}
```

You'll also notice that both functions have the void return type as they do not ever return anything but instead have side effects.

For example, in the setup function the LED_BUILTIN (which is a shortcut name for the pin that controls the voltage to the LED) is set to be an output for the duration of the loop. In general you will need to set all of the pins you use as either inputs or outputs during the setup function. If you wired up the camera yourself you have already explored a lot of the special names reserved for the pins as shown on the pinout diagram.

In the loop function you'll notice that we are alternating between writing a HIGH (aka turning on the LED) and writing a LOW (aka turning off the LED). The delay of 1000 milliseconds (1 second) between each step is crucial as otherwise the light would turn on and off so fast that it would be imperceptible. In fact if you make the delay too short the light will simply seem to be dim.

This is a trick called Pulse Width Modulation that is actually used often in industry to e.g., control motors. If you'd like, feel free to experiment with modifying these delays and redeploying the code to see its effect!

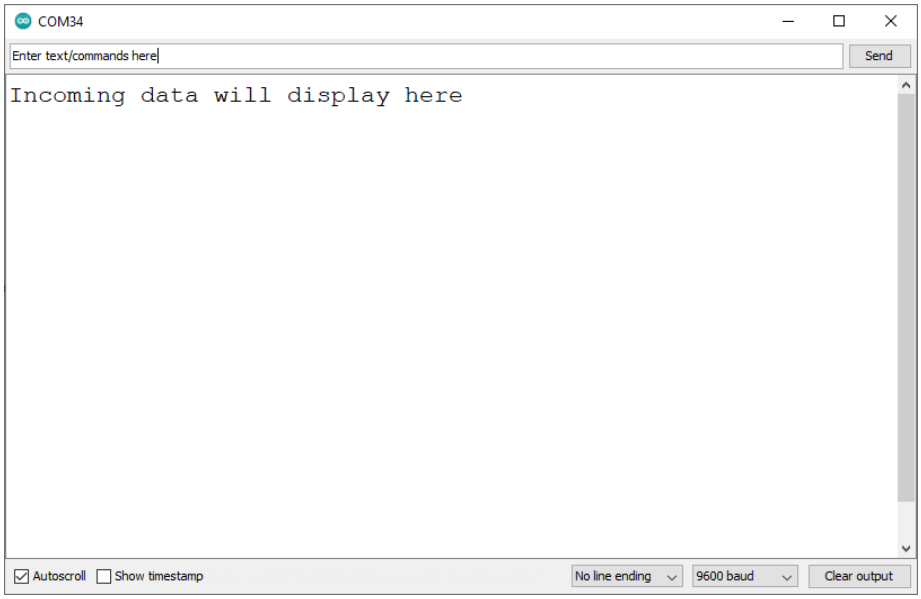
Testing the Sensors

With tests of the microprocessor itself we can turn our attention to verifying that the sensors required for this course (IESTI01) are functioning properly. To do so, we'll be calling on a predefined script that the Harvard staff have put together. By the end of this reading, you will have seen live data streams for the on-board microphone, the STMicroelectronics, MP34DT05, the on-board internal measurement unit (IMU), the STMicroelectronics, LSM9DS1, as well as at least a static image return from the off-board OV7675 camera module, the OmniVision OV7675. We will also detail the optional extensions you'll need to do to obtain a live video feed from the camera as well.

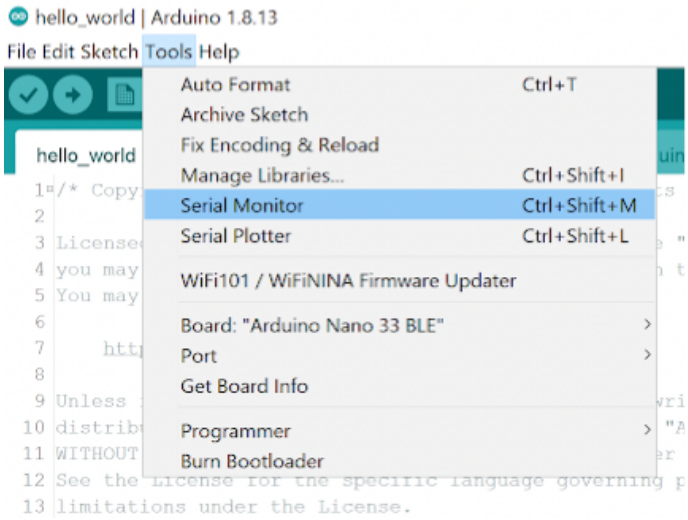
The Serial Monitor & Plotter

The Serial Monitor & Plotter are two core tools that you can use to get information from your Arduino when it is connected to your computer with a data USB cable. In fact, as you develop your own Arduino applications you'll probably find yourself opening up the Serial Monitor and using the Serial.println() command in your Arudino sketches much like how you have used the print() function in Python or printf() function in C++. Since the Arduino sketch runs in an infinite loop you may find that it may be easier to plot the graph of the data you are sending back from the Arduino instead of printing the raw values, and that is where the Serial Plotter will come in.

While the primary function of the Serial Monitor (and the Serial Plotter) is to view data incoming from the MCU, the Serial Monitor also features a text entry field that we will use to issue pre-determined commands conveyed in the console. Below we have included a screenshot of the Serial Monitor elements to differentiate where we expect to see incoming data and where we can enter commands to send to the microcontroller:



You can open up the Serial Monitor (and Serial Plotter) by navigating through the menu to Tools → Serial Monitor or Tools → Serial Plotter.



Testing the Microphone

1. Use a USB cable to connect the Arduino Nano 33 BLE Sense to your machine. You should see the green LED power indicator come on when the board first receives power.
2. Open the test_microphone.ino sketch, which you can find via the File drop-down menu. Navigate, as follows: File → Examples → Harvard_TinyMLx → test_microphone.
    - Note that this library could be found as: File → Examples → INCOMPATIBLE → Harvard_TinyMLx → test_microphone.
3. As always, use the Tools drop down menu to select appropriate Port and Board.

    a. Select the Arduino Nano 33 BLE as the board by going to Tools → Board: <Current Board Name> → Arduino Mbed OS Boards (nRF52840) → Arduino Nano 33 BLE. Note that on different operating systems the exact name of the board may vary but/and it should include the word Nano at a minimum. If you do not see that as an option then please go back to Setting up the Software and make sure you have installed the necessary board files.
    b. Then select the USB Port associated with your board. This will appear differently on Windows, macOS, Linux but will likely indicate 'Arduino Nano 33 BLE" in parenthesis. You can select this by going to Tools → Port: <Current Port (Board on Port)> → <TBD Based on OS> (Arduino Nano 33 BLE). Where <TBD Based on OS> is most likely to come from the list below where <#> indicates some integer number
        i. Windows → COM<#>
        ii. macOS → /dev/cu.usbmodem<#>
        iii. Linux → ttyUSB<#> or ttyACM<#>

4. Use the rightward arrow next to the 'compile' checkmark to upload / flash the code.

    You'll know the upload is complete when you red text in the console at the bottom of the IDE that shows 100% upload of the code and a statement that says something like "Done in <#.#> seconds."

If you receive an error you will see an orange error bar appear and a red error message in the console (as shown below). Don't worry -- there are many common reasons this may have occurred. To help you debug please check out our FAQ appendix with answers to the most common errors!

5. Open the Serial Monitor. You can accomplish this three different ways as shown below. **If the Serial Monitor fails to open check to make sure the appropriate Port is selected.** Sometimes the port is reset during the upload process.
    1. Use the menu to select: Tools → Serial Monitor
    2. Click on the magnifying glass at the top right of the Arduino Desktop IDE
    3. Use the keyboard shortcut: CTRL + SHIFT + M or CMD + SHIFT + M

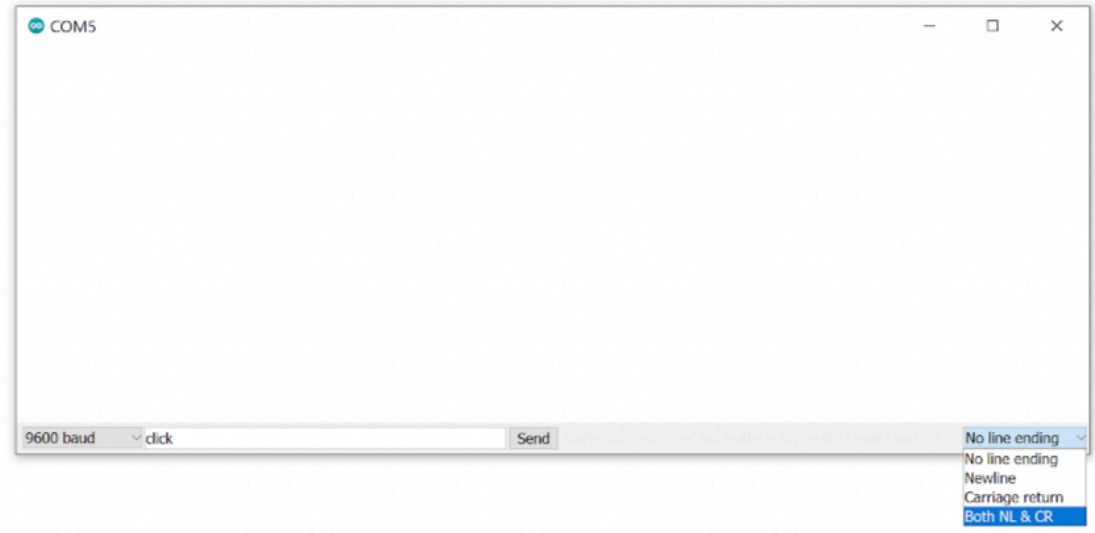
    depending on your operating system.

6. When the Monitor opens, you should see the following text appear:

```
Welcome to the microphone test for the built-in microphone on the Nano 33 BLE Sense

Use the on-shield button or send the command 'click' to start and stop an audio recording
Open the Serial Plotter to view the corresponding waveform
```

7. Press the on-shield button to start an audio recording. After you do, a stream of data should appear in the Serial Monitor. Press the button again to stop the recording. If the numbers were changing then your microphone is most likely correctly recording audio! Congratulations! If you do not have the shield you can also control the starting and stopping of the waveform by sending the command click through the serial monitor. In the drop down menu that reads "No line ending" by default at the bottom right of the Serial Monitor, you need to select "**Both NL & CR**". This tells the Serial Monitor to send both a NL = new line character as well as a CR = carriage return character every time you send a message. This is important for our application as our Arduino sketch is looking for that to differentiate between each input.

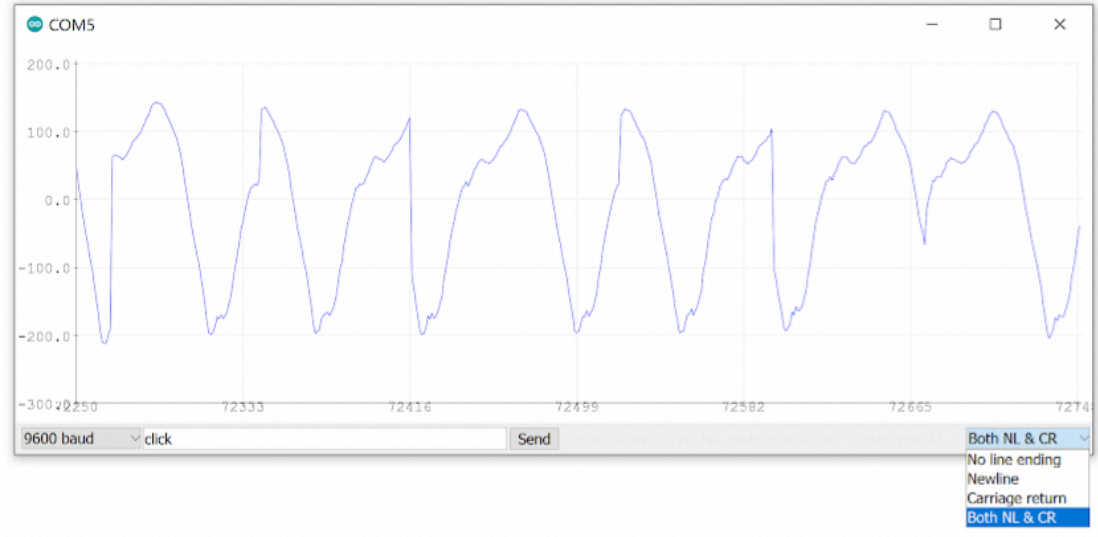


8. To help visualize this a little better we are going to use the Serial Plotter. **Note that you cannot have both the Serial Monitor and Serial Plotter open at the same time as the Arduino can only communicate over a single serial port. Importantly, this also means that you cannot upload new code to the Arduino if either the Serial Monitor or Plotter is open!** You can open the Serial Plotter in two ways:
    1. Use the menu to select: Tools → Serial Plotter
    2. Use the keyboard shortcut: CTRL + SHIFT + L or CMD + SHIFT + L

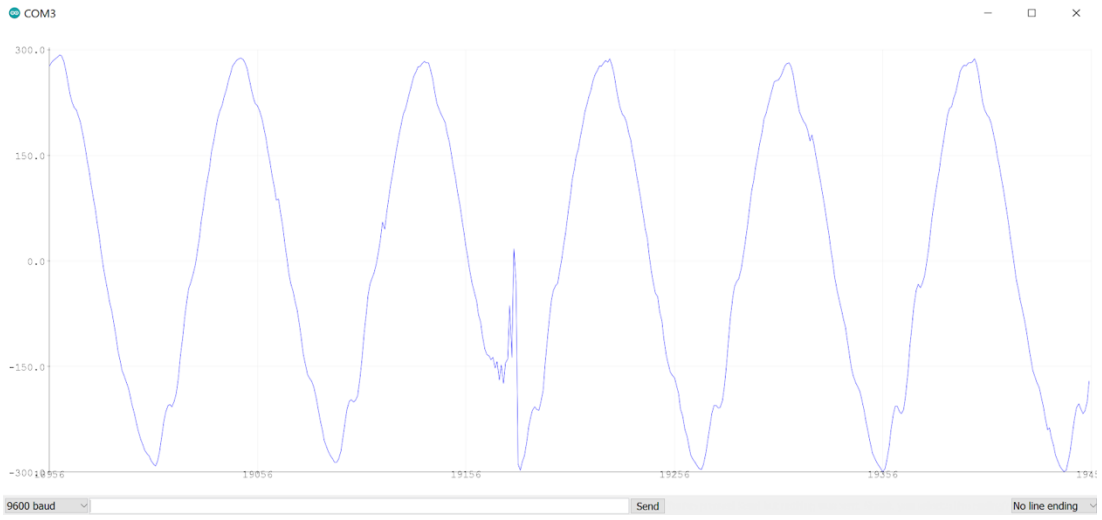
    depending on your operating system.

9. Perform the same experiment, press the on-shield button (or send the serial command click) to start an audio recording and again to stop it. You should see a waveform

    appear on the Serial Plotter. This is simply a graphical representation of the numbers you saw earlier on the Serial Monitor. Note that the vertical axis of the Serial Plotter scales dynamically so that the magnitude of the data conveyed is not fixed with respect to the enclosing window, but rather scaled to occupy most of the headroom.



10. Now record some audio again. This time try to whistle or hum a constant tone. If you can keep the tone constant you should see the waveform start to look like a sinusoid. An example the course staff made is below. Don't worry if you can't get this to work perfectly. You should in general find that a constant tone has a more constant pattern. If when you make different sounds you find that the waveform changes, you can be confident that your microphone is working.



Testing the Inertial Measurement Unit

1. Now let's open the test_IMU.ino sketch, which you can find via the File drop-down menu. Navigate, as follows: File → Examples → Harvard_TinyMLx → test_IMU.
2. As always, use the Tools drop down menu to select appropriate Port and Board.
3. Then use the rightward arrow next to the 'compile' checkmark to upload / flash the code. If you get the error "Arduino_LSM9DS1.h: No such file or directory" then please go back to Setting up the Software and make sure you install the accelerometer, magnetometry, and gyroscope library! To help you debug please check out our FAQ appendix with answers to the most common errors!
4. Open the Serial Monitor and you should see the following. As a reminder you can open the serial monitor in one of three ways shown below. **If the Serial Monitor fails to open check to make sure the appropriate Port is selected.** Sometimes the port is reset during the upload process.
    1. Use the menu to select: Tools → Serial Monitor
    2. Click on the magnifying glass at the top right of the Arduino Desktop IDE
    3. Use the keyboard shortcut: CTRL + SHIFT + M or CMD + SHIFT + M depending on your operating system.

```
Welcome to the IMU test for the built-in IMU on the Nano 33 BLE Sense

Available commands:
a - display accelerometer readings in g's in x, y, and z directions
g - display gyroscope readings in deg/s in x, y, and z directions
m - display magnetometer readings in uT in x, y, and z directions
```

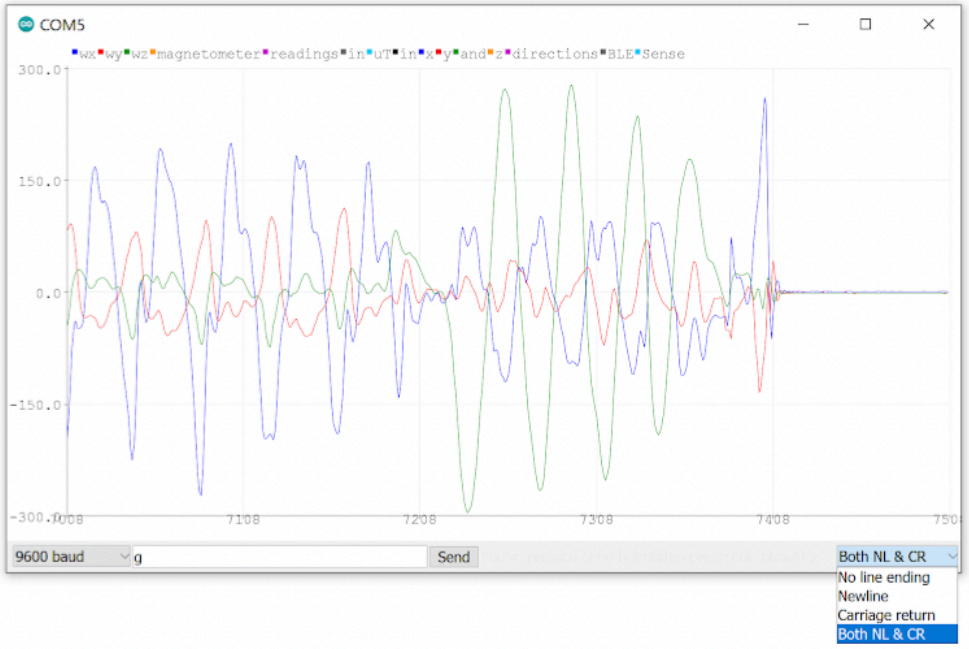
5. Like with the microphone, in the drop down menu that reads "No line ending" by default at the bottom right of the Serial Monitor, you need to select "**Both NL & CR**".
6. Now you can enter one of the possible arguments (a, g, or m) into the text entry bar across the top of the Serial Monitor and click "Send". As mentioned in the Serial Monitor

this will start to print the data coming from the [a]ccelerometer (computing acceleration in the x, y, or z direction), the [g]yroscope (computing the angular velocity -- the change in the roll, pith, and yaw), or the [m]agnetometer (computing the magnetic forces present on the IMU). Depending on your selection, you should now see a stream of corresponding data. However like with the microphone this data is hard to interpret in raw form. So instead lets us the Serial Plotter.

7. Close the Serial Monitor and open the Serial Plotter. As a reminder you can open the Serial Plotter in two ways:
    1. Use the menu to select: Tools → Serial Plotter
    2. Use the keyboard shortcut: CTRL + SHIFT + L or CMD + SHIFT + L

    depending on your operating system.

8. With the Plotter open, you should see the same stream of data presented graphically. The most interesting one to plot is the [g]yrocope. Again make sure you have selected “Both NL & CR” and type “g” into the text entry box (now at the bottom of the window) and click “Send.”



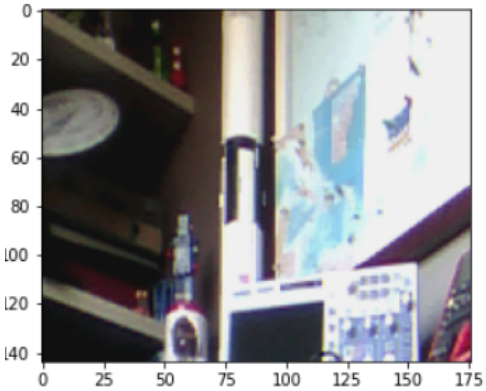
9. Move the board around and observe the response. Can you figure out which axis of rotation is the x, the y, and the z (also known as roll, pitch, and yaw)? If by moving the board around in different directions you get different responses from the various lines in the Serial Plotter you can feel confident that you have a working IMU.

Testing the OV7675 Camera

1. Now let's open the test_camera.ino sketch, which you can find via the File drop-down menu. Navigate, as follows: File → Examples → Harvard_TinyMLx → test_camera.
2. As always, use the Tools drop down menu to select appropriate Port and Board.
3. Then use the rightward arrow next to the 'compile' checkmark to upload / flash the code. If you get the error "Arduino_OV767X.h: No such file or directory" then please go back to Setting up the Software and make sure you install the accelerometer, magnetometry, and gyroscope library! To help you debug please check out our FAQ appendix with answers to the most common errors! **Note:** if you get an error message that contains something like 'OV7675' was not declared in this scope this means you already had a conflicting copy of the Arduino_OV767X library installed on your system. We have bundled a forked copy of it with our library so please uninstall the conflicting version.
4. If you get a message that "Your board was not found", this could be due that the MCU was "busy" running the previous code. Click the Reset button twice (you will see the Build in LED blinking slowly). Upload the sketch again.
5. Open up the Serial Monitor and you should see:

```
Welcome to the OV7675 test
Available commands:
single - take a single image and print out the hexadecimal for each pixel (default)
live - the raw bytes of images will be streamed live over the serial port
capture - when in single mode, initiates an image capture
```

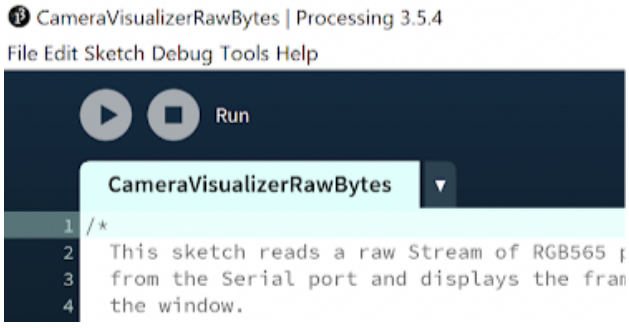
6. Type "single" in the text entry field and press send or hit the enter / return key. The camera is now primed to take a picture. Now you can either text "capture" or press the on-shield button to take a selfie (or photo). To get the photo oriented correctly make sure the "OV7675" text on the camera module (or the "Tiny Machine Learning Shield" on the shield) is oriented such that it would be readable by the subject of the photo (e.g., you if you are taking a selfie). **The camera does not have a large field of view so if you are taking a selfie make sure to hold the camera far away from your face.**
7. To view your image you will need to copy the sequence of hexadecimal digits that will print to the Serial Monitor and paste them into the notebook: OV7675_Image_Viewer.ipynb, that is saved on this class GitHub.



(Optional) Real Time Video with Processing

If you are interested in seeing a live stream capture from the OV7675, you can do that but not through the Arduino IDE. You'll have to use an additional piece of software, Processing. You can download the application for your OS, here (we've tested with version 3.5.4). We would like to note that we have found this to be buggy on Windows, which is why we provide this as an optional extension and not the default suggested process.

1. Launch the test_camera example through the Arduino IDE, open the Serial Monitor, and then type "live" in the text entry field and press send or hit the enter / return key. Your camera will now be live streaming the image to the computer at 1 frame per second.
2. With Processing installed (which for some operating systems is as simple as unzipping the pre-built application folder), open the Processing Application. You'll find that the application looks a lot like the Arduino Desktop IDE.
3. Then find your Arduino folder. On most operating systems it is located either in your home or Documents directory.
4. In Processing open the file Arduino → Libraries → Harvard_TiynMLx → extras → CameraVisualizerRawBytes → CameraVisualizerRawBytes.pde
5. Now click the "run" play button at the top left of the UI. A popup should appear streaming the camera image. Again, we have found this to be buggy on Windows so don't be surprised if the image looks odd on Windows.



Powering Your TinyML System

Now that you have your Arduino development board up and running, let's talk about how you could deploy it independent of your computer! While some embedded systems call on AC-DC converters (or “wall warts,” colloquially) to provide low voltage power to their electronics (the Google home speaker, for example), others are battery powered. Both of these paradigms are applicable to real-world deployment of tinyML and both are achievable using your TinyML kit.

USB Power Delivery

To this point, we have provided power over USB to our microcontroller via the microB port on the Nano 33 BLE Sense. The 5V that USB carries is then down regulated on the development board to 3.3V, the logical reference for the MCU. While there’s nothing wrong with this in development, a prototype of your application ought not depend on drawing power from your computer. Instead, you could call on an AC-DC converter with USB output. This has the added benefit of likely raising the current capacity of the 5V power rail, from at most 500 mA via a computer to whatever the specification happens to be for a given wall-bound converter. This could be meaningful for driving certain power hungry actuators, like speakers.

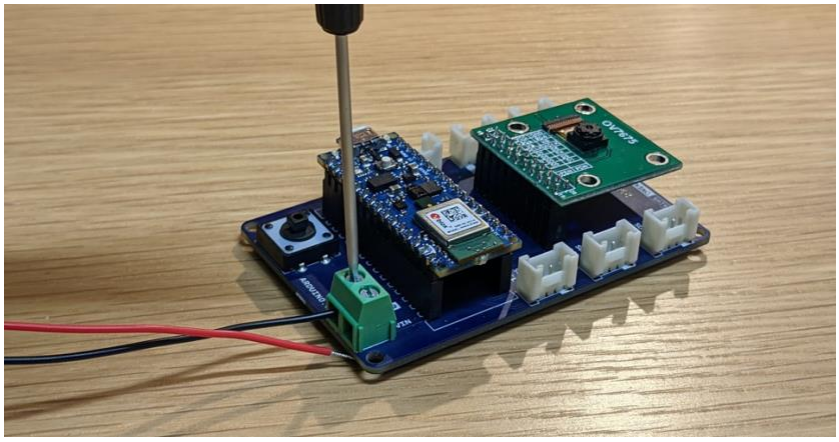
Battery

While the above solution removes the need for the computer, you’re still tethered to a wall. To go fully mobile you’ll want to call on a battery. So what are our options? The idea of using a voltage regulator (specifically, the MPM3610) to cut down an input voltage to a nice, stable 3.3V level applies here as well. If you were to take a closer look at the linked datasheet, you’d find that the MPM3610 accepts input voltages from 4.5V to 21V. The 5V delivered over USB is within this range, and any compatible battery will need to be as well. This unfortunately eliminates the possibility of calling on single cell 3.7V lithium batteries, but makes the selection of a 9V alkaline battery fairly obvious.

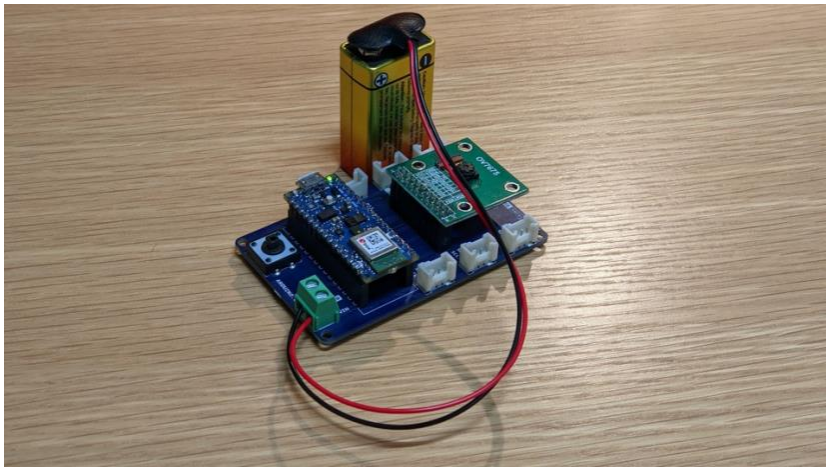
You might be wondering how any battery might connect to the boards in front of you, but never fear, we’ve got you covered. At one corner of the Tiny Machine Learning Shield you’ll find a green terminal block with silkscreen labels that read, “VIN” and “GND,” where GND is our reference voltage and as such should be connected to the negative terminal of any compatible battery. This green terminal block is where you’d want to screw in wires carrying 4.5V to 21V, and we’ll add that 9V clip, like this, that terminates in pre-stripped hook-up wire makes this quite easy!

Assembly Steps

1. Screw down a wire leading from the negative battery terminal (black) to GND (Most < 3mm flat-head screwdrivers will suffice here).



2. Repeat this process for the positive battery terminal (red) to VIN. And that's it you're all set to power your Arduino from a battery!



Important Notes

1. While there is clever circuitry on board to handle such an exception, it is generally good practice to avoid having competing power sources, so we'd recommend that you unplug the Nano 33 BLE Sense from USB power before connecting a battery
2. With about 550 mAh capacity, a 9V battery can source 15 mA for about 37 hours before you will need to get a new one.