Debugging Microcontroller Code

One of the challenges of developing embedded software is that it can be difficult to know how the code you've written behaves once it has been deployed. Because the corresponding computation is happening on the microcontroller, rather than on your laptop, say, you won't necessarily have the same access to live variables or breakpoint-level debug that you may be accustomed to. In this reading, we outline a hierarchy of potential solutions and underline which of these we intend to use during this course.

Isolate Variables

As pedantic as the following suggestion may be, it is genuinely helpful to remember that when you're developing a new system, including software, it can be extremely beneficial to monitor the change in output for a singular change in input as a way of tracking the effect a given input has on the output. In the context of developing microcontroller code, this manifests as adding (or removing) one line or block of code at a time to discern how this affects performance in isolation. In so doing, it should become clear whether or not the input under test has bearing over the aspect of behavior that you are evaluating.

Note: This is a viable method for debug in this course

Indicators by Actuation

In the Blink example, we saw how easy it is to control simple actuators, like toggling an LED on and off. While we won't call on these in this course, the same could be said for piezo buzzers or vibration motors, for example. We can take this concept further still by calling on the RGB LED featured on the Nano 33 BLE Sense to emit not only varied intensity of light, but also color to encode status and will use this as a way to indicate classifier performance.

Note: This is a viable method for debug in this course

Serial Monitor

While indications of variable magnitude (binary or analog) can be encoded in actuating behavior, if precision is required, we can leverage the Serial Monitor we explored in the sensors test script to read and write to a data link between your computer and the MCU on the Nano 33 BLE Sense. Beware, however, that Serial.print() can affect program timing.

Note: This is a viable method for debug in this course

Full-featured Debug

Finally, we want to mention that some microcontrollers boards, including the Nano 33 BLE Sense, have the capacity for what we'll call full-featured debugging, although this almost always requires an external interface, many of which are costly. At best, you can gain access to line-by-line breakpoints and track variables and registers as your program slowly steps through the code you have written.

For the Nano 33 BLE Sense, there are pads on the bottom of the board that give access to the SWD, or Serial Wire Debug, interface of the microcontroller for full-featured debugging. This requires soldering wires to these pads to break them out to an external SWD debug probe.