Building a better Sketch-Y-Etch

Sensored Hacker

May 4, 2025

Abstract

Initial versions of the sketch-y-etch had issues related to haphazard planing, and inconsistent use of parts. It has been two years since the initial creation filled the Bangor Makerspace with retro drawing for all, and we have heard many things about what makes the sketch-y-etch hard to use. Here in v2, we are going to try an fix the issues.

ISSUES:

- **Drawing Performance:** Performance was limited by encoder issues.
- **Build Quality:** V1 used available parts with no specific plan for integration, negatively affecting the overall experience.
- **Stability:** The wobble made drawing difficult. While the Sketch-Y-Etch has never tipped over, it feels like it could—and that can be fixed.
- **Documentation:** Existing documentation focused on the history and developer notes but lacked detailed build instructions.
- **Bill of Materials:** A BOM was non-existent in V1. Not everyone has a stockpile of miscellaneous computer parts.

Improvements

- **Improved Components:** A concerted effort was made to use higher-quality parts throughout the build.
- **Specialized Hardware:** A new conceptual design in v2 maintains compatibility with a wide range of hardware and platforms, while offering many new design options.
- **Focused Documentation:** We want you to build your own Sketch-Y-Etch! A detailed tutorial is now available to guide you through the process.
- **New Features:** More colors, adjustable pen size, dynamic scaling, and additional enhancements.

- **Bill of Materials:** All required parts can be easily acquired, with a total estimated cost of about $50.

Recommended Software

The only essential software requirement for the Sketch-Y-Etch is a working installation of Python 3:

- Python 3: `https://www.python.org`

You will also need the `numpy` and `smbus2` libraries, which can be installed with:

```
pip install numpy smbus2
```

Download the Code

Download the source repository for the Sketch-Y-Etch project, available at: `https://github.com/FOSSBOSS/SketchyEtch`

Once you have the repository, extract it if necessary, and navigate to the `v2_WIP` folder. In your IDE of choice, try running `SketchyV2_WIP.py`.

If you encounter import errors, you may be missing required libraries, or your Python 3 path may be incorrect.

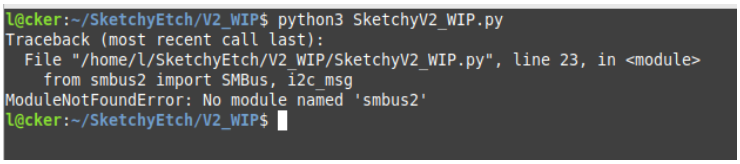


Figure 1: Missing `smbus2` library error example.

Sketch-Y-Etch app keys:

- Arrow keys: Left Right, Up, Down, moves the pen respectively.
- c, clears the screen.
- g, get a new color. Press it until you find what you want.
- l, lift pen. press again to set pen down.
- s, save your work.
- q, quit, close the program.

If you are choosing to use the Sketch-Y-Etch software for fun drawing, without building a whole thematic retro electronics installation, then you can stop reading now. Else: continue.

Let's Build It!

To begin building your Sketch-Y-Etch, you will need a set of materials and tools. The Bangor Makerspace offers complete kits for purchase if you are interested, or you can source your own components independently.

Required Tools

Basic tools required for assembly include:

- Screwdrivers
- 3D printer
- Wire cutters
- Soldering station
- Drill and drill bits
- Personal protective equipment (PPE)

Electronic Components

The Bangor Makerspace developed the HDMI-TAP specifically for this project. While its use is highly recommended for simplicity and reliability, it is also possible to build a compatible alternative at relatively low cost.

This project utilizes Qwiic connectors for inter-device communication over I^2C. A common practice in electronics is to color-code wires according to their function. Although wire insulation color does not affect circuit operation, following a standard color convention greatly simplifies visual inspection and troubleshooting.

In this project, the following color scheme is used:

- **Red:** +5VDC (power)
- **Black:** Ground (common)
- **Blue:** SDA (I^2C data line)
- **Yellow:** SCL (I^2C clock line)

As a mnemonic, you can remember that *yellow* (like the sun on a sundial) represents the clock signal.

This color convention matches the standards used by Adafruit STEMMA QT and SparkFun Qwiic connect systems, utilized in this project.

Bill of Materials

Item	Part Number / Name	Qty	Notes
Rotary Encoder	Adafruit Seesaw Encoder	4	I2C, with built-in pull ups
Display	HDMI Monitor	1	Any 1080p-capable screen
Cables	Assorted Qwiic Wires	5	Male-male and male-female
Knobs	Custom 3D-Printed Knobs	1	STL files available
Computer	Any with HDMI Output	1	Laptop, desktop, or SBC
HDMI-TAP	v2 w/qwiic connects	1	easy access HDMI i2c bus.
Optional	I2C bus splitter	1	can make routing easier.

Install Hardware

The Sketch-Y-Etch application will run on Any platform where python3 is installed. Windows, Mac, or Linux, Arm, i686, x86_64 .... it can run on any standard PC hardware. If you just want to draw, and don't care about making it look cool, can skip this step, and the steps below.

Embrace Complexity

This project has many possible configurations. Here are two basic setups you might consider:

1. Connect a computer to the Sketch-Y-Etch, then to a display screen.
2. Connect a laptop's HDMI output directly to the Sketch-Y-Etch.

Many possibilities and functionalities exist at this stage. The zeitgeist of this project is to embrace experimentation, and creativity with flexibility. Take some time to consider how you want to configure your Sketch-Y-Etch.

While video pass-through is an option, if you have multiple display outputs, you may route your controls in various ways. You might choose to embed the Sketch-Y-Etch into a TV, build a standalone control panel, make multiple controllers on one computer, use a projector, or simply run the app on your computer without specialized electronics.

Is HDMI strictly required? Not necessarily. HDMI is modern and widely available, but it is not the only option. Other video or data interfaces could be adapted with additional development effort. Adaptions may change the path of these instructions, which should conceptually work regardless of your desired implementations. I am however not going to write a new tutorial for every conceivable build path, or video technology in existence.

Please keep in mind: this project remains experimental. It is not intended as a polished, professional consumer product. While it is feasible to create a more plug-and-play experience, the practical setup will depend heavily on the specific hardware you have available. This flexibility can be confusing, the key is to clearly define what you want to build before you start building it.

Optional Steps

These steps are not strictly required but are highly recommended to simplify testing and setup. 3D printing the knobs, and experimental build platform may take approximately an hour of printing, depending on your printer. Print 4 knobs, and 1 table. Please note that the current knob.stl file is sized for the encoders used in this project, if you are using some other unspecified component you may need to adjust the models to meet your specific needs.

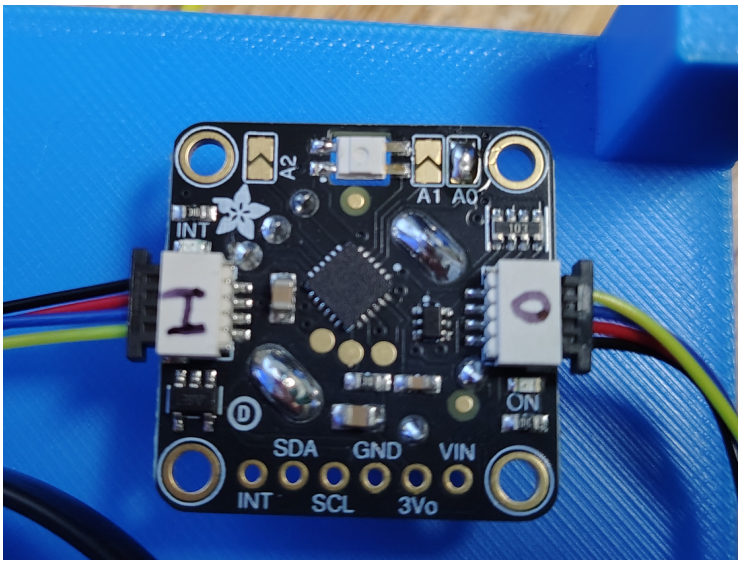
1. 3D print the `table.scad` build platform and install your devices onto it.
2. Drill or melt holes into the table to match your layout preferences.
3. Install the knobs by pushing them on to the encoder shafts. Using alternate knobs is also acceptable, we're just trying to make the encoders easy to use.
4. Test and verify all hardware and software functionality using the build platform before embedding the electronics into a TV.
5. If embedding a computer into an other device, its a good idea to have an easy means of access. Ssh, wireless keyboard, or leave the covers off until you are sure everything is running properly.

Wiring

Assuming you are following along with building the test platform first for testing, and your 4 rotary encoders are now installed, flip the table to view the encoders for wiring. The encoders have directionality, which may be counter intuitive, and limited a specific models. There is an input side, and an output side. If looking at the underside of the encoder, with the LED at the top, or away from you, the input port is on the left, and the output Qwiic connect port is on the right. You may opt to label the ports I for input and O for output. When connecting the encoders, you will connect from the output of the first encoder to the input of the next encoder in line. Here, you can observe the exposed A0, A1, and A2 pads. These pads control the I^2C addresses your encoders use to communicate on the bus. By default, all pads are disconnected, which is how the encoders are shipped from the factory. The default I^2C address is `0x36`.

By bridging these pads with solder, you manually set the device's address. You can attach your encoders in any order; however, each encoder must have a unique address. Up to eight address combinations are possible. Please note that whichever addresses you assign must be configured correctly in the software to ensure that each encoder performs its designated function.

Once you have assigned your 4 distinct addresses by soldering, record those addresses, and their respective functions per your board layout. You will need this information when you go to adapt the SketchyEtch.py code to your specific installation. The findbus function will search for your 4 encoders, when



the program loads. If you set your encoder addresses to something other than 0x36,0x37,0x38,0x39, you will need to change the known_addrs values to reflect the values you have set. The specific encoder addresses also control which encoder controls what Sketch-Y-Etch function. If in the code the findbus function does not find your encoders, the Sketch-Y-Etch program will default to using a keyboard for drawing. Depending on your project construction, this may be a problem. If your computer is embedded into your tv, or projector, and attaching a keyboard is difficult, then be sure to configure the known_addrs before hand. The sketch-y-etch program does not continuously scan for known addresses, rather is only scans during initialization.

More wiring

I used an ADS115 Analog to Digital Converter (ADC) to add 4 buttons to this project. Its wild overkill for the project, but Im going to build other projects with that part, so I wanted to figure out the logic for it anyway. As in using it was educational for me. For you, this part is cheap, and readily available. The logic of using the ADC is also flexible, and exceedingly simple once you have the code to listen to the device set up. If a button normally tied high, goes low, perform a function. How does that work? I used 4, 10K ohm resistors tied to +5VDC to the "A" inputs of the ADC, and the normally open switch tied to ground. when the button is pressed, the voltage on the input goes low. This is not a short circuit, because of the resistor. So then our code sees a low value on the channel where the button was pressed, and we respond accordingly by calling a function.

ADDR	A0	A1	A2
0x36	H	H	H
0x37	L	H	H
0x38	H	L	H
0x39	L	L	H
0x3A	H	H	L
0x3B	L	H	L
0x3C	H	L	L
0x3D	L	L	L

Figure 2: Encoder addresses `Empty pad is high by default`

Inspect

Inspection is an important step in every electronics project. Mistakes happen, the world is a distracting place, and the fragility of human existence can lead to some unfortunate side effect. Your electronics projects do not need to suffer from this. Look for loose wires, stray strands of wire poking out of terminals, connectors ajar, messy incoherent wiring... if you can spot problems before they start, it will be safer, and less confusing later. If everything on your test rig looks good, try plugging it in.
projects rarely work perfectly the first time around, even with expertly drafted and bespoke instructions.

Plug your test rig in. Fire? unplug it.
No fire? no smells?
Great!! you have passed the first step of electronics tinkering! Did your externally attach monitor resume video playback? Did the power lights on the encoders illuminate?

If every thing seems to be working, then lets proceed. If you start to smell something, or your computer begins to behave erratically, I would suspect the newly attached experimental electronics project, and unplug it as best practice.

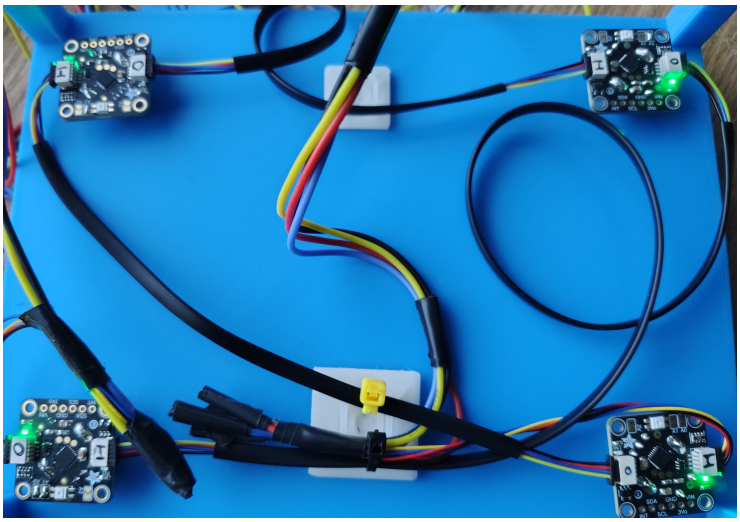


Figure 3: Lights on! `no fire!`

Test Hardware

Adapt the code

Play Time!

Troubleshooting

0.1 No video:

If you ran the solitary signal ground though the qwiic connectors, a disconnect on any of the encoders could prevent video pass thru.
Check your sensors, and encoders. The kit version should not be as susceptible to this issue. The i2c bus is not strictly required by the HDMI video connection, its use is in identifying connected devices. The signal ground however is required for the video signals. You could add a ground cable, or just check your wiring.

0.2 Flashy Flashy:

flickering, or stuttering can occur when the Hot Plug Detect is wired straight through, and or not tied to the +5VDC line. The GPU, or video device is basically always searching for, and confirming the eeprom of the attached screen, TV, projector etc. When the detection is confirmed, the device sends a signal in the form of +5VDC to the video driver verifying the connection. If the HPD passes straight through, your screen will flicker at the rate of the polling. Tying

the HPD high prevents the flicker, but does not stop the polling. Again, check your wiring.

0.3 screen tearing:

Depending on the size of your screen and the computer you use to supply video, you could run out of video memory. This is exceedingly close on screens larger than 52 inches on devices like the raspberry pi which do not have a lot of video memory.

0.4 Video output issues:

try removing the pass through. custom connections may be wired incorrectly.

Contact

1. You love this project, and want to contribute!
2. You have commentary about this document, project, code and want further detail.
3. Complaints. This project sucks! Believe me, I KNOW. With a wanton disregard for more performant hardware, and methods I bravely undertook this project despite the stern remarks from veteran Electrical Engineers and programmers alike. Don't tell me how to have fun!

Nobody asked:

0.5 I2C bus topologies:

Venturing into the realm of TMI, learning about I2C bus topologies can give you insight into various ways you can build future projects.

0.6 i2c Sucks!

Well, yeah, kinda.

- **Very Slow**: Typical speeds are **100 kHz**, to maybe **400 kHz**
- **Short Distance**: Designed for communication across a single PCB or a few inches of wire. Beyond ∼1 meter, signal integrity issues like reflections, capacitance, and noise become problematic.
- **Shared Bus = Traffic Jams**: All devices share the same two wires (SDA and SCL). A slow or malfunctioning device can stall the entire bus.

- **Limited Error Handling**: Only basic ACK/NACK mechanisms are provided; no robust error correction or retry protocols.
- **Limited Bandwidth**: As a shared bus, more devices mean reduced effective performance.
- **Requires Pull-Up Resistors**: Since I^2C uses an open-drain configuration, pull-up resistors are needed, leading to:
  - Slow signal rise times
  - Higher active power consumption
  - Sensitivity to wiring quality and length

But its also easy to implement, as I hope this project shows, and devices using it are prevalent.

1 Aversions to hardware hacking:

yes we took a devices and modified its normal operation to garner our own distinct operation. I've heard all manner or cries about the feasibility of interrupting the I2C bus on a video port. I have news for you: while the typical operation is to use video output devices to communicate with internal memory of screens, and TVs and projectors, a wide array of devices also interrupt the I2C bus to garner their own functionality. So one, this operation is not all so outlandish. Further still, any given I2C buss can communicate with up to 127 devices, and the typical utilization is just using 1. The same folks screaming about interrupting the i2c bus have no problem with a 20 foot long HDMI cable which subjects the circuit to the same sorts of noises and losses.

Helpful Tools

The Sketch-Y-Etch was developed primarily on Linux using Python 3 alongside a variety of additional software tools:

- OpenSCAD (for 3D design): `https://openscad.org`
- Geany (lightweight code editor): `https://www.geany.org`
- KiCad (PCB design and schematic capture): `https://www.kicad.org`
- i2c-tools (I^2C communication utilities): `https://archive.kernel.org/oldwiki/i2c.wiki.kernel.org/index.php/I2C_Tools.html`

While the design should, in principle, work on any operating system, platform-specific nuances have not been extensively tested. The tools listed above reflect the preferences of the original developer. Although not strictly required, they are highly recommended for a smoother and more educational development experience.

References

- Turtle Graphics API (Python Standard Library): `https://docs.python.org/3/library/turtle.html`
- Smbus2 (Python I^2C library) : `https://pypi.org/project/smbus2`
- I^2C Implementation Guide (SparkFun): `https://learn.sparkfun.com/tutorials/i2c/all`
- Adafruit encoder pin outs `https://learn.adafruit.com/adafruit-i2c-qt-rotary-encoder/pinouts`
- Adafruit Seesaw Library: `https://github.com/adafruit/Adafruit_Seesaw`
- Component Data sheets: Included in this repository.
- The Book of I^2C by Randal Hyde `https://nostarch.com/book-i%C2%B2c`

All references are provided for educational purposes. They are not required to complete the project but may offer valuable background information.

burgeons:

A fun side project is to modify your TV, projector, or screen to report its identity as a Sketch-Y-Etch, or whatever you want I suppose. But this is a topic for a different discussion.