UNTZtrument: a Trellis MIDI Instrument

Created by Phillip Burgess



https://learn.adafruit.com/untztrument-trellis-midi-instrument

Last updated on 2022-12-01 04:15:25 PM EST

Table of Contents

Overview 3

- What UNTZtrument is:
- What UNTZtrument is not:

First Steps 6

- Let's Get Started!
- Introducing Adafruit Trellis
- Troubleshooting

Assemble Case 13

Software 23

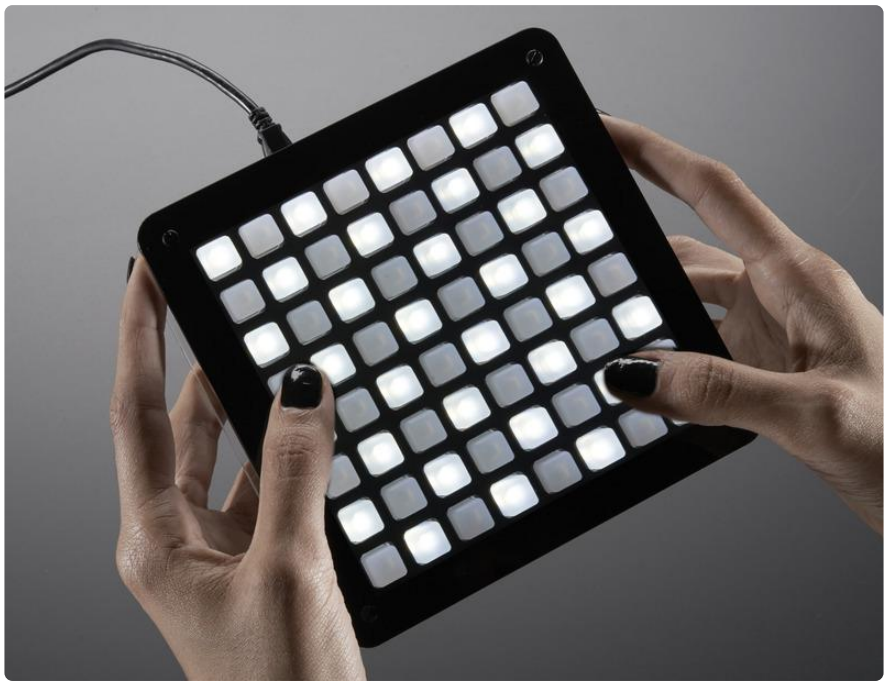
- Installing Arduino Libraries
- Uploading Code
- Host-Side Software
- Using the UNTZtrument_Hello_World Example
- Using the UNTZtrument_Step_Seq Example

Hacking 27

- Hardware Considerations
- Software Considerations
- Using the UNTZtrument Arduino Library

Downloads 36

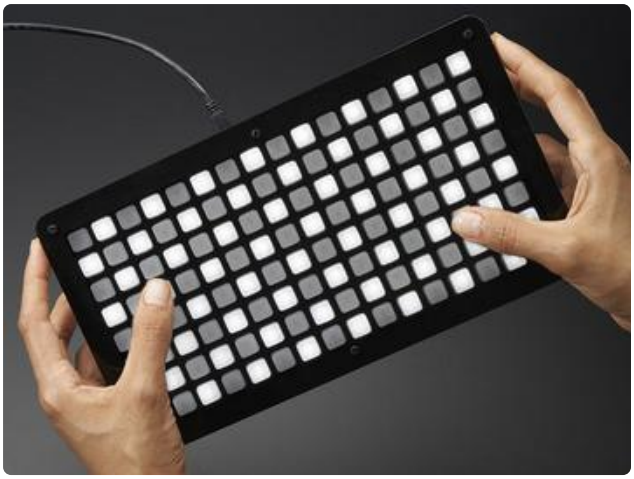
Overview



Build and customize your very own open-source button grid controller with UNTZtrument! This DIY kit comes with delicious translucent button pads, driver boards, diffused white LEDS and a custom laser cut enclosure. The result is a sturdy and elegant but also super-hackable controller for music, video...or something else???

We designed this kit for ease of use and ultimate flexibility. All you need is an Arduino Leonardo, basic soldering tools and an afternoon. Once assembled and programmed with the Arduino IDE, this box turns into a USB MIDI device that works with any computer and has 64 buttons (128 on the HELLA UNTZtrument) with individually-controllable LEDs. Our example programs send simple MIDI Note On and Off messages, but with a little programming ingenuity you can send and receive any kind of MIDI command. Since its USB MIDI it can work instantly with just about all synth software. Don't like MIDI? The Arduino Leonardo can also emulate a USB keyboard or plain old USB serial.

Since it's Arduino-powered, adding more stuff like accelerometers, potentiometers, rotary encoders, etc. is straight-forward using existing libraries available on the Internets.

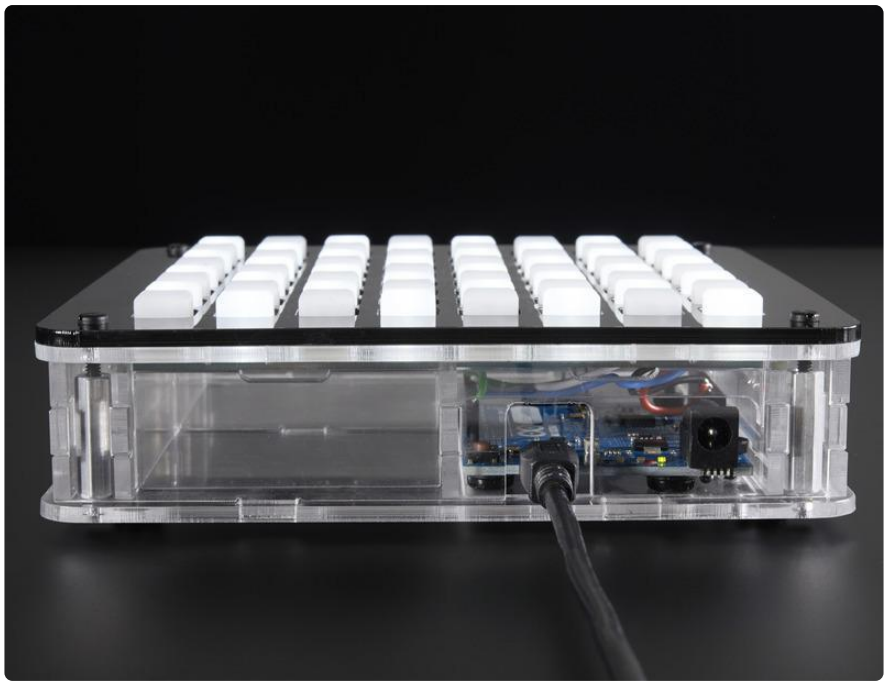


HELLA UNTZtrument ups the ante with 128 LED-backed buttons. It's huge!

What UNTZtrument is:



- UNTZtrument is a 64- or 128-button MIDI device that works alongside music software on your computer.
- It's a kit, requiring some soldering, a few tools and a little prior experience with the Arduino microcontroller.
- UNTZtrument is open source. The software is free and you can mix it up to add your own new features, or use code that others have written.



What UNTZtrument is not:

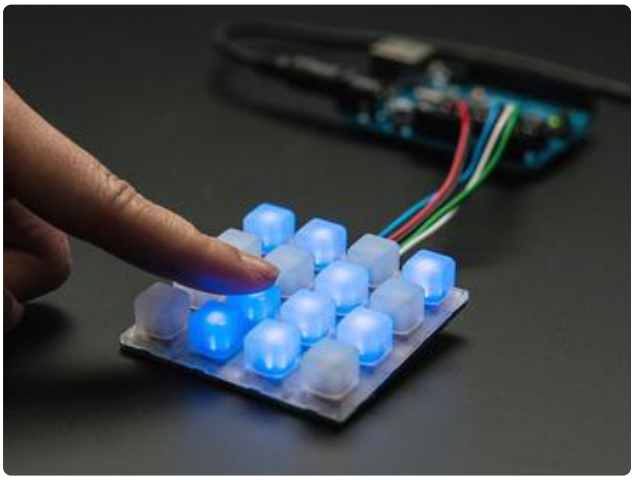
- UNTZtrument is not a self-contained musical instrument. It generates no audio and must be connected via USB to a computer to either create sounds or forward MIDI data to a synthesizer.
- UNTZtrument is not a Monome (or Arduinome), but looks similar. Those are seria l USB devices requiring a software bridge to the Max visual programming language. UNTZtrument speaks MIDI, which is native to most music software.*

* UNTZtrument could certainly be adapted to be compatible with these...yay for open source...but we've found said bridge software to currently be quite finicky. Using MIDI now doesn't preclude using serial for other things later...we might revisit this as the situation evolves.

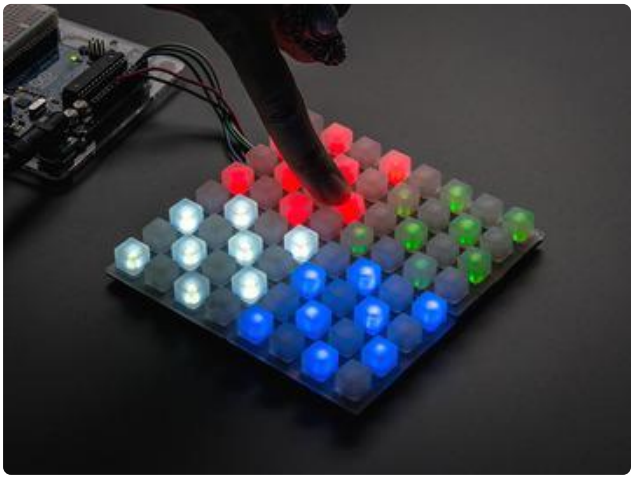


Here Collin Cunningham explains the basics of MIDI:

First Steps



UNTZtrument is based on Adafruit's Trellis, a 4x4 backlit keypad kit.



Four Trellises are combined to make a single large 8x8 matrix for UNTZtrument. The HELLA UNTZtrument has eight Trellises in a 16x8 matrix!

Normally you have to buy three separate parts for each Trellis (PCB, elastomer keypad and LED pack), but the UNTZtrument kits have everything you need.

We selected white LEDs for the UNTZtrument kits. If you have a large stash of 3mm LEDs in some other color you can certainly use those instead.

You will also need an Arduino Leonardo microcontroller board, either the regular version () or the headerless variety () if you want permanent connections.

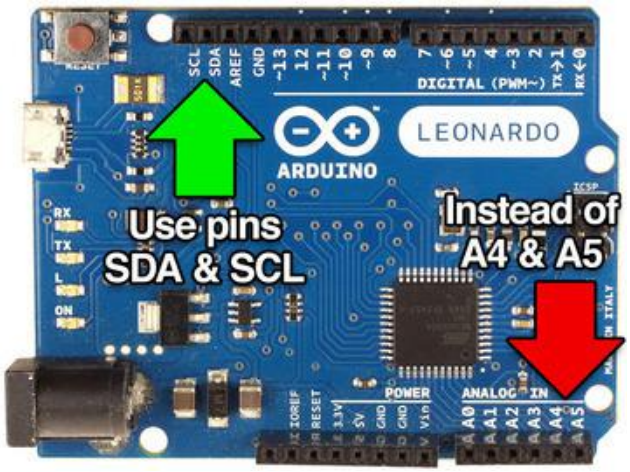
UNTZtrument will not work with the Arduino Uno, Mega or other boards. Must be the Arduino Leonardo, or a 100% compatible board based on the ATmega32U4 microcontroller. Because MIDI.

Arduino Leonardo. Period.

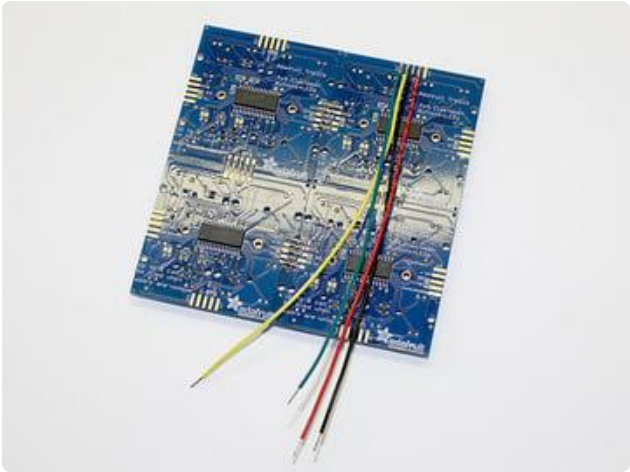
Additionally, you'll need some wire (22 gauge solid-core () wire is ideal, but stranded can work in a pinch), a soldering iron & solder, basic hand tools and a Micro USB cable.

Let's Get Started!

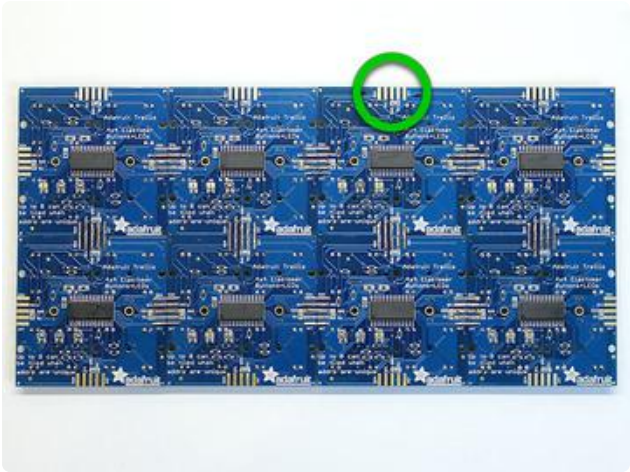
So, as a first step to building your UNTZtrument kit, work through our introductory Trellis guide () first. But with a few important changes:



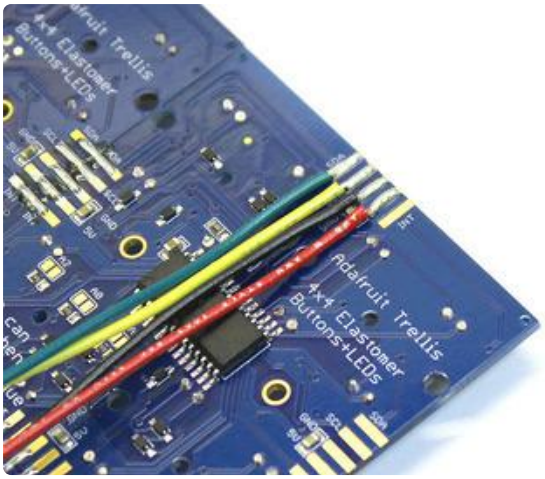
Because UNTZtrument is based around the Arduino Leonardo, it requires slightly different wiring: Use the SDA and SCL pins instead of A4 and A5 as shown in the guide.



Looking at the back of the tiled Trellis boards...with the text upright, in the normal orientation for reading...connect the wires to the header along the top edge, toward the right. They should be about 6 inches (15 cm) long, or a little longer.



On the HELLA UNTZtrument — a 4x2 assembly of Trellises — connect the wires to the third header along the top edge, not the rightmost fourth header.



Try pointing the wires “inward” rather than off the edge of the board. This makes it easier to fit in the case.

The INT pin is not used by UNTZtrument and does not need to be connected.



When setting the board addresses (by bridging the solder points on the back), use the values shown here.

Using a different order is not catastrophic, you’ll just need to edit the code to match. Following this standard makes it easier to share code with other UNTZtrument users.

Check the orientation of the boards and use the large Adafruit silkscreen logo to make sure you have them oriented right and the correct addresses set.



Here’s the address map for the HELLA UNTZtrument.

Use the code below for testing your UNTZtrument matrix rather than the code in the Trellis guide; it’s already set up for the tiled matrix.

For the HELLA UNTZtrument, change NUMTRELLIS to 8.

```
/*************************************************************
  This is a test example for the Adafruit Trellis w/HT16K33.
  Reads buttons and sets/clears LEDs in a loop.
  "momentary" mode lights only when a button is pressed.
  "latching" mode toggles LED on/off when pressed.
  4 or 8 matrices can be used.  #define NUMTRELLIS to the
  number in use.

  Designed specifically to work with the Adafruit Trellis
  ----&gt; https://www.adafruit.com/products/1616
  ----&gt; https://www.adafruit.com/products/1611

  Adafruit invests time and resources providing this
  open source code, please support Adafruit and open-source
  hardware by purchasing products from Adafruit!

  Written by Limor Fried/Ladyada for Adafruit Industries.
  MIT license, all text above must be included in any redistribution
 *************************************************************/

#include &lt;Wire.h&gt;
#include "Adafruit_Trellis.h"

#define NUMTRELLIS 4        // **** SET # OF TRELLISES HERE

#define MOMENTARY 0
#define LATCHING  1

#define MODE      LATCHING  // **** SET MODE HERE

Adafruit_Trellis matrix[NUMTRELLIS] = {
  Adafruit_Trellis(), Adafruit_Trellis(),
  Adafruit_Trellis(), Adafruit_Trellis()
#if NUMTRELLIS &gt; 4
 ,Adafruit_Trellis(), Adafruit_Trellis(),
  Adafruit_Trellis(), Adafruit_Trellis()
#endif
};

Adafruit_TrellisSet trellis = Adafruit_TrellisSet(
  &amp;matrix[0], &amp;matrix[1], &amp;matrix[2], &amp;matrix[3]
#if NUMTRELLIS &gt; 4
 ,&amp;matrix[4], &amp;matrix[5], &amp;matrix[6], &amp;matrix[7]
#endif
);

#define numKeys (NUMTRELLIS * 16)

// Connect Trellis Vin to 5V and Ground to ground.
// Connect I2C SDA pin to your Arduino SDA line.
// Connect I2C SCL pin to your Arduino SCL line.
// All Trellises share the SDA, SCL and INT pin!
// Even 8 tiles use only 3 wires max.

void setup() {
  Serial.begin(9600);
  Serial.println("Trellis Demo");

  // begin() with the addresses of each panel.
  // I find it easiest if the addresses are in order.
  trellis.begin(
    0x70, 0x71, 0x72, 0x73
#if NUMTRELLIS &gt; 4
   ,0x74, 0x75, 0x76, 0x77
#endif
  );

  // light up all the LEDs in order
```

```
  for (uint8_t i=0; i&lt;numKeys; i++) {
    trellis.setLED(i);
    trellis.writeDisplay();
    delay(50);
  }
  // then turn them off
  for (uint8_t i=0; i&lt;numKeys; i++) {
    trellis.clrLED(i);
    trellis.writeDisplay();
    delay(50);
  }
}

void loop() {
  delay(30); // 30ms delay is required, dont remove me!

  if (MODE == MOMENTARY) {
    // If a button was just pressed or released...
    if (trellis.readSwitches()) {
      // go through every button
      for (uint8_t i=0; i&lt;numKeys; i++) {
    // if it was pressed, turn it on
    if (trellis.justPressed(i)) {
      Serial.print("v"); Serial.println(i);
      trellis.setLED(i);
    }
    // if it was released, turn it off
    if (trellis.justReleased(i)) {
      Serial.print("^"); Serial.println(i);
      trellis.clrLED(i);
    }
      }
      // tell the trellis to set the LEDs we requested
      trellis.writeDisplay();
    }
  }

  if (MODE == LATCHING) {
    // If a button was just pressed or released...
    if (trellis.readSwitches()) {
      // go through every button
      for (uint8_t i=0; i&lt;numKeys; i++) {
        // if it was pressed...
    if (trellis.justPressed(i)) {
      Serial.print("v"); Serial.println(i);
      // Alternate the LED
      if (trellis.isLED(i))
        trellis.clrLED(i);
      else
        trellis.setLED(i);
        }
      }
      // tell the trellis to set the LEDs we requested
      trellis.writeDisplay();
    }
  }
}
```

So...with those changes in mind...here's a link to the starter guide:

Introducing Adafruit Trellis ()

Don't continue with the UNTZtrument guide until you have a tested and working 8x8 or 16x8 Trellis.

Troubleshooting

Some of the LEDs don't light up!

- If the positions are somewhat random: some LEDs might have been installed backwards, or might've been damaged from excessive heat. Happens all the time, not to worry. This is why we include lots of spares. De-solder the problem LEDs and clean up the holes using a solder sucker, then replace them with new ones (in the correct orientation).
- If it's a complete quadrant of the 8x8 Trellis, the address jumpers on the back of the board might be improperly set. Refer to the diagram above.

None of the LEDs light up!

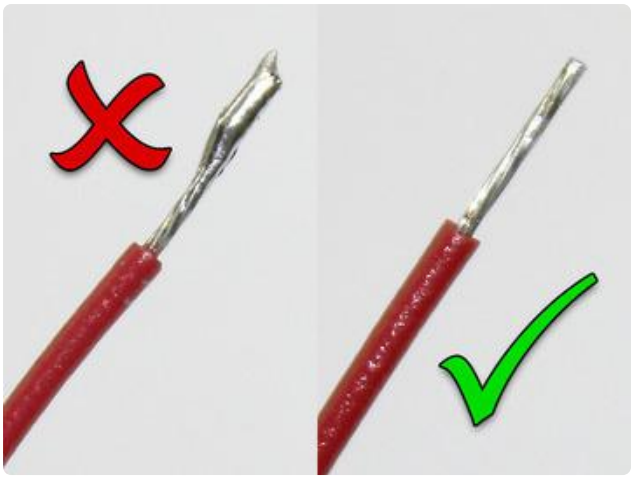
Might be the wiring. The Arduino Leonardo requires the use of different pins when communicating with the Trellis boards, or you might just have the wires swapped. Refer to the diagram above.

The wires keep snapping off the board!

Wires shearing off are usually due to where the solder has not fully wetted the pad and flowed smoothly between the pad and wire. Make sure you're fully heating the pad and wire first adding solder... melt solder on the tip of the iron and then carry it to the wire, that's a recipe for failure.

I can't fit the wires into the Arduino sockets!

It's a little easier to build UNTZtrument with solid-core wire (it slides right into the Arduino headers), but sometimes stranded is all you've got. Too-fat wires can happen if you've tinned the tip of a stranded wire with excessive solder...



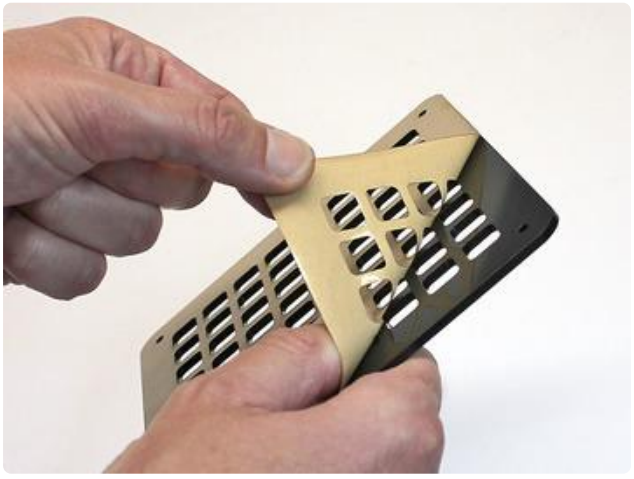
When tinning stranded wires (to prevent fraying and shorts), use just a tiny bit of solder, and make sure it fully flows into the strands. Sometimes you'll want to clip off the very tip of the wire if the iron left a glop or spike there.

The Adafruit Guide to Excellent Soldering () has lots of advice for common soldering problems.

Reminder: don't proceed until you have a fully tested and working Trellis + Leonardo on your desk. It's not easy to debug soldering once its inside the case!

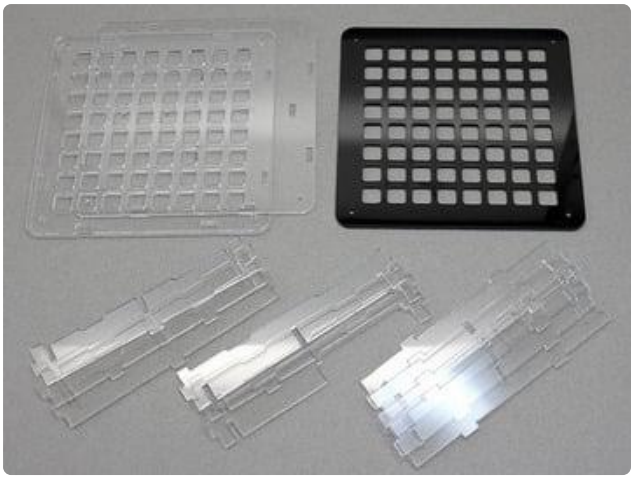
Assemble Case

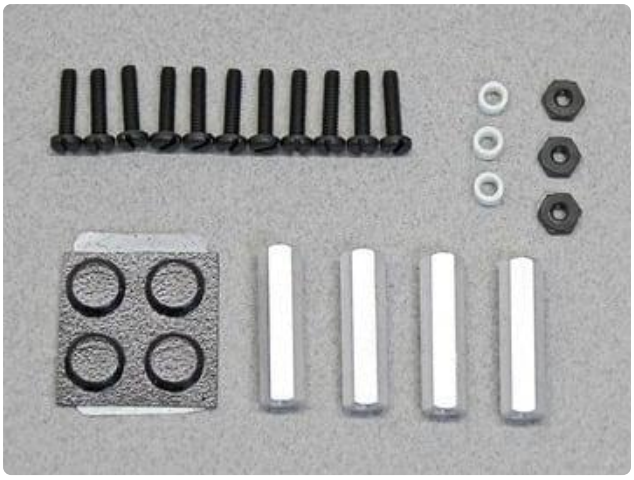
You should have a tested and fully working Trellis (without case) at this point. Don't proceed until you've reached that milestone.



Start by peeling the backing paper off both sides of all the laser-cut parts. It's easiest to start at a corner, catching the edge of the paper with a fingernail.

The laser-cutting process sometimes leaves a little paper soot at the edges. If you like, you can wash these off with soap and water, just be absolutely certain that all the parts are completely dry before proceeding!





All told, there should be 13 laser-cut parts (15 for HELLA UNTZtrument). Most are clear except for the one black grid piece.

Your kit should also include the following hardware:

Eleven (11) nylon screws
Three (3) nylon nuts
Three (3) 1/8" board spacers
Four (4) 1" threaded standoffs
One set of 4 peel-and-stick rubber feet
HELLA UNTZtrument has 2 more standoffs and 4 more screws for the wider case.

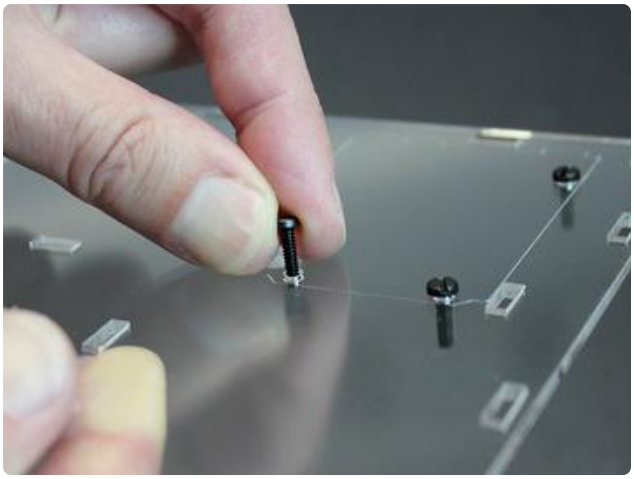
If any parts are missing or damaged, contact support@adafruit.com to arrange for a replacement.

You've already assembled and tested the Trellis and Arduino Leonardo boards; they're implied but not listed in the above inventory.



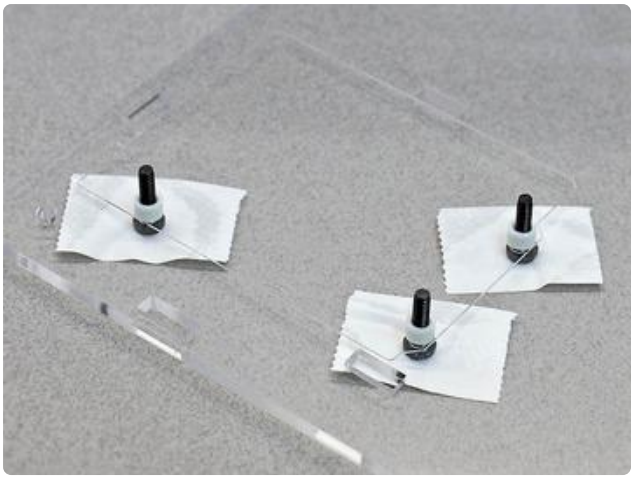
We'll start with the base piece...it's one of the large squares, the one without the waffle grid.

This piece has the Arduino footprint scored on one face. This helps identify the top surface.

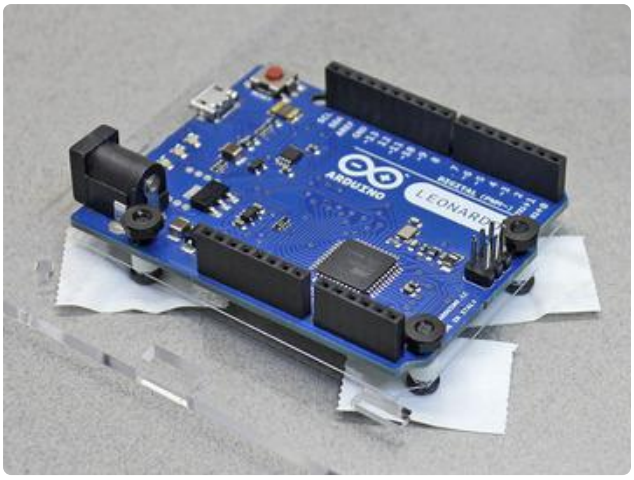


Turn the base over and install three screws in the Arduino footprint area.

This piece needs to be turned back over for additional work. You can either put a little masking tape over the head of each screw to hold them in place...or, if you're dextrous, grip the three screws from the other side as you turn it over.



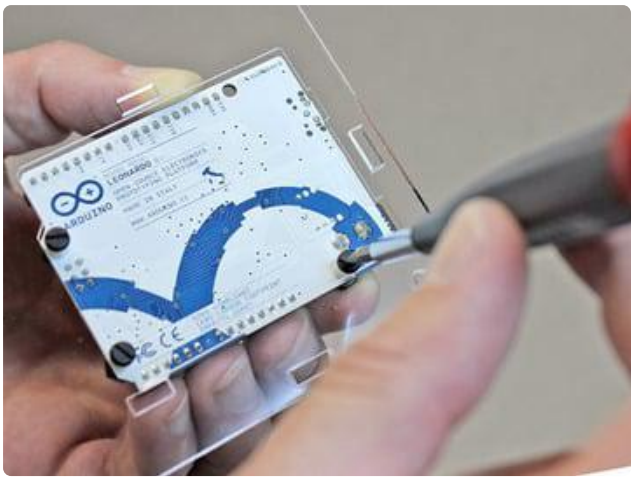
Set the base piece down so the screws are now pointed up.



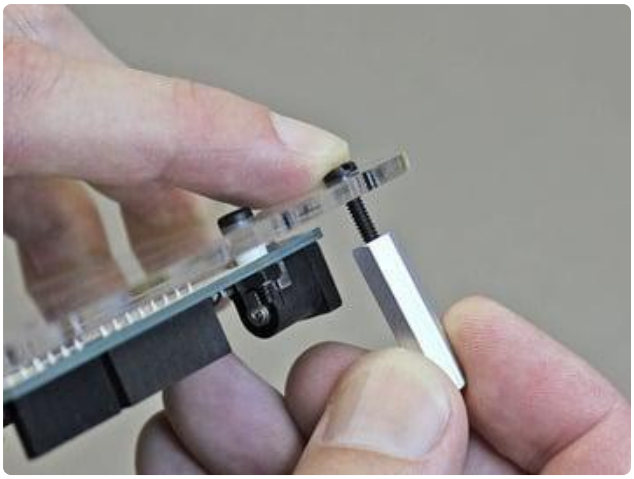
Add a 1/8" nylon standoff over each screw.

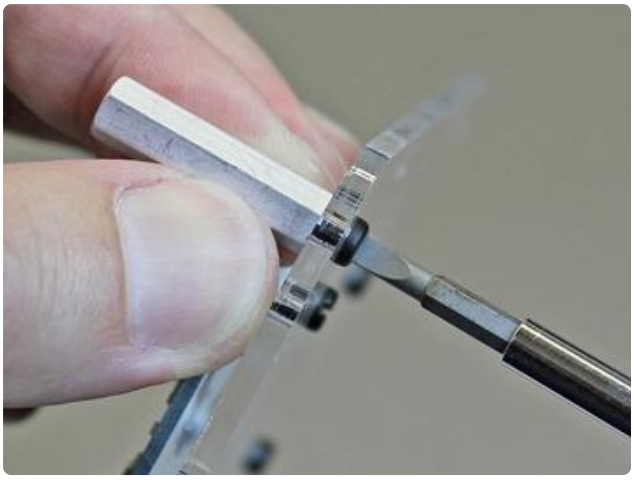
Install the Arduino board with the mounting holes over these three screws (there's a fourth hole, but we're not using it here).

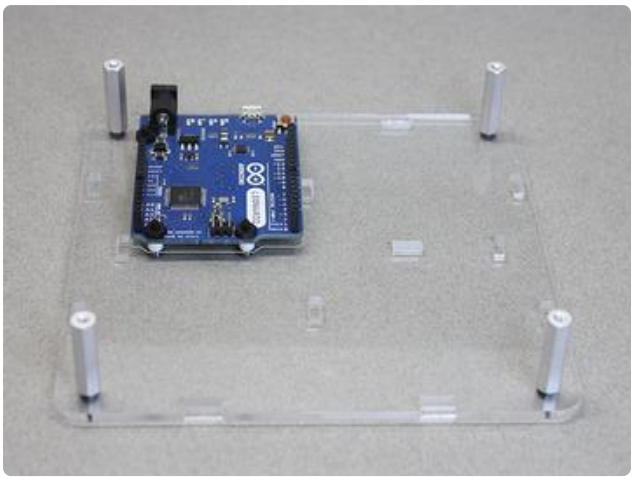
Add a nut to each screw. There will probably be some mechanical interference from nearby headers and parts...that's okay, you just need to get the nuts started.



Now turn the base over, remove the tape (if used) and gently tighten the three screws with a small screwdriver.



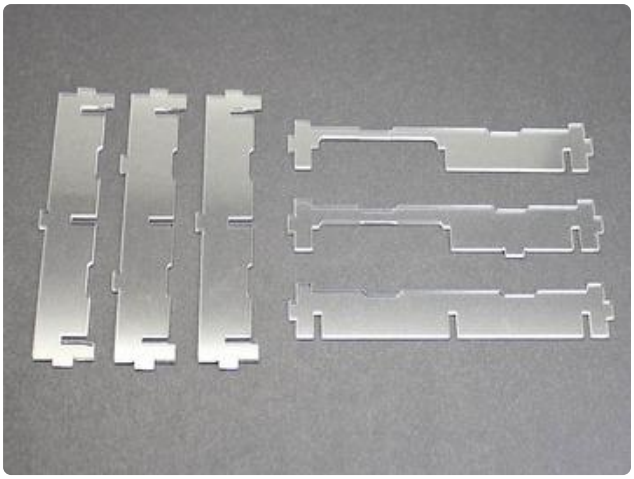


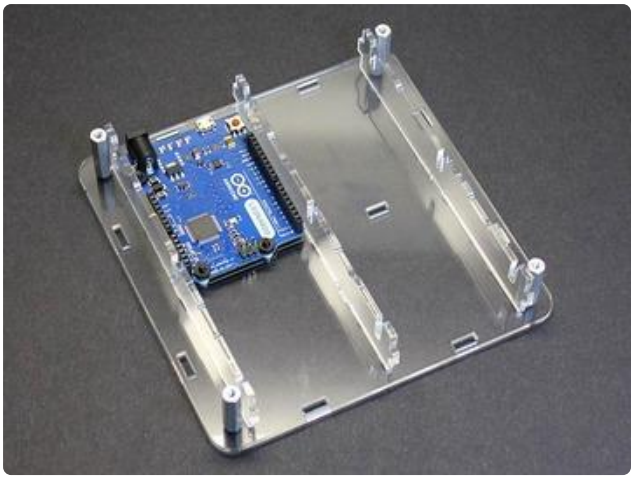


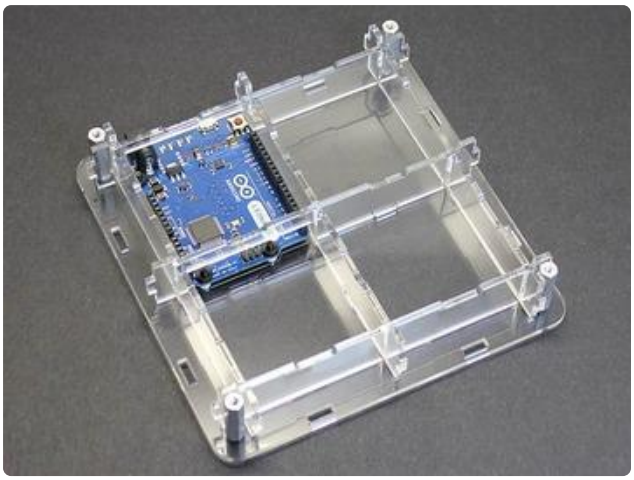
The standoffs are easier, we can do them one at a time. No need for tape.

Insert a screw into one of the four corner holes. Come up from the underside, as you did with the Arduino. Catch the screw in the threads of the standoff and turn it into place. Finger pressure is usually sufficient, or you can gently use a screwdriver.

Repeat until all four standoffs are installed (six for HELLA UNTZtrument). You should then be able to set the base down with the Arduino and standoffs all on the top side.



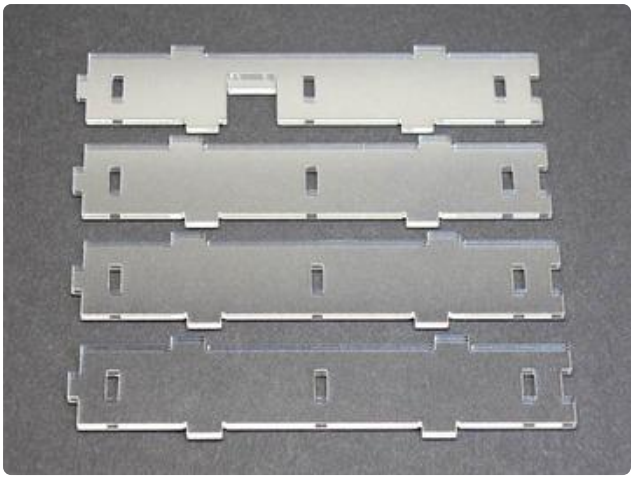


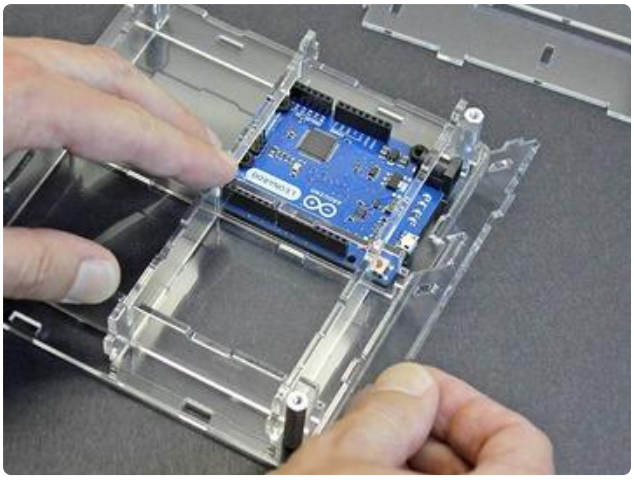


Next we'll install the vertical and horizontal braces.

The vertical braces (which run parallel to the Arduino's longer axis) all look similar, but notice the "nubs" on the bottom are different: one, two, one. These fit into corresponding slots on the base. (On the HELLA UNTZtrument, there are five vertical braces — again, they're keyed to only fit specific positions).

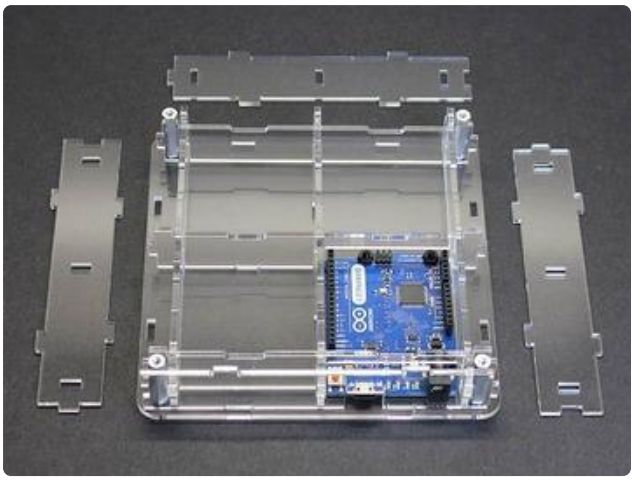
The horizontal braces now slot into notches on the verticals. Each of these is a different shape. The thinnest one is designed to clear the Arduino's power jack. The slightly thicker (but still pretty thin) brace goes down the middle, while the full-thickness one is at the bottom.

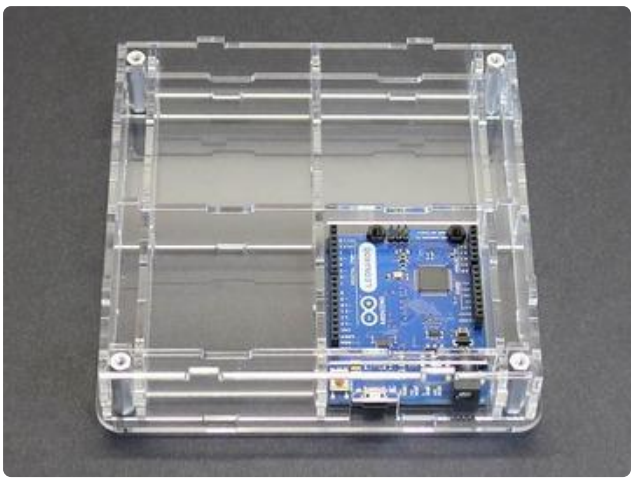


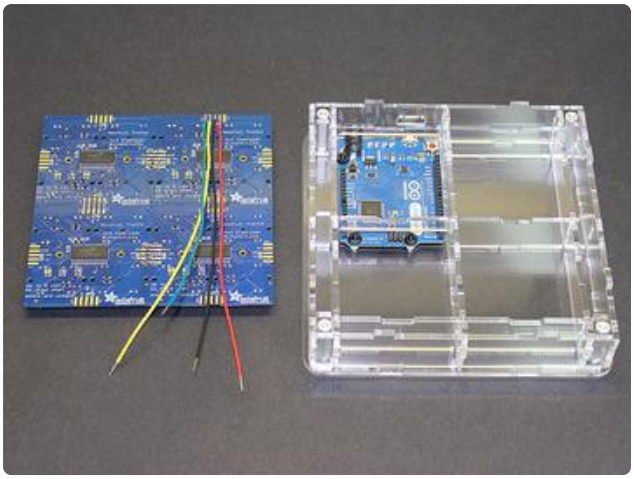


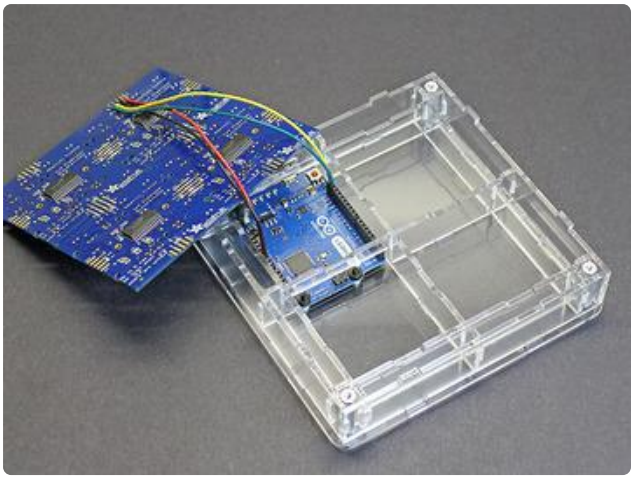
The four side pieces are almost identical... except for one which has a notch for the Arduino's USB port. Install this side first. Align the bottom tabs with the slots in the base and tilt it up into place. If the USB port is covered, you've got it backwards — turn it around and try again.

Once the first side is in place, the alignment of the remaining three should be straightforward.







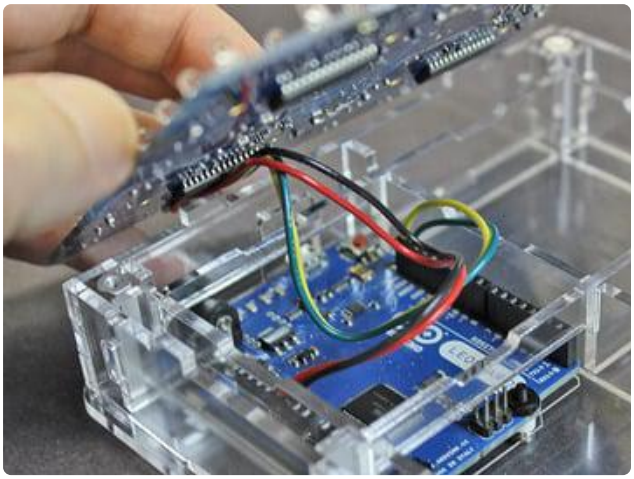


Now get the Trellis board ready. Remove the rubber buttons for the time being and turn the board face-down.

Orientation is important. Once installed, the top edge of the Trellis will face be aligned with the Arduino's USB port.

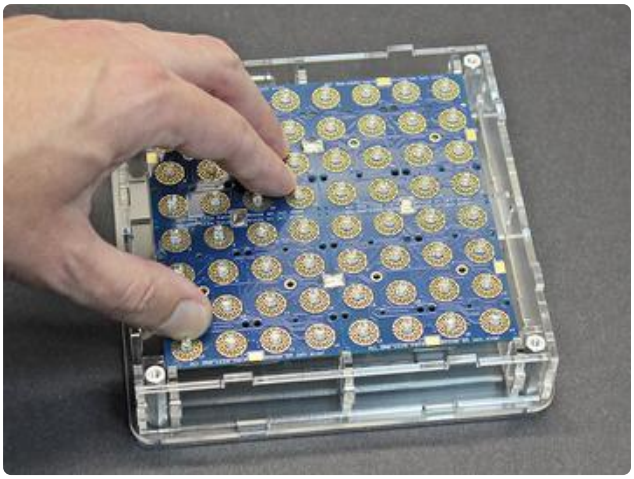
Insert the power and signal wires from the Trellis to the headers on the Arduino:
5V from Trellis to 5V on Arduino
GND from Trellis to any GND on Arduino
SCL from Trellis to SCL on Arduino (this pin is nearest the mounting hole with no screw through it)
SDA from Trellis to SDA on Arduino (this is the second pin, next to SCL)
The INT pin is not used for this project; no connection is necessary.

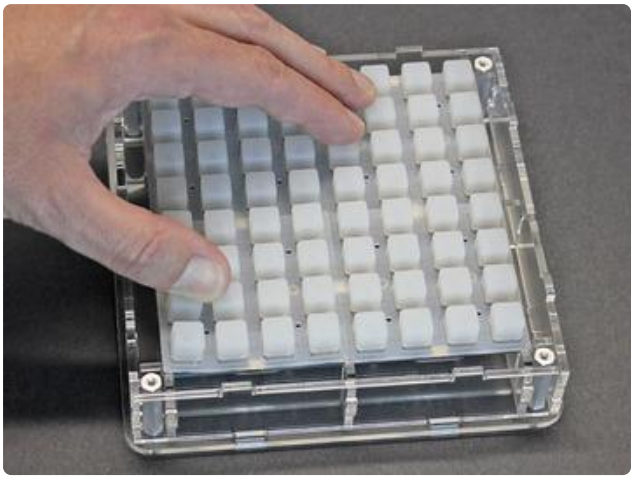
Now the Trellis gets turned over, LED side up, and is lowered into place in the case. You'll see the horizontal and vertical braces have notches into which the board neatly fits.



The wires will fight you at this stage, either pushing back against the trellis or springing out of the Arduino sockets. This part goes a little easier if you bend a couple zig-zag kinks in the wires, keeping them all within that quadrant of the case. Be patient and put each wire back in place if they pop out.

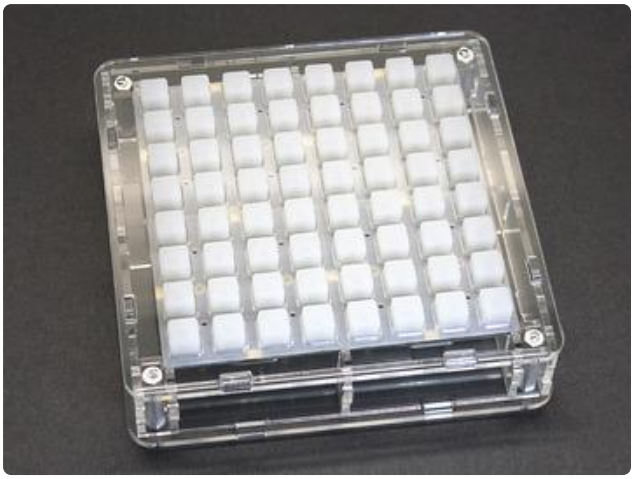
If your wires are connected to a different location on the Trellis, that's okay...the braces have notches to allow wires to run from quadrant to quadrant.





Once the Trellis is in place, you'll probably need to hold it down as the wires press against it. This is normal.

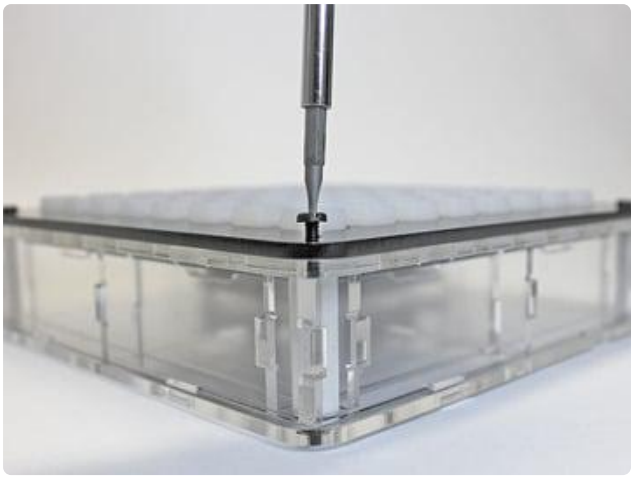
While still holding this down, install the rubber button elastomers over the Trellis, aligning the "nubs" with the corresponding holes in the board. You'll need to scoot these around a little until they all sit flush.



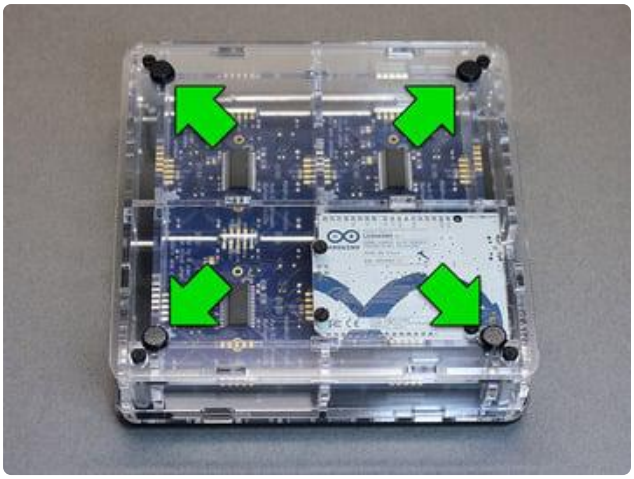


Install the clear top “waffle” piece. Since you’ll still be holding the Trellis down, start from one edge, aligning the notches with the tabs on the side pieces, and swing this down into place as you remove your other hand.

When everything is aligned and sits flush, you can place the black surface piece over this.

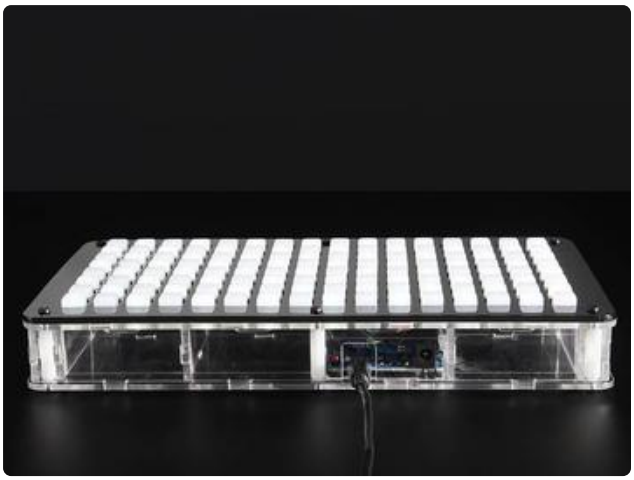


Install four corner screws through both top waffles and into the standoffs.



Turn the whole thing over and peel-and-stick the four rubber feet near the corners.





Turn it back over and you're ready to rock! Plug in a USB micro B cable and we'll turn our attention back to the computer...

Software

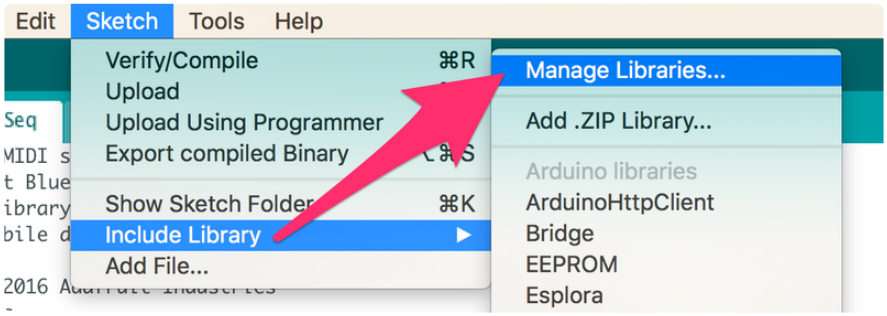
Getting an Arduino talking MIDI over USB has traditionally been a bag of hurt, but thanks to a few smart folks it can now be accomplished with the right bits of software.

You'll start with a plain vanilla Arduino software installation, version 1.6.9 or later. This can be downloaded from the Arduino web site (). Do not use the 1.5.x beta software for this, nor older releases!

Installing Arduino Libraries

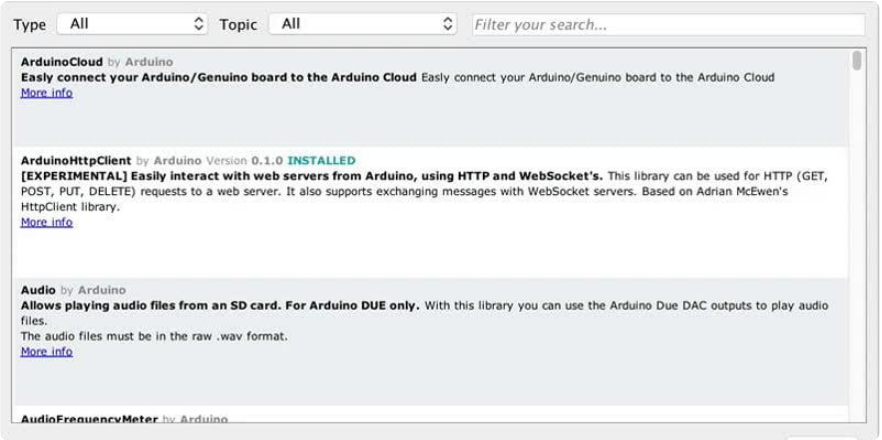
Open up your Arduino IDE and open the Manage Libraries... dialog under the Sketch→Include Library menu:

If you don't see the Manage Libraries option, make sure you downloaded the Arduino IDE from arduino.cc! Older versions and derivatives of the IDE may not have it.



You will need to install the following libraries using the Library Manager:

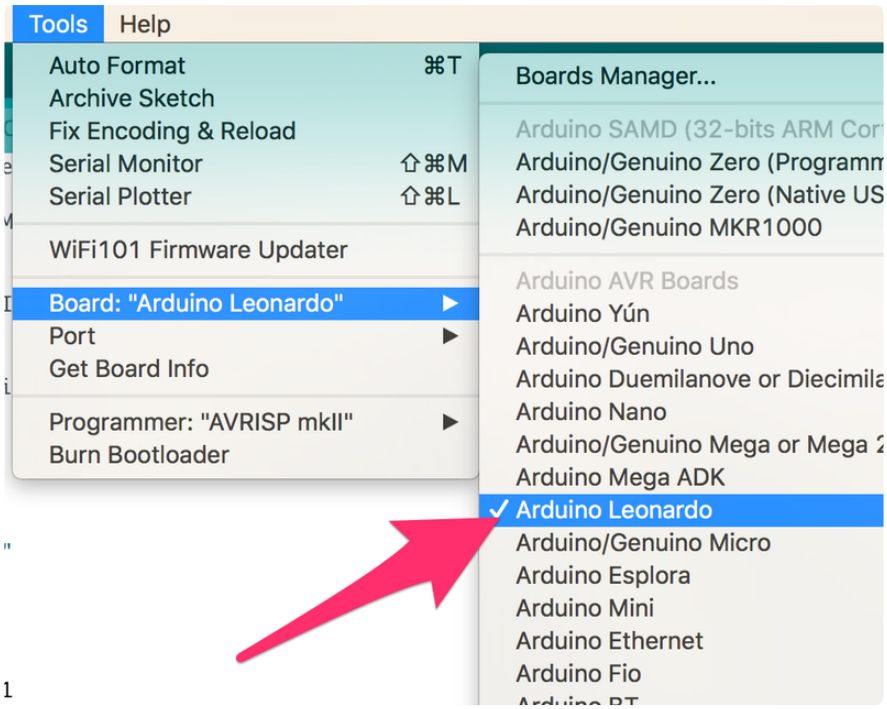
- MIDIUSB
- Adafruit Trellis
- Adafruit UNTZtrument



Uploading Code

There's still a little song and dance to be done, but it's easier than MIDI The Old Way™.

First, select Arduino Leonardo from the Tools→Board menu.



Now, from the Tools→Serial Port menu, select the port corresponding to the Arduino Leonardo board.

Open one of the UNTZtrument example sketches in the File→Examples menu. If using a HELLA UNTZtrument (16x8), add this line near the top of the code:

```
#define HELLA
```

You don't need this change for a regular 8x8 UNTZtrument.

Before uploading the code to the board, unplug and re-plug the USB cable to restart the bootloader again. There's a 10 second window to then click Upload. If you miss it on the first try, just try again with the cable-replugging and uploading.

There's just one more piece we'll need to finish the setup...

Host-Side Software

UNTZtrument issues MIDI data over USB. Now you'll need software on the computer at the other end of that cable.

This requires either a software synthesizer — a program that generates sounds using your computer's audio hardware — or a MIDI patchbay utility, which forwards the MIDI data to a hardware synthesizer.

If you've done music on your computer before, you probably already have one or both of these tools. Otherwise, there are a ton of options out there, many of them free. For example, on Windows there's the freeware Firebird2. Mac has the very basic SimpleSynth or the soup-to-nuts GarageBand. Even Linux has options. Google around for "software synthesizer", "free" and "Windows", "Mac" or "Linux" and see what you find.

There are so many options, we can't walk you through this for every app or operating system. It's usually pretty straightforward though, we're confident you can find something and get it installed and running.

Using the UNTZtrument_Hello_World Example

If you haven't already uploaded this example, run the Arduino IDE, open this sketch and select "Arduino Leonardo" from the Tools→Board menu and "MIDI" from the Tools→USB Type menu. Unplug and replug the USB cable from the device and then quickly press the Upload button (if this fails the first time, try again, there's about a 10 second window for uploading the code).

(Remember, you'll need to add one line as previously described if using a HELLA UNTZtrument.)

This is a minimal Trellis-and-MIDI example that simply maps each button to a different MIDI note...lowest at the top-left, highest at the bottom right. It's not trying to mimic a specific instrument, nor is there any relationship between rows and octaves or such. It's basically to test communication between the device and synthesizer.

Using the UNTZtrument_Step_Seq Example

This is a simple example of a step sequencer, sometimes used for creating rhythms. Each of the eight horizontal rows of the matrix corresponds to a different sound, while the eight vertical columns are scanned in sequence...anywhere there's an intersection (a pixel that's previously been lit by pressing its button) you'll get a sound.

Popular among electronic dance music. Hence the name, UNTZtrument.

Hacking

UNTZtrument is just the start. We hope you'll hack it to your heart's content. Some ideas:

- New types of MIDI musical instruments or sequencers.
- Beyond music...UNTZtrument could trigger video clips and effects for live "veejay" performances.
- The Arduino Leonardo can also act as a HID keyboard or mouse...how about a game controller or an assistive device?

Not just code...there's a little space for hardware hacking as well:

- Encoders or potentiometers could be added to control different music parameters (volume, tempo, etc.).
- A popular mod for the Arduinome (an inspiration for the UNTZtrument) is an accelerometer, for more expressive performance.
- "Bling it up" with internal LEDs.

UNTZtrument hacking is not for the meek; it requires patience, tools and a willingness to improvise. Tolerances inside the case are very tight. If you've done "circuit bending" before (modifying electronic instruments), you can probably handle this.

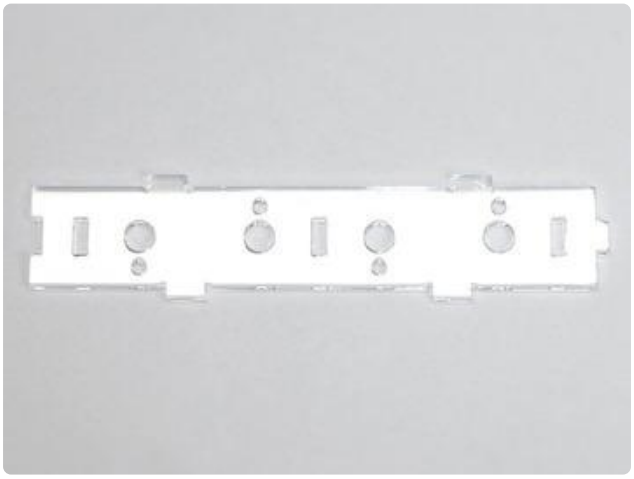
Hardware Considerations

SDA and SCL are connected to digital pins 2 and 3 on the Arduino Leonardo. Those pins are not available as inputs or outputs. Everything else is fair game!

A USB port provides current up to 500 milliamps. UNTZtrument uses about 150 mA, allowing up to 350 mA for your own additions (but aim a little lower to play it safe). That's enough for a few NeoPixels or other LEDs, or a small piezo buzzer or vibration motor for feedback. HELLA UNTZtrument uses close to 300 mA, so there's about 200 mA available for adding your own bling.

Small slots between each of the "cells" in the UNTZtrument case can be used for routing wires. Unruly wiring can be kept at bay with some clear tape or a dot of hot glue.

There are three available GND pins on the Arduino board, one of which is used by the Trellis. Because we're using USB for power, both 5V and VIN provide 5 Volts (VIN is normally the higher unregulated voltage from the DC jack, but that's not being used here); UNTZtrument uses one of these 5V pins, leaving the other available. If you need to split power in more directions, 3-way () and 5-way () block connectors are available, as are smaller 2-way cold splices (http://adafru.it/1494).

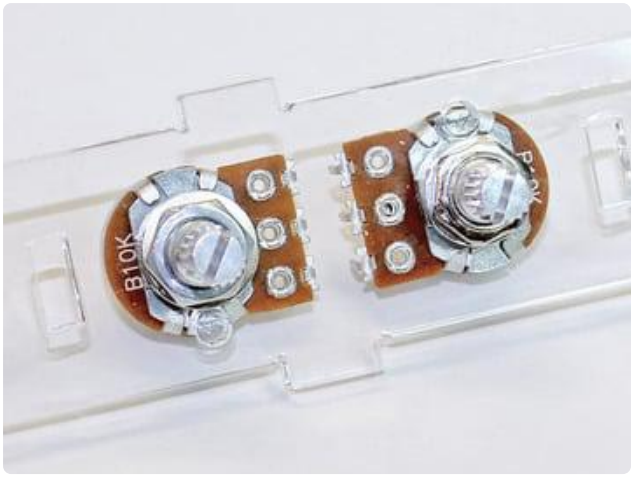


Acrylic requires special techniques for drilling; normal drill bits will crack it! This optional plastic part (separately available from Adafruit) (http://adafru.it/1945) has cutouts for adding encoders or potentiometers.

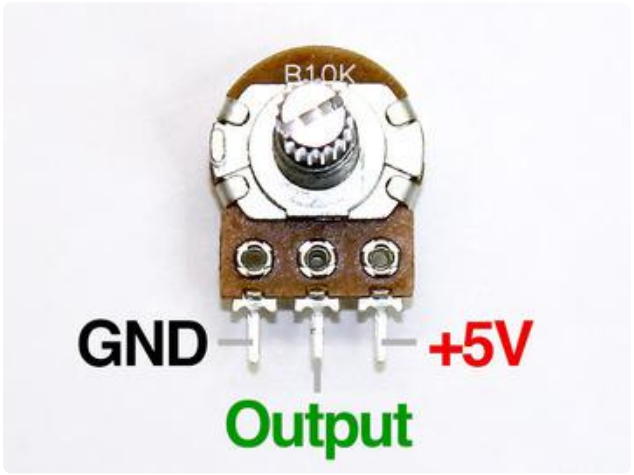
Three sides of UNTZtrument are identical, so you can add controls to any (or all) sides. On the HELLA UNTZtrument, these fit only on the left and right sides, or there's a longer front part specifically for the HELLA ().

If you have access to a laser cutter, the UNTZtrument case design can be downloaded from the last page of this guide, so you can make any special cutouts you like!





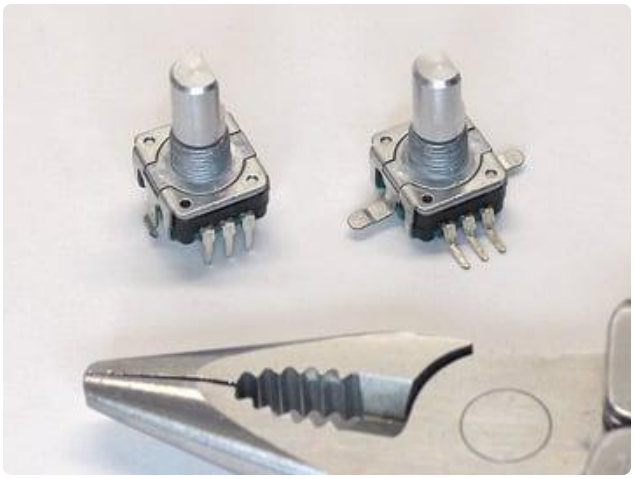
If adding more than one potentiometer, the legs must be bent back to fit inside the case. If two potentiometers are installed in the same quadrant, they install back-to-back.



Potentiometers don't care which outer leg is ground or +V, but the choice does affect which direction has increasing output.

If you want increasing values when turning clockwise, and if using the 10K pots in the Adafruit shop (), wire them as shown here.

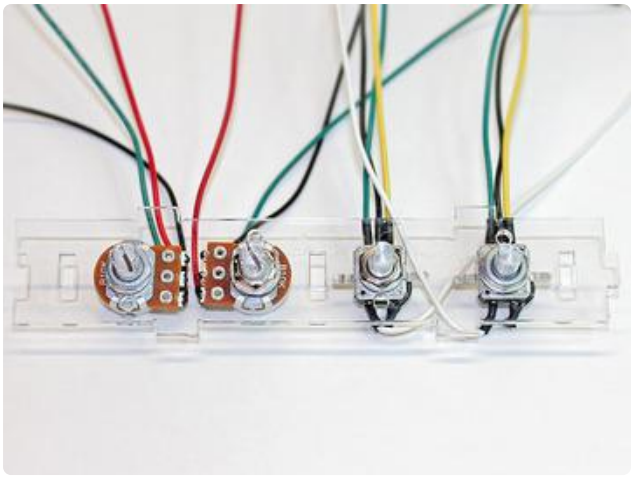
Other pots may behave differently. If the output is reversed from what you wanted, just swap the outer connections, or compensate in code using the map() function.

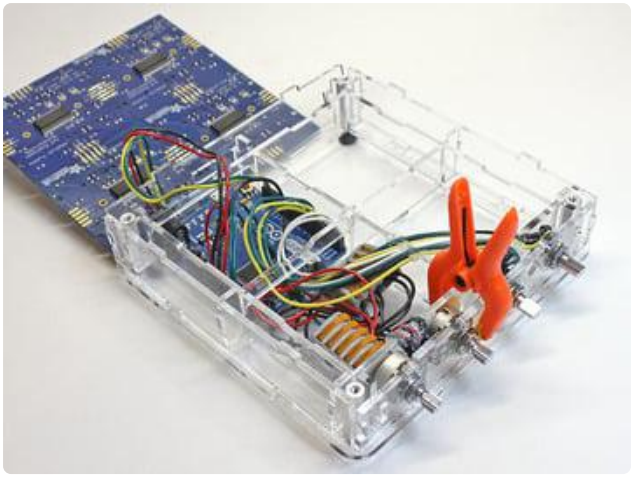


If using rotary encoders, we have the opposite problem when fitting in the case: rather than bending the legs back, we need to splay them outward.



You need to be REALLY CAREFUL when doing this. DO NOT straighten the existing bends — the legs will crack off! Instead, add extra bends a little further down.





Whatever controls you add, you'll then need to get really creative with routing wires. Narrow gauge (24 or 26 AWG) fits in tight spaces a little better.

Potentiometers especially are a tight squeeze.

In the second photo you can see all of the hackishness that was required for one project. A clamp provides a temporary hold while working. Hot glue keeps loose wires at bay. Cold splices provide multiple +5V and GND connections.

Software Considerations

If adding I2C devices (such as certain accelerometers or other sensors), be aware that our example code does a crass thing to activate 400 KHz I2C (this makes the LED updates a bit more responsive), and it might not be compatible with other devices you're interfacing. If this creates a problem, one option is simply to disable this line of code, reverting to the standard 100 KHz communication.

On 8-bit AVR boards, the offending line, commented out, would be:

```
// TWBR = 12;
```

On 32-bit SAMD-based boards (e.g. M0), the line in question is:

```
// Wire.setClock(400000L);
```

Another option (on AVR) is to save the original value of the TWBR register in a global variable following initialization, then set it to the 100 or 400 KHz rates as needed.

Global variable, before setup():

```
uint8_t i2c_save;
```

In setup() function, after trellis.begin():

```
i2c_save = TWBR; // Original 100 KHz value
```

In loop() or other functions:

```
TWBR = 12; // 400 KHz
untztrument.writeDisplay();
…
TWBR = i2c_save; // 100 KHz
// Read/write other I2C devices here
```

These are architecture-specific hacks! If you're thinking of adapting the UNTZtrument code to any other microcontroller, don't do this, it's non-portable and may be more trouble than it's worth.

Using the UNTZtrument Arduino Library

The Adafruit_UNTZtrument library () provides a few utilities to help with creating UNTZtrument-specific sketches. We foresee every application being quite distinctive, so the library is pretty minimal and doesn't impose a lot of policy.

To use the library, you need to include three header files:

```
#include &lt;Wire.h&gt;
#include &lt;Adafruit_Trellis.h&gt;
#include &lt;Adafruit_UNTZtrument.h&gt;
```

The first enables I2C communication. Second is the core Trellis library for reading button presses and changing the LEDs. The third has our UNTZtrument-specific utilities.

The Adafruit_UNTZtrument library contains two object classes. The first is called Adaf ruit_UNTZtrument and is simply a slight extension of the Adafruit_TrellisSet class from

the Adafruit_Trellis library...in fact, declaring and initializing the Adafruit_UNTZtrument object is identical to a TrellisSet:

```
Adafruit_Trellis     T[4];
Adafruit_UNTZtrument untztrument(&amp;T[0], &amp;T[1], &amp;T[2], &amp;T[3]);
const uint8_t        addr[] = { 0x70, 0x71, 0x72, 0x73 };
void setup() {
  untztrument.begin(addr[0], addr[1], addr[2], addr[3]);
}
```

Four Adafruit_Trellis objects are declared (eight for a HELLA UNTZtrument) and passed to the UNTZtrument constructor. Their I2C addresses are passed to the begin() function in setup(). All the same Adafruit_Trellis or Adafruit_TrellisSet functions are then available to the Adafruit_UNTZtrument object.

The Adafruit_UNTZtrument object provides two new functions for converting between button/LED index numbers (as normally used by the Trellis library) and (X,Y) coordinates (more useful for programs like the step sequencer).

i2xy() converts a Trellis button or LED index (0–63 for an 8x8 UNTZtrument, or 0–127 for a HELLA UNTZtrument) to separate X (column) and Y (row) values. You must provide the addresses of two unsigned char (or uint8_t) variables to store the results:

```
uint8_t x, y;
untztrument.i2xy(i, &amp;x, &amp;y);
```

xy2i() is the complementary function, converting an X (column) and Y (row) pair to a button/LED index:

```
uint8_t i = untztrument.xy2i(x, y);
```

Adafruit_UNTZtrument also provides the enc object for interfacing quadrature encoders. Unlike analog potentiometers (which have a set range of 0–1023 using analogRead()), encoders are more flexible (and they can spin around and around, no end stops).

Each encoder requires two pins (sometimes called the “A” and “B” channels). The third pin connects to GND. The shaft “click” function of some encoders is not explicitly handled by the library...but it works just like any normally-open contact switch and is pretty straightforward to code for.

Do not connect encoders to Leonardo pins 2 or 3 — those link up to the SDA and SCL pins required for I2C communication. Anything else is fair game though (even pins 0 and 1, which Arduino Uno coders often have to avoid).

To use an encoder, just pass the index of the A and B pins to the constructor:

```
// You can declare encoders onesy-twosey:
enc e(4, 5);

// Or create a whole array of them:
enc e[] = {
  enc( 4, 5),
  enc( 6, 7),
  enc( 8, 9),
  enc(12, A2) };
```

The encoder pins do not need to be sequential, nor are they limited to pin-change-interrupt pins. Any two pins will do.

An optional third parameter to the enc() constructor enables or disables the Arduino's internal pull-up resistor on that pin. Some encoders (such as the ones previously shown) use the pull-ups...this is the default case and the third parameter is not needed. Others ("active" encoders, requiring a 5V connection) drive their output high or low on their own; the pull-up is not needed. Pass a third parameter of false when using these. This is rare.

In your setup() function, you must initialize the encoders as follows:

```
void setup() {
  // Initialize any/all previously-declared encoders:
  enc::begin();
}
```

The encoders need to be frequently polled in order to update their values (this is the downside to not using interrupt pins). There are a couple of ways to do this. Simplest is just to put this line in your loop() function:

```
void loop() {
  enc::poll();
}
```

You only need one of these; it's not necessary to poll every encoder individually, it handles the lot.

This requires the loop() function to iterate quickly. But if your program occasionally has to do something more time-consuming, the encoders may "lose counts" (the value won't change as the knob is turned). To avoid this, you can use a timer interrupt to call enc::poll() at regular intervals (e.g. once per millisecond). This requires some homework...the library does not provide this — timers are a limited resource and every application or timer library sets its own rule for these — so you'll have to wrap a

bit of your own code around it.

To read the value from an encoder, use the getValue() function:

```
// Reading from one encoder named "e":
int16_t bpm = e.getValue();

// Reading one encoder in an enc array:
int16_t bpm = e[0].getValue();
```

The value is a signed 16-bit integer value. By default, this allows a range from -32768 to +32767. The range can be limited using the setBounds() function:

```
// Limit encoder "e" to 0-100 (inclusive):
e.setBounds(0, 100);
```

An optional third parameter sets whether the value stops (further turns won't further change the value) or "wraps around" (turning past the upper limit returns the value to the lower limit, and vice-versa). Pass true to enable wrap-around, otherwise pass false (or just leave off the third parameter — false is the default behavior).

The value of an encoder can be set or changed with the setValue() function:

```
e.setValue(42);
```

When setting bounds, if the current value of the encoder falls outside the requested range it will be clipped or wrapped as appropriate. Therefore, if you need to change both the range and value of an encoder, it's recommended to set the range first:

```
e.setBounds(0, 100);
e.setValue(42);
```

Some encoders have detents — little clicky stops you can feel as the knob is turned. With the Adafruit encoders, there are four "counts" per detent...that is, it tends to stop on the values 0, 4, 8, 12, etc. But maybe you're using other encoders that spin freely. The library doesn't dictate the use of encoders with detents, so you may want to compensate in code by setting a 4X larger range and dividing the value read by 4 (or whatever detent size your specific encoder uses).

For example, let's suppose we wanted an encoder for setting a tempo between 60 and 480 beats per minute (240 by default). We're using an Adafruit encoder or another type with the 4-count detents, so the relevant parts of our code might look like this:

```
enc e(4, 5); // Encoder on pins 4 &amp;amp; 5
```

```
// in setup():
e.setBounds(60 * 4, 480 * 4 + 3);
e.setValue(240 * 4);

// in loop():
bpm = e.getValue() / 4;
```

This then provides the expected one step per detent.

I instinctively always want to say détente, but no, that's a different word for a different thing. Detent. De-tent, like having one's tent removed. Even Wikipedia notes this confusion. () :)

Downloads

You can laser cut your own enclosure — or make modifications with the following vector files — from 1/8" (3mm) acrylic in any colors you wish!

This ZIP file contains three .SVG files for the 8x8 UNTZtrument:

Untztrument-8x8.zip

"untztrument-8x8-clear.svg" and "untztrument-8x8-black.svg" are the main two files we use when producing an 8x8 UNTZtrument in clear and black acrylic, respectively... of course you can use whatever colors you choose (fluorescent acrylic is pretty rad). A third file, "untztrument-8x8-opt-cutouts," is an optional piece that can be used for adding pots or encoders to the bottom edge, typically using the same material chosen for the "clear" file.

This ZIP file contains three .SVG files for the 16x8 HELLA UNTZtrument:

Untztrument-16x8.zip

These files are what we use when producing a 16x8 HELLA UNTZtrument...

- "untztrument-16x8-black.svg" for the black top layer (or use whatever color you like).
- "untztrument-16x8-clear-1.svg" is the first of two "clear" files...this one has the two large flat pieces comprising the upper and lower sandwich layers.
- "untztrument-16x8-clear-2.svg" is the second of two "clear" files...this is all the side pieces and internal cross supports, including an optional piece with pot/encoder cutouts (not a separate file like the 8x8 version).

You can find CAD files for Trellis elastomers and PCBs on github ()