November 15, 2020

Rotary Encoder Sketches for Arduino

There are two sketches presented here for Arduinos using the ATMEGA328P microcontroller, which include the Uno, the Nano and the Pro Mini.

Standard Method

The first sketch demonstrates the standard lookup table method of servicing a mechanical rotary encoder using pin-change interrupts. The ISR increments or decrements the transition count based on which of the 16 possible current/previous pin state changes has occurred. Then at each detent, if the correct net number of transitions has occurred (+4 or -4), a tick and its direction are reported to the main loop for further processing.

The sketch flashes an LED on A1 for a CW tick and an LED on A2 for a CCW tick. In addition, the ISR keeps track of the number of times it has been called since the last reported tick, and that total is printed to the Serial Monitor. So the sketch offers a way to assess how bouncy or noisy a particular rotary encoder is. For a perfect encoder with an equal number of pulses and detents per revolution, there would be only four interrupts between ticks, one for each transition of each switch.

Alternate Method

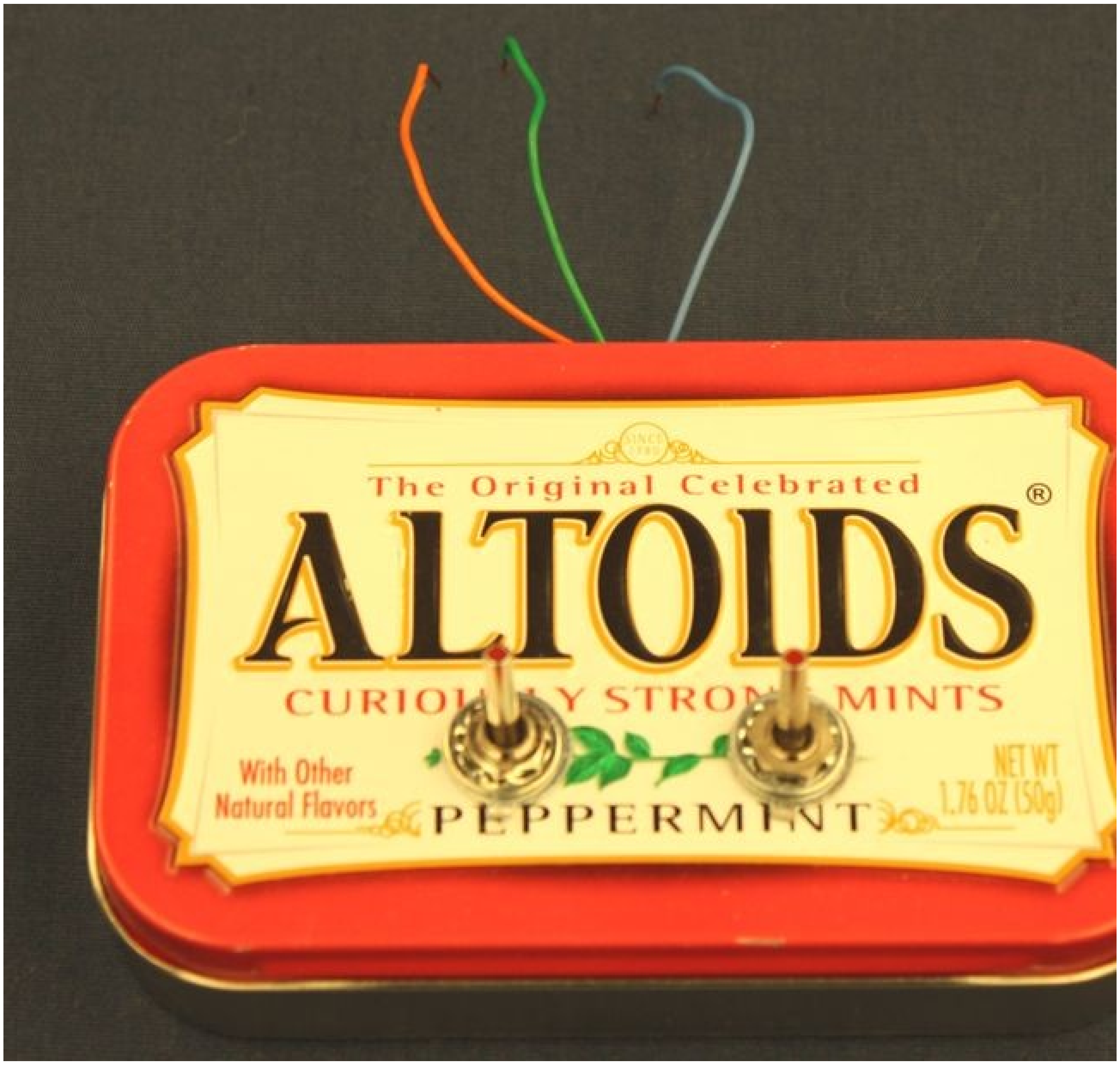
The second sketch is an attempt to minimize servicing overhead by reducing the number of interrupts. It does that by enabling only one interrupt at a time. When an interrupt is triggered, the ISR immediately disables that interrupt and enables the interrupt on the other pin, which transitioned some time ago, and which should have finished bouncing by now. So any bouncing on the current pin produces no interrupts at all because its interrupt is no longer enabled. This method should produce the ideal four interrupts per tick even if the switches bounce - so long as each switch finishes bouncing and settles down before the other switch changes state.

This method produces a complication when the direction of rotation changes. That's because we have the interrupt enabled on the wrong pin. What we expected to be the leading switch has now become the trailing switch, so we have completely missed one transition. The solution is to expand the lookup table index to five bits, with the extra bit representing the identity of the pin currently interrupting. The lookup table then becomes 32 entries long, and now includes some +2 and -2 entries in addition to the +1, -1 and 0 entries in the standard version. While there is only one interrupt flag bit for the entire port under pin change interrupts, we know which pin interrupted because we know which pin is enabled, and only one is enabled at a time.

One problem with using pin interrups to service rotary encoders is the delay which occurs between the time the interrupt is triggered and the time the port is read. Bouncing may cause an eroneous value to be read from the port. But in the Alternate method, we keep track of where the pin is, so we know which edge triggered the interrupt, and we can use that value instead of the port read value for that pin. Such an adjustment is not possible under the Standard method because we don't know which pin triggered the interrupt.

Performance

Under limited circumstances, the Alternate method may be a better choice than the Standard method, producing the same ticks on fewer interrupts. For example, this "Minty" encoder is an analog of a rotary encoder that allows manually stepping through each transition of each switch:



A CW sequence would start with both high, then go to A low, then B low, then A high, and finally B high. That gives a correct +1 tick under either method. Going through this 4-transition sequence 10 times gives this interrupt count for the Alternate method:

4 4 4 4 4 4 4 4 4 4

But the Standard method produces this:

```
52 53 130 73 234 46 15 42 75 30
```

This difference happens only with an "ideal" encoder, which is one that may bounce like crazy as these Minty switches do, but which bounces only at the transitions, and settles down pretty quickly. But most real encoders are not like that - they may continue to bounce as the closed switch contacts are dragged along during rotation. One pretty crappy real encoder produced these interrupt counts:

Alternate:

```
6 4 4 4 4 8 4 4 4 18 4 3 4 4 4 4 4 4 6 6
4 4 3 4 4 4 6 4 4 4 8 4 8 6 3 4 8 6 4 6
4 4 4 6 6 4 5 4 4 6 4 4 4 4 4 4 6 12 11 6
6 4 4 4 4 6 4 6 8 4 3 6 4 6 4 4 4 4 4 4
4 4 15 4 4 4 4 4 4 4 4 8 4 7 4 4 4 4 4 4
```

Standard:

```
12 5 7 9 28 41 27 29 45 19 13 7 13 67 24 27 14 24 11 13
11 17 22 32 13 18 21 14 33 7 9 47 16 38 28 18 73 47 21 11
25 9 12 17 16 7 6 15 44 82 32 29 62 10 43 10 10 34 23 77
31 8 6 17 9 18 10 21 30 26 24 23 20 23 72 10 12 23 5 15
25 14 16 24 42 50 16 95 9 23 12 39 11 16 11 34 38 67 22 36
```

but the Alternate method made more mistakes, including producing false direction ticks. The Stardard method produced no noticeable mistakes. So the Alternate method would be useful if low overhead servicing is really important, and a few errors in results aren't so important - or if the encoder is well-behaved, and finishes bouncing fairly quickly, so no errors occur..

In any case, the two sketches can be used to test an encoder or compare encoders to see how bouncy or noisy they are.