Garden Railways Magazine - October 2018

This innocuous looking track-inspection gondola is actually a working odometer that can measure the length of your track. Read here how you can make one.

Build a TRACK ODOMETER

Measure the length of your track using an Arduino

by Del Tapparo | Windsor, Colorado | PHOTOS BY THE AUTHOR

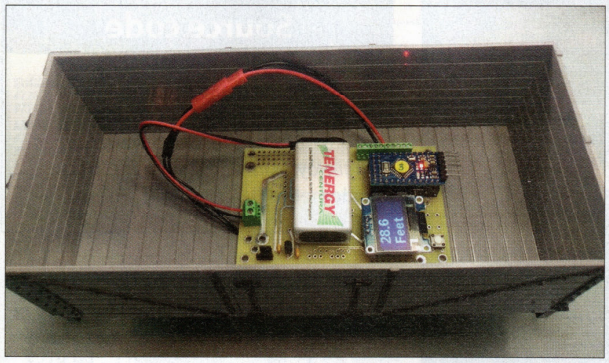
In searching for fun things to do with my recent interest in Arduino microcontrollers, I decided it would be nice to have a track-inspection car of sorts that could act as an odometer. This would measure the distance traveled over the mainline, giving a definitive answer to the common question, "How long is your mainline run?"

The concept is simple. If you count the number of revolutions a wheel of the inspection car makes over the course of the run, you could calculate the distance traveled based on the circumference of the wheel. Since every piece of rolling stock has different size wheels, though, I wanted to make the program work for any car. A calibration routine would solve that. Record the number of counts obtained over a 10'-section of track and store the counts per foot in non-volatile memory (EEPROM). When measuring the mainline, the distance traveled in feet equals the current number of counts divided by counts per foot. The results are displayed on a little OLED display.

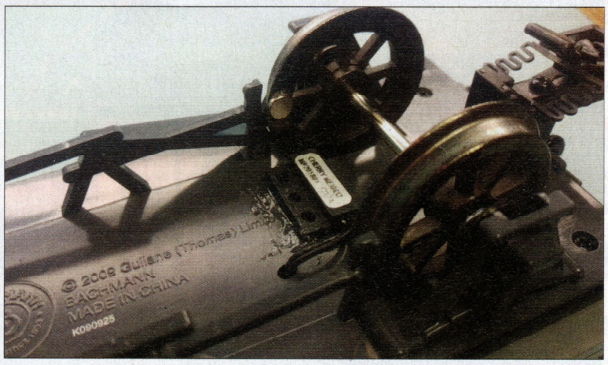
Hardware

To count wheel revolutions, we can glue a magnet to the inside of a wheel and place a reed switch adjacent to the wheel. Each revolution will cause the reed-switch contacts to briefly close. Two magnets will improve the accuracy. We will also need a simple operator interface—a momentary push-button switch. This will be used to enter the calibration procedure and to zero the measurement on the display. An Arduino has many different I/O (Input/Output) pins. The digital pins can be configured as either inputs or outputs. They detect something as being either on or off, or they can be used to actually turn something on or off. There are also analog pins that can be used for inputs or outputs.

For this project, we will need two digital inputs: one for each switch. Both inputs will use internal pull-up resistors to keep the input high until the switch closes, causing it to switch low. The digital input used for the reed switch must be an "Interrupt pin" (explained later in the software section). The reed switch also requires a small capacitor across it to



Simple electronics, based on a low-cost Arduino, are carried in the car.



A magnet, attached to a wheel, actuates an adjacent reed switch as the car rolls.

eliminate switch bounce. When the reed switch closes, it actually opens and closes many times before it settles down and stays closed. Our software will be fast enough to detect these changes. The capacitor smoothes out this noise, so the software will only see one switch closure and, thus, one count. Switch bounce won't be critical for the operator switch.

The OLED display uses I2C communication with the Arduino. It just needs power and ground connections, plus two wires called SCL (clock) and SDA (data). These are pins A4 (SDA) and A5 (SCL) on the Arduino. Using the I2C bus can be a daunting task but a huge advantage of using the Arduino programming language (similar to "C") is software libraries. We simply download a free library for I2C, and we can then write to the display using a few simple commands. The OLED display allows us to choose different font sizes and print anywhere on the screen, so it is quite versatile.

The Arduino can be powered from two different sources. While connected to your PC, it gets 5VDC power from the USB connector. This can be used as you develop your program. Eventually, however, the project needs to be self powered, without the PC. This is done using a 9V battery connected to the Arduino RAW input, which will regulate the internal power down to either 3.3V or 5V, depending on your Arduino.

Construction

There are few components, so this project can easily be done on a "perf board" or "Protoboard" using point-to-point wiring. Since I have several Arduino projects in the works, all with similar components, I had a printed circuit board (PCB) made that will, I hope, accommodate all of them (time will tell). All components are carried on the circuit board except the reed switch and the 9V battery. Circuit diagrams are shown in **figure 1**.

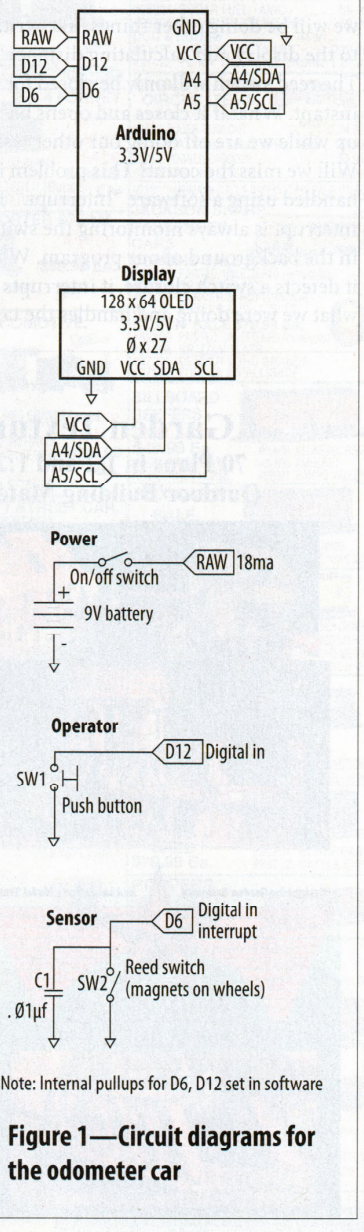
A gondola car is best suited for the project, since it provides easy accessibility for operation. There are two wires to the battery and two more wires to the reed switch. The reed-switch wires have a connector that facilitate easy removal of the circuit board.

Software

Software development for our project is done using the Arduino IDE (Integrated Development Environment). The code is written using the built-in editor. It can then be compiled and checked for errors. When error free, it is then uploaded to the Arduino via a USB port with a simple click of the mouse.

A nice feature of the IDE is a built-in serial monitor, which allows us to open a window and print information to it from within our program. For example, to see if our counter is working, we tell it to print the value of the counter. The program line is: Serial.print(counter);Serial.println("counts per foot"). This displays the value of the variable named counter, then prints the text ("counts per foot"). When the program runs, we'll see if it works or not. Insert these print statements anywhere you need to check the status of something. They can be removed or commented out when the program is finalized.

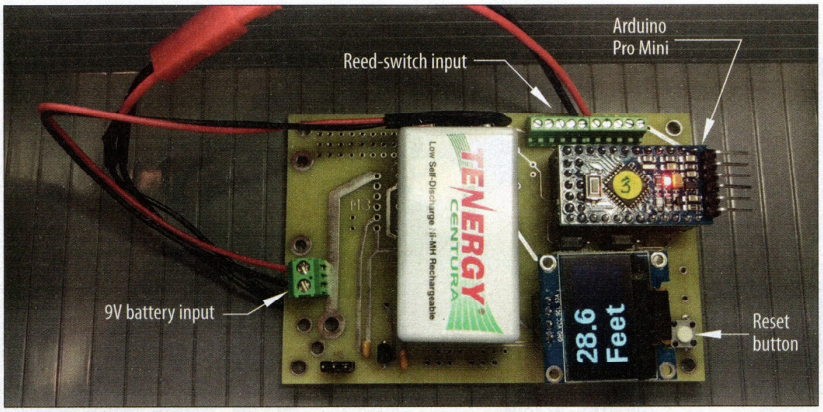
Arduino programs, like many others, run in a continuous loop. During this loop, we need to keep checking the status of our two switches to determine what to do next. Inbetween checking the switches,



Note: Internal pullups for D6, D12 set in software

Figure 1—Circuit diagrams for the odometer car

ILLUSTRATION BY MARC HOROVITZ



The author designed his own circuit board but perf board could be used as easily for the simple circuitry.

we will be doing other things, like writing to the display and calculating distance. The reed switch will only be closed for an instant. What if it closes and opens back up while we are off doing our other tasks? Will we miss the count? This problem is handled using a software "Interrupt." The interrupt is always monitoring the switch in the background of our program. When it detects a switch closure, it interrupts what we were doing and handles the task associated with the switch closure—in our case, counting. Then it returns to continue the program from where it left off, so we never miss a count.

Operation

The operator button will reset the count (distance) to zero when we press it. The count is then incremented every time we get a reed-switch closure. The distance is calculated and the display is updated as the car travels down the track.

If the button is held closed for more than three seconds, we enter the calibration mode. The operator will be prompted on the display to roll the car exactly 10 feet, and then press the button to record the calibration count. The count is stored in non-volatile memory, which means it will still be there the next time we power up.

So, put your gondola in a train, pick a starting point on your track, reset the counter, make a run around the loop (or to the end of track), and read the display for an accurate reading of the total distance traveled. Now you will know for sure how many feet of mainline your railway has.

Source code

You can download the Arduino program for this project in an .ino file from our website. Go to *www.GardenRailways.com* and search for "track odometer."