Logging Ultrasonic Sensor Data
September 30, 2020

1. Introduction

As part of another Arduino project, using an HC-SR04 ultrasonic sensor to monitor a vehicle parking in a garage, I needed information on a vehicle's velocity as a function of position pulling into a garage. The easiest way to log the distance data was to connect an Arduino Uno to an attached HC-SR04, and connect the Arduino to an ancient laptop, through a USB serial port. The following process represents one possible approach to log Arduino ultrasonic sensor data to a computer. This is not the mother of all ways to log Arduino data, but may be useful to others as it stands, or as a starting point. All the code is adapted from examples online, just modified and compiled into a more useful one stop approach of various operations.

Some potential advantages of the current process:

- Attempts to automatically find the correct serial port. (though potential issues with MACs)
- Auto-creates files to save logged data, without overwriting previous data.
- Works with Arduino Uno clones using official Arduino schematics (other cases unknown).
- Works with Windows 10, Linux (Ubuntu), and *almost* MAC (see discussion).

2. Hardware Used:

- Arduino Uno R3 (Elegoo clone, uses official Arduino schematics)
- Ultrasonic sensor HC-SR04
- Laptop (ancient, circa 2007): 1.66 GHz Intel processor, 2 GB RAM; 256GB hard drive.
- Windows 10 desktop system; 4.4GHz, 16 GB RAM,

3. Software Used:

- Arduino IDE; vers. 1.8.13
- Linux; Ubuntu distro 20.4, running on laptop.
- Windows 10
- Python 3.61 or 3.8.2

4. Overview

Logging data to a computer requires two programs: On the computer side, a python module to get the data from the Arduino's serial port; on the Arduino Uno side, a sketch to interface with the HC-SR04. The choice of python is based on ubiquitous use, personal familiarity, and especially the PySerial library, which has many online examples and discussions about interfacing, including with the Arduino. On the Arduino side, the NewPing ultrasonic sensor library is used.
Note that the python module and Arduino sketch are based on sending and reading streaming data continuously up to a certain number of pings dictated by the Arduino sketch, but there is a timeout limit. If no data appears within that timeout limit, the python module assumes logging is complete, terminates data logging, and outputs the data.

There may be issues discovering the serial port with Arduino Uno clones that do not use the Arduino schematics, especially with MAC Darwin.

The code is based on connecting an Arduino Uno. It might also work with a MEGA256 and Nano, but has not been tested. Beyond that, especially the port determination function in the python file might be finicky, especially with Arduino clones that do not follow Arduino protocols for the USB UART chip.

5. Downloads

- For each computer, download python 3.x or higher, and the current Arduino IDE. For python, make sure the PySerial library is installed or updated via pip or pip3. (Python versions < 3.x will require a bit of code tweaking.

- Download the files in this repository: *arduino_serial.py*, and the Arduino sketch: *usonic_streaming.ino*.

6. Setup

Place the downloaded sketch in your Arduino projects folder. The best way to do this is download the file, open in the Arduino IDE, and save as a project. The default project folder for Windows 10 is usually: *C:/Users/user/Documents/Arduino*, where "user" is your user name. For Ubuntu, installation creates the directory */home/user/Arduino,* where projects end up by default.

The python module does not create any folders to dump collected data files; this must be done manually. For example, with reference to the Settings Section in *arduino_serial.py,* the python module was placed in a Windows folder on my F: drive, *F:/python_files/,* or for Linux in */home/pythonfiles. B*e sure to make sure your paths are correct. For Windows, I also created a sub folder, */python_files/csvfiles. F*or Linux, I created the *csvfiles* directory under the */home/* directory, */home/csvfiles*. This is where the csv output files will be stored. You do not have to use this directory arrangement or names, but you will need to edit the python Settings section of *arduino_serial.py* to indicate your destinations.

6.1. Check and edit unsonic_streaming.ino sketch:

The Arduino sketch is straightforward. There are only a few variables or constants to check:
Check that the pin assignments for connecting the ultrasonic sensor match reality. The current setup is for a 4-wire sensor.
Check the *iteration* variable, currently set to 2000 pings. With a ping interval of 33 ms, the total data recording time is ~66 seconds. Final file output time is *iteration*ping_interval + time_stop* (*time_stop* set in python module)

6.2. Check and edit arduino_serial.py:

The csv file prefix cannot be arbitrarily changed. The code was written such that the string length is important in finding csv files from the root search directory specified in the module's Settings section. A number of lines of code also need to be edited to change the prefix. (In retrospect, this was an oversight.)

6.2.1. Review the Setting Section in the Python module:

Review the three path variables, *win_path, linux_path,* and *mac_path* for default paths to search for csv files. Only one of these may be necessary to change.

6.2.2. Review the default port names for the os:

Keep in mind that the pyton port determination function, *serial_ports()*, uses the search term 'rduino' in the manufacturer info. For boards that do not follow the Arduino protocols, the identification may not work, and the port will not be connected, unless you have manually figured out which port is being used. On that basis, the port function might work with the Uno, MEGA256, and Nano.

6.2.2.1 Windows port - An Arduino Uno clone designed to the Arduino specifications will likely supply a description or manufacturer serial port descriptor, such as "Arduino www.arduino.cc". The function, *serial_ports()* searches ports to find 'rduino', otherwise it will use the default provided in the python file Settings section. Good practice would be to set this default COM port, based on what is found in the Arduino IDE. For example, my Uno was always on COM3, but no other HID (human interface devices) devices were connected to a serial port. If the serial port being used is still not clear, record the ports showing up, disconnect the device, and recheck which devices show up. The one missing is the port to use.

6.2.2.2. Linux port - Before dealing with the default serial port to use in Linux, if you have never set up a serial port for devices check several things. First, in terminal issue the command, '*group*s' and see if there is a '*dialout*' group. If not, found, issue the cmd: *sudo adduser $USER dialout*, where $USER is the name you login with. This will add the dialout group.

The form of the serial port name may be */dev/ttyACM#*, where "#" is often "*0*", but may be different if you have other HID devices connected, or are using Arduino clone knockoffs that do not support the original Arduino Uno schematics. The system will search for a choice, but with non standard Uno clones, it is likely there will be no definitive descriptor to indicate which port is being used. It is imperative you identify the port being used, and place it as the default.

Depending on many factors, finding the right port can be a mess. Try connecting the Arduino to the computer, and issue the command, *'dmesg | grep tty'*. Hopefully, it will be clear what port is being used, and what the port name should be. Another possible aid is the command, *udevadm info --query=all --name=/dev/ttyACM0*, where ttyACM0 would be changed one by one to whatever port names, *dmesg* prints out. An additional aid to identifying the port may be the vid:pid product id code assigned to the UART (See the Improvements-Extension section for a reference.)

A lot depends on the Arduino Uno board being used. A Uno clone that follows the official Arduino schematics will likely follow the patterns suggested above. If the connection between the Arduino and computer uses a serial (e.g. 9 pin) connector on computer to usb port cable, or is not using an ACM compliant device, then the port may show up as /dev/ttyUSB#. If somehow, using a 9 pin to 9 pin cable, then it may be /dev/ttyS0.

With Linux, there might also be a serial port permission problem to overcome. Issue the command,

'groups yourusername'.

If the dialout group is not listed, try,

sudo adduser YourUserName dialout

then logout and log back in for the change to take effect.
You might also need to explicitly set permissions:

sudo chmod a+rw /dev/ttyACM0, assuming your port number is '0'.

6.3. Review the Baudrate
Next check the *baudrate* variable setting. The baud rates must be the same in both the sketch and python file. This is currently set to 9600 baud. Whether there will be any data acquisition, and send and receive collision issues with higher baud rates, was not determined.

6.4. Review time_stop
Lastly, review the *time_stop* variable is Settings. This variable determines how long the python data acquisition function waits, if it receives no data, before dumping data to a csv file. If the time is exceeded, python closes the serial port connection and exports the data to the csv file. Obviously, *time_*stop must be longer than the time expected between successive or longest ping intervals. Note that this time will also affect the wait time before data is printed to a file.

7. Order of Operation

1. As necessary, check the Settings section of *arduino_serial.py* as discussed above.
2. Plug the Arduino into the USB serial port of the computer.
3. Load or make sure the sketch, *usonic_streaming.ino* is loaded into the Arduino; it is not necessary that the Arduino IDE be kept open once the sketch is loaded, but there should be no issue if it is.
4. Unless you have the paths set globally, go to the folder containing the python file, e.g., *cd python_files or pythonfiles*, or wherever you placed the python file.
5. Run the python file: *python3 arduino_serial.py*. (Alternatively, run the python file from the terminal in IDLE or other IDE, such as Visual Studio Code.)

Starting the python file will cause the Arduino to reboot and begin sending microsecond echo data through the serial port. The incoming data will be mirrored in the terminal output. Output will consist of a single column of data, with a header 'u s e c' . (Yes, its messed up.)

8. Output

The incoming data will be mirrored in terminal. This was found under the current settings and hardware to not create any breakdown in transmission, even with a lowly 1.66 GHz processor and 2GB of RAM.
The final output of the python module is a **.csv* file with the following structure, *'streaming_ddmmyy_##.csv',*

- '*streaming_* ' is a constant prefix name. it is used in a search function to find the latest csv file.
- '*ddmmyy_* ' is the current date as two digit: month, day, year form; this date is automatically generated and added by the module to the prefix.
- '*##*' is an index of form 00 to 99. If the module finds the latest date of a csv file is the same as the new one to be generated, the index is automatically incremented, so no csv file created on the same day is overwritten. As suggested, the limit is 99 files, before the module will error on output.

8.1. To stop the logging process before it completes, use a CTRL-C in terminal.

If a Ctrl-C is issued to prematurely terminate logging, any data in the receiving array, will be printed to a typical data cvs file.

9. Notes

1. Data transfer uses the PySerial *readline()* to complement the sketch *Serial.println(...)* sending data to the USB port (and the Serial Monitor). Thus, the line feed character (10) is the signal that a microsecond echo string of bytes is complete and should be saved to the holding array of echo values. The data stream on the python side is read into an array and only saved to a *.csv* file when data transmission has stopped. If the data input to python is extraordinarily large, and there is insufficient RAM, logging will fail.
If multiple data elements need to be sent per line, the code will need to be changed to accommodate it. Right now the time constant between successive values is taken to be the same as the *ping_interval* variable used in the sketch. However, concern about timings is the reason conversion to distance is not done. Instead, the raw microsecond values can be converted to distance later (using Excel) via the formula: (*time x sound_velocity)/2*. Sound velocity is on the order of 0.03445 cm/µs.

2. With the inherent exception of python resetting the Arduino at runtime, writing commands from the python module to the Arduino is not implemented.

3. If you have Excel 2016 or higher, you can directly log Arduino output to an Excel worksheet. Search "data streamer add-in Windows". (My version of Excel is a lot older...2007.)

Improvements - Extensions

If it is not already obvious, I am not an expert on a lot of this stuff. I have learned a lot, still stumble a lot, and do a lot of reading. Most I understand, and some I never will.

There is clearly room for improvement to generalize the code. For me, the current set up solved a specific need with modest effort, and I learned new stuff along the way. I did spend a little time adding some generality, with the intent that if I needed to log data from Arduino for other projects, I would have a modest skeleton to start with. Possible, extensions might be:

- Add code to read multiple values per line, or stream data with a delimiter.

- There are weaknesses in determining the serial port; all situations cannot be anticipated. The current setup may be specific to Uno clones that follow the official Arduino schematics. Two particular issues are in the Linux and MAC port determination. It appears that in the case of a good Arduino Uno clone, the port manufacturer element holds the term "Arduino" or "arduino", or both. For instance, my Uno had *'Arduino www.arduino.cc'* as manufacturer. The current port discovery process searches the port object's manufacturer info, to determine the port to connect to. For Windows based systems, the port is a 'COM#' port. For Linux, the description varies depending on hardware configurations, often the port is /dev/ttyACM0, but certainly not always, especially when other devices connected, or non Arduino knockoffs are used. The vid:pid info is another possible way to get the Uno port and possibly the MEGA256. mine was VID:PID 2341:0043, but again, this is not at all universal. See https://forum.arduino.cc/index.php?topic=446788.0 and links there, for an interesting discussion on the potential UART serial chips. Whether the VID:PID would be any more robust is unclear. Certainly a large, ever changing, dictionary might be required to more easily identify the proper port.
- Setting a port on a MAC is confusing (for me). What I found with a cursory review was that the search string, can be: '/dev/cu.*' or /dev/tty* . The "*" type depends on the specific hardware and associated drivers sending and receiving data.
- Corral some of the code lines outside the main(), or create a class module.
- Resolve/fix a way to tell the python code that looping is done, save data, via some value from Arduino.
- Fix the issue with allowing only *'stream_* ' as the cvs file prefix.
- Possibly auto-detect baud rate: https://stackoverflow.com/questions/47965838/pyserial-loop-through-baud-rates; https://github.com/devttys0/baudrate