ID: 520

Data-acquisition with MDSplus and custom devices.

Fernando Santoro
Thomas Fredian, Stephen Lane-Walsh and Joshua Stillerman
PSFC at MIT
fsantoro@psfc.mit.edu

ABSTRACT

MDSplus is a software tool dedicated to data acquisition, storage and analysis used for complex scientific experiments. We will show how to set up a very simple experiment, manage data retrieval, storage and consumption using MDSplus and Python. JupyterLab is used as the interactive development environment.

BACKGROUND

As a data acquisition software, designed as a new paradigm for data analysis, MDSplus has been extensible used as one of the main software tools to acquire and organize the vast amount of data coming from magnetic fusion energy programs. The intent of this poster is to show that it can easily be used for any kind of data-acquisition systems: from the most complex to the simplest.

METHODS / IMPLEMENTATION

METHOLOGY

A custom device is used as the data provider. This device is an open-source electronics platform hardware that consists of a microcomputer (Fig.1) and sensors (Fig. 2).

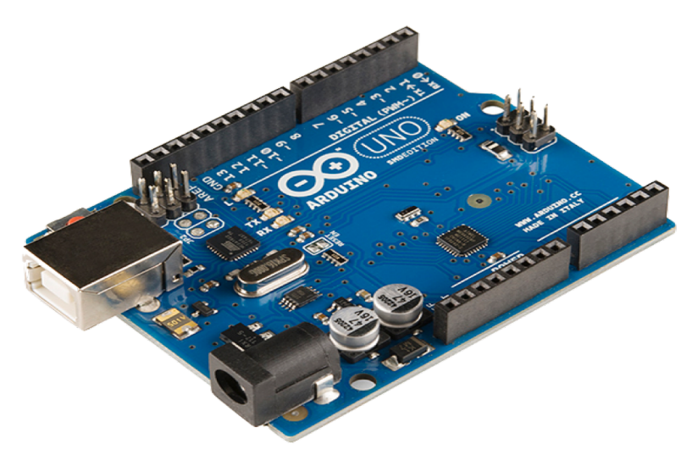


Fig 1. Arduino UNO microcontroller.

Fig 2. Garmin Lidar-Lite r3HP (left) and Temperature/humidity sensors.

Fig. 3 shows the JupyterLab IDE. It allows for code development, notebooks and data visualization.

IMPLEMENTATION

MDSplus Device represents the hardware in the data acquisition system. Information associated with a given device will be stored in a set of nodes of the pulse file. Figure 5 shows what the tree structure looks like for our particular device.

The code that represent the device (see Figure 3), written in Python, has the following structure:

- The **Device**: an Python subclass that defines the Arduino device.
- The **Tree** structure and nodes: defined in MDSplus parts[].
- The **Port** communication with the device and sensors: PySerial library.
- The **INIT** and **STOP** methods: to start and stop the data acquisition.
- The **TREND** method: for acquisition using launchd/systemd.
- The **STREAM** method: for acquisition using the data streaming.

Additionally, an Arduino **sketch** needs to be uploaded into the microcontroller.

HARDWARE

The device (Fig. 6) itself is composed by the following:

- An Arduino UNO: a microcontroller board (Figure 1)
- A Garmin LIDAR-Lite v3HP sensor. (Figure 2)
- A DHT11 temperature and humidity sensor.

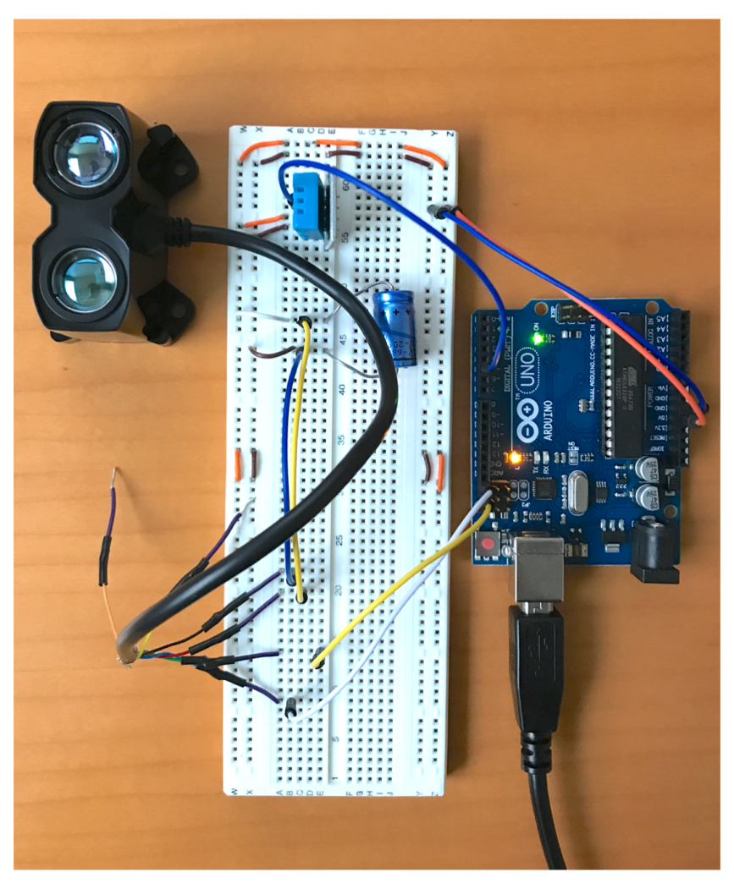
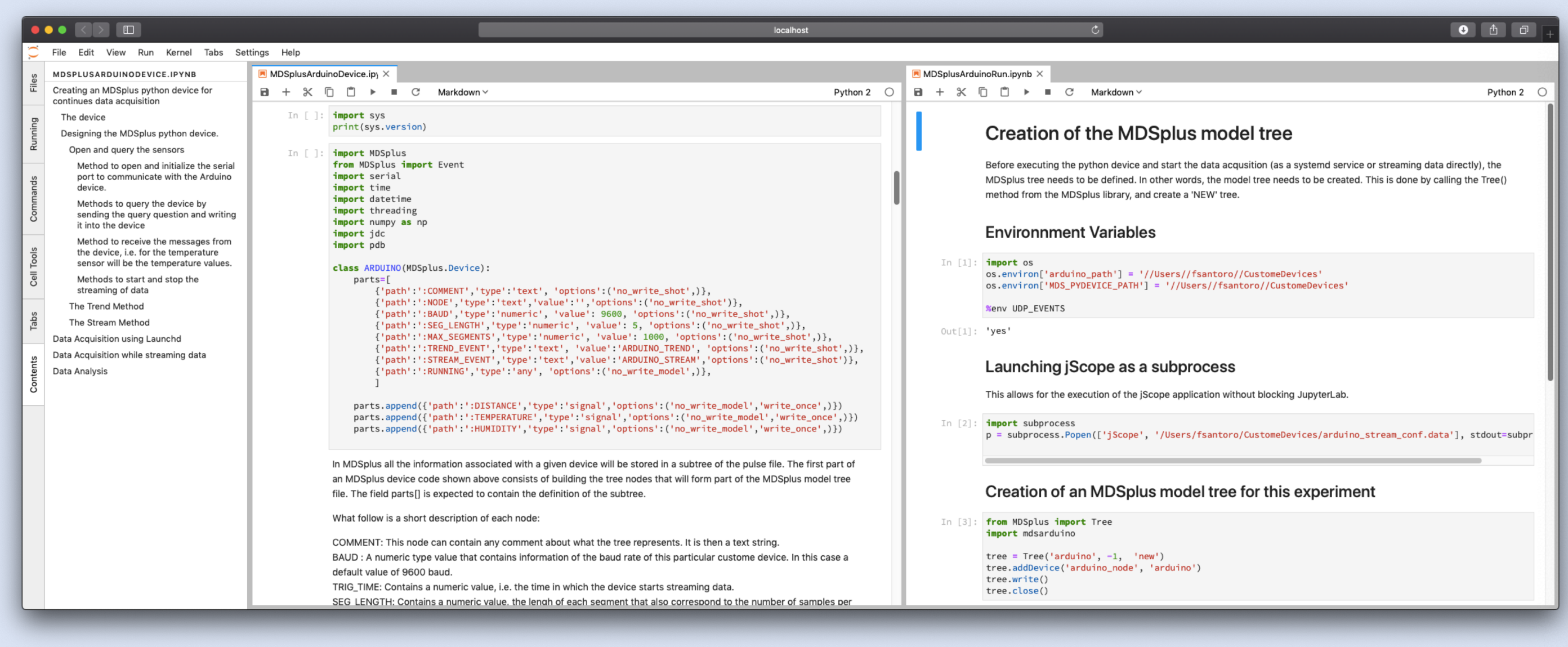


Fig. 6. The Device.

REFERENCES and ACKNOWLEDGEMENTS

MDSplus online documentation (http://www.mdsplus.org); Arduino UNO (https://www.arduino.cc); Garmin (http://www.garmin.com)



MDSPLUSARDUINODEVICE.IPYNB
Creating an MDSplus python device for continues data acquisition
The device
Designing the MDSplus python device.
Open and query the sensors
Method to open and initialize the serial port to communicate with the Arduino device.
Methods to query the device by sending the query question and writing it into the device
Method to receive the messages from the device, i.e. for the temperature sensor will be the temperature values.
Methods to start and stop the streaming of data
The Trend Method
The Stream Method
Data Acquisition using Launchd
Data Acquisition while streaming data
Data Analysis

```
In [ ]: import sys
        print(sys.version)

In [ ]: import MDSplus
        from MDSplus import Event
        import serial
        import time
        import datetime
        import threading
        import numpy as np
        import jdc
        import pdb

        class ARDUINO(MDSplus.Device):
            parts=[
                {'path':':COMMENT','type':'text', 'options':('no_write_shot',)},
                {'path':':NODE','type':'text','value':'','options':('no_write_shot')},
                {'path':':BAUD','type':'numeric', 'value': 9600, 'options':('no_write_shot',)},
                {'path':':SEG_LENGTH','type':'numeric', 'value': 5, 'options':('no_write_shot',)},
                {'path':':MAX_SEGMENTS','type':'numeric', 'value': 1000, 'options':('no_write_shot',)},
                {'path':':TREND_EVENT','type':'text', 'value':'ARDUINO_TREND', 'options':('no_write_shot')},
                {'path':':STREAM_EVENT','type':'text','value':'ARDUINO_STREAM','options':('no_write_shot')},
                {'path':':RUNNING','type':'any', 'options':('no_write_model',)},
                ]

            parts.append({'path':':DISTANCE','type':'signal','options':('no_write_model','write_once',)})
            parts.append({'path':':TEMPERATURE','type':'signal','options':('no_write_model','write_once',)})
            parts.append({'path':':HUMIDITY','type':'signal','options':('no_write_model','write_once',)})
```

In MDSplus all the information associated with a given device will be stored in a subtree of the pulse file. The first part of an MDSplus device code shown above consists of building the tree nodes that will form part of the MDSplus model tree file. The field parts[] is expected to contain the definition of the subtree.

What follow is a short description of each node:

COMMENT: This node can contain any comment about what the tree represents. It is then a text string.
BAUD : A numeric type value that contains information of the baud rate of this particular custome device. In this case a default value of 9600 baud.
TRIG_TIME: Contains a numeric value, i.e. the time in which the device starts streaming data.
SEG_LENGTH: Contains a numeric value, the lengh of each segment that also correspond to the number of samples per

Creation of the MDSplus model tree

Before executing the python device and start the data acqusition (as a systemd service or streaming data directly), the MDSplus tree needs to be defined. In other words, the model tree needs to be created. This is done by calling the Tree() method from the MDSplus library, and create a 'NEW' tree.

Environnment Variables

```
In [1]: import os
        os.environ['arduino_path'] = '//Users//fsantoro//CustomeDevices'
        os.environ['MDS_PYDEVICE_PATH'] = '//Users//fsantoro//CustomeDevices'

        %env UDP_EVENTS

Out[1]: 'yes'
```

Launching jScope as a subprocess

This allows for the execution of the jScope application without blocking JupyterLab.

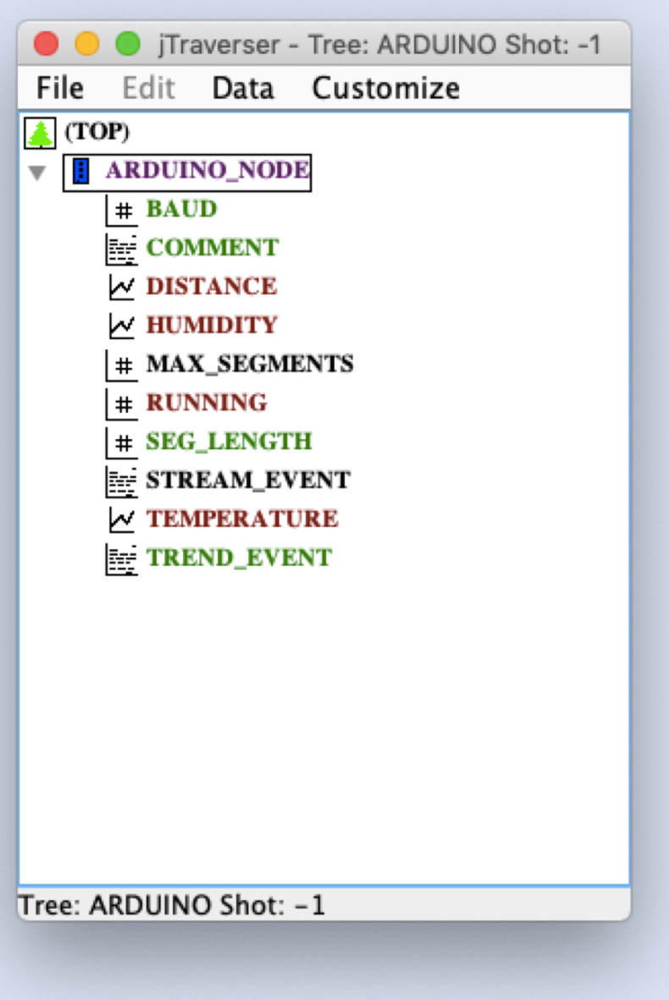
```
In [2]: import subprocess
        p = subprocess.Popen(['jScope', '/Users/fsantoro/CustomeDevices/arduino_stream_conf.data'], stdout=subpr
```

Creation of an MDSplus model tree for this experiment

```
In [3]: from MDSplus import Tree
        import mdsarduino

        tree = Tree('arduino', -1,  'new')
        tree.addDevice('arduino_node', 'arduino')
        tree.write()
        tree.close()
```

Fig. 3. JupyterLab IDE showing code and model tree creation.



```
(TOP)
▼ ARDUINO_NODE
    BAUD
    COMMENT
    DISTANCE
    HUMIDITY
    MAX_SEGMENTS
    RUNNING
    SEG_LENGTH
    STREAM_EVENT
    TEMPERATURE
    TREND_EVENT
Tree: ARDUINO Shot: -1
```

Fig. 5 The data tree structure

EXCECUTION and OUTCOME of the EXPERIMENT

STEPS TO RUN THE DATA ACQUISITION SOFTWARE

- Set up the system environment: *os.environ['[treeName]_path']*
- Launch the data viewer: MDSplus jScope
- Create an MDSplus Model Tree: *addDevice()*
- Configure the data acquisition parameters.
- Set the current shot and create the pulse: *createPulse()*
- Execute the INIT method to start the data acquisition: *init()*
- Execute the STOP method to stop the data acquisition: *stop()*

VISUALIZING THE RESULTS

The results of the experiment can be visualized in three ways:

- Dynamically: Real time visualization using jScope (Fig. 4)
- Data Analysis: using Python libraries, shown inside JupyterLab.
- Tree content: the data structure visualization using jTraverser (Fig. 5)

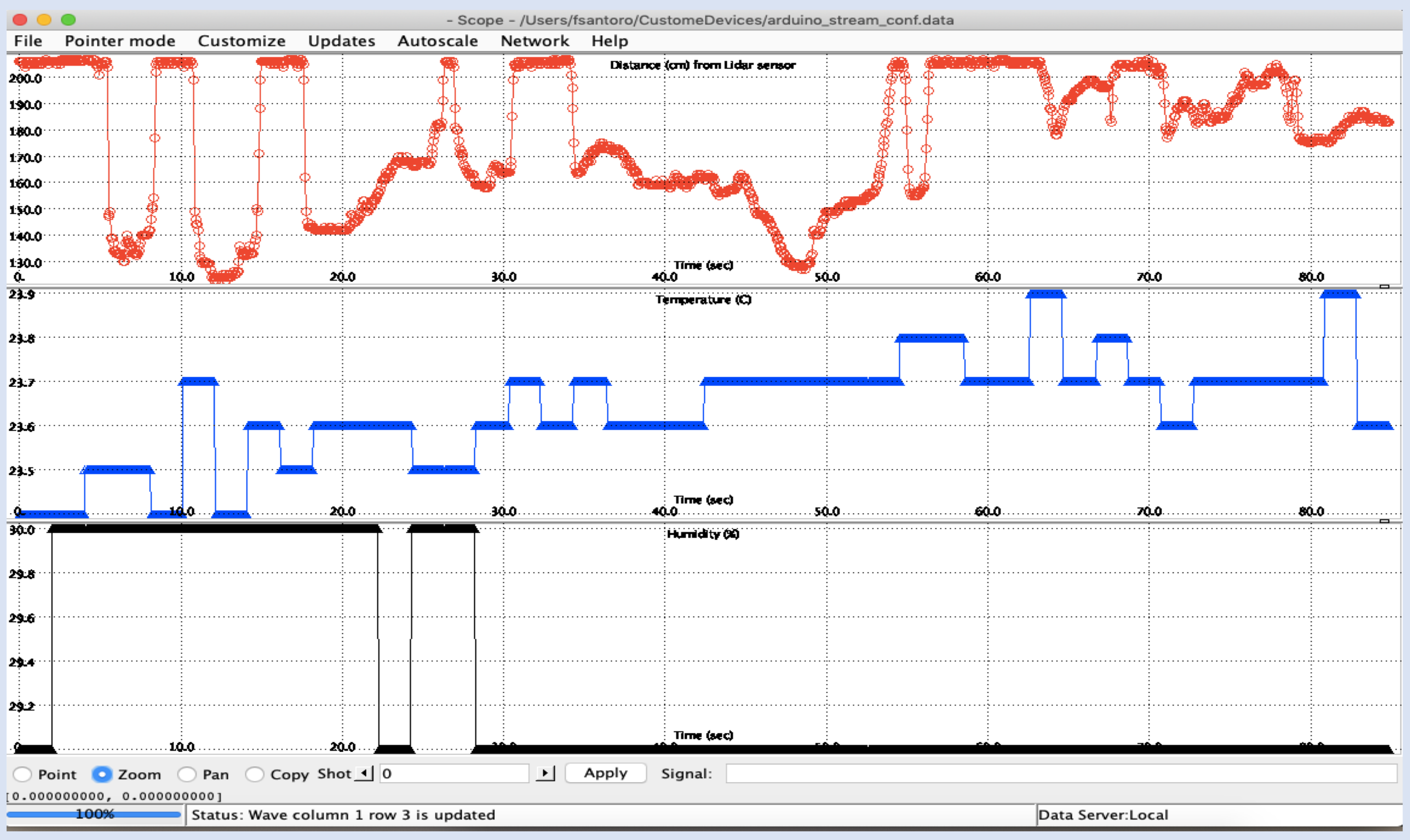


Fig. 4. MDSplus jScope showing data acquisition in real-time.

RESULTS and CONCLUSIONS

- Real-time results can be seen in Fig 4. Data analysis can be done within JupyterLab.
- Limitation: jScope needed to be run as a subprocess from within JupyterLab to be able to see real-time data streaming.
- Limitation: it is difficult to communicate with a serial port device from within JupyterLab.
- The structure of the code that represent the device can be used as a template for a variety of devices, sensors and systems in general.

This work was funded under DOE cooperative agreement DE-SC0012470.