Dash DAQ

PACKAGE GUIDE

Contents

Welcome to Dash DAQ!

Thank you for purchasing Dash DAQ! This package contains the `dash-daq` toolkit, instructions to help you get started with Dash DAQ, and the source code for several example apps. Included in this package are apps that can be used with:

- Multimeters
- Spectrometers
- Stepper motors
- LEDs
- Pressure gauges

Along with other lab equipment, tools, and personal projects.

Support

If you run into any issues with the apps in this package, please e-mail us at dash.daq@plot.ly.

About this package

This document contains brief overviews of all of the apps included in the package. Each app will contain an `app.py` file, which is what will be used to run it on your computer. Each folder also contains a demo app, which you can use to test out any UI components you want to add without having to configure it to work with the hardware.

Directory structure

```
dash-daq-monorepo/
| - Dash_DAQ_Package_Guide.pdf
| - dash-apps/
    | - dash_daq-0.0.3.tar.gz
    | - dash-daq-hp-multimeter/
        | - app.py
        | ... [other necessary Python files]
    | - dash-daq-iv-tracer/
    | - dash-daq-led-control/
    | - dash-daq-omega-pip/
    | - dash-daq-pressure-gauge-kjl/
    | - dash-daq-pressure-gauge-pfeiffer/
    | - dash-daq-robotic-arm-edge/
    | - dash-daq-sparki/
    | - dash-daq-stepper-motor/
    | - dash-ocean-optics/
    | - dash-tektronix-350/
```

Where to find documentation

- Dash DAQ components: http://dash-daq.netlify.com
- Dash User Guide and Documentation: https://dash.plot.ly/plugins
- Dash source code: https://github.com/plotly/dash

About Dash DAQ

Dash DAQ is an extension of Plotly's Dash that empowers you to create simple, intuitive web interfaces for interacting with lab equipment. Visit our website at https://www.dashdaq.io for more information.

About Plotly

Plotly creates leading open source tools for composing, editing, and sharing interactive data visualizations via the Web. Visit us at https://plot.ly to find out more about the products and services that we offer.

SETUP

Installing dash-daq

You must have pip installed for dash-daq, and for the setup for all of the example apps, to work. Please also note that these installation instructions are given for UNIX machines (OS X and Linux). If you have a Windows machine, then make sure to either install pip separately or install Python 3, as it ships with pip.
After you've installed pip, you need to install the Dash DAQ package. It is contained in the tarball dash_daq-0.0.3.tar.gz. To install, simply run:

```
$ pip install dash-apps/dash_daq-0.0.3.tar.gz
```

You can then test out whether or not it worked by typing in:

```
$ pip freeze | grep dash
```

And the output should contain the line:

```
dash-daq==0.0.3
```

Any other necessary Dash packages (e.g., dash-core-components) or external packages will be installed on a per-app basis through the requirements.txt files.

Using virtualenv

Unless otherwise specified, you should run each app in a virtual environment created with virtualenv. If you don't have this installed, then run

```
$ pip install virtualenv
```

Then make sure you're in the same directory that contains app.py, and create a virtual environment with the version of Python that is specified in the "Requirements" section of the app's documentation. First, find the path to the desired Python executable (say, for example, we want to run the app in Python 3.6):

```
$ which python3.6
```

Then create the virtual environment and activate it:

```
$ virtualenv -p [path to Python 3.6] .
$ source bin/activate
```

Now, you will be able to install all of the app-specific requirements and run the app.

APPS

dash-daq-hp-multimeter

Introduction

An application that allows the user to control a multimeter. Test out the app yourself at https://dash-gallery.plotly.host/dash-daq-hp-multimeter/.

Multimeters

Multimeters are popular electric data acquisition devices that are used in laboratories around the world. The key features of a multimeter are its ability to measure voltage and current. Voltage is measured by creating a parallel line from point A to B, which reads out the voltage difference between these two points. Current is measured by passing the current of a circuit through itself, in series.

Hardware

This app was developed specifically using the HP Agilent 34401A

Requirements

The virtual environment for this app should be using any version of Python 3. After setting up the environment with `virtualenv`, you can install the required packages:

```
$ pip install -r requirements.txt
```

How to use the app

There are two versions of this application. A mock version for the user to play with without any instruments connected, and a local version that can be connected to a device.

Mock application

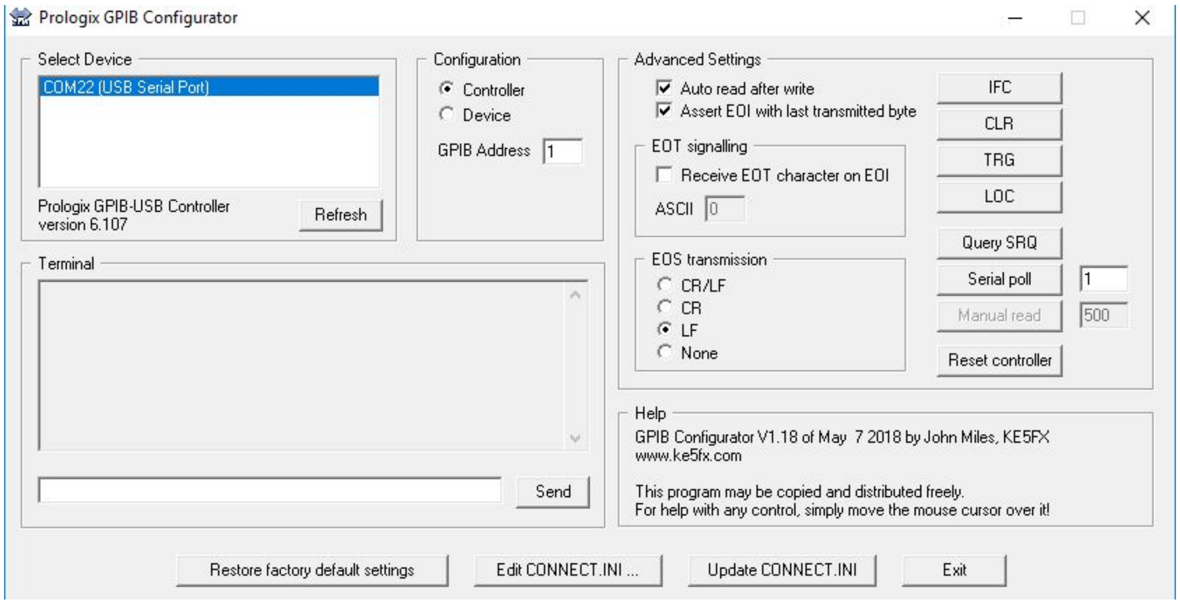
To run the mock application, simply run in the command line:

```
$ python app_mock.py
```

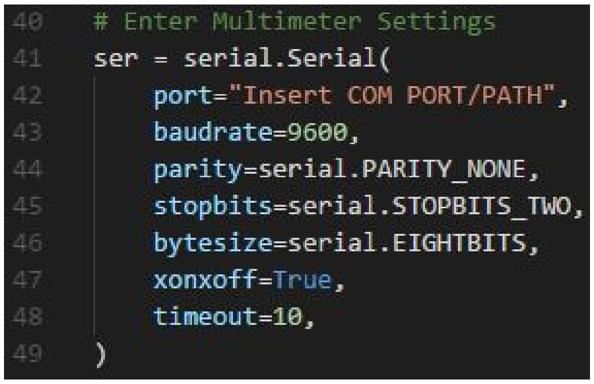
Next, open the web address that will display in the terminal.

Local application

If you would like to run the local version, please connect the Prologix GPIB located at the back of multimeter to the PC via USB.
If this is your first time connecting the multimeter to your PC, ensure the serial communication address of the HP multimeter and the Prologix GPIB are the same. You can check/change the address of the multimeter by going to:
Settings -> I/O Menu -> HP-IB ADDR -> Address
For the Prologix GPIB, you will have to use the Prologix GPIB Configurator which can be found in the Prologix resources, under ‘‘Utilities" here. Here is a screen shot of what the Prologix GPIB settings should be (note that the COM port may vary):


Prologix GPIB Configurator
Select Device
COM22 (USB Serial Port)
Prologix GPIB-USB Controller version 6.107
Refresh
Configuration
Controller
Device
GPIB Address 1
Advanced Settings
Auto read after write
Assert EOI with last transmitted byte
EOT signalling
Receive EOT character on EOI
ASCII 0
EOS transmission
CR/LF
CR
LF
None
IFC
CLR
TRG
LOC
Query SRQ
Serial poll 1
Manual read 500
Reset controller
Terminal
Send
Help
GPIB Configurator V1.18 of May 7 2018 by John Miles, KE5FX
www.ke5fx.com
This program may be copied and distributed freely.
For help with any control, simply move the mouse cursor over it!
Restore factory default settings
Edit CONNECT.INI ...
Update CONNECT.INI
Exit

Before starting the application, please change `Insert COM PATH/PORT HERE` to your COM port/path.

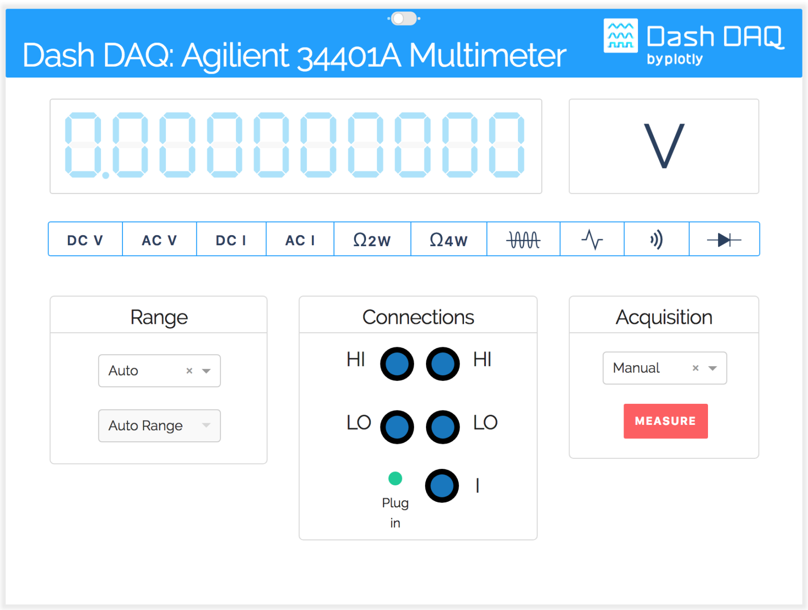


```
# Enter Multimeter Settings
ser = serial.Serial(
    port="Insert COM PORT/PATH",
    baudrate=9600,
    parity=serial.PARITY_NONE,
    stopbits=serial.STOPBITS_TWO,
    bytesize=serial.EIGHTBITS,
    xonxoff=True,
    timeout=10,
)
```

When you are ready, run the following line of code in your command line to initialize the application:

```
$ python app.py
```

Next, open the web address that will display in the terminal.
In the browser, you should see something like this:



undo

Controls

- DC V: Measure DC voltage.
- AC V: Measure AC voltage.
- DC I: Measure DC current.
- AC I: Measure AC current.
- 2 W: Measure resistance upto 2 watts.
- 4 W: Measure resistance upto 4 watts.
- Frequency: Measure frequency in Hz.
- Period: Measure period in seconds.
- Continuity: Check if continuous circuit (not open).
- Diode: Diode check.
- Acquisition: Capture data from measurements. Manual is every button press, and continuous captures data every second.
- Measure: Press to start capturing data.
- Range: The range of the measurement measured. Reads out data based on a set scale (e.g.m reading 10 V DC at a scale of 100 V DC or 5 VDC).

Resources

PySerial was used for serial communication. The API for PySerial can be found here. For a further understanding of how the multimeter works, and the SCPI language used to communicate, check out the manual for the multimeter here. If you need help connecting the Prologix GPIB, need drivers, or the GPIB configurator refer to the Prologix website here and specifically to the listed resources here.

dash-daq-iv-tracer

Introduction

An application that allows the user to acquire current-voltage (I-V) curves from a multimeter. Test out the app yourself at https://dash-gallery.plotly.host/dash-daq-iv-tracer/.

I-V curves

I-V curves are a good way to characterize electronic components (diode, transistor or solar cells) and extract their operating properties. They are widely used in electrical engineering and physics.

Hardware

This app was developed specifically with a Keithley 2400 SourceMeter.

Requirements

The virtual environment for this app should be using any version of Python 3. After setting the environment up with `virtualenv`, you can install the required packages:

```
$ pip install -r requirements.txt
```

How to use the app

There are two versions of this application. A mock version for the user to play with, without any instruments connected, and a local version, that can be connected to a device.

Mock application

You can run the mock application from `app.py`. For this method, ensure that in `app.py`, the `mock_mode` attribute is set to `True`:

```
iv_generator = keithley_instruments.KT2400(
  mock_mode=True,
  instr_port_name=[your instrument's COM/GPIB port]
)
```

Then, simply run in the command line:

```
$ python app.py
```

Alternatively, you can just run the following:

```
$ python app_mock.py
```

Next, open the web address that will display in the terminal.

Local application

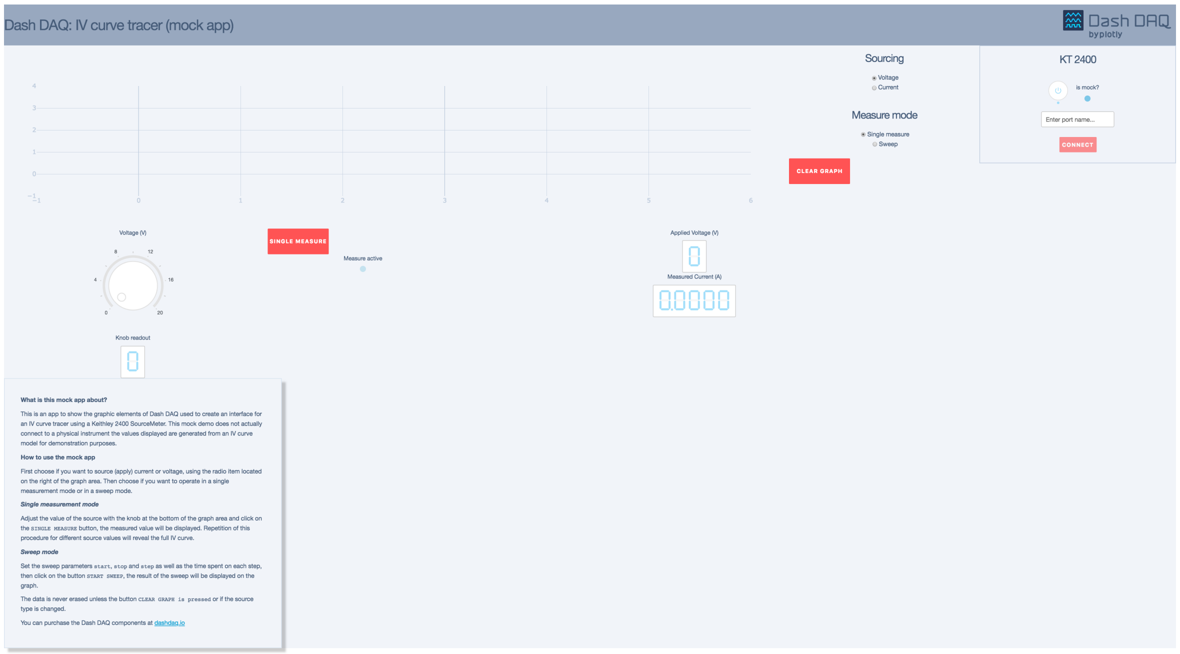
To control your SourceMeter, you need to set the `mock_mode` attribute to `False`:

```
iv_generator = keithley_instruments.KT2400(
  mock_mode=False,
  instr_port_name=[your instrument's COM/GPIB port]
)
```

Then, simply run in the command line:

```
$ python app.py
```

In the browser, you should see something like this:



What is this mock app about?

This is an app to show the graphic elements of Dash DAQ used to create an interface for an IV curve tracer using a Keithley 2400 SourceMeter. This mock demo does not actually connect to a physical instrument the values displayed are generated from an IV curve model for demonstration purposes.

How to use the mock app

First choose if you want to source (apply) current or voltage, using the radio item located on the right of the graph area. Then choose if you want to operate in a single measurement mode or in a sweep mode.

Single measurement mode

Adjust the value of the source with the knob at the bottom of the graph area and click on the `SINGLE MEASURE` button, the measured value will be displayed. Repetition of this procedure for different source values will reveal the full IV curve.

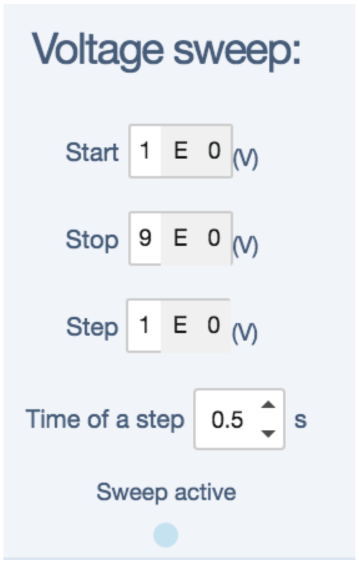
Sweep mode

Set the sweep parameters `start`, `stop` and `step` as well as the time spent on each step, then click on the button `START SWEEP`, the result of the sweep will be displayed on the graph.

The data is never erased unless the button `CLEAR GRAPH is pressed` or if the source type is changed.

You can purchase the Dash DAQ components at dashdaq.io

If you have "Measure mode" set to "Sweep", you will see this in place of the voltage knob and display on the left:



Controls

- Power button: When in the ''on" state, this enables all of the controls.
- Sourcing: Determines whether the voltage or the current is controlled by the instrument.
- Measure mode: Determines whether one measurement is taken, or whether the source is varied over a range and multiple measurements are taken throughout that range.
- Voltage/current knob, Knob readout: Controls and displays the magnitude of the source voltage or current, depending on which one is selected.
- Single measure/Start sweep: Takes a single measurement, or initiates the sweep.
- Clear graph: Removes all traces from the graph.
- Port name/Connect: If a port name is entered and ''Connect" is clicked, the application will connect to the instrument at the specified port.

Resources

The manual for the Keithley 2400 can be found here.

dash-daq-led-control

Introduction

An application that allows the user to control an LED stick. Test out the app yourself at https://dash-gallery.plotly.host/dash-daq-led/.

Light Emitting Diode Stick

The LED BlinkStick is a stick consisting of eight semi-conductor light emitting diodes. A diode is an electrical component that only allows electrical current to travel in one direction. When sufficient current passes through an LED, electrons cause this special diode to release energy, as photons. Photons are simply light particles, which is why we see the LED light up! This process is known as electroluminescence. Learn more about this here.

Hardware

This application was developed specifically with an LED BlinkStick.

Requirements

The virtual environment for this app should be using any version of Python 3. After setting the environment up with `virtualenv`, you can install the required packages:

```
$ pip install −r requirements.txt
```

How to use the app

There are two versions of this application. A mock version for the user to play with, without any instruments connected, and a local version, that can be connected to a device.

Mock application

To run the mock application, simply run in the command line:

```
$ python app_mock.py
```

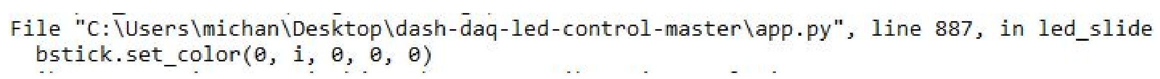
Next, open the web address that will display in the terminal.

Local application

If you would like to run the local version, please connect the device to the USB port on your computer, and run in the command line:

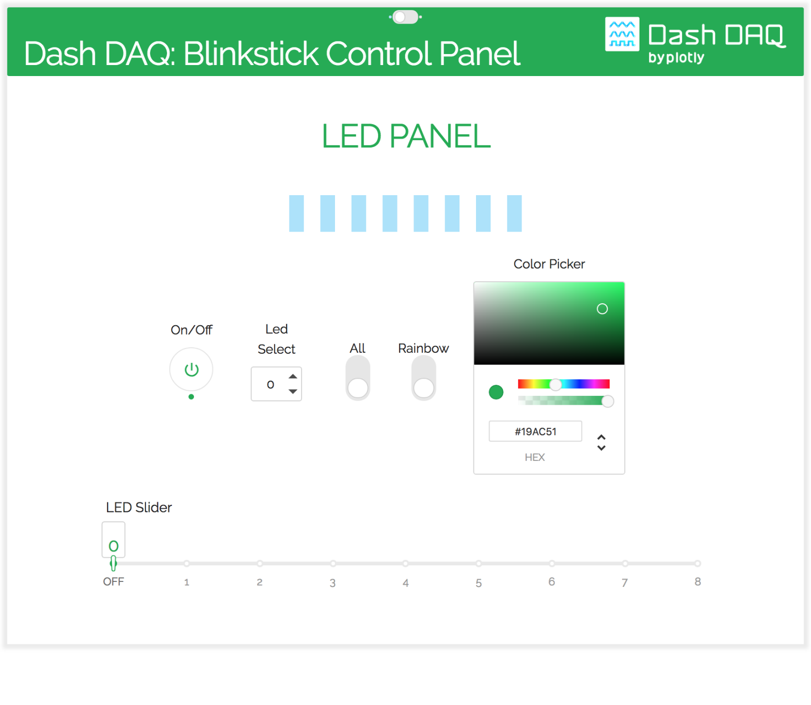
```
python app.py
```

Next, open the web address that will display in the terminal.
If the app is run, but the device is not connected, you will see something like this:



```
File "C:\Users\michan\Desktop\dash-daq-led-control-master\app.py", line 887, in led_slide
  bstick.set_color(0, i, 0, 0, 0)
```

If this occurs, check the connection to the BlinkStick and restart the app.
In the browser, you should see something like this:



Controls

- On/Off: Turn BlinkStick LEDs on or off.
- LED Select: Select individual LEDs and change their color, with the color picker.
- All: Select all LEDs and modify their colors simultaneously, with the color picker.
- Rainbow: All LEDs light up according to the rainbow color spectrum.
- LED Slider: Turn on LEDs one through to the selection on the slider, and change the color with color picker.
- Color Picker: Change the color of the LED/LEDs to the color selected.

Resources

The BlinkStick API is used in this application, and has a multitude of features. If you need some help figuring out the BlinkStick API, check out the documentation here. To install the BlinkStick API, go here. If you have trouble using the API with Python 3, check out this fix.

dash-daq-omega-pid

Introduction

An application to control a PID controller. Try the app out yourself at: https://dash-gallery.plotly.host/dash-daq-omega-pid/

Propotional, Integral, Derivative (PID) Gain Controllers

PID controllers are widely used in closed-loop control systems. A closed loop control system has an input that is reliant on the output of the system. This process is known as feedback. In this system the desired output is defined as the setpoint, where the difference between the setpoint and actual output defines the error of the system. This error is what is known as the feedback, and is then looped back to the input in what is known as the feedback loop. The proportional, integral, and derivative gains, as well as the error, are used in the PID equation to solve for the percentage of output applied to the system to reach the setpoint (refer to mathematical form). For more information and detail on this system please refer here.

Hardware

This application was developed specifically using a CN32PT-440-DC Omega PID Controller.

Requirements

The virtual environment for this app should be using any version of Python 3. After setting the environment up with `virtualenv`, you can install the required packages:

```
$ pip install -r requirements.txt
```

How to use the app

There are two versions of this application. A mock version for the user to play with, without any instruments connected, and a local version, that can be connected to a device.

Mock application

To run the mock application, simply run in the command line:

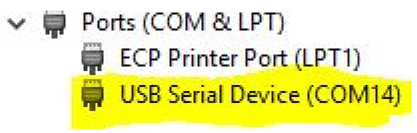
```
$ python app_mock.py
```

Next, open the web address that will display in the terminal.

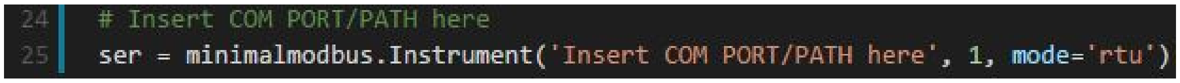
Local application

If you would like to run the local version, please ensure whether your device is configured using the LOW VOLTAGE POWER OPTION (12 VDC - 36 VDC or 24 VAC) or the AC POWER OPTION (110 VDC - 300 VDC or 90 VAC - 240VAC), and power it accordingly. For the setup used in testing this application, the LOW VOLTAGE POWER OPTION was set, where the PID controller was powered with 12 VDC. This application is made to use OUTPUT 1 of the system, with Thermocouples as the input. If you are having troubles connecting the device, you may need to install the drivers which can be found here:

When the drivers are properly installed you will see something like this highlighted port:



Open `app.py` and insert the COM PORT/PATH of the PID controller connection, where it is asked:

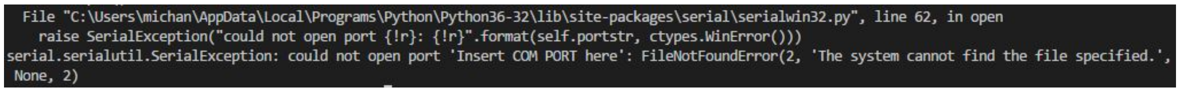


```
# Insert COM PORT/PATH here
ser = minimalmodbus.Instrument('Insert COM PORT/PATH here', 1, mode='rtu')
```

When the device is ready, run in the command line:

```
$ python app.py
```

Next, open the web address that will display in the terminal.
If the app is run, but the device is not connected you will see something like this:

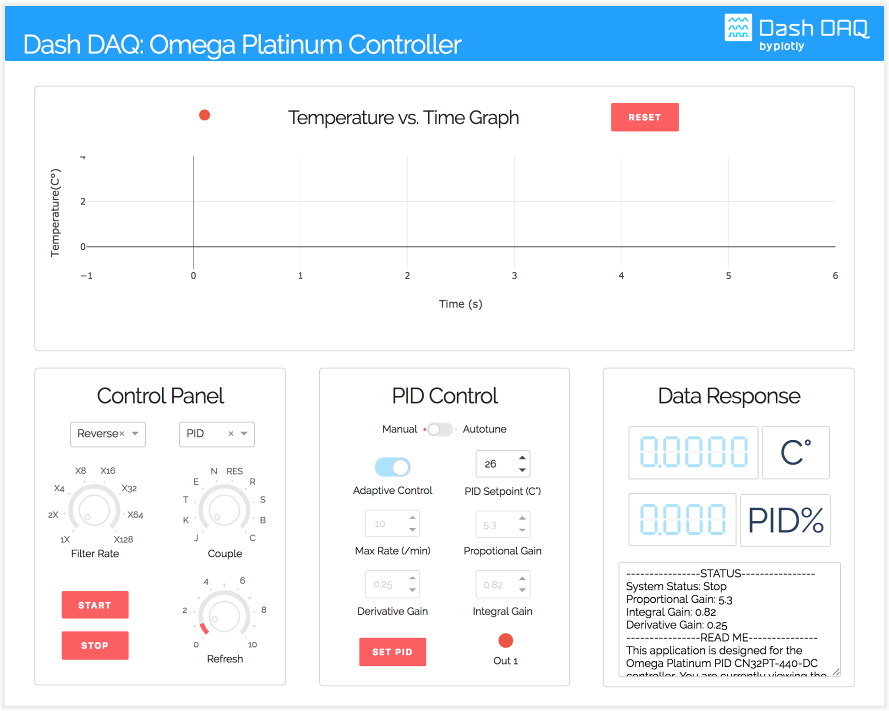


```
  File "C:\Users\michan\AppData\Local\Programs\Python\Python36-32\lib\site-packages\serial\serialwin32.py", line 62, in open
    raise SerialException("could not open port {!r}: {!r}".format(self.portstr, ctypes.WinError()))
serial.serialutil.SerialException: could not open port 'Insert COM PORT here': FileNotFoundError(2, 'The system cannot find the file specified.',
 None, 2)
```

If issues arise, try the following:

1. Unplugging all connections and plugging them back in.
2. Checking to see if correct COM PORT/PATH is selected.
3. Ensuring proper voltage is applied to the controller.
4. Performing a factory reset on the controller (refer to heading 6.13 of the manual).
5. Reinstalling the drivers.

In the browser, you should see something like this:



```
----------------STATUS----------------
System Status: Stop
Proportional Gain: 5.3
Integral Gain: 0.82
Derivative Gain: 0.25
----------------READ ME---------------
This application is designed for the
Omega Platinum PID CN32PT-440-DC
```

Controls

- Reset: Reset graph.
- PID%: The percentage of output applied to system. C^o: Current temperature.
- Manual (Boolean Switch): Switch to manual tuning PID.
- Autotune (Boolean Switch): Switch to autotuning PID.
- Autotune Timeout: Timeout for autotuning period.
- PID Setpoint: The desired output to reach.
- Max Rate (/min): Maximum rate of change per miniute.
- Proportional Gain: The proportional gain of the controller.
- Derivative Gain: The derivate gain of the controller.
- Integral Gain: The integral gain of the controller.
- Out 1: Output on/off.
- Set PID (Button): Set PID parameters in manual.
- Autotune (Button): Start Autotune, for optimized PID parameters. If Autotune fails (running status: fault), your system is not optomized for this and you will have to manually tune your PID controller.

- Adaptive Control: Fuzzy logic control more here Couple: The thermocouple type used.
- Refresh: Refresh rate of graph.
- Filter Rate: Sensitivity of thermocouple.
- Action Mode: Heating, cooling, or heating and cooling system.
- PID: The output is dependent on the PID parameters.
- Start: Start PID controller and graphing data.
- Stop: Stop PID controller and stop graphing data.

Resources

This application was controlled through the MODBUS using the `minimalmodbus` library, which can be found here. In order to determine outputs, inputs, and any other information related to the controller refer to the manual here. For a list of commands that can be written/read to the controller refer to the MODBUS manual here.

dash-daq-pressure-gauge-kjl

Introduction

An application that allows the user to control Kurt J. Lesker pressure gauge controllers. Test out the app yourself at https://dash-gallery.plotly.host/dash-daq-pressure-gauge-kjl.

Low pressure gauges

Low pressure gauges (1 atmosphere and below) are used in vacuum environments.

Hardware

This app was developed specifically with the Kurt J. Lesker MGC4000 multi-gauge controller.

Requirements

The virtual environment for this app should be using any version of Python 3. After setting the environment up with `virtualenv`, you can install the required packages:

```
$ pip install -r requirements.txt
```

How to use the app

There are two versions of this application: a mock version for the user to play with, without any instruments connected, and a local version that can be connected to a device.

Mock application

You can run the mock application from `app.py`. For this method, ensure that in `app.py`, the `mock` attribute is set to `True`:

```
pressure_gauge = MGC4000(
        instr_port_name=[your instrument's com port],
        mock=False,
        gauge_dict={'CG1': 'mbar', 'CG4': 'mbar'}
)
```

Then, simply run in the command line:

```
$ python app.py
```

Alternatively, you can just run the following:

```
$ python app_mock.py
```

Next, open the web address that will display in the terminal.

Local application

To control your gauge controller, you need to input your COM port number in the app.py file and set the mock attribute to `False`:

```
pressure_gauge = MGC4000(
        instr_port_name=[your instrument's com port],
        mock=False,
        gauge_dict={'CG1': 'mbar', 'CG4': 'mbar'}
)
```

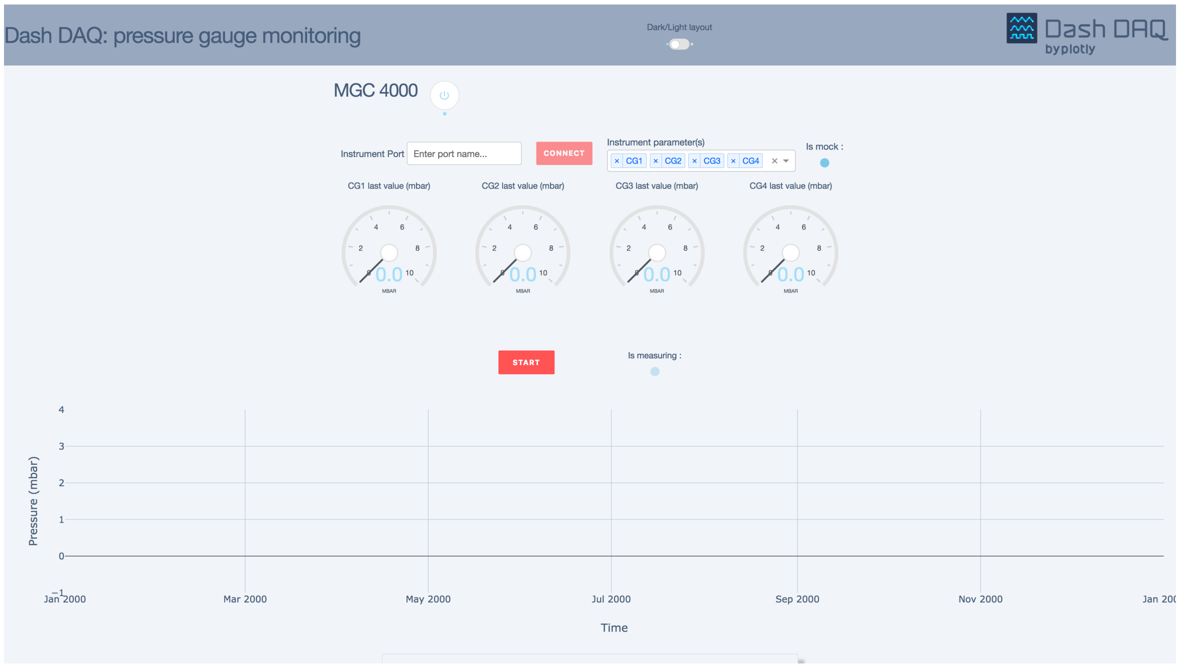
Your gauges in `gauge_dict` should be named after the convention `[Gauge type][Gauge number]`. The gauge types implemented up to now are the following two types:

- CG stands for ''Convection gauge" and is a Pirani pressure gauge valid in the pressure range 1.3E-4 to 1333 mbar (millibar).
- IG stands for ''Ion gauge" and is a cathode ion gauge valid in the pressure range 1.3E-9 to 6.7E-2 mbar.

You can then run the app:

```
$ python app.py
```

Next, open the web address that will display in the terminal.
In the browser, you should see something like this:



Controls

- Power button: When in the ''on" state, this enables all of the controls.

- ‘‘Instrument Port": When used in non-demo mode, allows you to choose a COM port or GPIB port to connect to. Note that you will only be able to do so when the input follows the format of `com[number]` or `gpib0::[number]` (or the uppercase variants of either of those).
- ‘‘Connect": Initiates a connection with the instrument entered in the ‘‘Instrument Port" field.
- ‘‘Start": Begins the measurement process, and the measurements are displayed on the plot below the controls.
- ‘‘Instrument parameter(s)": Chooses which traces are displayed on the graph.

Resources

The manual of the Kurt J. Lesker MGC4000 can be found here.

dash-daq-pressure-gauge-pfeiffer

Introduction

An application that allows the user to interface with a multi-gauge controller. Test out the app yourself at https://dash-gallery.plotly.host/dash-daq-pfeiffer-gauge.

Low pressure gauges

Low pressure gauges (1 atmosphere and below) are used in vacuum environments.

Hardware

This app was developed specifically with the Pfeiffer TPG256A multi-gauge controller.

Requirements

The virtual environment for this app should be using Python 3.6. After setting the environment up with `virtualenv`, you can install the required packages:

```
$ pip install -r requirements.txt
```

How to use the app

There are two versions of this application: a mock version for the user to play with, without any instruments connected, and a local version that can be connected to the device.

Mock application

You can run the mock application from `app.py`. For this method, ensure that in `app.py`, the `mock` attribute is set to `True`:

```
pressure_gauge = TPG256A(
        instr_port_name=[your instrument's com port],
        mock=True,
        gauge_dict={'P1': 'mbar', 'P4': 'mbar'}
)
```

Then, simply run in the command line:

```
$ python app.py
```

Alternatively, you can just run the following:

```
$ python app_mock.py
```

Next, open the web address that will display in the terminal.

Local application

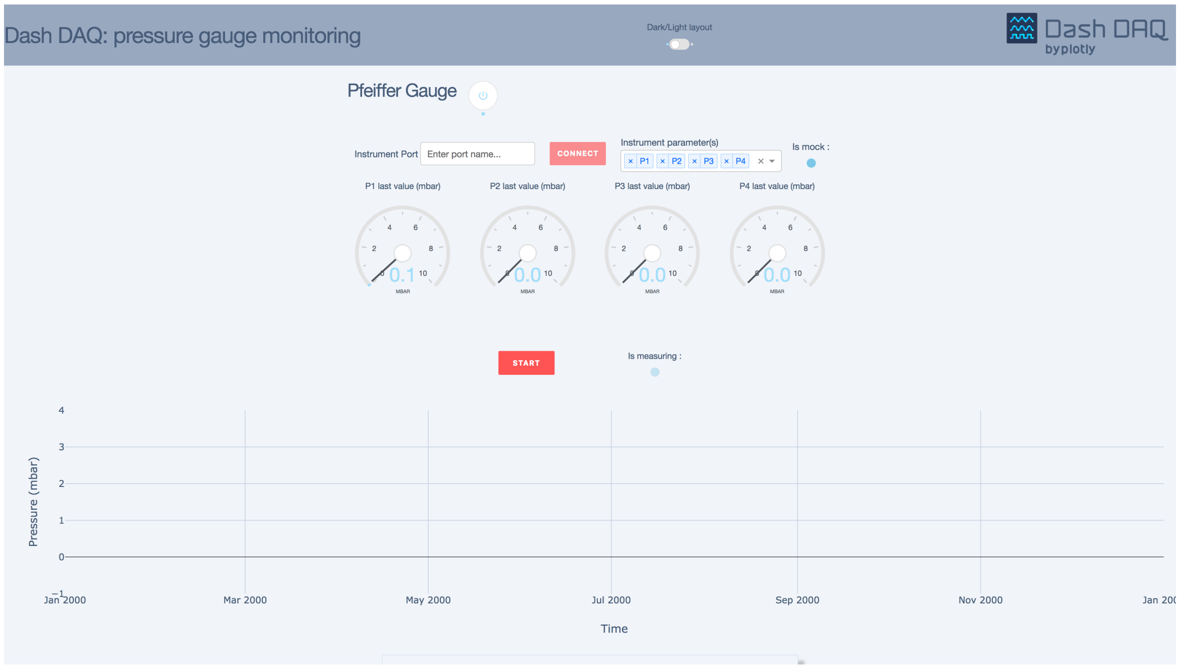
To control your gauge controller, you need to input your COM port number in the `app.py` file and set the mock attribute to `False`:

```
pressure_gauge = TPG256A(
        instr_port_name=[your instrument's com port],
        mock=False,
        gauge_dict={'P1': 'mbar', 'P4': 'mbar'}
)
```

Your gauges in `gauge_dict` should be named after the convention `P[input number of the gauge]`. You can then run the app:

```
$ python app.py
```

Next, open the web address that will display in the terminal.
In the browser, you should see something like this:



Controls

- Power button: When in the "on" state, this enables all of the controls.
- "Instrument Port": When used in non-demo mode, allows you to choose a COM port or GPIB port to connect to. Note that you will only be able to do so when the input follows the format of `com[number]` or `gpib0::[number]` (or the uppercase variants of either of those).
- "Connect": Initiates a connection with the instrument entered in the "Instrument Port" field.
- "Start": Begins the measurement process, and the measurements are displayed on the plot below the controls.
- "Instrument parameter(s)": Chooses which traces are displayed on the graph.

Resources

The manual of the Pfeiffer TPG256A can be found here.

dash-daq-robotic-arm-edge

Introduction

An application that allows the user to control a robotic arm. Test out the app yourself at https://dash-gallery.plotly.host/dash-daq-robotic-arm-edge/.

Robotic arms

This robot arm is a nice hobby project for anyone who loves electronics. It is controlled by multiple stepper motors, allowing it to rotate left and right, and of course move forwards and backwards. It also includes an LED light, located near the grippers. Robotic arms are used in industry and manufacturing all over the world. They have a huge variety of applications.

Hardware

This application was developed specifically with a Robotic Arm Edge.

Requirements

The virtual environment for this app should be using any version of Python 3. After setting the environment up with `virtualenv`, you can install the required packages:

```
$ pip install -r requirements.txt
```

In order to use this application you will also need the USB interface component.

How to use the app

If you are running Windows 10, installing the drivers may present some difficulties. Follow the instructions at the bottom of this page, titled Windows, to install working drivers for this app and the required `roboarm` library.
There are two versions of this application - a mock version for the user to play with, without any instruments connected, and a local version, that can be connected to a device.

Mock application

To run the mock application, simply run in the command line:

```
$ python app_mock.py
```

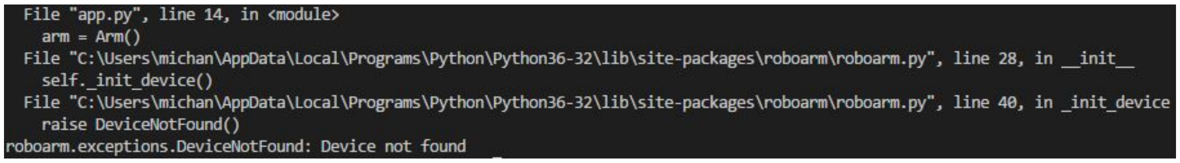
Next, open the web address that will display in the terminal.

Local application

If you would like to run the local version, please connect the USB interface of the Robotic Arm Edge to the USB port on your computer, and run in the command line:

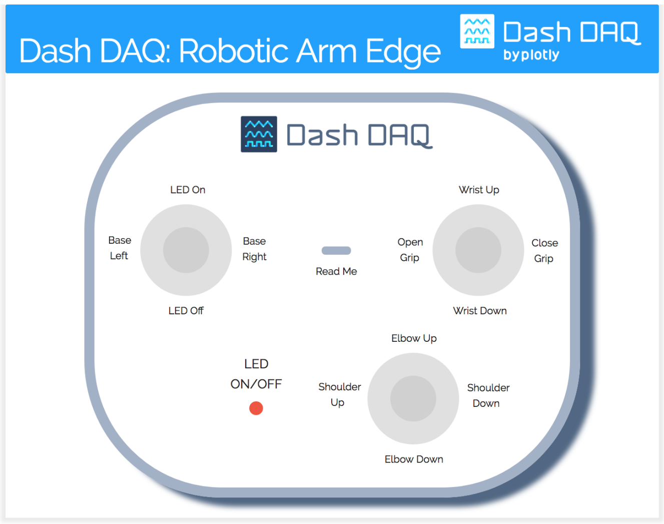
```
$ python app.py
```

Next, open the web address that will display in the terminal.
If the app is run, but the device is not connected you will see something like this:



```
  File "app.py", line 14, in <module>
    arm = Arm()
  File "C:\Users\michan\AppData\Local\Programs\Python\Python36-32\lib\site-packages\roboarm\roboarm.py", line 28, in __init__
    self._init_device()
  File "C:\Users\michan\AppData\Local\Programs\Python\Python36-32\lib\site-packages\roboarm\roboarm.py", line 40, in _init_device
    raise DeviceNotFound()
roboarm.exceptions.DeviceNotFound: Device not found
```

If this occurs, check the connection to the arm and restart the app.
In the browser, you should see this:



Controls

- Left Joystick: Moves base left and right. Turns LED on and off.
- Read Me: Displays read me.
- Top Right Joystick: Moves wrist up and down. Opens and closes grippers.
- Bottom Right Joystick: Moves elbow up and down. Moves shoulder up and down.

Resources

The RoboArm library was used to control the Robotic Arm Edge. The API can be found here.

dash-daq-sparki

Introduction

An application that allows the user to wirelessly control an Arduino powered robot. Test out the app yourself at https://dash-gallery.plotly.host/dash-daq-sparki/.

Hardware

This app was developed with Sparki, a pre-built robot, consisting of: an ultrasonic sensor, piezometer, stepper motor, servos, an RGB LED, and a bluetooth module. This robot is built upon a Arduino microcontroller, and is programmed with Sparkiduino, an Ardunio based IDE.

Requirements

The virtual environment for this app should be using any version of Python 3. After setting the environment up with **`virtualenv`**, you can install the required packages:

```
pip install −r requirements.txt
```

How to use the app

There are two versions of this application. A mock version for the user to play with, without any instruments connected, and a local version, that can be connected to a device.

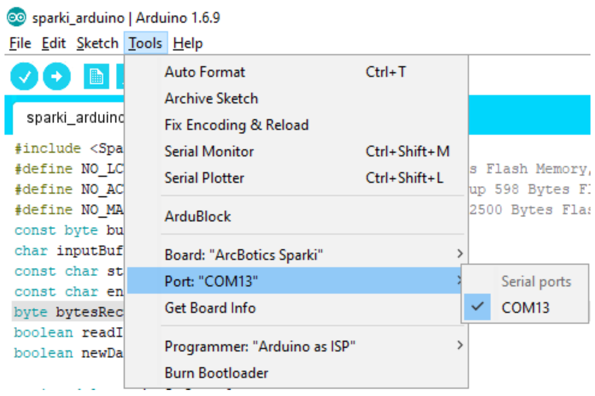
Mock application

To run the mock application, simply run

```
$ python app_mock.py
```

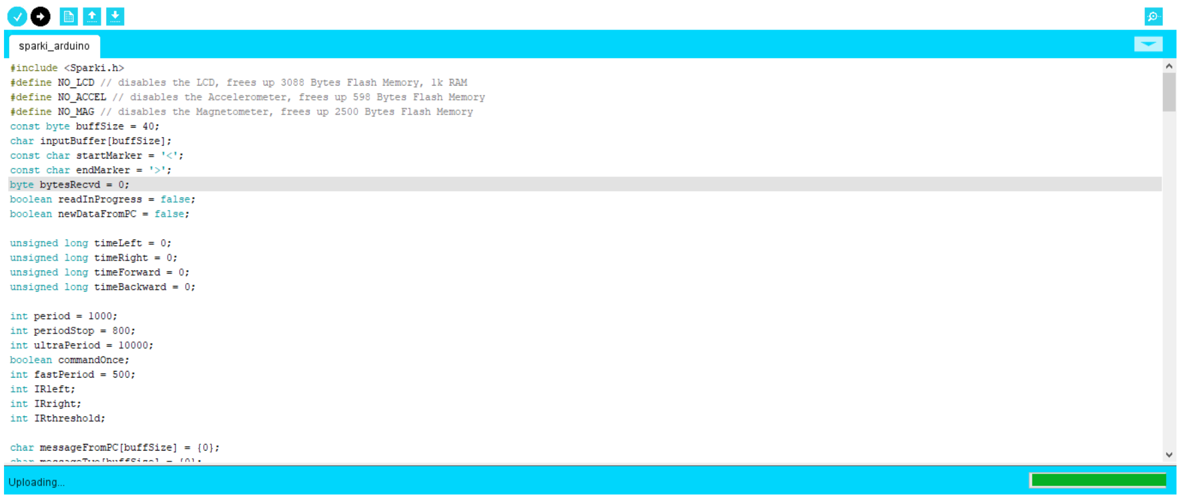
Local application

If you would like to run the local version, first ensure that the following file is uploaded to Sparki using Sparkiduino.
Open the provided file with Sparkiduino **`sparki_arduino.ino`**, and ensure the board ArcBotics Sparki and correct port are selected:


sparki_arduino | Arduino 1.6.9
File Edit Sketch Tools Help
sparki_arduino
Auto Format Ctrl+T
Archive Sketch
Fix Encoding & Reload
Serial Monitor Ctrl+Shift+M
Serial Plotter Ctrl+Shift+L
ArduBlock
Board: "ArcBotics Sparki"
Port: "COM13"
Get Board Info
Programmer: "Arduino as ISP"
Burn Bootloader
Serial ports
COM13

```
#include <Spa
#define NO_LC                         s Flash Memory,
#define NO_AC                         up 598 Bytes F
#define NO_MA                         2500 Bytes Fla
const byte bu
char inputBuf
const char st
const char en
byte bytesRec
boolean readI
boolean newDa
```

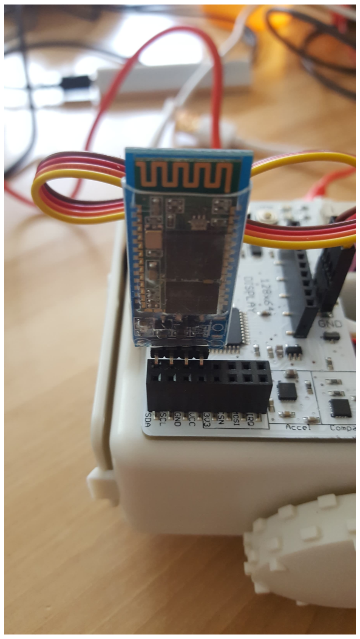
Next, upload the file to Sparki:



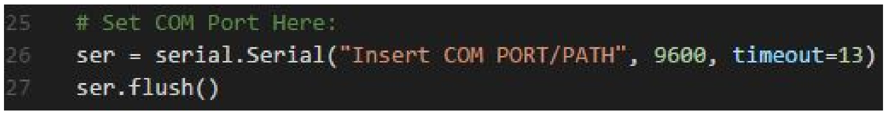
```
#include <Sparki.h>
#define NO_LCD // disables the LCD, frees up 3088 Bytes Flash Memory, 1k RAM
#define NO_ACCEL // disables the Accelerometer, frees up 598 Bytes Flash Memory
#define NO_MAG // disables the Magnetometer, frees up 2500 Bytes Flash Memory
const byte buffSize = 40;
char inputBuffer[buffSize];
const char startMarker = '<';
const char endMarker = '>';
byte bytesRecvd = 0;
boolean readInProgress = false;
boolean newDataFromPC = false;

unsigned long timeLeft = 0;
unsigned long timeRight = 0;
unsigned long timeForward = 0;
unsigned long timeBackward = 0;

int period = 1000;
int periodStop = 800;
int ultraPeriod = 10000;
boolean commandOnce;
int fastPeriod = 500;
int IRleft;
int IRright;
int IRthreshold;

char messageFromPC[buffSize] = {0};
```

Uploading...

If you have issues uploading to Sparki, you can try performing a manual upload.
When the file is uploaded, plug the HC-05 Bluetooth module into Sparki like so:



Now, you can set up the Bluetooth connection between your computer and Sparki by following the instructions here for Windows, and here for Mac.
Once you have finished setting up the connection bewteen Sparki and your PC, take the serial port path or COM PORT and modify it in `app.py`:

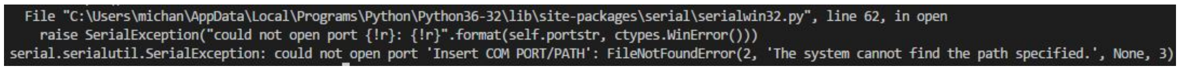


```
    # Set COM Port Here:
    ser = serial.Serial("Insert COM PORT/PATH", 9600, timeout=13)
    ser.flush()
```

Start the app by running the following in the command line:

```
$ python app.py
```

Next, open the web address that will display in the terminal.
If the app is running, but the device is not connected, you might see something like this:

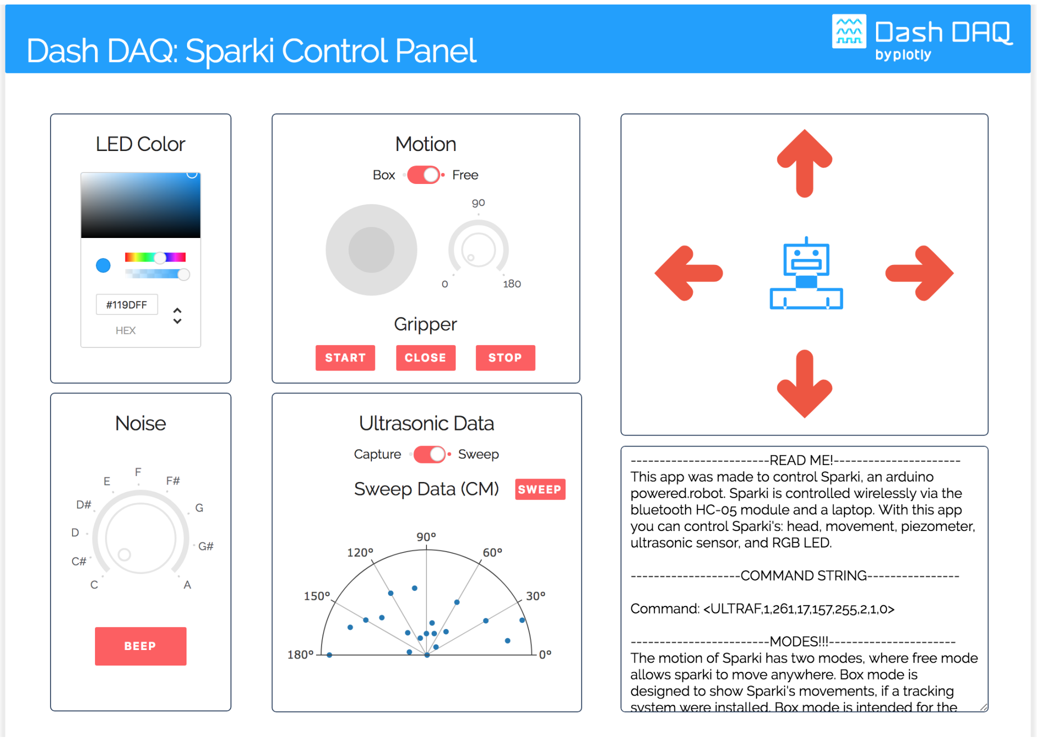


```
  File "C:\Users\michan\AppData\Local\Programs\Python\Python36-32\lib\site-packages\serial\serialwin32.py", line 62, in open
    raise SerialException("could not open port {!r}: {!r}".format(self.portstr, ctypes.WinError()))
serial.serialutil.SerialException: could not open port 'Insert COM PORT/PATH': FileNotFoundError(2, 'The system cannot find the path specified.', None, 3)
```

Sometimes the connection drops with Sparki; if you have issues with connection, try the following:

1. Turning the Bluetooth on and off on the PC
2. Pressing the reset button on Sparki
3. Running `app.py`

In the browser, you should see this:



------------------------READ ME!----------------------
This app was made to control Sparki, an arduino powered.robot. Sparki is controlled wirelessly via the bluetooth HC-05 module and a laptop. With this app you can control Sparki's: head, movement, piezometer, ultrasonic sensor, and RGB LED.

-------------------COMMAND STRING----------------

Command: <ULTRAF,1,261,17,157,255,2,1,0>

------------------------MODES!!!----------------------
The motion of Sparki has two modes, where free mode allows sparki to move anywhere. Box mode is designed to show Sparki's movements, if a tracking system were installed. Box mode is intended for the

Controls

- Color Picker: Changes colors of the RGB LED, and the Sparki icon.
- Motion Joystick: Controls the stepper motors in Sparki's wheels, allowing it to move left, right, forwards, and backwards.
- Motion Knob: Controls Sparki's ultrasonic sensor location.
- Gripper Start: Open grippers.

- Gripper Close: Close grippers.
- Gripper Stop: Stop grippers.
- Sweep (Boolean Switch): Puts ultrasonic sensor in sweep mode.
- Sweep: Ultrasonic sensor scans a 180 degree area, and graphs distance of objects every 10 degrees.
- Capture (Boolean Switch): Puts ultrasonic sensor in capture mode.
- Capture: Ultrasonic sensors scans area directly in front of it, and returns distance.
- Detected LED: Green indicates object detected. Red indicated error, or object not detected.
- Beep: Plays a frequency from the piezometer according to the knob frequency.

Resources

PySerial was used for serial communication, over bluetooth. The API for PySerial can be found here. For programming Sparki, use the Sparkiduino IDE, which also comes with the necessary drivers; you can find the link here.

dash-daq-stepper-motor

Introduction

An application that allows the user to control the speed and absolute positioning of a stepper motor. Test out the app yourself at https://dash-gallery.plotly.host/dash-daq-stepper-motor/.

Stepper motors

When DC voltage is applied to one of the coils in the motor, a magnetic field is produced, due to electromagnetic induction. This field can either attract or repel the magnetized gear depending on the direction of current flowing through the coils, which can best be understood through Lenz' Law. The job of the magnetic field is to rotate and align the magnetic gear, with the produced magnetic field (coil). This movement is known as a step. The direction of current flow is controlled by a driver, which directs current/voltage to the coils based on the demand of the user. The coils act synchronously, in groups called phases, that can be directed by the driver to move to precise locations, or rotational speeds. For a further understanding, read more here.

Hardware

This application was developed specifically with a Lin Engineering SilverPak17C Stepper Motor.

Requirements

The virtual environment for this app should be using any version of Python 3. After setting the environment up with `virtualenv`, you can install the required packages:

```
pip install -r requirements.txt
```

How to use the app

There are two versions of this application. A mock version for the user to play with, without any instruments connected, and a local version, that can be connected to a device.

Mock application

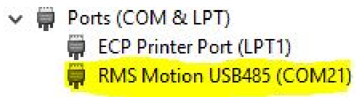
To run the mock application, simply run

```
$ python app_mock.py
```

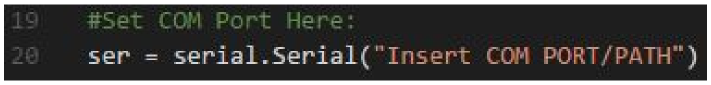
Next, open the web address that will display in the terminal.

Local Application

If you would like to run the local version, please ensure that the stepper motor voltage (12 VDC - 40 VDC) is being run to the stepper motor, the stepper motor is plugged into the USB 485 converter card, and the USB 485 converter card is plugged into the PC. If this is your first time using the stepper motor, you may need to install the drivers to use the USB 485 converter card. Instructions and the necessary files to install the driver can be found on the Lin Motors website here under "Downloads". When the drivers are properly installed, you will see something like this highlighted port:



Open `app.py` and insert the COM PORT/PATH of the stepper motor where it is asked:

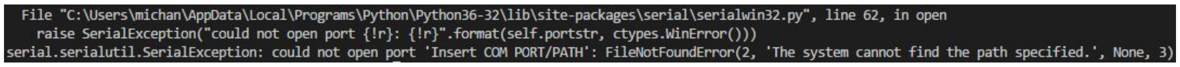


```
#Set COM Port Here:
ser = serial.Serial("Insert COM PORT/PATH")
```

When the device is ready, run in the command line:

```
$ python app.py
```

Next, open the web address that will display in the terminal.
If the app is run, but the device is not connected, you will see something like this:



```
  File "C:\Users\michan\AppData\Local\Programs\Python\Python36-32\lib\site-packages\serial\serialwin32.py", line 62, in open
    raise SerialException("could not open port {!r}: {!r}".format(self.portstr, ctypes.WinError()))
serial.serialutil.SerialException: could not open port 'Insert COM PORT/PATH': FileNotFoundError(2, 'The system cannot find the path specified.', None, 3)
```

If this occurs, re-connect the motor to the computer and restart the app.
Sometimes the application may freeze and hang due to the serial communication. You can fix this by:

- Unplugging all connections and plugging them back in.
- Checking to see if correct COM PORT/PATH is selected.
- Ensuring proper voltage is applied to the motor (12 VDC - 40 VDC).
- Checking the ground wire from the USB485, stepper motor, and power supply are connected.
- Seeing if voltage is running to the stepper motor.
- Reinstalling the drivers.

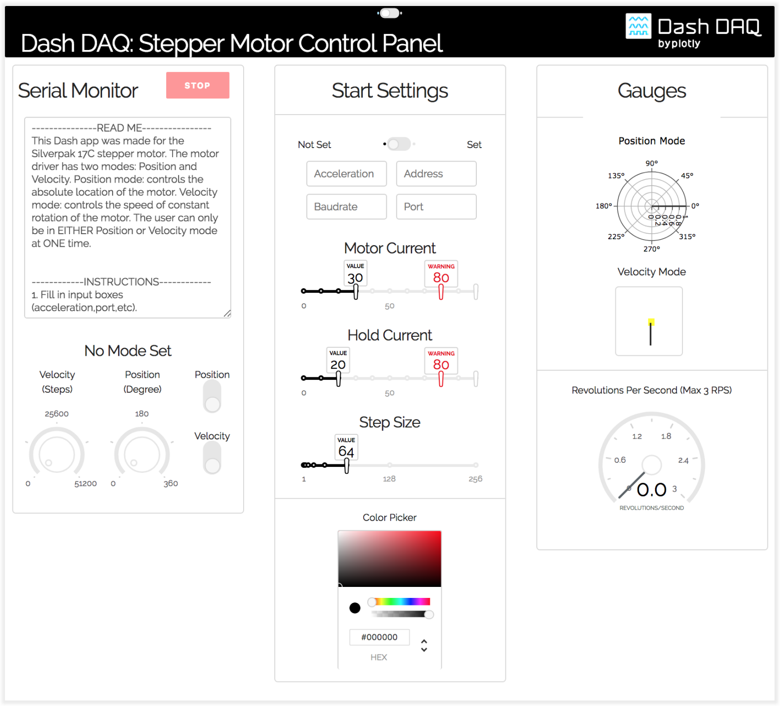
If the app freezes, it is because the device is not sending back data, so the serial port is floating. In this case change all the following blocks from:

```
response = str(ser.read(7))
```

to

```
response = None
```

In the browser, you should see this:



Dash DAQ: Stepper Motor Control Panel

Serial Monitor

---------------READ ME----------------
This Dash app was made for the Silverpak 17C stepper motor. The motor driver has two modes: Position and Velocity. Position mode: controls the absolute location of the motor. Velocity mode: controls the speed of constant rotation of the motor. The user can only be in EITHER Position or Velocity mode at ONE time.

------------INSTRUCTIONS------------
1. Fill in input boxes (acceleration,port,etc).

Controls

- Dark Theme: Boolean switch located in the banner, allows the user to switch to the dark theme.
- Color Picker: Changes colors of components and banner.
- Acceleration: Sets rate to get to specified velocity in microsteps per second squared.
- Address: The port address, set on the reciever and sender.
- Baudrate: The rate at which data is sent through serial communication. This should be same for reciever and sender.
- Port: Sets the port in the global variable version of the application.
- Motor Current: The percentage of max current to run the motor at.
- Hold Current: The percentage of max current to hold the motor in idle.
- Step Size: The resolution in microsteps of the full step size.
- Position Mode (Graph): The absolute positioning of the motor, in position mode.
- Revolutions Per Second: The amount of rotations per second in velocity mode.
- Not Set/Set: Sends a command through PySerial, setting the stepper motors, acceleration, address, baudrate, port, motor current, hold current, and step size.
- Stop: Terminates any commands running, and flushes the serial port.

- Velocity (Switch): Sets the motor to velocity mode.
- Position: (Switch): Sets the motor to position mode.
- Velocity (Steps): The velocity of the motor in microsteps/second.
- Position (Degree): The absolute positioning of the motor based on it's initial starting point as 0 degrees.

Resources

PySerial was used for serial communication. The API for PySerial can be found here. More detail and information on the commands that the stepper motor can take can be found in the Lin Engineering documentation. For a better understanding of this motor read the motor manual here.

dash-daq-ocean-optics

Introduction

An application that allows the user to control and read data from Ocean Optics spectrometers. Test out the app yourself at https://dash-gallery.plotly.host/dash-ocean-optics.

Absorption spectroscopy

Certain wavelengths of electromagnetic radiation correspond to frequencies that allow the electrons in certain atoms to transition to higher or lower energy levels; as these wavelengths are absorbed by the sample, the resulting spectrum can yield insight into the chemical composition of the sample. Read more about spectroscopy here.

Hardware

This app was developed specifically using the Ocean Optics USB2000+. Connect the spectrometer to your computer using the USB cord that came with it.

Requirements

It is advisable to create a separate `conda` environment (not `virtualenv`) running Python 3 for the app and to install all of the required packages there. To do so, run (any version of Python 3 will work; for the example, we use Python 3.4):

```
$ conda create -n       [your environment name] python=3.4
$ source activate [your environment name]
```

To install all of the required packages to this `conda` environment, simply run:

```
$ pip install -r requirements.txt
$ conda install -c poehlmann python-seabreeze
```

and all of the required `pip` packages, as well as the `python-seabreeze` package, will be installed, and the app will be able to run.

How to use the app

There are two versions of this application: a mock version for the user to play with, without any instruments connected, and a local version that can be connected to a device.

Mock application

To run the mock application, simply run:

```
$ python3 app.py demo
```

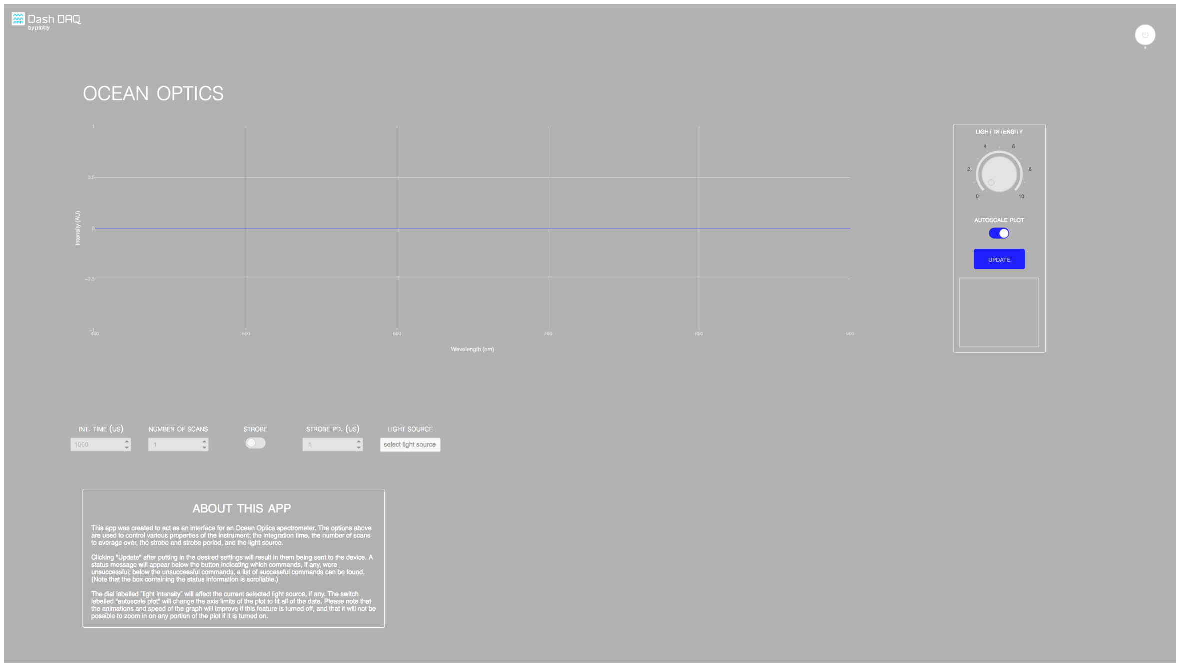
Next, open the web address that will display in the terminal.

Local application

You can run the app from command line:

```
$ python3 app.py
```

Next, open the web address that will display in the terminal. If the app is run, but the device is not connected properly, you will see an error in the terminal window. If this occurs, unplug the device (without exiting the app) and plug it back in.
In the browser, you should see something like this:



INT. TIME (US): 1000
NUMBER OF SCANS: 1
STROBE
STROBE PD. (US): 1
LIGHT SOURCE: select light source

ABOUT THIS APP

This app was created to act as an interface for an Ocean Optics spectrometer. The options above are used to control various properties of the instrument; the integration time, the number of scans to average over, the strobe and strobe period, and the light source.

Clicking "Update" after putting in the desired settings will result in them being sent to the device. A status message will appear below the button indicating which commands, if any, were unsuccessful; below the unsuccessful commands, a list of successful commands can be found. (Note that the box containing the status information is scrollable.)

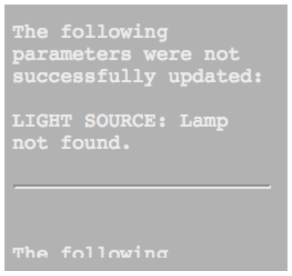
The dial labelled "light intensity" will affect the current selected light source, if any. The switch labelled "autoscale plot" will change the axis limits of the plot to fit all of the data. Please note that the animations and speed of the graph will improve if this feature is turned off, and that it will not be possible to zoom in on any portion of the plot if it is turned on.

Controls

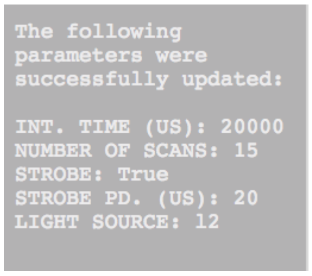
- Int. time (us): The integration time, in microseconds.
- Number of scans: The number of scans to average over.
- Strobe: Enables/disables the continuous strobe.
- Strobe pd. (us): The period of the continuous strobe, in microseconds.
- Light source: The light source to be used.
- Update: Sends all of the above options to the spectrometer. (Note that the values are not automatically updated; every time one of the parameters above is changed, you must click ''Update".)
- Power button: Enables or disables the controls.
- Light intensity: Controls the intensity of the light source, if one is selected.

- Autoscale plot: Controls whether the axes of the graph automatically change according to the limits of the data. To improve the animations and speed of the graph, it is necessary to switch this off.

The window below the "update" button displays the commands that failed, with the associated error messages, and the commands that succeeded, with the new values. Please note that this window is also scrollable.



```
The following
parameters were not
successfully updated:

LIGHT SOURCE: Lamp
not found.

The following
```



```
The following
parameters were
successfully updated:

INT. TIME (US): 20000
NUMBER OF SCANS: 15
STROBE: True
STROBE PD. (US): 20
LIGHT SOURCE: 12
```

Advanced

Configuring the colours The colours for all of the Dash and Dash-DAQ components are loaded from `colors.txt`. Note that if you want to change the appearance of other components on the page, you'll have to link a different CSS file in `app.py`.

Adding your own controls In order to add a control yourself, you must:

1. Create a new `Control` object in `app.py`; note that the `component_attr` dictionary must have the key `id` in order for the callbacks to be properly triggered. Append this new object to the list `controls` within `app.py`.

2. Add the key-value pair "[dash component id]", "[function object associated with control]" to the dictionary `self._controlFunctions` in the `PhysicalSpectrometer` and `DemoSpectrometer` class definitions (if you don't want this control to have any effect in the demo mode, then set the value to "`empty_control_demo`").

Adding your own spectrometers Although this app was created for Ocean Optics spectrometers, it is possible to use it to interface with other types of spectrometers. The abstract base class `DashOceanOpticsSpectrometer` contains a set of methods and properties that are necessary for the spectrometer to properly interface with the app. Please note that you should be using the communication and spectrometer locks as necessary to avoid issues with two different callbacks trying to modify/read the same thing concurrently.

Resources

The `python-seabreeze` library is required to communicate with the spectrometer; its source code and some documentation can be found here.

dash-daq-tektronix-350

Introduction

An application that allows the user to control and read data from a function generator and an oscilloscope. Test out the app yourself at https://dash-gallery.plotly.host/dash-tektronix-350.

Oscilloscopes

Oscilloscopes and function generators are widely used for diagnostic purposes involving electronic equipment.

Hardware

This application was developed specifically with a Tektronix TDS350 oscilloscope and a Tektronix AFG3021 function generator.

Requirements

The virtual environment for this app should be using any version of Python 3. After setting the environment up with `virtualenv`, you can install the required packages:

```
pip install —r requirements.txt
```

How to use the app

There are two versions of this application. A mock version for the user to play with without any instruments connected, and a local version that can be connected to a device.

Mock application

To run the mock application, simply run in the command line:

```
$ python app_mock.py
```

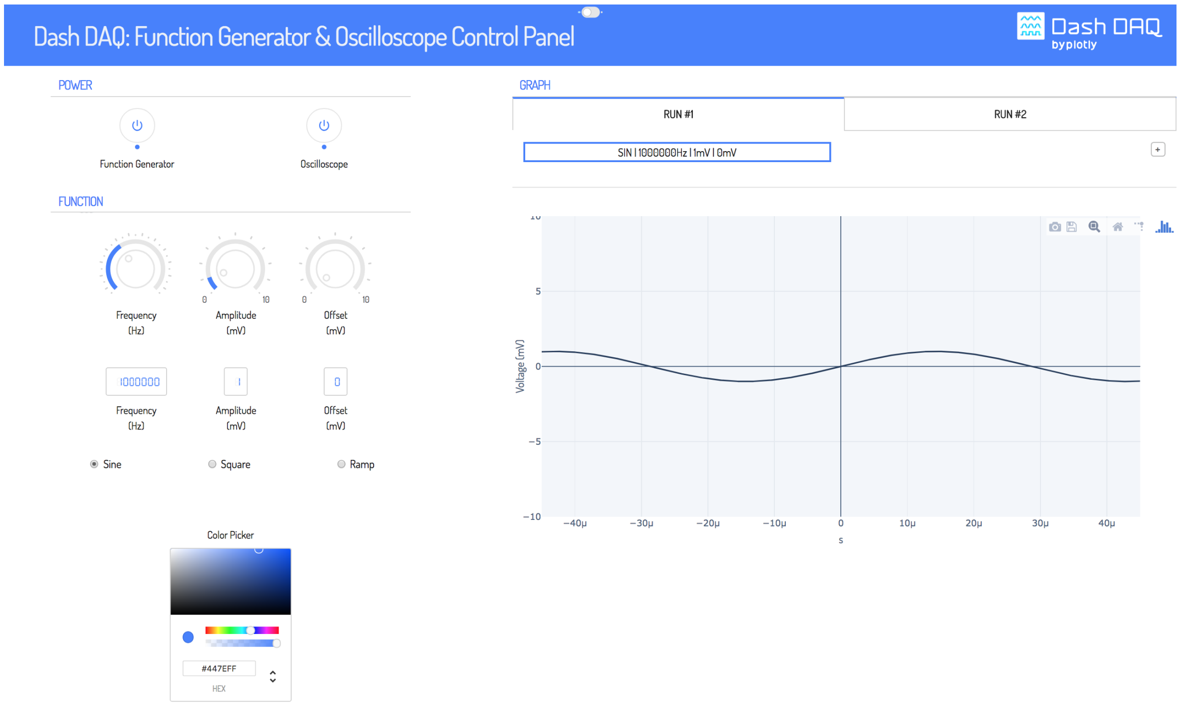
Next, open the web address that will display in the terminal.

Local application

First, ensure that the connection(s) between the oscilloscope and/or the function generator and the computer have been made correctly. You should be using a GPIB-USB-HS adapter.
When the device is ready, run in the command line:

```
$ python app.py
```

Next, open the web address that will display in the terminal.
In the browser, you should see this:



Controls

- Power: Turns the function generator or oscilloscope on or off.
- Dark Theme: Boolean switch located in the banner, allows the user to switch to the dark theme.
- Frequency (Hz): Controls the frequency of the function generator's output.
- Amplitude (mV): Controls the amplitude of the function generator's output.
- Offset (mV): Controls the DC offset of the function generator's output.
- Sine/Square/Ramp: Controls the form of the function generator's output.
- Color picker: Changes the color of components.
- Graph: Displays the output of the function generator or the oscilloscope. Each tab represents a different run. The + sign above the graph generates a new run with specified parameters.

Resources

The manuals for the oscilloscope and the function generator can be found here and here, respectively.