APPJ Communication Guide

The APPJ is operated via communication to an Arduino. This Arduino accepts commands to change various parameters of the APPJ setup. Additionally, it sends various measured data in a comma-separated line. See the Firmware folder for more information and the associated firmware for the Arduino.

Communication to the Arduino can be established by connecting to the device and reading its serial output. This may be achieved directly through the Terminal (Unix)/Command Prompt (Windows)\*. The following command(s) may be used:

1. Initialize the connection to the Arduino:  
   $stty -f [ARD\_ADDR] raw 38400 -hupcl & cat [ARD\_ADDR]  
   where [ARD\_ADDR] is the absolute path to the Arduino which includes the device address. An example when connected to a Mac computer may be: /dev/cu.usbmodem141301 (This changes with device and computer and should be determined when a new connection is made.)
2. Send commands to change operating parameters:  
   $echo "[char],[val]" > [ARD\_ADDR]  
   where [char] is a character representing the operating parameter to be changed and [val] is the value to which the operating parameter should be changed. A list of the operating parameters may be found in Table 1.  
   By default, the operating parameters that must be sent to ignite the plasma are the power (w), flow rate (q), and duty cycle (p). Operating parameters are saturated to be within appropriate operating ranges.

Table 1: Table of operating parameters for the APPJ.

|  |  |  |
| --- | --- | --- |
| **Operating Parameter** | **Character Representation** | **Range [units] (default)** |
| Duty Cycle | p | 0 – 100 [%] (0) |
| Primary Gas Flow | q | 0 – 10 [slm] (0) |
| Secondary Gas Flow\*\* | o | 0 – 20 [sccm] (0) |
| Frequency | f | 10 – 20 [kHz] (20) |
| Power | w | 1.5 – 5 [W] (0) |
| X Position\*\* | x | -50 – 50 [mm] (0) |
| Y Position\*\* | y | -50 – 50 [mm] (0) |
| Z Position\*\* | d | 0 – 20 [mm] (4) |
| Peak-to-Peak Voltage\*\*\* | v | 0 – 10 [kV] (0) |

\*Windows connections have not been tested.

\*\*These operating parameters have not been verified for use after the 2020 move to 2nd floor of Tan.

\*\*\*The default setup has V14 firmware preloaded to the Arduino (which allows for embedded control of the power via manipulation of the peak-to-peak voltage). Peak-to-peak voltage manipulation is not supported on this version, but is supported with V12 firmware.

Logging data sent to and received from the APPJ may be done via a Python script. This requires the following Python packages to be installed:

* matplotlib\*
* numpy\*
* scipy\*
* ipykernel\*
* seabreeze
* libusb1
* pyserial
* opencv-python\*\*
* opencv-python-headless\*\*
* pyvisa
* python-usbtmc
* pyusb
* crcmod

\*only necessary for data visualization and manipulation

\*\*choose only one of opencv-python or opencv-python-headless

Additionally, install [**libuvc from groupgets**](https://github.com/groupgets/libuvc).

Once necessary packages have been installed, connect your device to the setup and preview the collectDataOL.py file for an example of how to automate the process of collecting data from the APPJ. In summary, the following components should be included in your script:

1. create a connection to all devices within setup (Serial connection to Arduino, oscilloscope connection, spectrometer (fiber optic cable OES) connection, thermal camera connection)
2. create a set of run options (see RunOpts class in APPJFunctions.py) to decide data collection, data saving, and sampling time of the data collection.
3. Initialize data collection by running the asynchronous task functions once:  
   (see)
4. Initialize container variables for saving data. Tip: It is recommended to initialize the entire array for data collection first rather than appending entries as the code loops.
5. Run experiment:
   1. If collecting open loop data, you should initialize various inputs to the APPJ before running the experiment loop
   2. If collecting closed loop data, you should initialize all parts of your control scheme (e.g., controller, model, observer/state estimator, etc.) before running the experiment loop (to avoid excess computation time within the sampling loop)

Tip: it may be useful to write your own class for experiments to make your code modular. Be sure to pass in all necessary information to this experiment loop (i.e., don’t forget to pass in device information from which to read data)!

1. SAVE your data! Remember to put in statements to save all of the desired data you wish to collect. Tip: Before running long experiments, make sure this part of your code works!

Notes/Troubleshooting

Compatibility with different operating systems (OSs)

Most of the code provided is typically agnostic to different OSs (e.g., Python scripts can be run on any machine, USB devices and serial communication can be achieved on any OS). However, there may be some special modifications to code that is OS-specific (and even OS-version-specific). It is imperative that the user understands how to troubleshoot problems themselves, if using the setup on their own machine. Much of the procedure described above is tested and verified to be successful on macOS Mojave, Catalina, Big Sur\*, and Monterey\*.

\*There is a known bug with thermal camera use on these versions.

Additionally, there are known tricks/procedures on Ubuntu 20 (Focal Fossa) +:

* By default, Linux distributions lock out USB-connected device permissions for non-superusers. To fix this for each device that is connected to the laptop (Arduino, thermal camera, spectrometer, oscilloscope), you must add udev rules [[1](https://linuxconfig.org/tutorial-on-how-to-write-basic-udev-rules-in-linux), [2](http://manpages.ubuntu.com/manpages/xenial/man7/udev.7.html), [3](https://opensource.com/article/18/11/udev)].
  + For the spectrometer, after installing seabreeze, run  
    $seabreeze\_os\_setup
  + For the thermal camera, add the following to the /etc/udev/rules.d
    - A file named 99-pt1.rules with the following line
    - SUBSYSTEMS==”usb”, ATTRS{idVendor}==”1e4e”, ATTRS{idProduct}==”0100”, SYMLINK+=”pt1”, GROUP=”usb”, MODE=”666”
  + [For USBTMC devices](https://github.com/python-ivi/python-usbtmc/blob/master/README.md), add the following to the /etc/udev/rules.d
    - A file named usbtmc.rules with the following line
    - SUBSYSTEMS==”usb”, ACTION==”add”, ATTRS{idVendor}==”0957”, ATTRS{idProduct}==”1755”, GROUP=”usbtmc”, MODE=”0660”
    - NOTE: As mentioned in the instructions, you should find the device’s vendor and product info and change them accordingly in the line above. If you have multiple devices, add multiple lines to this file. To find the vendor and product info, connect the device and run lsusb
* The Arduino device name may typically be /dev/ttyACM0.
* Only one of the opencv packages needs to be installed. Opencv-python is recommended.
* maplotlib requires a GUI backend to work. You can install any; the following command installs one that works  
  $sudo apt-get install python3-tk
* s

Extra Tips

* The circuitry is FRAGILE; try to touch it as little as possible, and mitigate disturbances to it.
* Sometimes interference causes problems in plasma operation. Foil may potentially be used to insulate the circuits.
* Sometimes the signals are not sent to the system in time; try to increase the sampling time. Sometimes this occurs at the beginning of operation (e.g., the flow rate is not changed at the correct time) and can be fixed by flipping the switch on the mass flow controller.
* Sometimes the thermal camera freezes after too frequent use and/or between infrequent use. This can usually be fixed by resetting it by either unplugging it and plugging it back in or pressing the reset button on the device itself.
* Measurements from the oscilloscope must use a larger sampling time. If these are included in the data collection, make sure to increase the sampling time to roughly 0.3-0.5 seconds per measurement from the oscilloscope.
* Unplugging and plugging back in often fixes many problems.

Archive

Previous information

Operation

Ctrl+alt+t open the terminal (open two or three: read, control, MPC(python code))

Ctrl+r search

Press the button in Arduino to reset it(to default)

ctrl+c stop

$stty -F /dev/arduino\_m raw 38400 -hupcl (start the Arduino)

$cat /dev/arduino\_m (read the status-arduino)

$echo “w, 2”> /dev/arduino\_m

To access external python code (OES for example)

$cd xx

$workon sci3

$python spectroscopy.py –loop –live

On Mac

please put MCP4922.h and MCP4922.cpp to the same path, also change the <MCP4922.h> to "MCP4922.h" in the code.

**Initialize Arduino**

$stty -f /dev/cu.usbmodem1434401 raw 38400 -hupcl & cat /dev/cu.usbmodem1434401

**Send commands to ignite the plasma**

$echo "p,100" > /dev/cu.usbmodem141301

$echo "w,2" > /dev/cu.usbmodem141301

$echo "q,1.5" > /dev/cu.usbmodem141301

**Change directory**

$/Users/adbonzanini/Box\ Sync/APPJ\ control\ codes/APPJ\_Control\_Dogan/Software/

**Install libraries (only the first time):**

$pip install seabreeze

$pip install libusb1

$pip install pyserial

$pip install opencv-python

$pip install opencv-python-headless

$pip install pyvisa

$pip install python-usbtmc

$pip install pyusb

$brew install libusb

Using homebrew ^

**Also install** [**libuvc from groupgets**](https://github.com/groupgets/libuvc)

**Remember to run everything using Python3**