QP-CryoSwitch Controller User Guide

Preparation

GIT repository

For user convenience, a small repository containing all the necessary files can be found here.

Inside the repository, you will find a copy of the installation guide (this file), the controller datasheet, and finally a library containing python code to interface with the CryoSwitch Controller.

Python setup

The controller's library doesn't need any specific python distribution, but some extra python libraries must be installed.

The easiest way to do so is by using 'pip'. Inside the repository folder and using your base or virtual environment run:

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

Preparation Summary

- Clone the GIT repository.
- Install the necessary python dependencies.

Connections and ports information

The QP-CryoSwitch Controller is specifically designed for driving inductive loads at room or cryogenic temperatures.

The controller supports a maximum of 4 ports (version dependent) and 7 contacts or channels per port. The following figure shows the back plate and pinout of the controller.

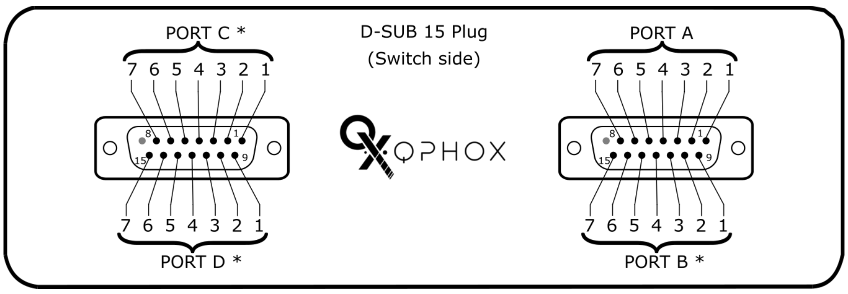

Figure 1: Controller output side

For example, if an SP6T cryogenic relay/actuator is used the connections would be as follows:

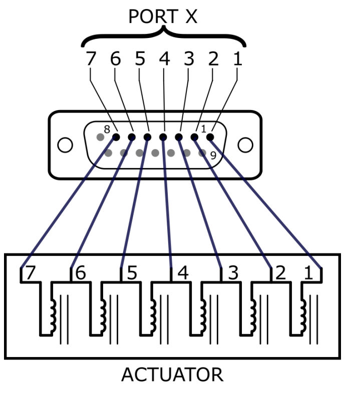

Figure 2: Application example

Python Library

The library contains the ‘CryoSwitch’ class which is a user-friendly way of interfacing between the QP-CryoSwitch Controller and a computer.

A basic implementation of the CryoSwitchController class can be done with the following functions:
- start()

- Input: None
- Default: None
- Enables the voltage rails, voltage converter and output channels

- set_output_voltage(*Vout*)

- Input: Desired output voltage (Vout)
- Default: 5V
- Sets the converter voltage to *Vout*. The output stage later utilizes the converter voltage to generate the positive/negative pulses.

- set_pulse_duration_ms(ms_duration)

- Input: Pulse width duration in milliseconds (ms_duration).
- Default: 15ms.
- Sets the output pulse (positive/negative) duration in milliseconds.

- connect(*port*, *contact*)

- Input: Corresponding port and contact to be connected. *Port*={A, B, C, D}, *contact*={1,...,6}
- Default: None.
- Connects the specified switch contact of the specified controller port (switch).
- Returns: Current waveform.

- disconnect(*port*, *contact*)

- Input: Corresponding port and contact to be disconnected. *Port*={A, B, C, D}, *contact*={1,...,6}
- Default: None.
- Disconnects the specified switch contact of the specified controller port (switch).
- Returns: Current waveform.

Typical application case

The CryoSwitch controller pulsing features use precise voltage and timing control to reduce heat dissipation, this way every user can optimize power dissipation while engaging the actuator.

Traditional relays use an electromagnetic field in order to actuate a switch. This way, normal mechanical switches (offering high isolation and low crosstalk) can be electrically actuated. Essentially, they are an electromagnet (coil) with a moving plate. Since the magnetic field is proportional to the coil current, a certain amount of current is needed in order to be able to engage the actuator. Temperature differences (mK or room temp.), connectors, and cabling resistance will influence the equivalent line resistance (R_{line}), hence limiting the current for a certain voltage pulse. **figure 3** shows the equivalent diagram of a typical application including the line resistance. It's important to note that the line resistance is not an actual resistor, but an equivalent for the entire system.

By adjusting the voltage and pulse width, it's possible to adjust the pulse current and compensate for the effective line resistance.

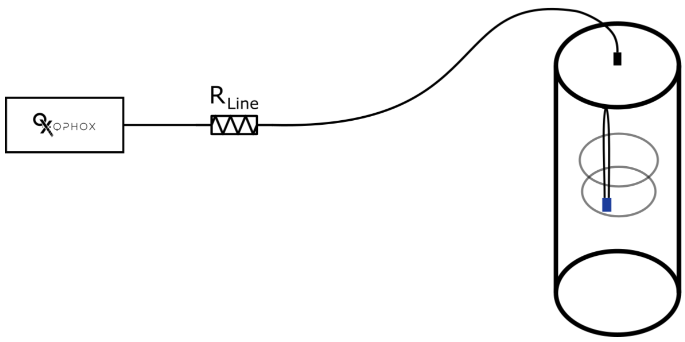

Figure 3: Equivalent circuit

Going back to the SDK, a typical application would look like this:

First, we initialize the controller software by declaring an instance of the Cryoswitch class:

switch = Cryoswitch()

Then we start the controller hardware by enabling the different voltage rails and setting the protections. By default, the output voltage is set to 5V and pulse width to 15ms. This can be done by running the **start()** command:

switch.start()

At this point the controller is running, the last step before switching the actuator is setting the appropriate output voltage. As we mentioned earlier, the appropriate pulse voltage varies from user to user, usually, we recommend a 5V pulse for cryogenic operation and a 20-25V for room temperature operation.

An easy way to check if the actuator was engaged is by looking at the current waveform. To do this, the plotting function must be enabled by setting

switch.plot = True

Figure 4 shows a typical current waveform at room temperature.

When the pulse starts, the coil is energized and the rising current exerts a force on the actuator. When the force on the actuator exceeds the retaining force, the actuator starts to move. The motion of the actuator (usually made from a ferromagnetic material) induces a back electromagnetic force in the coil, effectively reducing the current. As the movement continues, the current continues to drop until the actuator reaches the final position.

Once the actuator is at rest, the current starts to rise, saturating the coil until it reaches its final DC current (given by the equivalent line resistance and pulse voltage). This part of the switching is not strictly necessary, therefore it's called the excess pulse. Although one might try to reduce the pulse width to avoid the excess pulse and therefore further power dissipation, it's difficult to know beforehand what the pulse duration should be. As discussed earlier, this depends on the fridge wiring and temperature. Having said that, pulse widths of 15-20ms are usually enough to engage the actuator.

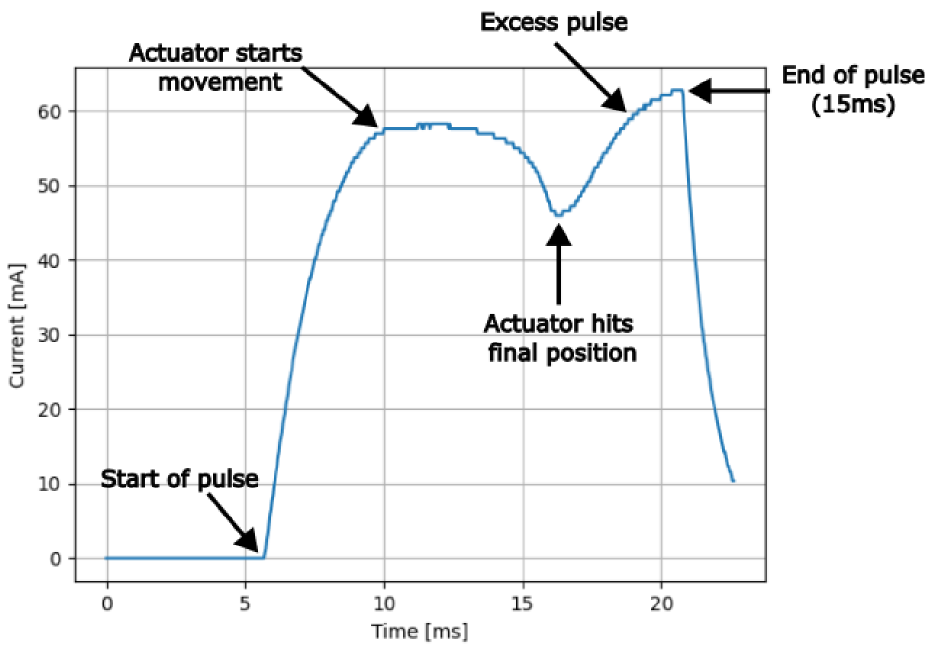

Figure 4: Normal operation waveform

How to recognize insufficient voltage settings? The following figure shows a fully engaged actuator waveform (blue) and a nonengaged waveform (orange). Since the current is directly proportional to the pulse voltage, reducing the pulse voltage will also reduce the pulse current and vice versa. The actuator needs a minimum amount of current to overcome the actuator retaining force (around 60mA). This is what the orange waveform is showing, in which the pulse didn't have enough current to overcome the actuator retaining force. This can be simply solved by increasing the pulse voltage.

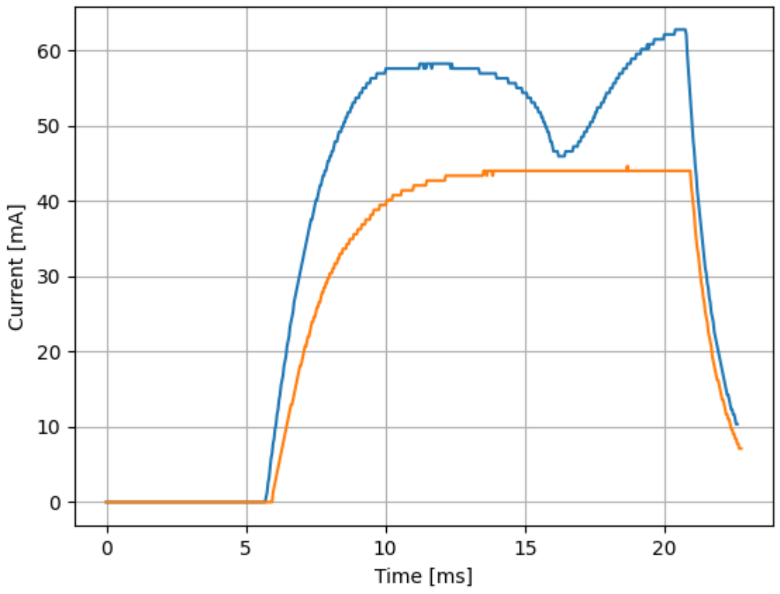

Figure 5: Low voltage waveform

Another common scenario is unconnected/broken cables, either inside the fridge or along the line. Internally, for every pulse, the controller uses some of the pulse current to bias the output stage. The bias current depends on the pulse voltage and is between 5-25mA. **Figure 6** shows a typical waveform when the outputs are not connected. Although the plot shows there is some current flowing (bias current), the user can easily distinguish this from an actual pulse and infer that the output is probably not properly connected.

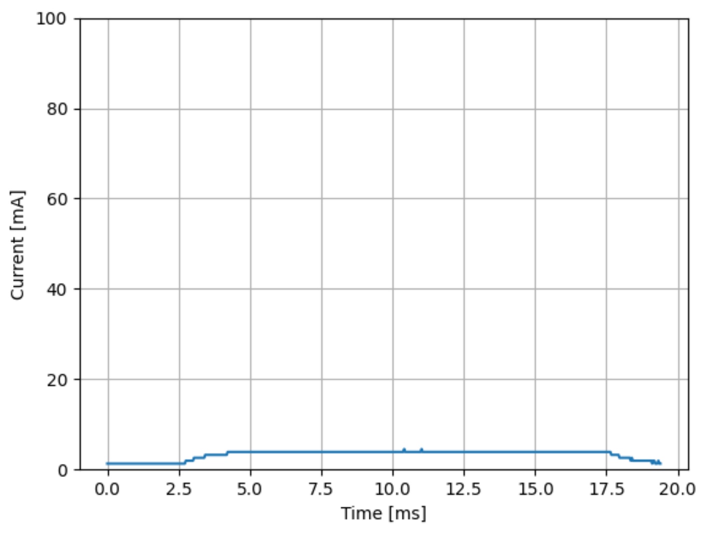

Figure 6: Unconnected waveform

In summary, the user should set the appropriate pulse voltage and width depending on their fridge configuration but a good starting point would be:

Case	Voltage	Width
Room temperature	20V	15ms
Cryogenic	5V	10ms

Recommended settings

The command sequence discussed above can be summarized as:

```
switch = Cryoswitch() ## -> CryoSwitch class declaration and USB connection
switch.start() ## -> Initialization of the internal hardware
switch.plot = True## -> Enable the current plotting function
switch.set_output_voltage(5) ## -> Set the output pulse voltage to 5V
```

Advanced functions

- enable_OCP()

- Input: None
- Default: None.
- Enables the overcurrent protection.

- set_OCP_mA(OCP_value)

- Input: Overcurrent protection trigger value (OCP_value).
- Default: 100mA.
- Sets the overcurrent protection to the specified value.

- enable_chopping()

- Input: None.
- Default: None.
- Enables the chopping function. When an overcurrent condition occurs, the controller will 'chop' the excess current instead of disabling the output. Please refer to the installation guide for further information.

- disable_chopping()

- Input: None.
- Default: None.
- Disables the chopping function. When an overcurrent condition occurs, the controller will disable the output voltage. Please refer to the installation guide for further information.

Overcurrent protection

The following image shows the current waveforms when the chopping function is enabled and disabled. The Blue trace shows the case when the chopping feature is disabled, as soon as the current exceeds the preset threshold the output is disabled.

On the other hand, the orange trace shows the case when the chopping feature is enabled. As soon as the current exceeds the preset threshold, the controller reduces the voltage in order to reduce the current also called “Chopping”. After the current drops, the controller will increase the voltage in order to increase the current. This cycle continues until the end of the pulse.

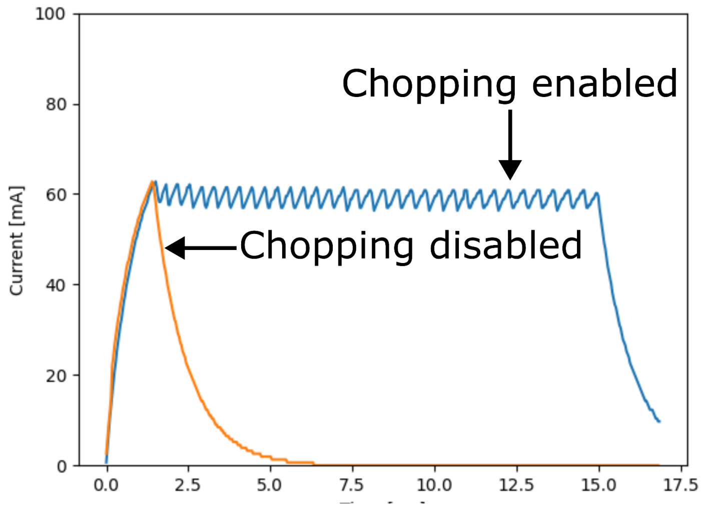

Figure 7: OCP modes of operation