Motor API

Revision 1 - Software v2023.2.2

irw

Summary

The following includes a description of the available methods used to interact with the motor controller via Python.

Contents

Arduino class

Instantiate a controller of the `Arduino` class with the following:

```
controller = Arduino(port = 'COM1') # Windows
controller = Arduino(port = '/dev/ttyACM0') # Linux
```

The available methods are listed below, including an example and description of each.

Note:

- Lobes are always specified in order of RU, RM, RL, LU, LL.
- Maneuvers may be specified as `'profile'`, `'inhale'`, or `'exhale'`

---

connect(self, port = None, baudrate = 19200)

```
controller.connect()
```

Connect to the Arduino at the specified port. If no port is specified, attempt to connect on the default port supplied with creating the `Arduino` object.

If successful, sets the `dev` instance variable from the serial connection.

---

close(self)

```
controller.close()
```

Closes the connection to the motor controller.

---

format(self, fragments)

```
controller.format(['SLS', '200', '11100'])
```

Formats a series of parameters to be read by the motor command protocol. The `fragments` argument can be a single string (e.g. `controller.format('SOI')`) or a list (e.g. `controller.format(['SDE', '5'])`).

Returns the formatted message but does not send to the controller.

---

query(self, message)

```
controller.query('?S')
```

Sends a message to the motor controller.

Returns the message response from the motor controller.

---

check_connection(self)

```
controller.check_connection()
```

Tests the serial connection to the motor controller.

Returns the controller response.

---

check_parameters(self)

```
controller.check_parameters()
```

Query the motor controller global parameters (breath count, maneuver delays, default values).

Returns the controller response.

---

print_parameters(self)

```
controller.print_parameters()
```

Aesthetic printing of default controller parameters.

---

check_lobe_delays(self)

```
controller.check_lobe_delays()
```

Query the delays stored in the current maneuver and in memory for each lobe/motor.

Returns the controller response.

---

print_lobe_delays(self)

```
controller.print_lobe_delays()
```

Aesthetic printing of default controller parameters and delays.

---

set_breath_count(self, n)

```
controller.set_breath_count(5)
```

Set the number of breaths to be run in a profile.

---

set_delay(self, maneuver, delay)

```
controller.set_delay('inhale', 0.1)
```

Set the delay for one of the global parameters. Maneuver options include:

- `'profile'`
- `'inhale'`
- `'exhale'`

Returns the controller response.

---

set_maneuver_order(self, maneuver)

```
controller.set_maneuver_order('inhale')
```

Set the first maneuver of a breathing profile. Maneuver options include:

- `'inhale'`
- `'exhale'`

Returns the controller response.

---

set_lobe_default(self, parameter, value, motors)

```
controller.set_lobe_default('delay', 1000, '11111')
```

Set the default value for lobe/motor parameters. Parameter can be one of `'steps'` or `'delay'`. Step values can be 0-700, and delay values can be a positive integer less than 65535. Delay values are related to processor cycles, not a strict time value.

---

update_lobe_delays(self, value = None, motors = None)

```
controller.update_lobe_delays()
```

Sets the delays for each lobe/motor to the default value for all steps.

Future method updates can interpret parameters in relation to the `SAC` command.

---

prepare_maneuver(self, maneuver, steps, motors)

```
controller.prepare_maneuver('inhale', 200, '10111')
```

Prepares an inhalation or exhalation maneuver for the defined step value and given motors.

Returns the controller response.

---

run_maneuver(self)

```
controller.run_maneuver()
```

Runs the last prepared maneuver for each lobe.

Returns the controller response.

---

run_profile_constant(self)

```
controller.run_profile_constant()
```

Runs a breathing profile based on the global settings and constant-delay breathing maneuvers.

Returns the controller response.

---

run_profile_variable(self)

```
controller.run_profile_variable()
```

Runs a breathing profile based on the global settings and variable-delay breathing maneuvers, where the variable delay values are retrieved from memory.
