01™ SuperModified – Python Manual

1. General Description

The SuperModified™ combo of miniature PCBs is an all-in one motor control solution. Incorporating a 15-bit magnetic absolute encoder, an 8-bit, 20MHz AVR ATMega328p microcontroller and a 5-Amp MosFet H-bridge at an astonishing outline of 16mm x 16mm x 13.2 mm it is ideal for space constrained applications. The overall dimensions allow for this motion control system to be installed inside a standard RC servo, transforming the device to a full functionality servo motor. The 01™ SuperModified is a highly cost effective solution, delivering closed loop PID control at 9.765 KHz, with advanced motion profiling capabilities and many other features.

In this document The Python Interface for the 01™ Supermodified controller is described.

ZERO ONE MECHATRONICS

01™ SuperModified

Miniature Controller for DC Motors

"*The robotic rebirth of the hobby servo"*

2. Prerequisites

Installed python 2.7 and packages serial, ttk*, Tkinter*, matplotlib*

* for visualization purposes only.

There are two ways to utilize the code given. The direct call of the script SMSLibrary.py and the import of the above script for usage in another python script. The first is more limited while the second more powerful.

Also as an interface you can apply a USB2RS485 adaptor.

3. Use

3.1 Run directly the python script

First open up a terminal/command prompt and head to the folder where the script is.
After acquiring root privilege (only for Linux Users / we open a serial port, that's why) run the script SMSLibrary.py directly with the following command:

$ [sudo] python SMSLibrary.py port_number command [Node Id] [value]

The commands with their arguments you can specify are:
start [Node Id]
stop [Node Id]
reset [Node Id]
startall*
stopall*
resetall*
getvel [Node Id]
getacc [Node Id]
getpos [Node Id]
move [Node Id] [value]
setvel [Node Id] [value]
setacc [Node Id] [value]
test [Node Id]

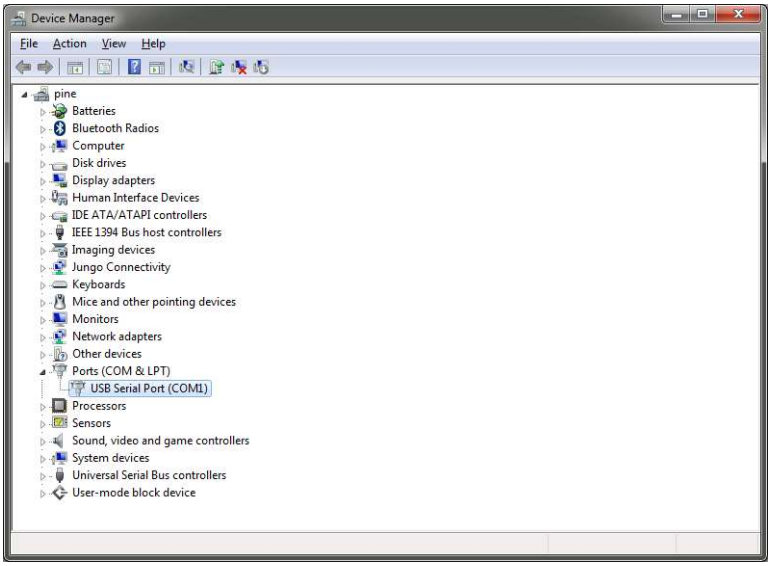
* It starts/stops/resets all the nodes from 0 to 10 by default.

Examples:

1) For Linux Users: (controller in ttyUSB0 and motor ID = 5)
$ sudo python SMSLibrary.py 0 start 5

2) For Windows Users: (controller in COM3 and motor ID = 4)
python SMSLibrary.py 3 getpos 4

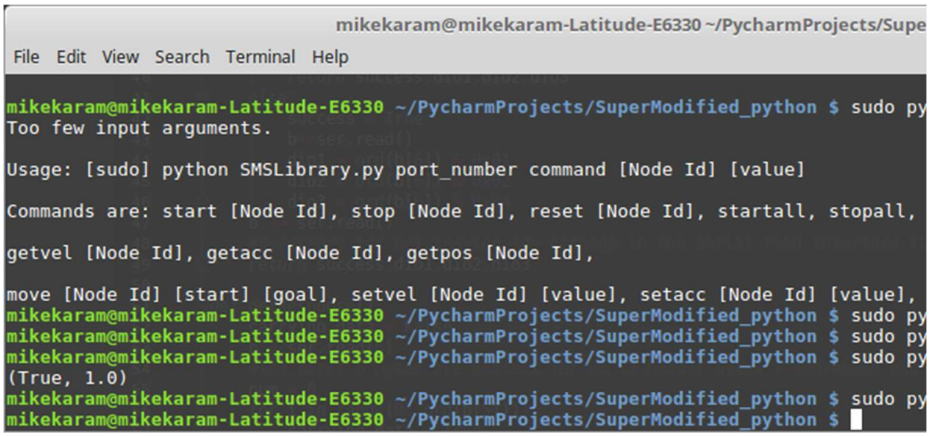
Note: If you don't know how to find which port your controller is attached to:
for Windows Users: To identify which COM port you are using in windows you can check the Device Manager.
The USB to RS485 / UART adapter should be listed as shown below:



In the advanced settings you can even change the COM port number to another one.

➔ **for Linux Users: use the command lsusb in a terminal and check the port to which the FTDI driver is connected.**

Screenshot Example:



2.2 Import the module and Usage

In the folder/directory you have extracted the zip provided, create a new python script with your preferred editor. The module SMSLibrary can be easily imported with a line like:

import SMSLibrary as sms *or*
from SMSLibrary import * (if you don't want to type the class name every time)

The first function of the module that must be called before anything else is the init(port_number) function, where port_number is the port number the controller uses.

The last function that must be used in order to close the serial port is: shut_down(port_number)

The 01TM Python Library contains all the functions needed to interface to the Supermodified controller. Each function is actually an implementation of a command of the 01TM protocol. The command set of the Supermodified controller is presented in detail in section 11 of the Supermodified datasheet.

There are generally 3 types of commands.

Set commands
All set commands are implemented as Python functions that accept 1 or more inputs and return one output.

The first (or only) input is always the motor ID on the bus. The motor ID can only be changed by using the SuperModified Commander (01TM stand-alone application) and must be unique for every controller on the bus. All Supermodified controllers are shipped with a default motor ID=4.

If the command does not transmit any additional information to the Supermodified controller then the function has only one input and only one output. The input is the motor ID and the output is whether the call was successful or not.

eg.
```
def start(mid):
    ...
    return not getResponse()[0]
```

The above is the start command and starts the control to the motor. It must be executed before any command that attempts to move the motor.

If the command has one or more data arguments that must be passed to the controller then the Python function is like below:

eg.
```
def moveWithVelocity(mid, val):
    ...
    return not getResponse()[0]
```

where mid is motor Id. val is velocity Ticks per second.

The above is the Move With Velocity Command that tells the motor to move at `val` velocity. Note that the `mid` input is always first.

Or another example:

```
def setDigitalOutputs(mid, dio1, dio2, dio3):
        ...
  return not getResponse()[0]
```

The above command sets or resets the controller's digital outputs according to inputs `dio1,dio2,dio3` and according to Digital IO configuration.

Note: The set command Functions return not getResponse()[0] for a reason explained in the comments of the code.

Get commands

All get commands are implemented as Python functions that accept 1 input and return 1 or more outputs.

The input is always the Motor ID and the first output is always the success or failure of the command. The rest of the outputs depend on the command.

For example:

```
def getPosition(mid):
            ...
    return getResponse()
```

This Python function can be used to read the position of the motor in encoder ticks (32768 ticks per revolution). The position will be stored in getPosition(mid)[1]. The success of the command will be in `getPosition(mid)[0]`.The motor that is communicated is the one with `mid`.

Accordingly commands that have more outputs,

eg.
```
def getDigitalIOConfiguration(mid, dio):
            ...
    return getDigitalIOResponse()
```

This Python function can be used to read the digital IO configuration. The bits dio1, dio2, dio3 are stored in getDigitalIOConfiguration `(mid, dio)`[1-3] respectively. The success of the command will be in getDigitalIOConfiguration `(mid, dio)`[0]. The motor that is communicated is the one with `mid`.

Broadcast commands

All broadcast commands are implemented as Python function with only one output which is always True.

eg.
```
def broadcastStart():
      ...
    return True
```

The above command starts all motors on the bus.

There is also an example script, in the zip provided, which shows the usage of the above commands.

Error handling and parsing will be implemented in a future release. Stay tuned!

4. Contents