01™ SuperModified – Matlab API 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 Matlab Interface for the 01™ Supermodified controller is described.

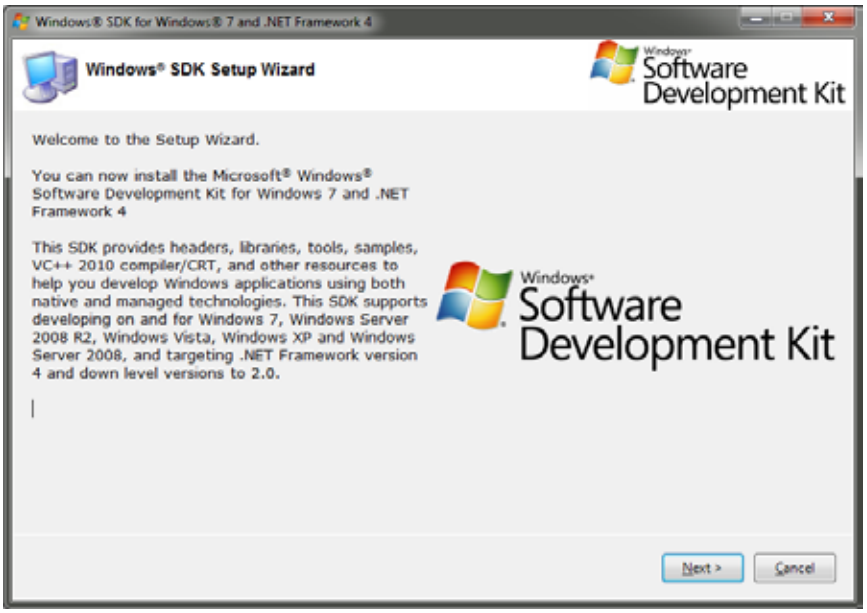
2. Prerequisites

- **Windows XP or later.**
- **Matlab 32bit or 64bit installed.**

3. Installation

A) Unzip the contents of the "ZeroOne Matlab API.zip" contents to a location of your preference. Please note that this location will then need to be added to Matlab's working directory.

B) Install the windows SDK. The web installer is included in the archive containing the matlab functions and the dll and is named: " winsdk_web.exe".



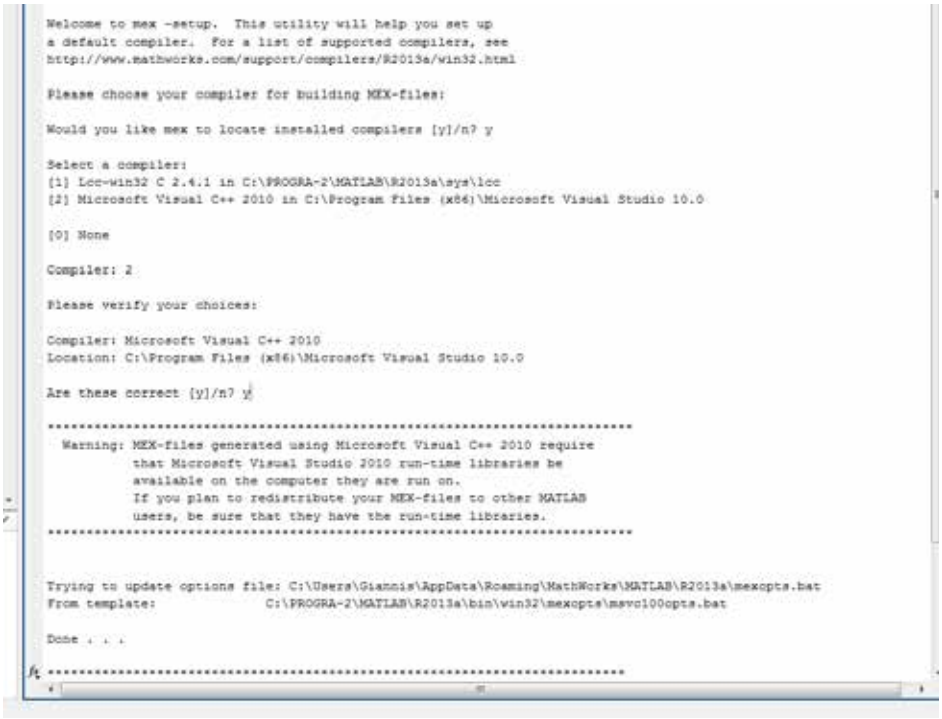
C) Open Matlab and in the command window type: "mex –setup" (without the quotes) . You should see with the following:

ZERO ONE MECHATRONICS

01™ SuperModified

Miniature Controller for DC Motors

"*The robotic rebirth of the hobby servo"*



When asked which compiler to use, select Microsoft Visual C++ 2010.

Note: The above procedure is needed only once.

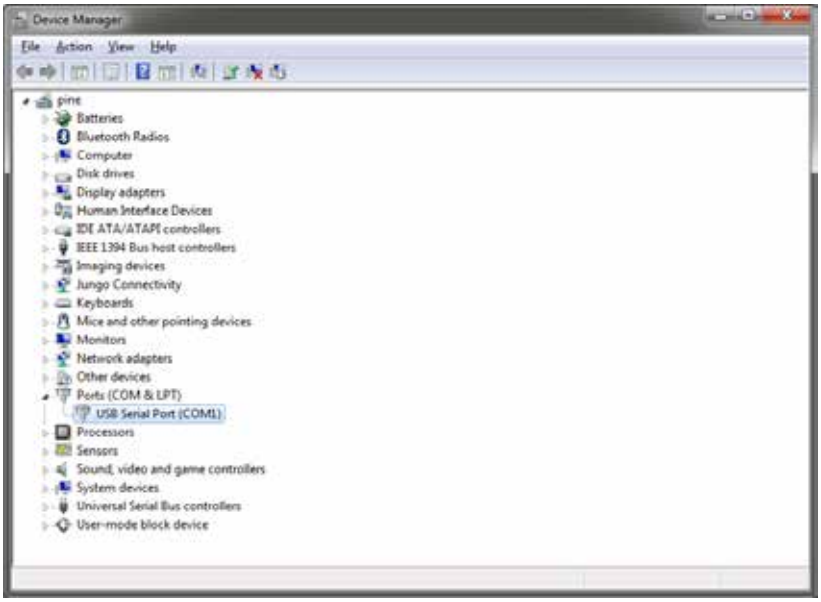
4. Usage

First make the directory that contains the dll and Matlab functions, Matlab's active directory. Please refer to Matlab help in order to do this.

4.1. Initialization

The first function that must be called before anything else is the command: zoSmsInit('comX'), where X is the number of the COM port you are using.

To identify which COM port you are using in windows you can check the Device Manager. The USB to RS485 / UART conveter should be listed as shown below:



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

You can try the command at the command window:

The last command that must be used in order to clear all variables and unload the library is : zoSmsShutDown()

4.2. Matlab Functions

The 01™ Matlab Library contains all the functions needed to interface to the Supermodified controller. Each function is actually an implementation of a command of the 01™ 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 Matlab 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 (01™ 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. `function [success] = zoSmsCmdStart(motorId)`

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 Matlab function is like below:

eg. `function [success] = zoSmsCmdMoveWithVelocity(motorId, velocityTicksPerSecond)`

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

Or another example:
`function [success] = zoSmsCmdSetDigitalOut(motorId,do1,do2,do3)`

The above command sets or resets the controller's digital outputs according to inputs `do1,do2,do3` and according to Digital IO configuration.

Get commands

All get commands are implemented as Matlab 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:

```
function [success,positionTicks] = zoSmsCmdGetPosition(motorId)
```

This Matlab function can be used to read the position of the motor in encoder ticks (32768 ticks per revolution). The position will be stored in `positionTicks.` The success of the command will be in `success.` The motor that is communicated is the one with `motorId.`

Accordingly commands that have more outputs, eg.

```
function [success,dio1,dio2,dio3] = zoSmsCmdGetDIOConfig(motorId)
```

Broadcast commands

All broadcast commands are implemented as Matlab function with only one output that indicates success or not.

eg. `function [success] = zoSmsCmdGlobalStart()`

The above command starts all motors on the bus.

4.3. Errors

During execution of a Matlab function an error might occur. All errors are stored in global variables and can be accessed by the use of one of the two following functions:

```
function zoSmsErrorsShow()
```

The above function displays the errors that occurred during the last executed command along with descriptions.

eg.

```
>> zoSmsErrorsShow()

 Errors that manifested during last operation
-----------------------------------------------
Transmit Parse Error: -1  : no input command error
Receive Parse Error: -1  : no output command error
Transmit System Error: -1  : (please refer to MSDN getLastError()
reference)
Receive System Error: -1  : (please refer to MSDN getLastError()
reference)
Transmit HW Error: -1  : null
Receive HW Error: -1  : null
Supermodified Controller Internal Errors: null  (please refer to
datasheet error reference)
```

NOTE: If there is a controller related error it must be explicitly cleared by the zoSmsCmdResetErrors function or it will never stop to appear and the controller will not

accept any other commands.
Another function that can be used to read errors is the private function:

```
function [txParseErr, txParseErrDesc, rxParseErr, rxParseErrDesc,
txSysError, rxSysError, txHwErr, txHwErrDesc, rxHwErr,
rxHwErrDesc, smsErr] = zoSmsErrorsGet()
```

which outputs all possible error codes and matching descriptions as strings.

4.4. Error codes reference

The following errors can appear during operation:

Parser Input Errors

PARSER INPUT ERRORS	
1001	empty data!
1002	corrupted bytes, %d character(s) not an even number, discarding pair consistency for hex data!
1003	non hex data, the %d-%d character pair (%c%c) is invalid!
1004	insufficient bytes, they are %d, less than minimum expected (%d) according to the protocol!
1005	overflown bytes, they are more (%d) than maximum expected (%d) according to the protocol!
1006	invalid first byte of header, it is %d (int) %c%c (hex) instead of %d (int) %c%c (hex)!
1007	invalid second byte of header, it is %d (int) %c%c (hex) instead of %d (int) %c%c (hex)!
1008	machine id %d (int) %c%c (hex) is not a valid target to send the command!
1009	machine id %d (int) %c%c (hex) is not a valid id to send the command since it belongs at math integer space [0, 3], values which are system reserved!
1010	invalid host id %d (int) %c%c (hex), it is impossible to set a command with a host id which is not equal to 1 (int) 01 (hex)!
1011	command id %d (int) %c%c (hex) is not a valid command id according to the protocol!
1012	command id %d (int) %c%c (hex) is not a broadcast command in order to use 0 (int) 00 (hex) listener id which is translated to all system listeners!
1013	overflown bytes (%d), this command id does not need any more data bytes than the protocol minimum (%d) cause no byte count data are supposed to be interfered!
1014	invalid transmit count of bytes, byte count is %d instead of %d according to the given command id!
1015	byte count data should be zero instead of %d for the given command id!
1016	overflown amount of byte count data (%d), byte count is greater than the maximum (%d) expected according to the protocol!
1017	insufficient amount of byte data given (%d) according to the byte count value (%d), crc seems to be missing!

1018	insufficient amount of byte data given (%d) according to the byte count value (%d), crc seem to be missing and at least one byte out of byte count data!
1019	overflown amount of byte data given (%d) according to the byte count value (%d), data should be %d bytes!
1020	%d (int) %c%c (hex) crc is invalid, it should be %d (int) %c%c (hex)!
3001	error reaction (special) command id %d (int) %c%c (hex) can not be set with at least one of the error reaction bytes out of math integer space [0, 2], first illegal byte grabbed is byte %d!

PARSER OUTPUT ERRORS

2001	empty data!
2002	insufficient bytes, they are %d, less than minimum expected (%d) according to the protocol!
2003	overflown bytes, they are more (%d) than maximum expected (%d) according to the protocol, since there is no error bytecount!
2004	invalid first byte of header, it is %d (int) %c%c (hex) instead of %d (int) %c%c (hex)!
2005	invalid second byte of header, it is %d (int) %c%c (hex) instead of %d (int) %c%c (hex)!
2006	motor answer should happen to primal host %d (int) %c%c (hex) but it happened to %d (int) %c%c (hex) which is invalid!
2007	motor answer id is %d (int) %c%c (hex) which is invalid, it should be %d (int) %c%c (hex)!
2008	command id %d (int) %c%c (hex) is not a valid error response id %d (int) %c%c (hex) or the same as input command id %d (int) %c%c (hex) expected as answer validation!
2009	overflown bytes (%d), this command id does not need any more data bytes than the protocol minimum (%d) cause no byte count data are supposed to be interfered!
2010	invalid receive count of bytes, byte count is %d instead of %d according to the taken command id!
2011	byte count data should be zero instead of %d for the taken command id!
2012	byte count error response must be at least one and not zero!
2013	overflown amount of byte count data (%d), byte count is greater than the maximum (%d) expected according to the protocol, since there is no error bytecount!
2014	insufficient amount of byte data taken (%d) according to the byte count value (%d), crc seems to be missing!
2015	insufficient amount of byte data taken (%d) according to the byte count value (%d), crc seem to be missing and at least one byte out of byte count data!
2016	overflown amount of byte data taken (%d) according to the byte count value (%d), data should be %d bytes!

2017	error code %d (int) %c%c (hex) is not a valid error code according to the protocol!
2018	%d (int) %c%c (hex) crc is invalid, it should be %d (int) %c%c (hex)!
3002	get analog inputs (special) command id %d (int) %c%c (hex) does not return byte count data in pairs, byte count %d is not an even number!
3003	error reaction (special) command id %d (int) %c%c (hex) returns at least one of the error reaction bytes out of math integer space [0, 2], first illegal byte grabbed is byte %d!

USER ERRORS

4002	invalid value integer array, it seems that at least one invalid character is interfered
4004	overflown unsigned value given
5002	invalid value integer array, it seems that at least one invalid character is interfered
5004	overflown (signed) positive value given
5005	underflown (signed) negative value given
6001	sign missmatch underrun, possible conflict between unsigned and negative symmetrics cause of simultaneous call
6002	invalid value for set error reaction command, it has insufficient length
6003	invalid value for set error reaction command, it seems that at least one invalid character is interfered

HARDWARE ERRORS

8001	cannot connect to COM port
8002	cannot set communication timeout parameters for COM port
8003	cannot get communication timeout parameters for COM port
8004	cannot set other communication parameters for COM port
9001	cannot disconnect COM port
10001	cannot write to COM port
10002	cannot flush COM port
11001	cannot read from COM port

5. Contents

Disclaimer: The information in this document is provided in connection with 01 Mechatronics products. No license, express or implied, by estoppel or otherwise, to any intellectual property right is granted by this document or in connection with the sale of 01Mechatronics products.

EXCEPT AS SET FORTH IN 01 MECHATRONICS TERMS AND CONDITIONS OF SALE LOCATED ON 01 MECHATRONICS WEB SITE, 01 MECHATRONICS ASSUMES NO LIABILITY WHATSOEVER AND DISCLAIMS ANY EXPRESS, IMPLIED OR STATUTORY WARRANTY RELATING TO ITS PRODUCTS INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL 01 MECHATRONICS BE LIABLE FOR ANY DIRECT, INDIRECT, CONSEQUENTIAL, PUNITIVE, SPECIAL OR INCIDENTAL DAMAGES (INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS OF PROFITS, BUSINESS INTERRUPTION, OR LOSS OF INFORMATION) ARISING OUT OF THE USE OR INABILITY TO USE THIS DOCUMENT, EVEN IF 01 MECHATRONICS HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

01 Mechatronics makes no representations or warranties with respect to the accuracy or completeness of the contents of this document and reserves the right to make changes to specifications and product descriptions at any time without notice. 01 Mechatronics does not make any commitment to update the information contained herein. Unless specifically provided otherwise, 01 Mechatronics products are not suitable for, and shall not be used in, automotive applications. 01 Mechatronics's products are not intended, authorized, or warranted for use as components in applications intended to support or sustain life.

© 2014 01 Mechatronics Corporation. All rights reserved. 01™ MECHATRONICS ®

01™ is a registerd trademark of 01 Mechatronics. Other terms and product names may be trademarks of others.