ORCA™ Motors with MATLAB

User Guide 231031

Version 1.3

CONTENTS

REVISION HISTORY

Version	Date	Author	Reason
1.0	August, 2023	rm	Initial Release
1.1	October, 2023	rm	Replace Modbus object with serial object, simulink and haptic examples. Actuator class
1.2	October, 2025	jg	Updated baud rate information.
1.3	March, 2026	rm	branding

Overview

As part of the Matlab source code package, the “Actuator” class is provided in the “Actuator.m” file. This object abstracts serial communication to the motor to make it easy to add ORCA motor interfacing into new and existing projects. The Actuator class can easily be expanded upon to include additional functionality as all motor features are available through reading and writing to registers.

The Matlab source code package, which contains the “Actuator” class as well as multiple examples is available at irisdynamics.com/downloads.

The Actuator Class makes use of MATLAB’s serialport object to establish and maintain a serial connection to the ORCA motor. More information about the MATLAB object can be found here https://www.mathworks.com/help/matlab/ref/serialport.html.

A serial to USB converter, like the ORCA-USB, must be used to create a virtual COM port for the ORCA motor’s RS422 serial interface.

The serial port will be closed when an object is destroyed. If additional Actuator objects are made with the same COM port, they will not be able to connect until the other is destroyed. Each Actuator class method will handle the formatting, sending, receiving and parsing of Modbus messages to the ORCA Motor.

Reading Data from the Motor

In this step a 'modbus' object is created and configured. A simple read command is implemented to read the ORCA motor's error register. The loop is exited when an error is read from the motor.
When the example is run, a prompt will appear to enter the COM port number of the RS422 serial interface of the ORCA motor.
Note: COM port number can be found in Device Manager.

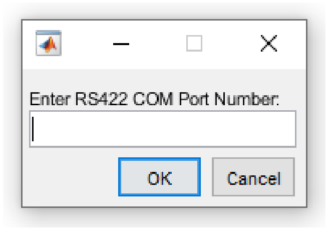


Figure 1: Prompt to enter motor's RS422 COM port number

read_motor_data.m

```
clear;

portnum = inputdlg('Enter RS422 COM Port Number:'); % Enter the com port
number used by the RS422 cable

port = strcat("COM", portnum);
orca = Actuator(port, 19200); %COM port of RS422 serial interface of ORCA
Motor and default Modbus baud rate

ERROR_0_address = 432% register contains the active errors of the motor
ERROR_value = 0;
% Loop while reading the error register and break if errors are encountered
while ERROR_0_value == 0
      ERROR_0_value = orca.read_register(ERROR_0_address, 1); % Read the value
      from the ERROR_0 register
end

fprintf("Program Exited due to Motor Error: %d", ERROR_0_value);
```

Plotting Motor Position

The motor's position is stored in micrometers (μm) as an int32 across 2 registers in 'little-endian' format. The parsing of this is all handled as part of the Actuator class methods. ORCA motors have custom function codes that allow for multiple pieces of relevant motor information to be returned in a single frame. While commanding a force, position or other command, the motor can return position, force, voltage, power, and error information in a single message allowing for increased data update frequency. The information returned is stored in the object's parameters.
Additionally in this step a 'walking' plot is added to plot the motor's position. As the motor's shaft is moved, the plot will update.

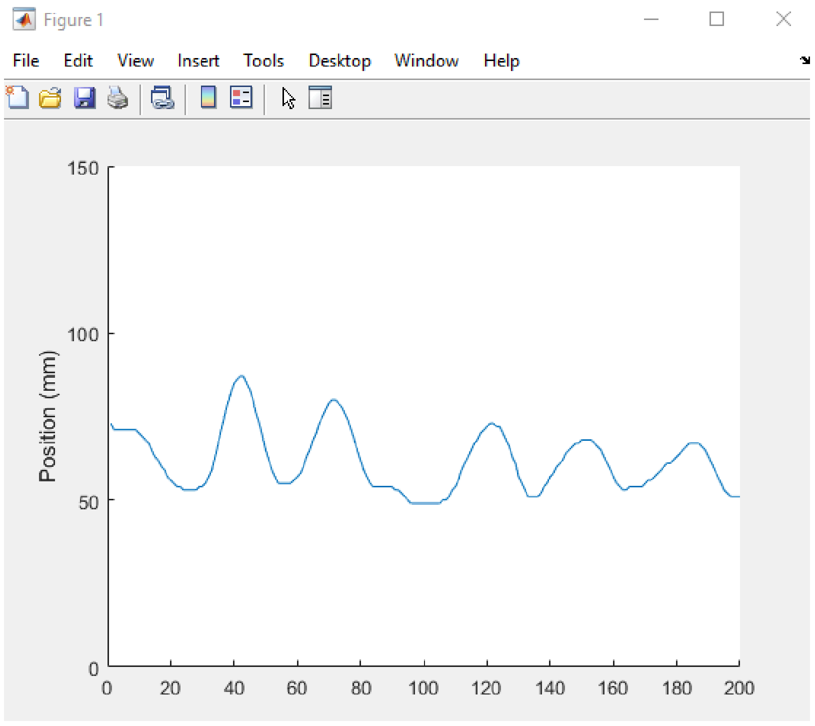


Figure 2: Motor position plot

plot_motor_position.m

```
clear;

portnum = inputdlg('Enter RS422 COM Port Number:'); % Enter the COM port of
the Motor’s RS422 serial interface.
port = strcat("COM", portnum);
orca = Actuator(port, 19200); %COM port of RS422 Modbus channel to ORCA Motor
and default Modbus baud rate

% a plot to monitor the position of the shaft
position_plot = figure;
hold on
ylim([0 150]);
num_samples = 200;
x = 1:num_samples;
y = zeros(size(x));
p = plot (x,y);
iteration = 1;
ylabel('Position (mm)');


% Loop while commanding zero force to motor and plotting position
while true
      orca.command_orca_force(0); %command 0 newtons to the motor
      % Create a 'walking' plot of the data
      iteration = iteration + 1;
      if iteration <= num_samples
            y(iteration) = orca.position/1000.;
      else
            y = circshift(y, -1);
            y(end) = orca.position/1000.;
      end
      set(p, 'XData',x, 'YData', y)
      drawnow
end
```

Configuring Position Targets and Triggering with Kinematic Mode

Position profiles can be configured by setting various kinematic motions and triggering them. This will allow for the position targets to be commanded locally on the motor and give a smooth motor response no matter the frame rate or consistency of commands coming from Matlab. A series of motions can be configured, chained together as desired and then triggered through (up to 32) or a single motion can be updated as needed.
In this case, we can also make note of the current motion being run on the plot, and trigger a new motion each time one is finished.

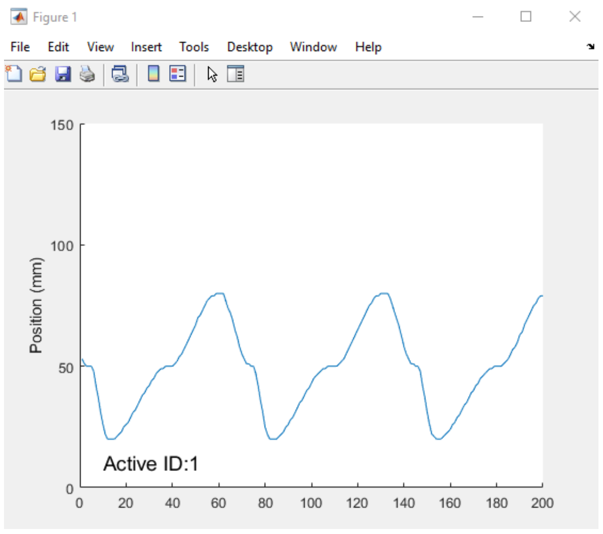


Figure 3: Motor position plot with currently running motion ID

trigger_configure_motion.m

```
clear;
portnum = inputdlg('Enter RS422 COM Port Number:'); % Enter the COM port of
the Motor’s RS422 serial interface.
port = strcat("COM", portnum);
orca = Actuator(port, 19200); %COM port of RS422 Modbus channel to ORCA Series
Motor and default Modbus baud rate

% a plot to monitor the position of the shaft
position_plot = figure;
hold on
h = text(10,10,'Active ID','FontSize',14);
ylim([0 150]);
num_samples = 200;
x = 1:num_samples;
y = zeros(size(x));
p = plot (x,y);
iteration = 1;
```

```
ylabel('Position (mm)');
%Memory Map addresses of registers to be read
KIN_STATUS_address = 319;
MODE_OF_OPERATION_address = 317;

%Put the motor to sleep to start in a known state and clear any active
%errors
orca.op_mode = 0;
while orca.op_mode ~= orca.SleepMode
      orca.change_mode(orca.SleepMode);
      orca.op_mode = orca.read_register(MODE_OF_OPERATION_address, 1);
end

%configure a set of kinematic motions with the final motion automatically
%looping to the first
orca.configure_motion(0, 50000, 1000, 200, 1, 0, 0);
orca.configure_motion(1, 80000, 800, 20, 2, 0, 0);
orca.configure_motion(2, 50000, 500, 0, 3, 0, 0);
orca.configure_motion(3, 20000, 300, 0, 0, 0, 1);

%Ensure the motor gets into kinematic mode. This will immediately trigger
%the home motion.
while orca.op_mode ~= orca.KinematicMode
      orca.change_mode(orca.KinematicMode);
      orca.op_mode = orca.read_register(MODE_OF_OPERATION_address, 1);
end

% Loop while triggering kinematic motions, plotting position and displaying
% currently active motion id
while true
      %kinematic status gives the running bit and the active ID
      kin_status = orca.read_stream(KIN_STATUS_address, 1); %read the current
      state of the kinematic controller
      active_id = bitand(kin_status, 0x7FFF);
      kin_running = bitand(kin_status, 0x8000);

      %When there is no active trigger next
      if(kin_running ==0)
            orca.kinematic_trigger(active_id + 1);
      end
      % Create a 'walking' plot of the data
      iteration = iteration + 1;
      if iteration <= num_samples
            y(iteration) = orca.position/1000.;
      else
            y = circshift(y, -1);
            y(end) = orca.position/1000.;
      end
      set(p, 'XData',x, 'YData', y)
      drawnow
      h.String = sprintf('Active ID:%d', active_id); %Update text on the
      figure with active motion IDs
end
```

Streaming Position Targets in Position Mode

Alternatively if a specific set of positions is desired, these can be streamed directly to the motor. The position target in this example is being generated by a sine wave signal. Alternatively the target could come from a file or be based on information returned from the motor.

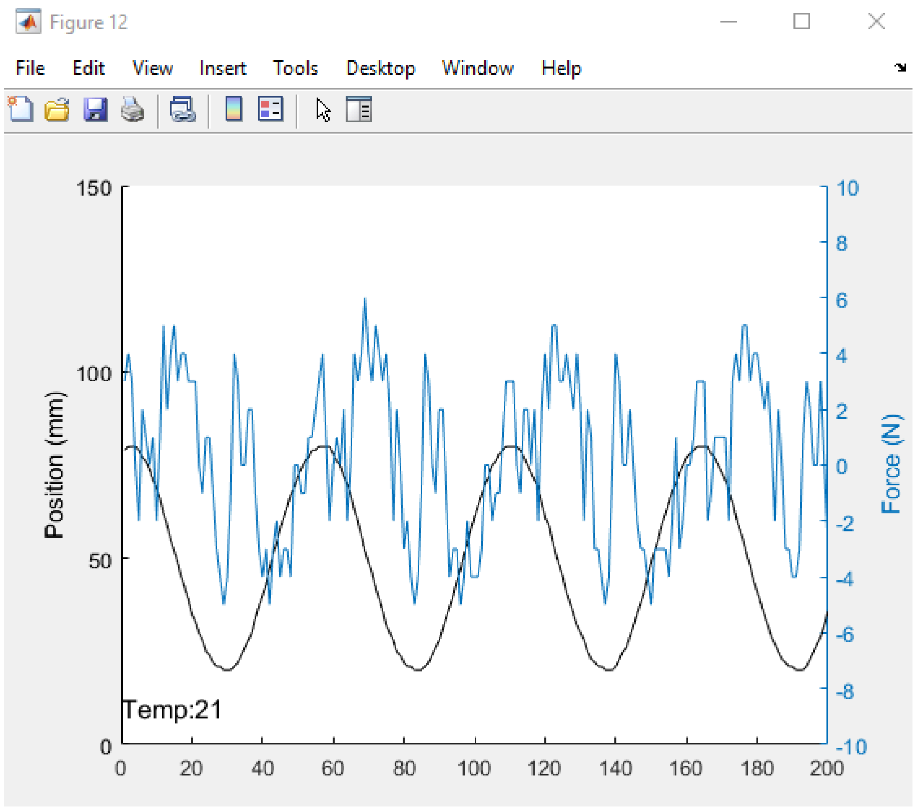


Figure 4: Motor position, force and temperature plot

stream_position_targets.m

```
clear;

portnum = inputdlg('Enter RS422 COM Port Number:'); %Enter the COM port of the
Motor’s RS422 serial interface.
port = strcat("COM", portnum);
orca = Actuator(port, 19200); %COM port of RS422 Modbus channel to ORCA Motor
and default Modbus baud rate

% a plot to monitor the position of the shaft
% also displays motor temperature and force values
position_plot = figure;
hold on
h = text(0,10,'Temp','FontSize',12);
```

```
ylim([0 150]);
num_samples = 200;
x = 1:num_samples;
y = zeros(size(x));
r = zeros(size(x));
iteration = 1;
yyaxis left
p = plot (x,y);
ylim([0 150]);
ylabel('Position (mm)');
%create a second axis for Force display
yyaxis right
p1 = plot (x,r);
ylim([-10 10]);
ylabel('Force (N)');

% Memory Map register address
MODE_OF_OPERATION_address = 317;
orca.op_mode = 0;

%Put the motor to sleep to start in a known state and clear any active
%errors
while orca.op_mode ~= orca.SleepMode
      orca.change_mode(orca.SleepMode);
      orca.op_mode = orca.read_register(MODE_OF_OPERATION_address, 1);
end
%Sine wave parameters
tic;
freq = 0.5;
Amplitude = 30000;
Offset = 50000;
% Loop while streaming a sine wave position target to the motor
%plot motor's position, and force output, display temperature
while true
      t = toc;
      orca.command_orca_position(Amplitude * sin(t*2*pi*freq)+ Offset);
      % Create a 'walking' plot of the data
      iteration = iteration + 1;
      if iteration <= num_samples
            y(iteration) = orca.position/1000.;
            r(iteration) = orca.force/1000.;
      else
            y = circshift(y, -1);
            r = circshift(r, -1);
            y(end) = orca.position/1000.;
            r(end) = orca.force/1000.;
      end
      set(p, 'XData',x, 'YData', y)
      set(p1, 'XData',x, 'YData', r)
      drawnow
      h.String = sprintf('Temp:%d', orca.temperature);
end
```

Using Haptic Effects

This example shows how haptic effects can be configured and enabled. In this case there is a certain zone of the shaft where an additional vibration effect is layered on top of a spring that is throughout the motion of the shaft.

using_haptic_effects.m

```
clear;

portnum = inputdlg('Enter RS422 COM Port Number:'); % Enter the COM port of
the Motor’s RS422 serial interface.
port = strcat("COM", portnum);
orca = Actuator(port, 19200); %COM port of RS422 Modbus channel to ORCA Motor
and default Modbus baud rate

%Memory Map addresses of registers
MODE_OF_OPERATION_address = 317;
S0_GAIN_N_MM_address = 644;
O0_GAIN_N_address = 664;
HAPTIC_STATUS_address = 641;

%% configure spring effect
gain = 150;
center = typecast(int32(65000), 'uint16');
coupling = 0;
deadzone = 0;
force_sat = 0;
configuration = [gain center coupling deadzone force_sat];
orca.write_multi_registers(S0_GAIN_N_MM_address, 6, configuration);

%% configure oscillation effect
gain = 10; % Newtons
type = 1; %sine wave
freq = 1000; %dHz
duty = 0;
configuration = [gain type freq duty];
orca.write_multi_registers(O0_GAIN_N_address, 4, configuration);
orca.enable_haptic_effect (orca.Spring0);

while read_register(orca, MODE_OF_OPERATION_address, 1) ~= orca.SleepMode
      change_mode(orca, orca.SleepMode); %put the motor to sleep, this will
      also clear errors
end

while read_register(orca, MODE_OF_OPERATION_address, 1) ~= orca.HapticMode
      change_mode(orca, orca.HapticMode); %put the motor into haptic mode to
      start effects
end

%stream a sine wave position target to the motor
%last_position = 0;
```

```
quiet_zone = 50000;
haptic_status = 0;
while true
      if ((orca.position <= quiet_zone) && (haptic_status ~= orca.Spring0))
            orca.enable_haptic_effect (orca.Spring0 );
            elseif ((orca.position > quiet_zone) && (haptic_status ~=
            (orca.Spring0 + orca.Osc0)))
            orca.enable_haptic_effect (orca.Spring0 + orca.Osc0);
      end
      %this is to maintain a stream of data coming from the motor ie,
      position, force, temp etc.
      haptic_status = orca.read_stream(HAPTIC_STATUS_address, 1);
end
```

Note that writing multiple registers is being used to configure the effect. The configuration of a haptic effect can be added as an Actuator class method for easier use.

In Actuator.m

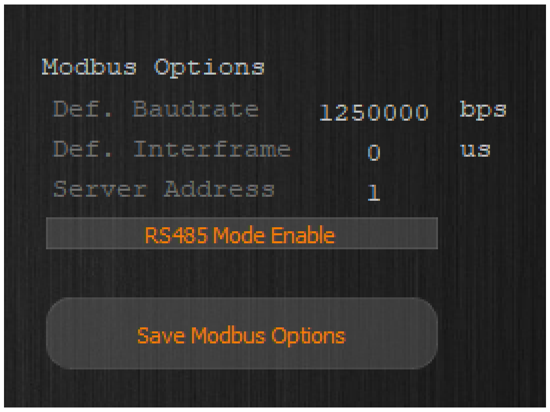
```
%% Configure Haptic Effect Spring A
function configure_springA(obj, gain, center, coupling, deadzone, force_sat)
    center = typecast(int32(center), 'uint16');
    configuration = [gain center coupling deadzone force_sat];
    write_multi_registers(644, 6, configuration); %write to spring A
configurations
end
```

In script

```
orca.configure_springA(150, 65000, 0, 0, 0);
```

Increasing Data Rates

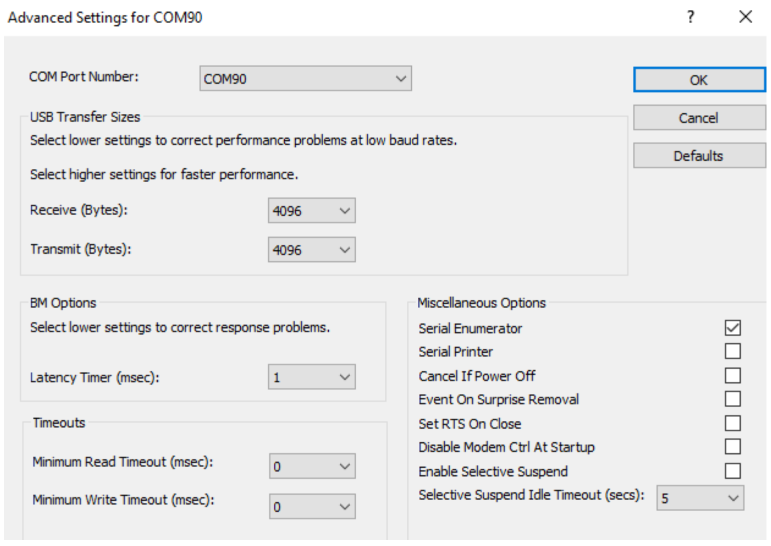
ORCA motors are set up to have a configurable baud rate and inter frame delay. By default these are 19200 bps and 2000 us respectively.
Through the motor's IrisControls GUI interface on the Modbus page, these can be increased to 1250000 bps and 0 us.
This change will also require that the Actuator object's BaudRate parameter be updated to match that set on the motor.



Modbus Options
Def. Baudrate 1250000 bps
Def. Interframe 0 us
Server Address 1

```
% Set up our Modbus client object
portnum = inputdlg('Enter RS422 COM Port Number:'); % Enter the COM port of
the Motor's RS422 serial interface.
port = strcat("COM", portnum);
orca = Actuator(port, 1250000);
```

To get maximum frame rates, it is also important to lower the Latency Timer from its default of 16 msec to 1 msec.



Advanced Settings for COM90

COM Port Number: COM90

USB Transfer Sizes
Select lower settings to correct performance problems at low baud rates.
Select higher settings for faster performance.
Receive (Bytes): 4096
Transmit (Bytes): 4096

BM Options
Select lower settings to correct response problems.
Latency Timer (msec): 1

Timeouts
Minimum Read Timeout (msec): 0
Minimum Write Timeout (msec): 0

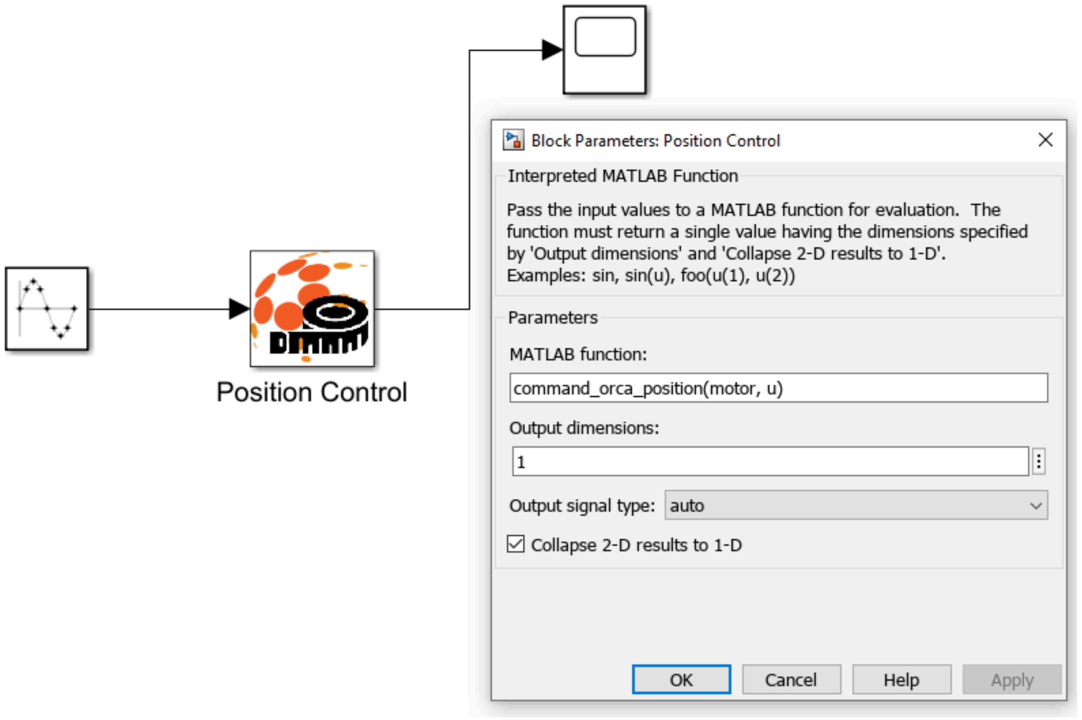
Miscellaneous Options
Serial Enumerator ☑
Serial Printer ☐
Cancel If Power Off ☐
Event On Surprise Removal ☐
Set RTS On Close ☐
Disable Modem Ctrl At Startup ☐
Enable Selective Suspend ☐
Selective Suspend Idle Timeout (secs): 5

Figure 5: Adjusting Latency Timer in Device Manager com port settings

Depending on other processing in the loop this can increase the communication rate to up to 800 messages per second.

Using Actuator Class with Simulink

Interpreted MATLAB Function blocks can be used to make blocks out of any of the Actuator class methods.

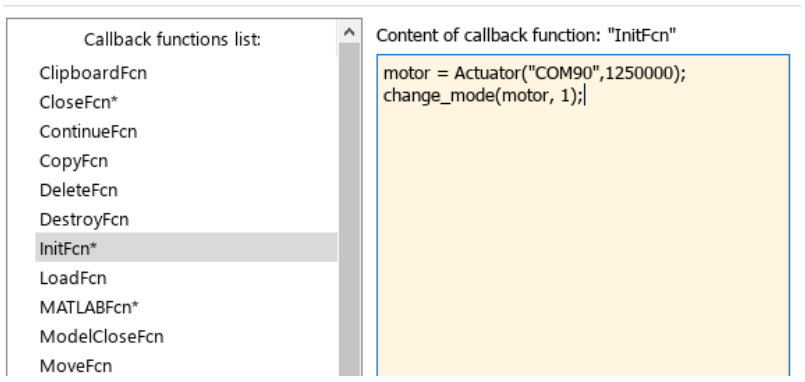

Position Control
Block Parameters: Position Control
Interpreted MATLAB Function
Parameters
OK
Cancel
Help
Apply

Pass the input values to a MATLAB function for evaluation. The function must return a single value having the dimensions specified by 'Output dimensions' and 'Collapse 2-D results to 1-D'.
Examples: sin, sin(u), foo(u(1), u(2))

MATLAB function: command_orca_position(motor, u)
Output dimensions: 1
Output signal type: auto
☑ Collapse 2-D results to 1-D

Figure 6: Simulink Position Control block parameters

Somewhere in the simulation an Actuator object will need to be initialized. In this example it is done through an InitFcn callback function in the position control block.



```
motor = Actuator("COM90",1250000);
change_mode(motor, 1);
```

Figure 7: Init callback function

The serial port is closed by clearing the object. Ensure there is a stop or close callback function that clears the Actuator object, or clear during the Init callback.

Appendix: Actuator Class Methods

Return	Method	Parameters	Description
obj	Actuator	com_port baud_rate	Constructor that opens a serial port at a specified baud rate.
position	command_orce_force	force_mN	Uses the motor command stream function code to send target force positions. Returns position and updates object's parameters (position, force, power, temperature, voltage, errors)
force	command_orca_position	position_um	Uses the motor command stream function code to send target position values. Returns force sensed and updates object's parameters (position, force, power, temperature, voltage, errors)
read_value	read_stream	register_address width	Uses the motor read stream function code to read from a specified register (16 or 32 bit) while also updating object's parameters (position, force, power, temperature, voltage, errors)
force	write_stream	register_address width value	Uses the motor write stream function code to write to a specified register while also updating the object's parameters (position, force, power, temperature, voltage, errors). Returns sensed force as an example.
void	change_mode	mode	Write to control register 3 to change the ORCA motor's mode of operation (Sleep, Force, Position, Kinematic, Haptic)
void	kinematic_trigger	motionID	Write to the kinematic software trigger register the ID of the motion that will be triggered (motor must be in kinematic mode to have an effect)
void	configure_motion	motionID position time delay nextID type autonext	Write to multiple registers that will configure a specified motion ID with all relevant motion parameters.
void	enable_haptic_effect	effect_bits	Turn on and off specific combinations of haptic effects. Bit flags found in class properties.
void	configure_springA	gain center coupling deadzone force_sat	Configure all parameters relevant to spring A's haptic behaviour.
void	tune_pid_controller	saturation p_gain i_gain dv_gain de_gain	Set the tuning of the motor's PID position controller. This controller will be active in Position and Kinematic modes.

void	write_register	register_address value	Write a value to a single register
void	write_multi_register	register_start_add num_registers register_data	Write an array of data to a series of consecutive registers.
read_value	read_register	register_start_add	Read a specified number of consecutive registers. Return the value contained in that register.