DC Motor Speed and Direction - Step Response

Part 1: Motor Step Response

Objectives:

- Use the DRV8833 or SN754410 motor driver and obtain the motor step response
  - In External mode
  - Directly through the serial port
- Understand the logic needed to control the magnitude and direction of a motor
- Observe the different effects of different logic schemes to drive a motor
- Observe the effect of sampling rate on a discrete sensor

Background Information:

A motor driver chip converts a lower power signal (from the microcontroller: 5v, 40mA) to a high power signal (9v, 1.5A) to drive the motor. It can be thought of as a switch or relay to the driver supply voltage (9v or 5v in this case). Since the supply voltage is fixed this would result in a fixed motor speed. To regulate the speed a PWM signal is used to quickly switch the output on and off so that the average output voltage can be controlled.

The term motor "driver" is also commonly called "amplifier" or "chopper". The DRV8833 driver is capable of providing 2.7-10.8v at 1.5A RMS on each channel.

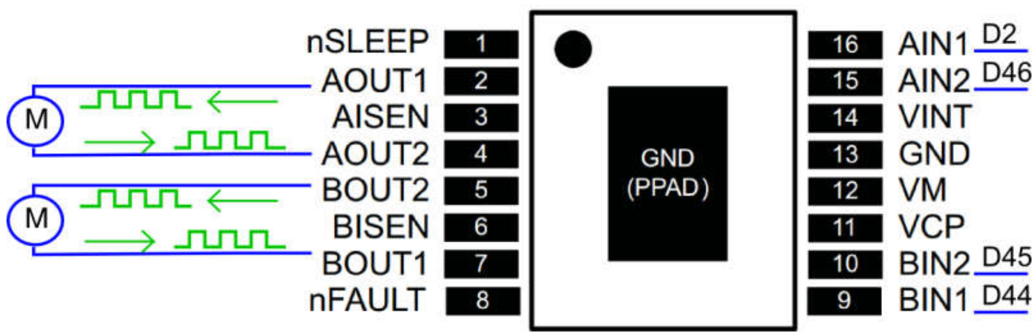


Figure 1: DRV8833pinout/wiring diagram

The SN754410 driver is capable of providing 3.5-36v at 1A on each channel.

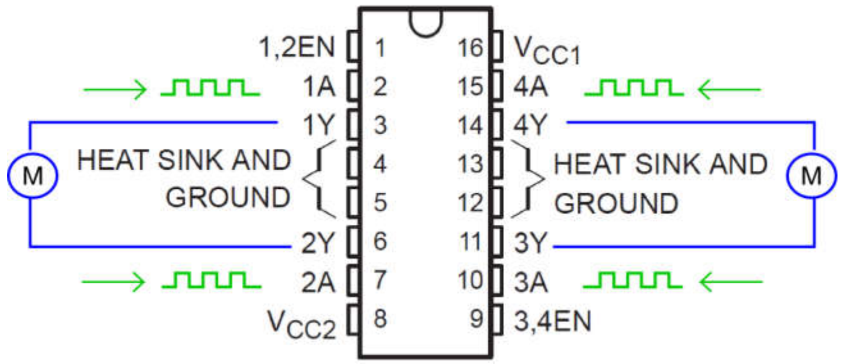


Figure 2: SN754410 pinout/wiring diagram

Simulink Model

Build and run the following Simulink diagram. Use the corresponding encoder/dual encoder block for **your system**, and the **digital pins that correspond to your motor driver**. Use the encoder blocks from RASPlib.

Shield or Board	Motor Number	Driver Pins	Encoder Pins
M1V4	1	5, 4	2, 3
	2	9, 11	6* or 0*, 10*
M2V3	1	6, 8	15, A8
	2	2, 5	18, 19
MinSegMega	1	2, 46	PE6, PE7
	2	44, 45	18,19

* Requires Jumpers on J4, J5, J6

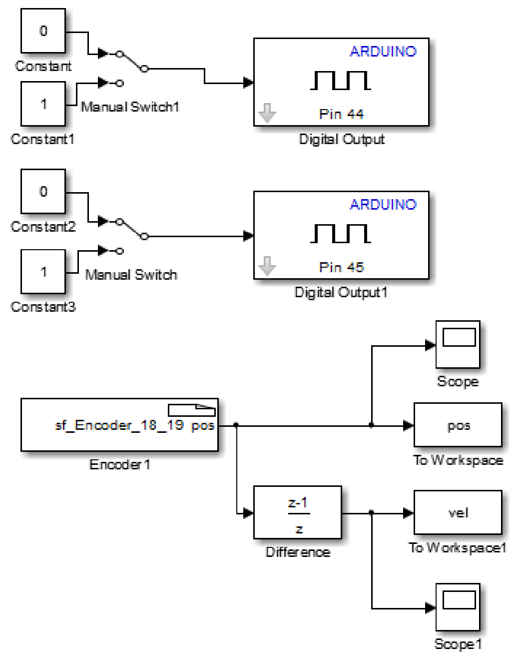


Figure 3: MinSegMega with generic encoder block

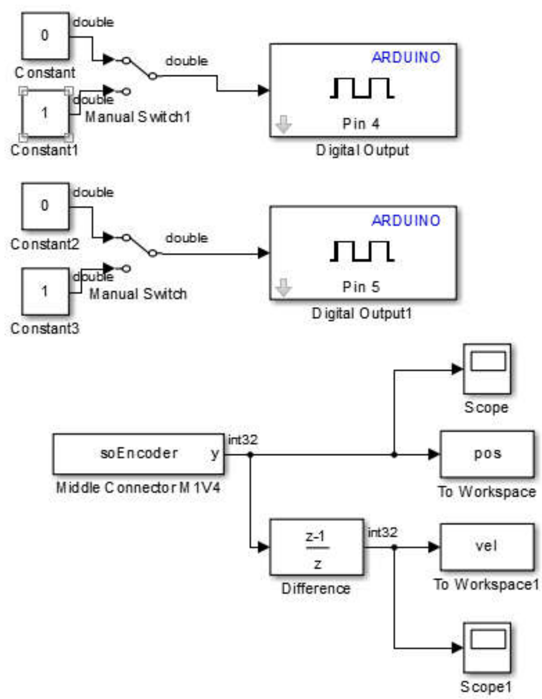
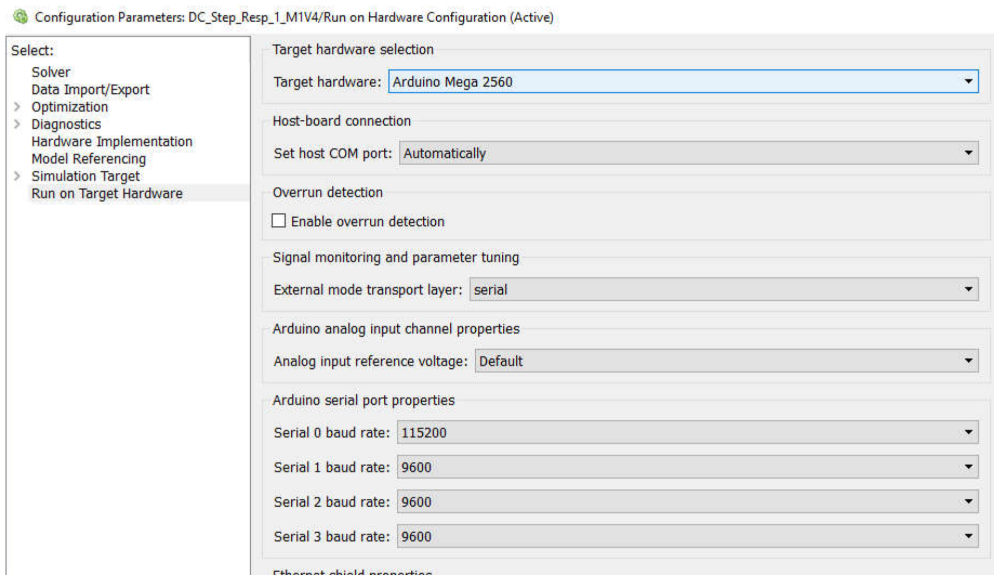


Figure 3: for M1V4 with dual encoder block

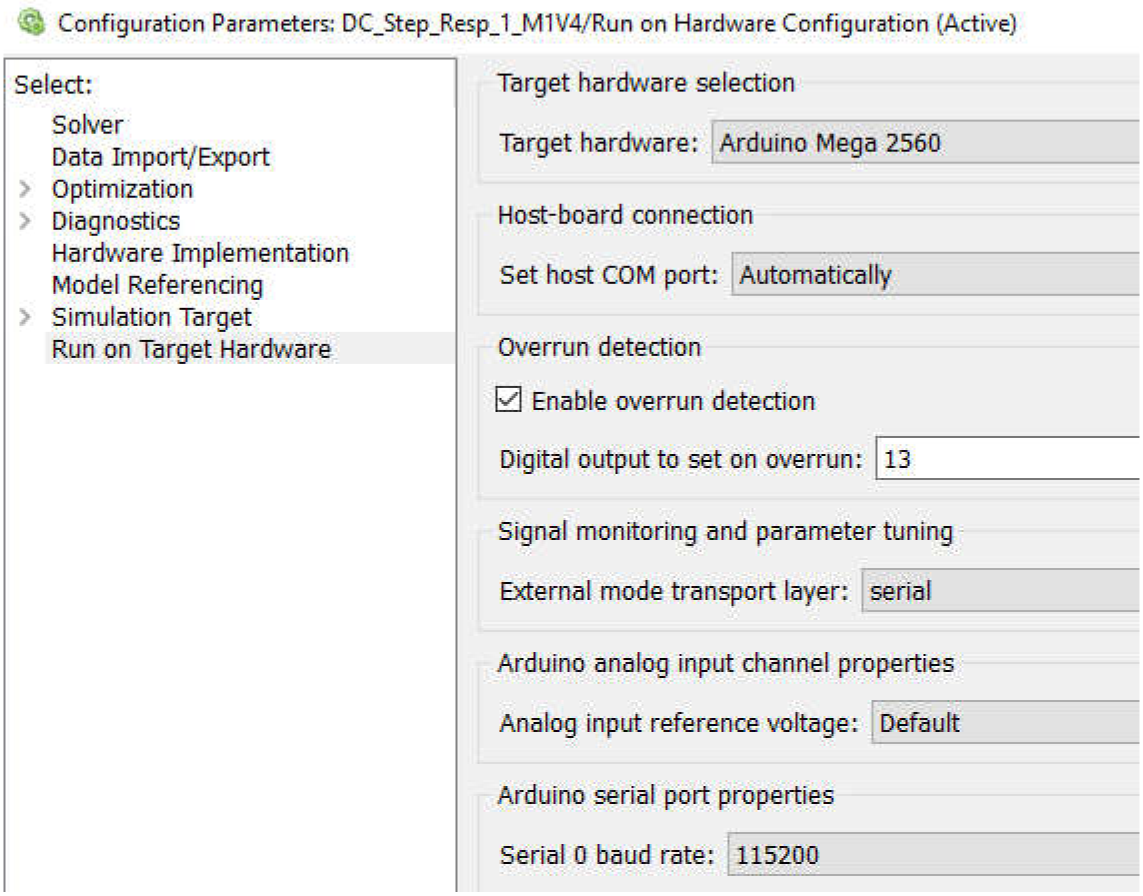
- Run the model in external mode to log the data
  - Sampling rate of .03 seconds
  - Change the baud rate to 115,200 to ensure the data can be sent fast enough



- Observe the response of the system
  - toggle the manual switches
  - toggle the On/Off switch on your board – this will determine the voltage source of the driver chip (ignore this step if you do not have batteries installed)
    - OFF - USB voltage (~4.5 volts)
    - ON – Battery voltage (~9v fully charged)
- Stop/disconnect from the system, save the data then plot the results
  - The commented line below stores the variables ‘pos’ and ‘vel’ into the data file ext_dat. Copy and paste the commented command in the command line to save the data. You do not want this uncommented in an m-file since it might cause you to accidentally overwrite previously stored data when you run the m-file.

```
%save ext_dat pos vel    % save the data
load ext_dat             % load the data
figure, plot(pos)
figure, plot(vel)
```

The velocity data obtained in external mode can be noisy. The primary reason seems to be that the time between samples is not actually fixed/constant. In the configuration parameters if you check the box “Enable overrun detection” then the digital pin you specify will be set high if the controller cannot execute your code in the sample time you specify. Check this box, select pin 13, and connect to the device. You will notice that the LED on pin 13 is always on. This indicates that in external mode the microprocessor has trouble meeting the sample time. This implies the time between each sample is not exactly 30 milliseconds. Any calculation assuming a fixed sample time, such as velocity, will then contain some error (noise) due to the sample time not being constant. Note that this noise is from the measurement system - not actual noise present in the signal or sensor.



- The actual motor velocity will not be fluctuating like the "noisy" data indicates. To make the data more useful use the "`smooth`" command in Matlab (if available). Type "`help smooth`" to obtain details on the smoothing method
    - `vel_smooth=smooth(double(Vel), 10)`
        - This function will use a moving average over 10 samples to smooth the velocity data
        - `double` – this converts the int16 values in Vel to doubles. If this is not done the function will complain

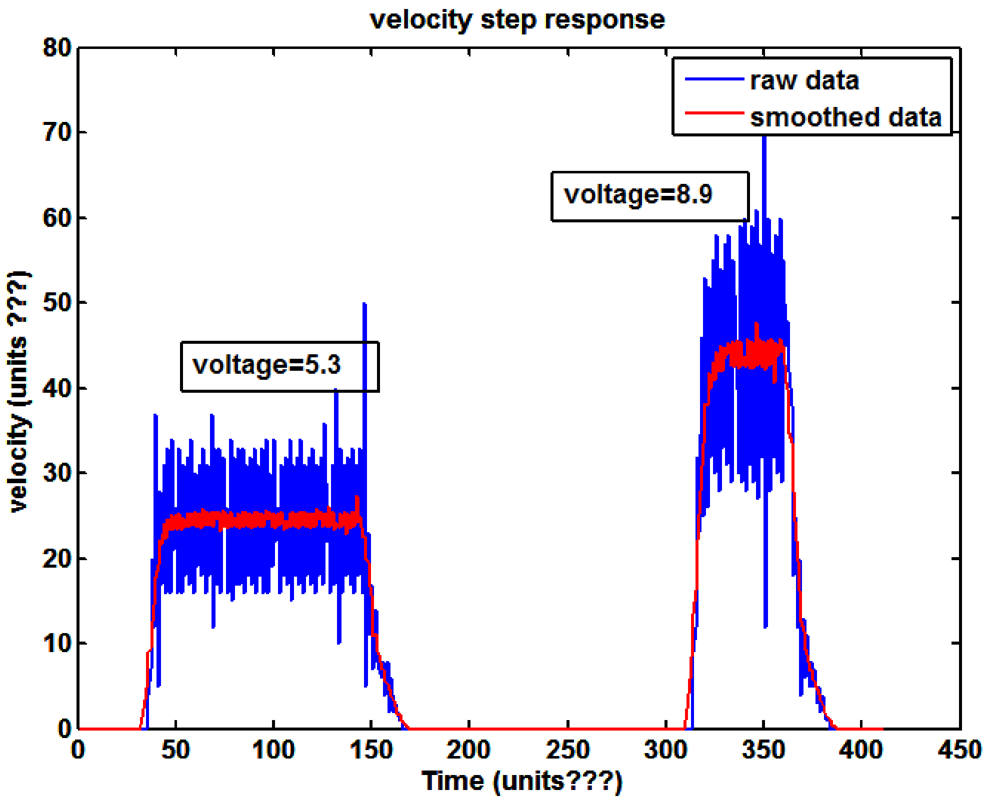


Figure 4: Step Response Graph

Questions:

- Provide a plot that contains the velocity step response. Provide the raw data and a smoothed response. Plot the velocity in RPM and time in seconds. An example plot (for two different voltage steps) is shown below for an unknown motor with unknown units (you will only need one voltage response if you do not have batteries installed)

Part 2: Step Response Data from Serial Port

The max velocity of the NXT motor is about 170 RPM at 9volts or about 18 radians/second. Since there are 720 pulses per revolution from the encoder (quadrature decoding), this is about 2040 counts/second. If we sample the system every .03 seconds this would give a maximum of 2040*.03 =61.2 counts every sample time. Since this the maximum value for the velocity we would expect, and this number is below 255, we can send the velocity data as uint8 data type.

Obtain same velocity information directly from the serial port:

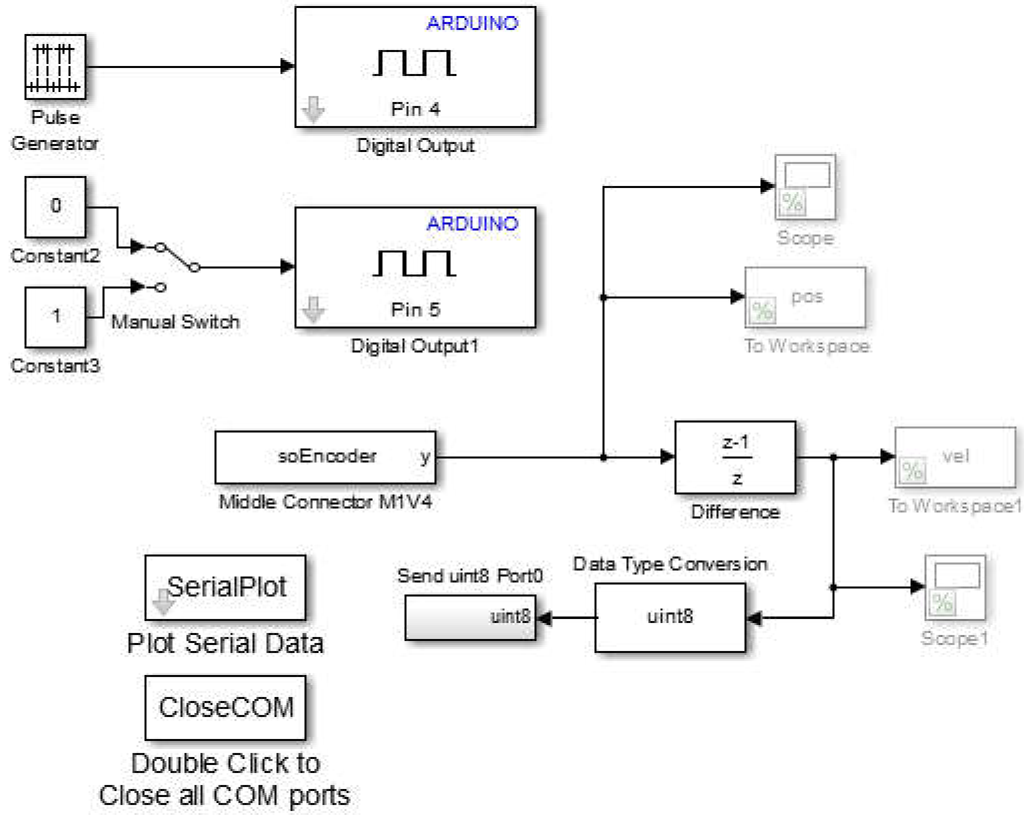
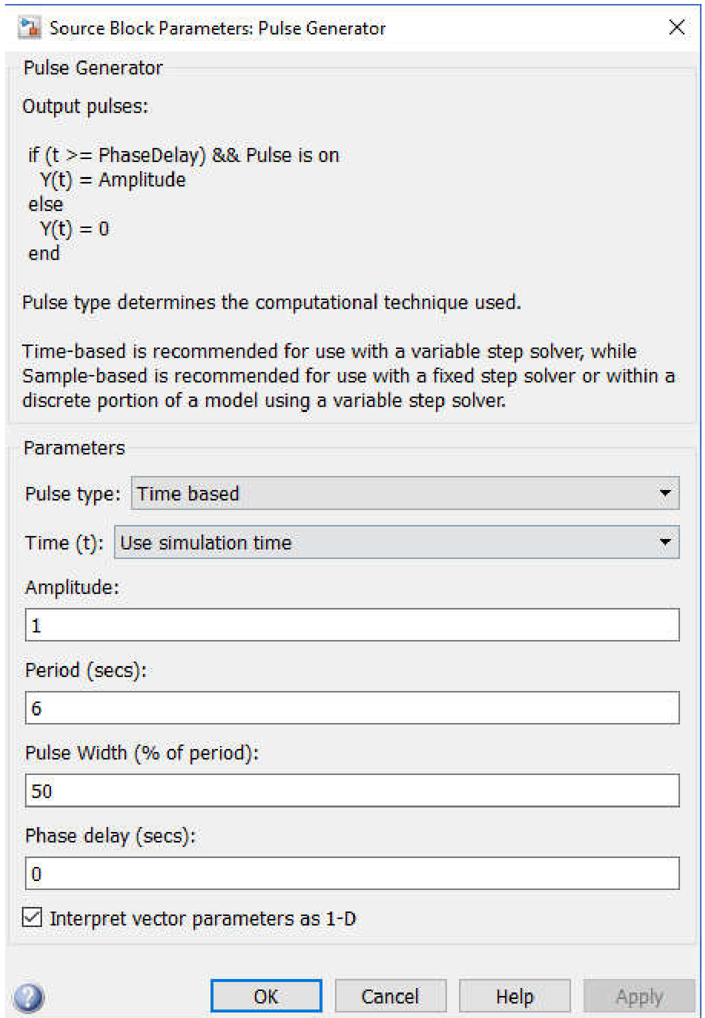


Figure 5: Serial Step Response

- The “ToWorkspace” and “Scope” blocks have been commented out because they cannot be used in Normal mode. You can comment a block out right-clicking it and selecting “Comment Out”. This is useful if you want to have one Simulink diagram for multiple purposes.
- Use the “Send uint8 Port0” block and the corresponding “SerialPlot” block to send and plot the data. The “CloseCOM” block is used to ensure all the COM ports MATLAB is trying to access are closed. Click this if for some reason you cannot download code to your board (because MATLAB has it open).
- Toggle the ON/OFF switch on the board to obtain the step response at USB voltage and at battery voltage if desired (optional if you do not have batteries)
- The switched shown in Figure 5 cannot be toggled when external mode is not used so use a pulse generator block to create the step input:



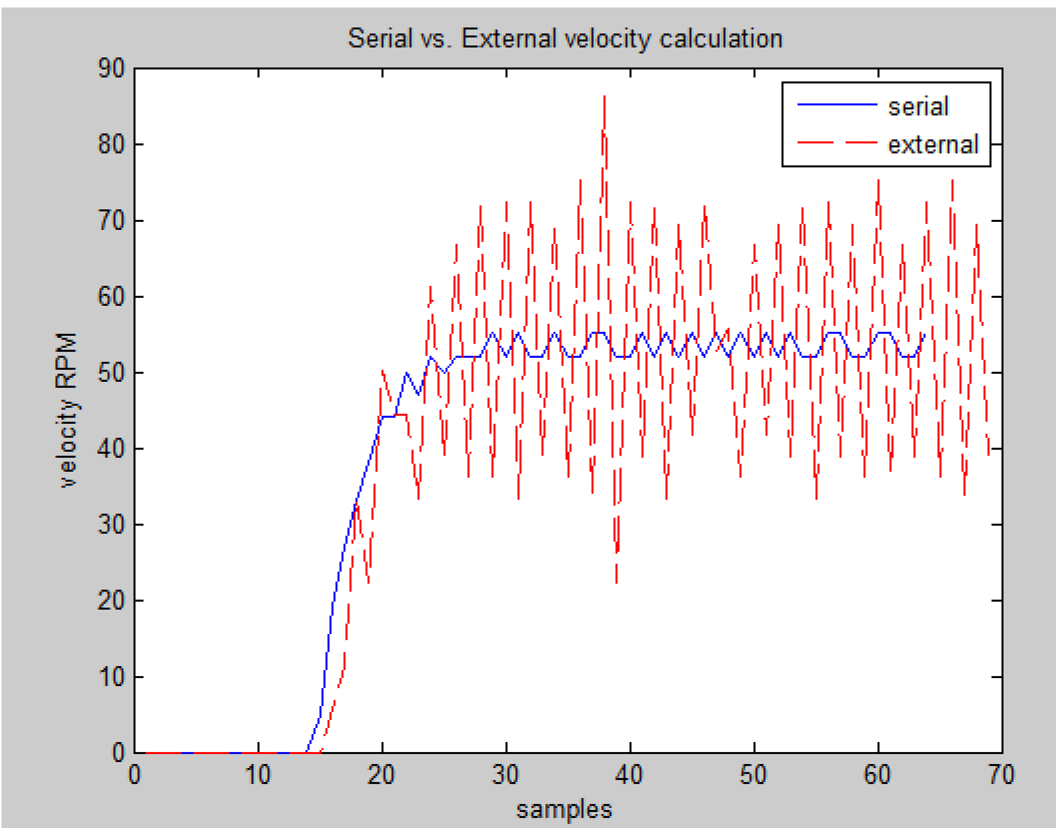
- Also use this model in external mode to obtain the data in the pos and vel variables if you uncomment the scope and to workspace blocks and comment out the ‘Send uint8 Port0’ block.
    - When the system is running toggle the ON/OFF button to get the response at ~9V in addition to the USB voltage

Use the following snippet to help plot this data and the previous data to observe the differences. You might have to modify and adjust your data appropriately.

```
%save ext_dat pos vel    % save the data
load ext_dat             % load the data
figure, plot(pos)
figure, plot(vel), hold on

% plot serial data:
% save ser_dat FullDat
load ser_dat
plot(FullDat, 'r')
legend('external mode', 'direct serial data')

% shift data so the the start of the step lines up
% only plot 100 data points
figure, hold on
plot(vel(118:118+100))            % external data
plot(FullDat(107:107+100), 'r') % serial data
legend('external mode', 'direct serial data')
```



Notice how the velocity obtained in serial may be better, this affirms the hypothesis that the calculation in external mode may be affected by a varying sampling time due to the processing overhead associated with external mode. Note: some newer computers with a USB 3.0 port have similar performance in serial and external modes.

Questions:

- Obtain the step response of motor in serial mode (Normal mode) at a time step of .03 seconds – be sure to keep this data to plot later
- Obtain the step response of the motor in serial mode (Normal mode) at a time step of .003 seconds – be sure to keep this data to plot later. You will need to modify the pulse generator parameters so the motor stays on for a longer time.
- Plot these two data sets (serial mode .03 and serial mode .003) on the same axis – you will have to modify the data so that they are both plotted on the same scale (you should use seconds instead of samples, and use the same velocity units). Explain why these look so different!
- KEEP This data - you will need this in future labs!

Part 3: Motor Logic: Direction and Magnitude

To regulate the speed a PWM signal is used to quickly switch the output on and off so that the average output voltage can be controlled. In addition, the correct switching logic needs to be implemented so that the motor can change direction based on the sign of the input.

The subsystem block below is used to correctly output the correct magnitude and logic of the PWM if the only input is a scalar input voltage (which could be negative**). You do not have to make this Simulink diagram – it is shown for reference only.**

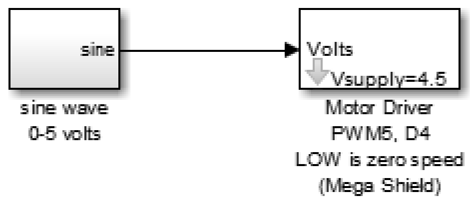


Figure 6: Motor driver block to control magnitude and direction

The contents of this subsystem are shown below:

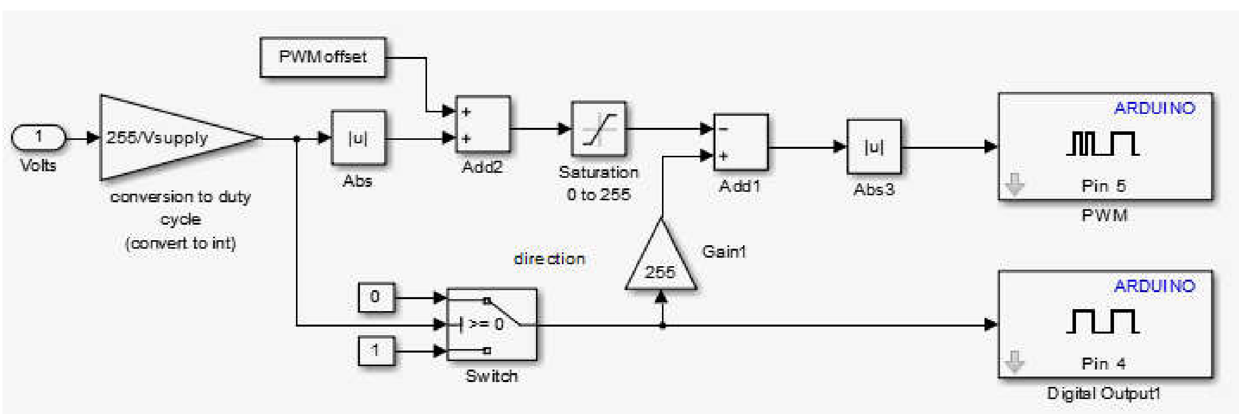
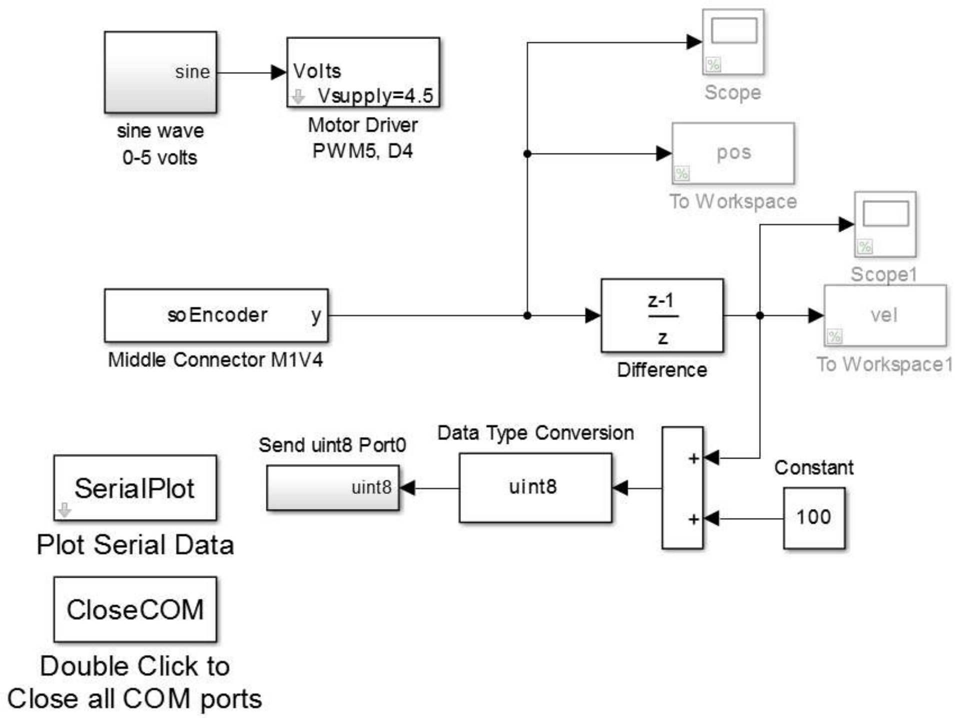


Figure 7: Example Logic for controlling magnitude and direction of motor voltage (for reference only - do not build this diagram, your driver block may look differently)

- Use this subsystem block in Figure 6 to generate a sinusoidal motor response:



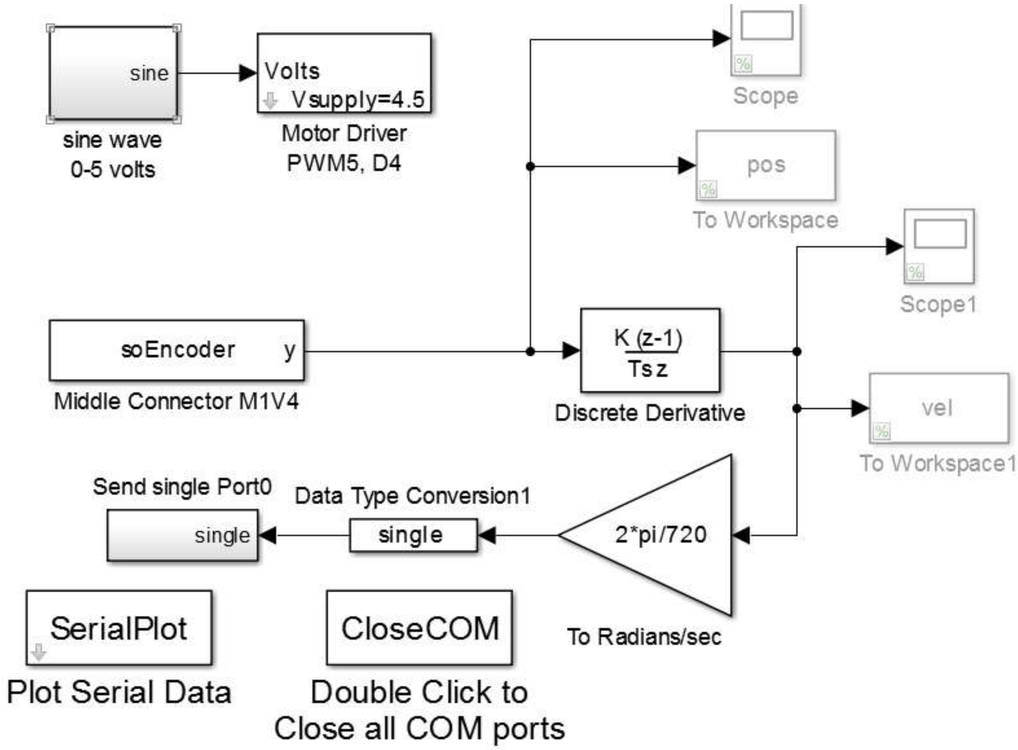
- Edit the Subsystem parameters to correctly specify the driver supply voltage (4.5 if power is from a USB, 9v if power is from the battery pack) and a PWM offset – normally zero.
- Set the parameters of the sine wave so it has a magnitude of 4.5 volts and a period of around 3 seconds. **Use the sine wave block from the demo file**. The default sine wave block in Simulink appears to use slow the serial port data transmission in external mode (might be using timers that interferes with serial timers).

Questions:

- Provide a plot of the sinusoidal velocity in external mode at .03 seconds and by obtaining data directly with the serial port at .03 seconds and .003 seconds.
- Why is a value of 100 added to the output data?

Part 4: Using the Discrete Derivative Block

In the previous sections the Difference block was used to capture velocity. The difference block ensured the resulting number would be integers (the encoder block outputs number of counts) this allowed the data to be sent as single byte (unsigned integer). After the data is received it then must be converted to useful units for velocity by dividing by the sample time and converting from counts to radians/second. This conversion can be done on the hardware, but the resulting floating point number must then be sent appropriately.



- Discrete Derivative will directly compute velocity considering the sample time of the system.
- Since the result could be a floating point number it must be converted to a single and sent as a single (4 bytes).