Motorcycle Assembly

Assuming we are using the Matlab license provided by Arduino Kit (https://www.mathworks.com/campaigns/products/arduino-kit-rev2-license.html).

1. Let us make sure the Simulink Support Package for Arduino Hardware is installed. Simulink Support Package for Arduino Hardware - File Exchange - MATLAB Central (mathworks.com). Download this file and install for each machine.
2. Also download the Matlab Support Package for Arduino Hardware. MATLAB Support Package for Arduino Hardware - File Exchange - MATLAB Central (mathworks.com). Download this file and install for each machine.
3. Courses | Arduino Cloud. Log into Arduino account, and access Arduino course information at this weblink. The weblink for registering kit is at Engineering Kit! (arduino.cc). The class may use the following group username and password username: nchunanorobot password: Nan0r0b0t.

Week 12

1. Open up the kit, and go to the webpage for the Engineering Kit Rev2 Arduino Education.
2. Please check to make sure all parts are in the kit (TA).
3. Please only use the parts needed for the motorcycle. Other kit contents should not be lost so that we are able to build the other projects later.
4. Two (or more?) sets of screwdrivers are available with the teacher. Please borrow and return to the front desk so others may also use the screws.
5. In this class we are going to make sure that the computers are able to connect to the Arduino board, check battery and ensure the inertial motor functions when electrically connected to Arduino. (https://engineeringkit.arduino.cc/aekr2/module/engineering/lesson/03-introduction-to-mechatronics). https://engineeringkit.arduino.cc/guide/cheatsheet.
6. If the link cannot be accessed, follow the instructions here.
7. There are two types of motors in the kit:

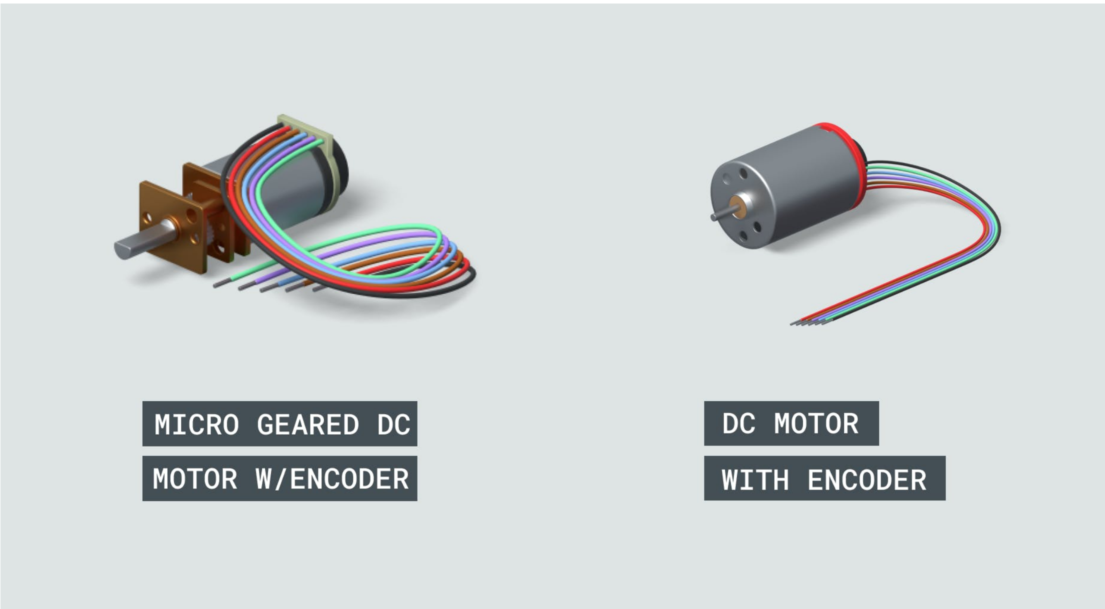


Figure 1. DC motors in the AEK.

8. We are going to work with the Micro geared DC Motor with Encoder (the rear wheel motor) which has the following pinouts.

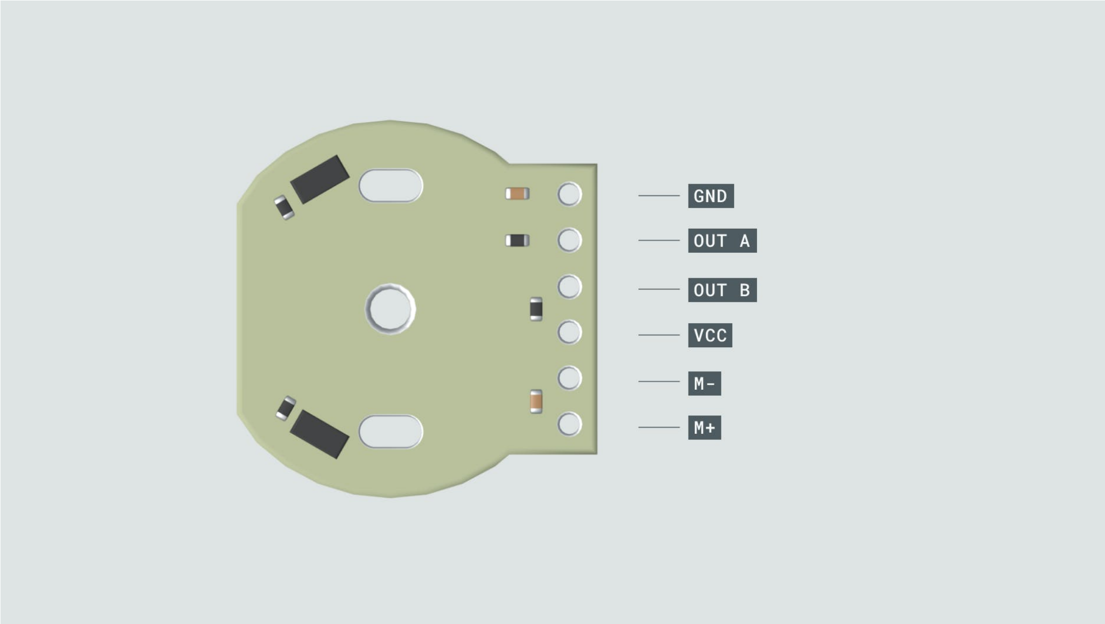


Figure 2. Pinouts for the micro geared DC inertial motor.

9. To have the kit function, 18650 type batteries are to be used: these are 3.7 V, 2500-2600 mAh. For a discharge rating of 5 C, the total current from the battery is 2500 mAh * 5 C = 12,500 mA. The maximum current discharge for these batteries should be at 20 A, and the battery weighs ~43.8 g. Please pay attention to polarity of the battery when installing to motorcycle.
10. The battery is important because we need the extra voltage to help drive the DC motors. The board itself is not sufficient to provide the power needed.

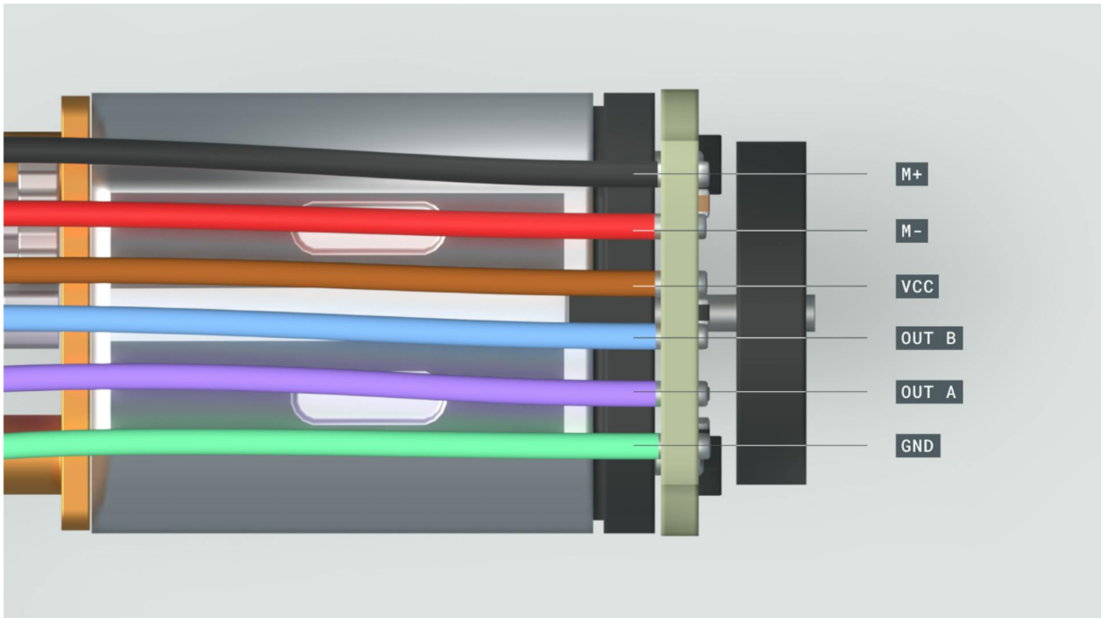


Figure 3. Wiring schematic for the micro geared DC inertial motor.

11. Connect the Arduino board correctly to the motor carrier as in Figure 4. Following

that let’s connect the micro geared DC motor wires to the Arduino NanoIoT33 board. Use for example the M1 motor connections.

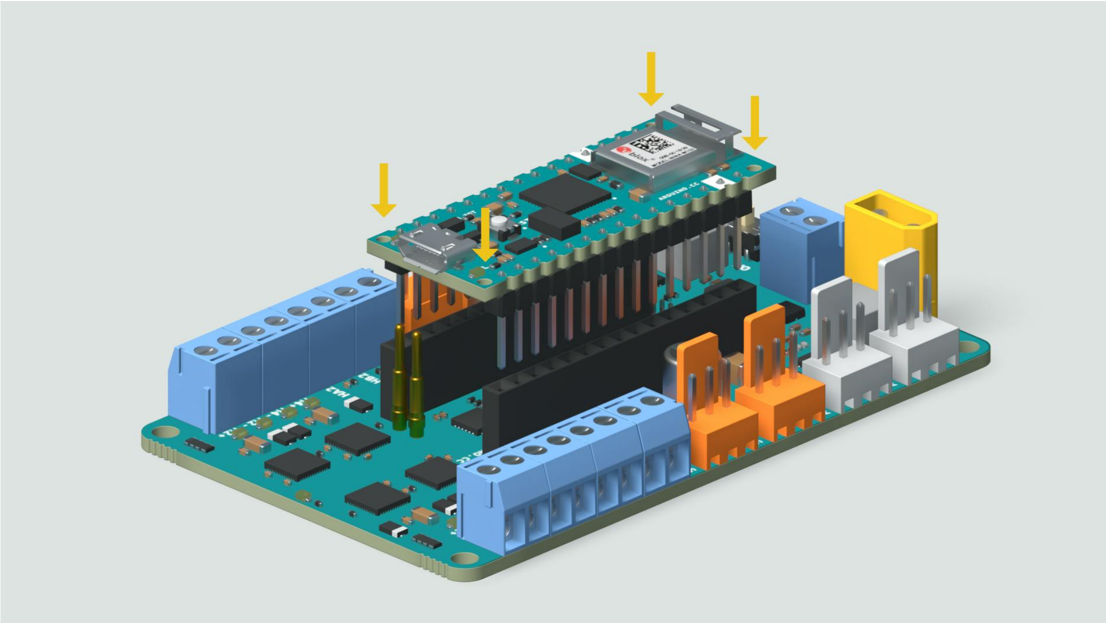


Figure 4. Connect the Arduino board correctly to the motor carrier.

12. Plug the USB cable into computer, turn the on-off switch to “On” position and ensure the LED is lit.

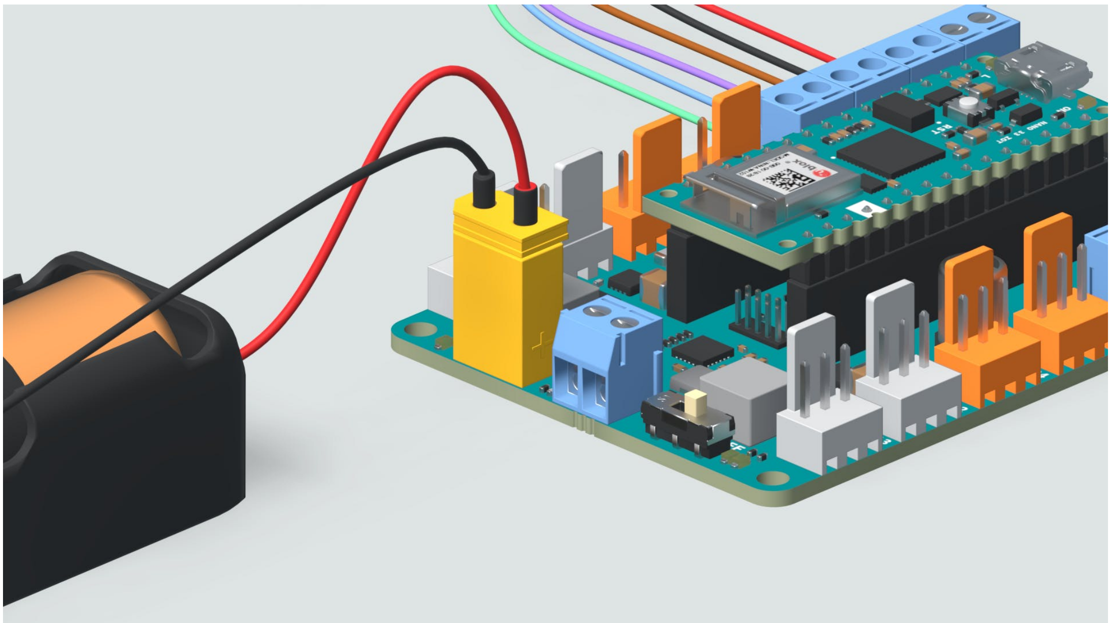


Figure 5. Wires from motor and battery are connected to the motor carrier board. The on-off switch remains in the “Off” position.

13. To make any mounting changes, turn off the switch.
14. Once Arduino is connected, type the following commands into Matlab command line:

```
>> clear a
>> a = arduino
```

15. The following command creates a carrier object in the Matlab workspace associated with the Arduino object.

```
>> carrier = motorCarrier(a)
```

16. The next, third object, gives control of the motor connected to M1 in Matlab. Use the following command to create a DC motor object that is associated with the Carrier object, and examine the displayed object properties in the Command Window.

```
>> dcm = dcmotor(carrier,'M1')
```

17. Now it is possible to drive the motor. The dcmotor() object has three properties: MotorNumber, Speed and Running. The Speed can be changed from -1 to 1. IsRunning is controlled with start and stop. Try the following commands.

```
>> start(dcm)
>> dcm.Speed = 0.5;
>> dcm.Speed = 0.3;
>> dcm.Speed = -0.3;
>> dcm.Speed = -0.1;
>> dcm.Speed = -0.05;
>> dcm.Speed = -0.01;
>> dcm.Speed = -0.3;
>> stop(dcm)
>> start(dcm)
>> stop(dcm)
>> dcm.Speed = -0.5;
>> start(dcm)
>> stop(dcm)
```

18. Next we are going to characterize the DC motor.
19. Set up as before.

```
>> clear a carrier
>> a = arduino
>> carrier = motorCarrier(a)
>> dcm = dcmotor(carrier,'M1')
>> enc = rotaryEncoder(carrier,1)
```

20. The encoder count buffer can be read and note the result.

```
>> readCount(enc)
```

21. The geometry of this encoder is such that there are three full cycles of quadrature when the motor shaft turns one full revolution. Recall that the quadrature signals

undergo four total changes in a full cycle. This means that there are 12 quadrature signal changes per revolution for the motor shaft. Thus, we can measure the angular position of the motor shaft with a resolution of 30 degrees, if we know the encoder count. Manually rotate the magnetic disk one full clockwise revolution (when looking down on the magnetic disk), and read the encoder count again with the previous command.

22. Note that switching the polarity of Out A and Out B would reverse the sign of the count readings.
23. Check again by rotating a full rotation and then check the counts.

```
>> shaftAngle = (readCount(enc)/12)*360
```

24. The gear motor has a gear ratio of 100:1. Use the following commands to get the angle of the output shaft in degrees, and then normalize to the range of 0 to 360 degrees.

```
>> axleAngle = (readCount(enc)/12)*360/100
>> axleAngleNorm = mod(axleAngle,360)
```

25. We can determine the speed of the output shaft in rpm and in degrees per second.

```
>> dcm.Speed = 0.5;
>> start(dcm)
>> rpm = readSpeed(enc)/100
>> degPersec = rpm/60*360
```

26. Now we are going to write a livescript to characterize the motor. Input the following code.

```
%% 1. Create test data

maxPWM = 1.00;                          % Maximum duty cycle
incrPWM = 0.05;                         % PWM increment
PWMcmdRaw = (-maxPWM:incrPWM:maxPWM);   % Column vector of duty cycles from -1 to 
1
```

27. Run the livescript and examine the Workspace. There should be a vector containing numbers ranging from -1 to 1 with 0.05 increments. This is going to be used to generate the PWM signal applied to DC motor.
28. The next line of the livescript prepares the motor.

```
%% 2. Create and initialize device objects

clear a dcm carrier enc              % Delete existing device objects

a = arduino;
carrier = motorCarrier(a);
dcm = dcmotor(carrier,'M1');         % Connect a DC motor at 'M1' port on the
Arduino Nano Motor Carrier board

enc = rotaryEncoder(carrier,1);      % Connect the encoder of 'M1' at the encoder
port 1 on the Arduino Nano Motor Carrier board
```

29. Now let's add a section to start the motor with PWM value, read the motor speed, and then stop the motor.

```
%% 3. Measure raw motor speed for each PWM command

dcm.Speed = 0;
gearRatio = 100;                            % As per the motor spec sheet, gear
ratio equals 100:1
start(dcm)                                  % turn on motor
dcm.Speed = PWMcmdRaw(1);
pause(1)                                    % wait for steady state
speedRaw(1) = readSpeed(enc)/gearRatio;     % read motor speed in rpm of the
output shaft
stop(dcm);                                  % turn off motor
dcm.Speed = 0;
```

30. Try running this section with different values of PWMcmdRaw and examine the Matlab Workspace.
31. Now update the third section with a for loop.

```
%% 3. Measure raw motor speed for each PWM command

dcm.Speed = 0;
gearRatio = 100;                                % As per the motor spec sheet,
gear ratio equals 100:1
start(dcm)                                      % turn on motor

for ii = 1:length(PWMcmdRaw)
    dcm.Speed = PWMcmdRaw(ii);
    pause(1)                                    % wait for steady state
    speedRaw(ii) = readSpeed(enc)/gearRatio;    % read motor speed in rpm of the
output shaft
end

stop(dcm)                                       % turn off motor
dcm.Speed = 0;
```

32. At this point, the livescript should be giving a warning to the assignment on speedRaw(ii). This warning appears because the script is assigning values to increasing indices of speedRaw, and as a result speedRaw needs to increase its size on every iteration to accommodate the new element. In some cases, this can lead to performance issues because the variable's memory may need to be reallocated many times. You can solve this problem by defining the vector in advance with the zeros function to allocate space for speedRaw before populating the vector with values. Make the update and test to ensure the code works.

```
speedRaw = zeros(size(PWMcmdRaw));              % Preallocate vector for speed
measurements
```

33. Now let's plot the Measured Speed versus the PWM Duty Cycle.

```
%% 4. Graph raw data

plot(PWMcmdRaw,speedRaw)      % raw speed measurements
title('100:1 Gearbox Motor Steady State Response')
xlabel('PWM Command')
ylabel('Measured Speed (rpm)')
```

34. A plot should open up in the output column of the Live Editor. Notice the features of the speed-PWM relationship.
35. Next try to examine speedRaw directly in the command line.

```
>> speedRaw
```

36. You can also compare to a first-order difference to see where non-increasing values are located.

```
>> diff(speedRaw)
```

37. Let's write a new section to perform post processing on the data.

```
%% 4. Post-process and save data

idx = (diff(speedRaw) > 0);             % find indices where vector is increasing
speedMono = speedRaw(idx);              % Keep only increasing values of speed
PWMcmdMono = PWMcmdRaw(idx);            % Keep only corresponding PWM values
PWMcmdMono(speedMono == 0) = 0;         % enforce zero power for zero speed
save motorResponse PWMcmdMono speedMono % save post-processed measurements
```

38. The following command allows to compare the raw and filtered values.

```
%% 5. Graph raw and post-processed data

plot(PWMcmdRaw,speedRaw)                            % raw speed measurements
hold on
plot(PWMcmdMono,speedMono)                          % non-monotonic measurements
filtered out
title('100:1 Gearbox Motor Steady State Response')
xlabel('PWM Command')
ylabel('Measured Speed (rpm)')
legend('Raw Data','Monotonic Data','Location','northwest')
```

39. When finished with this task, create a section to delete device objects.

```
%% 6. Delete device objects

clear a dcm carrier enc
```

40. The next task is to work with Matlab functions. Let's create a Live Function.
41. The inputs and outputs to the function need to be determined and these inputs are

PWMcmdRaw, dcm and encoder object enc. The outputs to the function are PWMcmdMono and speedMono.

```
function [PWMcmdMono,speedMono] = myMotorFunction(PWMcmdRaw,dcm,enc)
```

42. To make the entire algorithm, let's take some of the code from the livescript and copy to the live function.

```
function [PWMcmdMono,speedMono] = myMotorFunction(PWMcmdRaw,dcm,enc)

3. Measure raw motor speed for each PWM command
speedRaw = zeros(size(PWMcmdRaw));              % Preallocate vector for speed
measurements

dcm.Speed = 0;
gearRatio = 100;                                % As per the motor spec sheet,
gear ratio equals 100:1

start(dcm)                                      % Turn on motor

for ii = 1:length(PWMcmdRaw)

    dcm.Speed = PWMcmdRaw(ii);
    pause(1)                                   % Wait for steady state

    speedRaw(ii) = readSpeed(enc)/gearRatio;   % read motor speed in rpm of the
output shaft
end
```

```
4. Post-process and save data
idx = diff(speedRaw) > 0;                % find indices where vector is increasing
speedMono = speedRaw(idx);               % Keep only increasing values of speed
PWMcmdMono = PWMcmdRaw(idx);             % Keep only corresponding PWM values

PWMcmdMono(speedMono == 0) = 0;          % enforce zero power for zero speed

save motorResponse,'PWMcmdMono','speedMono' % save post-processed measurements
```

```
5. Graph raw and post-processed data
plot(PWMcmdRaw,speedRaw)   % raw speed measurements
hold on
plot(PWMcmdMono,speedMono) % non-monotonic measurements filtered out

title('100:1 Gearbox Motor Steady State Response')
xlabel('PWM Command')
ylabel('Measured Speed (rpm)')
legend('Raw Data','Monotonic Data')
```

43. Let's save the Live Function as myMotorFunction.mlx.
44. Now we can actually call the function in the Live Script.

```
1. Create test data
maxPWM = 1.00;                       % maximum duty cycle
incrPWM = 0.05;                      % PWM increment
PWMcmdRaw = (-maxPWM:incrPWM:maxPWM)'; % column vector of duty cycles from -1 to 1
```

```
2. Create and initialize device objects
clear a carrier dcm enc              % Delete existing device objects

a = arduino;
carrier = motorCarrier(a);
dcm = dcmotor(carrier,'M1');         % Connect a DC motor at 'M1' port on the
Arduino Nano Motor Carrier board

enc = rotaryEncoder(carrier,1);      % Connect the encoder of 'M1' at the
encoder port 1 on the Arduino Nano Motor Carrier board
```

```
3-5. Call motor characterization function
[PWMcmdMono,speedMono] = characterizeMotorFcn(PWMcmdRaw,dcm,enc);
```

```
6. Delete device objects
clear a carrier dcm enc
```

45. Note that the function call saves the analysis data to motorResponse.mat every time the function is called. We can update the code to save to a filename of choice.

```
function[PWMcmdMono,speedMono] = myMotorFunction(PWMcmdRaw,dcm,enc,filename)
```

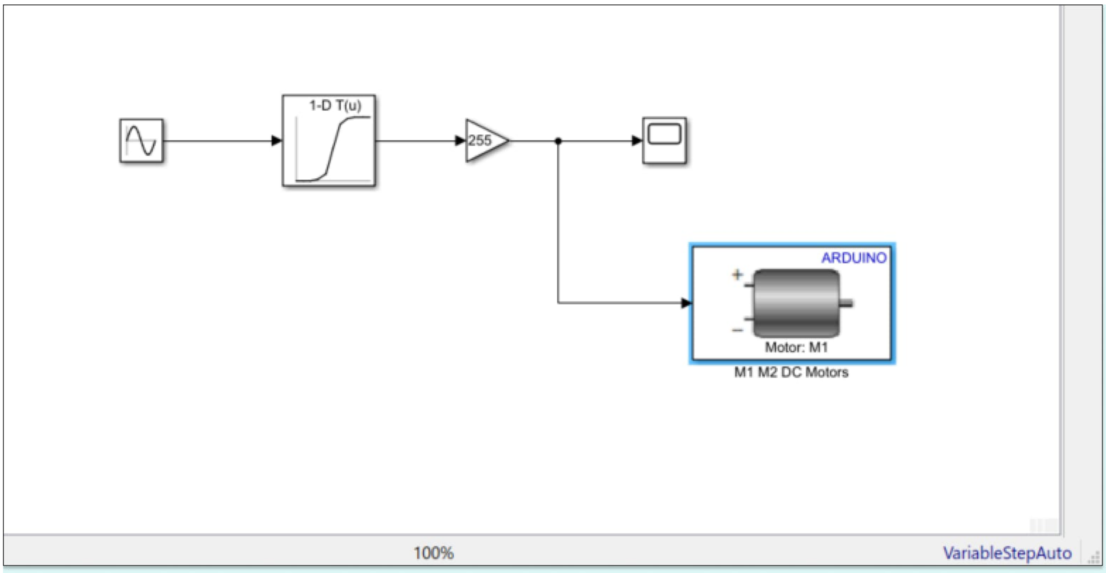
46. This input is written into the save command.

```
save(filename,'PWMcmdMono','speedMono')% save post-processed measurements
```

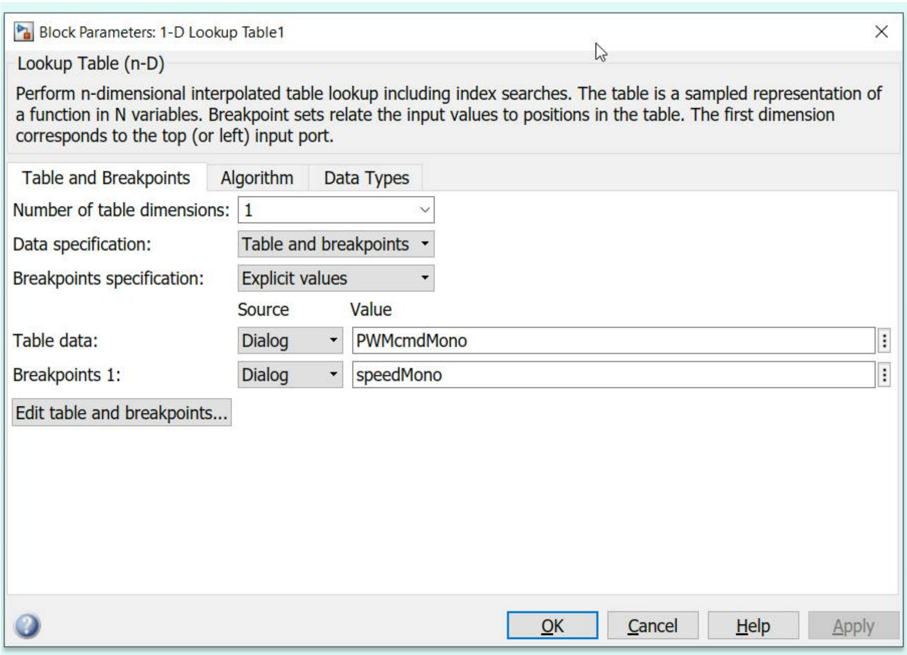
47. Depending on the motor used, we could give a specific name in the function call.

```
[PWMcmdMono,speedMono] = myMotorFunction(PWMcmdRaw,dcm,enc,'motorResponse');
```

48. The final task in this week is to try to setup a Simulink. Create the following in a blank Simulink canvas.



49. The Sine wave has output which is converted to a signal between -1 and 1 in a Lookup Table. The M1, M2 motors intake integer signals from -100 to 100 and the M3, M4 motors take -255 to 255. This is the purpose of the Gain block. The scope sink allows us to monitor the signal.



50. Configure the Sine block to have Amplitude 300, Bias 0, Frequency (rad/sec) 0.2, Phase (rad) 0 and Sample Time 0.
51. Have the Gain at 100, with Element-wise (K.*u) multiplication.
52. Open the Configuration Parameters Window by clicking on the Model Settings Gear Icon in Modeling.
53. Make sure the Hardware Implementation is set to “Arduino Nano 33 IoT” and the external mode is “Serial”.

54. Set the Stop Time to “Inf”.
55. Try to Run the Simulink canvas on the Arduino board.
56. The first Engineering Kit has course content at this webpage https://aek.arduino.cc. Access to learn more about the sensors on the boards.

Week 13

1. Open up the kit, and go to the webpage for the Engineering Kit Rev2 Arduino Education.
2. Please check to make sure all parts are in the kit (TA).
3. Please only use the parts needed for the motorcycle. Other kit contents should not be lost so that we are able to build the other projects later.
4. Two (or more?) sets of screwdrivers are available with the teacher. Please borrow and return to the front desk so others may also use the screws.
5. Go here to watch the assembly video, and then proceed to put together the motorcycle Content Preview (arduino.cc).
6. In the video, parts are labelled according to the boxes, Motorcycle, M2, for example.
7. Please try to complete motorcycle build in 45 minutes.
8. For the Matlab code, please download the kit files or go here (Arduino_Engineering_Kit_Project_Files_Rev_2 - File Exchange - MATLAB Central (mathworks.com)).
9. Connecting USB and motorcycle to PC. Matlab should send a message "Arduino detected. This device is ready for use with MATLAB Support Package for Arduino Hardware. Get started with examples and other documentation. This device is ready for use with Simulink Support Package for Arduino Hardware. Get started with examples and other documentation."
10. Proceed to test the battery, inertial motor, check to observe the scope changes when rotating manually the rotary encoder. Hope to see motorcycle balance function. Go to the Exercise 6_2.
11. Work through Battey_0.slx. Make sure the code is working before proceeding to the next list item.
12. Work through Encoder_0.slx and Encoder_1.slx. Make sure the code is working before proceeding to the next list item.
13. Work through IMU_0.slx and IMU_1.slx. Make sure the code is working before proceeding to the next list item.
14. Work through IW_Motor_0.slx. Make sure the code is working before proceeding to the next list item.
15. If the parts are all working, let us try to test in Exercise 6_3 the hardwareModel_5.slx.
16. When complete, please put all parts, including USB cable, back into kit box.

Week 14

1. If motorcycle was not disassembled in week 13, continue with testing.

2. When complete, please disassemble motorcycle parts and put all parts back into kit box.