Riley Rudeman

TCES 455, Autumn 2020

Laboratory 1: Intro to MATLAB/Simulink

Prior to lab:

- Install MATLAB/Simulink.

- Read about MATLAB basics on programming scripts & functions at: https://www.mathworks.com/help/releases/R2019b/matlab/scripts.html
  In particular, the following five sections
  a. Create scripts
  b. Loop control statements
  c. Conditional statements
  d. Add comments to programs
  e. Scripts vs functions

- Read the basics about Simulink at http://ctms.engin.umich.edu/CTMS/index.php?aux=Basics_Simulink

1. MATLAB scripts and functions development.

a. Create a ***script*** that adds all the terms in the series

$$1+\frac{1}{2}+\frac{1}{4}+\frac{1}{8}\ldots$$

until the sum exceeds 1.995. When you run the script, it will print out the sum and the number of terms needed to just exceed the sum of 1.995.

b. Write a ***function*** that can be used to calculate the equivalent resistance of n parallel connected resistors. In general, the equivalent resistance of resistors $R_1, R_2, R_3, \cdots, R_n$ is given by

$$\frac{1}{R_{eq}}=\frac{1}{R_1}+\frac{1}{R_2}+\frac{1}{R_3}+\cdots\cdots+\frac{1}{R_n}$$

c. For a triangle with sides of length $a, b,$ and c, the area A is given as

$$A=\sqrt{s(s-a)(s-b)(s-c)}$$

where

$$s=\frac{a+b+c}{2}$$

Write a ***function*** to compute the area given the sides of a triangle. Also use the function to compute the area of triangles with the lengths: (a) 56, 27 and 43; (b) 5, 12 and 13.

All in own scripts

2. Continue the following Simulink excises on modeling of an armature controlled DC motor.

Consider a permanent-magnet DC motor characterized by the following set of coupled differential equations:

Electrical Dynamics: $v_a = R_a i_a + L_a \frac{di_a}{dt} + K_V \omega_m$,

Developed Electromagnetic Torque: $T_e = K_V i_a$,

Mechanical Dynamics: $T_e = J_{tot} \frac{d\omega_m}{dt} + B_m \omega_m + K_L \omega_m^2$,

where v_a is the applied armature voltage, i_a is the armature current, ω_m is the angular velocity of the rotor shaft in radians/second, T_e is the developed electromagnetic torque in Newton-meters, and the output load torque is given by the $K_L \omega_m^2$ term.
The constants are:

R_a : armature resistance in ohms

L_a : armature inductance in Henries

K_V : the back emf of torque constant in volt-second or Newton-meter/amp

J_{tot} : sum of the armature inertia in kilogram-meter squared and the reflected load inertia (via the gear ratio)

B_m : viscous friction constant in Newton-meter-second

K_L : the load torque constant in Newton-meter-second squared

Ultimately the armature voltage will be the controllable output of a power converter and feedback system and we will be able to dynamically adjust the armature current and control the speed of the motor. For today, v_a will simply be a constant that we can specify.

<u>Starting SIMULINK</u>
To begin our investigation, open MATLAB and at the prompt type

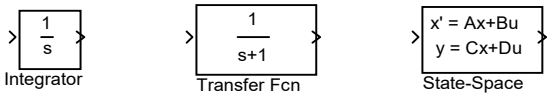
>> simulink

Note the extensive list of libraries and functions along the left portion of the Simulink window. In order to create a simulation model, you need to select

File → New → model

Where Components and Functions are Found

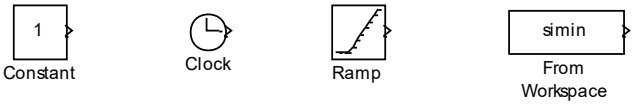
This new window that opens is where we will be dropping components from the libraries and interconnecting them. If you left-click the *Continuous* Library, it will display such items as



which will be the key building blocks for modeling dynamic systems. If you double-click the Transfer Function block, you can change the polynomial coefficients (highest order to lowest order) or make them variables to be assigned in a MATLAB M-file.

The *Math Operations* Library also has a couple of key components, where the Product block is typically reserved for creating terms like ω_m^2 whereas the Gain block is used to make $K_V \omega_m$.

Finally, in the *Sources* Library, we have the ability to specify constants, create ramps, input values of the Matlab workspace, and keep track of the time variable (Clock).



A component is placed into the model window by left-click-hold then moving the part into the desired position. Once in the model window, if you need to flip or rotate the element select

Right click → Format → flip block

If you double left click the block, you open a parameter window. For example for the summing block, you can change the icon shape, the number of inputs, and their respective polarities.

Connecting Components

Components may be connected by either

1. Left select the "from" block – hold the CTRL key – left click the "to" block
2. Start from the "out" port (left click and hold) and proceed to the

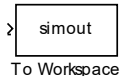
desired "in" port

You may always left-select a connection path and eliminate it by pressing delete. As diagrams become more complex, "From" and "GoTo" blocks in the *Signal Routing* Library may be used to manage the signal flow connections. Further for complex block diagrams, you can also subdivide your system into "subsystems" where only the inputs and outputs become visible at the top-most level. The feature is found in the *Parts & Subsystems* Library but will not be delved into today.

To Save Outputs to Matlab

First you will need to capture the time variable using the "Clock" function in the *Sources* Library. Next you will need to concatenate time along with other variables of interest using the mux block found in the *Signal Routing* Library

By double-left clicking, you can open a parameter block and specify the number of inputs. The mux output may be directed to a scope (*Sinks* Library) or to the Matlab workspace using the "To Workspace" block in the *Sinks* Library.



Make sure to double-click the "To Workspace" block and change the "Save Format" setting from *Structure* to *Array.* You may also change the outputted variable name from simout to something more descriptive. The variable simout will house column-wise the variables from the mux. So for instance, if time is the first element of the mux, we could then access it in Matlab and reassigning the name by typing

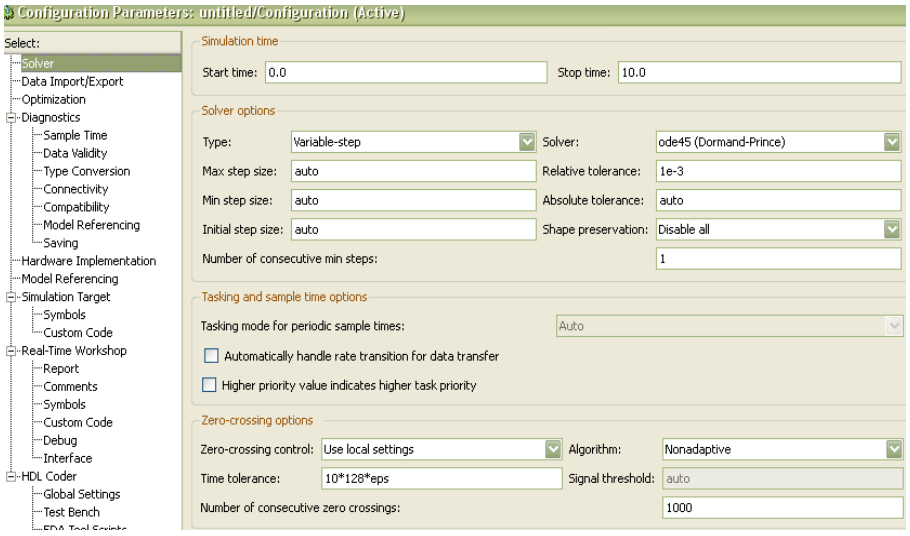
```
>> t = simout( : , 1 );
```

If we repeat this for the other variables, then we are set up to conveniently generate plots or perform data analysis.

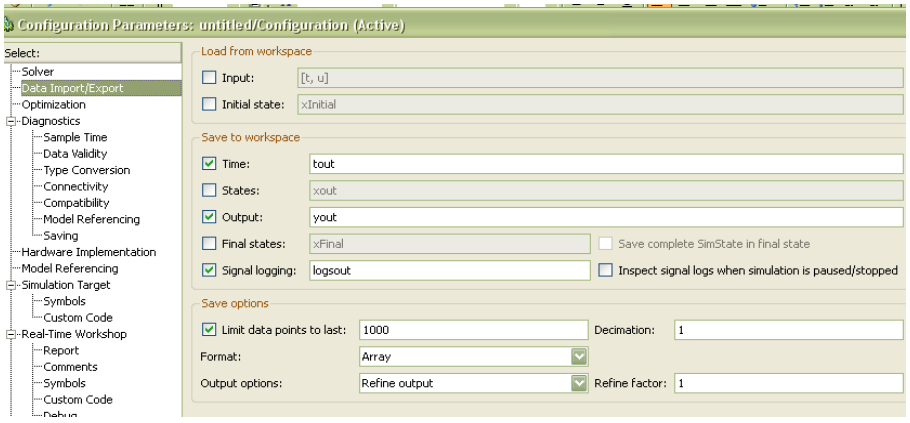
To Set Up a Simulation Run

After saving a completed model (making sure that no inputs are left unassigned), you next need to let SIMULINK know how long you want the simulation to execute, which numerical integration routine to use, and what time step to employ. In the model window select,

Simulation → Configuration Parameters



As shown, under the "Solver" tag (on the left menu), we can adjust the start and stop times and the solver routine. The variable-step solvers continuously adjust the algorithm step size to maximize the computational efficiency while maintaining a specified accuracy. The ode45 routine is excellent and is well suited for systems with discontinuities as we might find in power converters. On the "Data Import/Export" tag,



we see that we can automatically save the final values of the state variables to the Matlab workspace. We can then use these saved variables to initialize a future study (using the Initial state check box in the window). Essentially, state variables are established for each integrator block used and for and for each transfer function denominator power of 's'. We will explain more about state variables as we encounter them in practice.

<u>To Start and Run the Simulation</u>

Once you have built your model, established the desired outputs to Matlab, and configured your solver parameters, the simulation is executed by selecting

Simulation → Start

Or clicking the ▶ menu button. A "beep" will signify when the run is completed. OK, let's apply what we now know to modeling and simulating the PM DC motor.

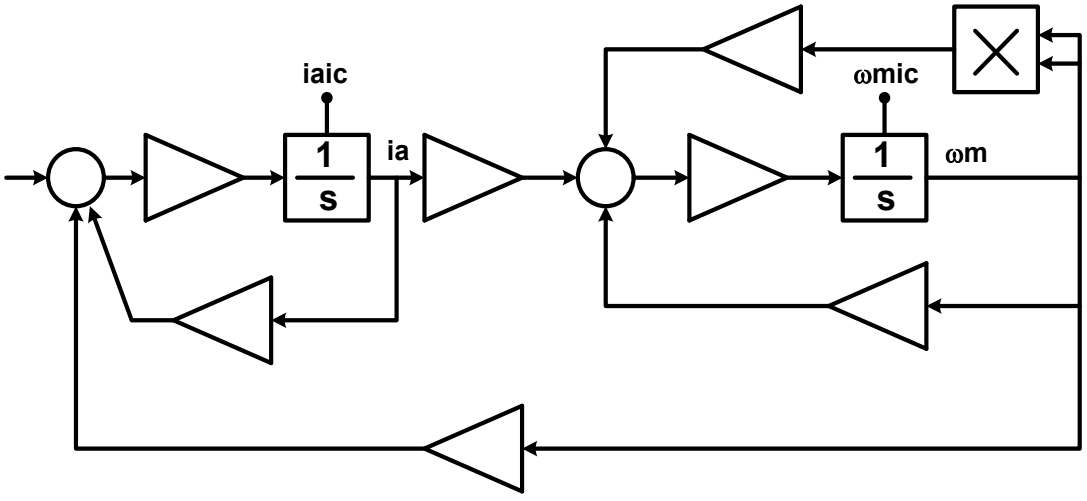
Step 1: Substitute T_e into the mechanical dynamics equation and solve for $\frac{di_a}{dt}$ and $\frac{d\omega_m}{dt}$ (do this with the parameter names maintained)

$\frac{di_a}{dt} =$

$\frac{d\omega_m}{dt} =$

The variables i_a and ω_m are our *state variables*. These are the quantities that will change with time.

Step 2: Next, build these equations using integrators, gain blocks, summing junctions, and a multiplier. Fill in the gain blocks with the appropriate terms and label the summing blocks with '+' or '-' as required.



The variables iaic and ωmic are the initial conditions on the integrators. These may be assigned to these variable names by double-left-clicking the integrator block and typing in the name.

Step 3: Open Simulink (if it is not open) and create the block diagram developed in Step 2. Use a "constant" block for the armature voltage. Include a time clock in the diagram. Use a mux to concatenate time, motor speed, and motor current. Then use a "To Workspace" block to send the matrix to Matlab (make sure to make the switch from structure to array as discussed earlier). Save the Simulink model.

Step 4: Create a Matlab M-file that will assign the following parameters (make sure the names are the same as you used in the Simulink diagram!). Recall, the total inertia is the sum of the motor and reflected load inertias. Run the file so these parameters are then available to your Simulink model.

$R_a = 32.5m\Omega \quad L_a = 14\mu H \qquad K_V = 0.0631 Vs \qquad J_{motor} = 0.0117 kg - m^2$

$B_m = 500 \times 10^{-6} Nms \qquad J_{load} = 0.6883 kg - m^2$

$K_L = 12 \times 10^{-6} Nms^2 \qquad v_a = 24V \qquad i_{aic} = 0.0A \qquad \omega_{mic} = 0.0 rad/s$

Step 5: Set-up a SIMULINK run that captures the startup transient (do this by trial and error as to when the speed settles out…you may choose to place a scope on the motor speed to do this more efficiently). In MATLAB, plot ω_m and i_a versus time.

Once these tasks are done, your notebook should contain

- Information you discovered in Steps 1-4, and how you discovered it
- Equations, Simulink model, M-files, and required plots. They should either be written down or printed out and included in your notebook.

Your individual one page lab summary should be submitted on Canvas before the next lab.

Step 1: Substitute T_e into the mechanical dynamics equation and solve for $\frac{di_a}{dt}$ and $\frac{d\omega_m}{dt}$ (do this with the parameter names maintained)

$$T_e = K_V i_a,$$

$$T_e = J_{tot}\frac{d\omega_m}{dt} + B_m\omega_m + K_L\omega_m^2,$$

$$v_a = R_a i_a + L_a\frac{di_a}{dt} + K_V\omega_m,$$

$\frac{di_a}{dt} =$

$\frac{d\omega_m}{dt} =$

$K_V i_a = J_{tot}\left(\frac{d\omega_m}{dt}\right) + B_m\omega_m + K_L\omega_m^2$

$i_a = \frac{J_{tot}\left(\frac{d\omega_m}{dt}\right) + B_m\omega_m + K_L\omega_m^2}{K_V}$

$\frac{di_a}{dt} = \frac{d}{dt}\left(\frac{J_{tot}\left(\frac{d\omega_m}{dt}\right) + B_m\omega_m + K_L\omega_m^2}{K_V}\right)$

$\frac{di_a}{dt} = \frac{J_{tot}\frac{d^2\omega_m}{dt^2} + B_m\frac{d\omega_m}{dt} + 2K_L\omega_m}{K_V}$

$\frac{di_a}{dt} = \frac{V_a - R_a i_a - K_V\omega_m}{L_a}$

$K_V i_a = J_{tot}\frac{d\omega_m}{dt} + B_m\omega_m + K_L\omega_m^2$

$\frac{d\omega_m}{dt} = \frac{K_V i_a - B_m\omega_m - K_L\omega_m^2}{J_{tot}}$

TCES 455, Autumn 2020

Laboratory 2: Intro to Arduino & MATLAB/Simulink Support for Arduino

What's for today?

- Enhance your understanding of engineering basics on LEDs, resistors, bread board, and wiring
- Your first Arduino sketch 'Blink'
    - Getting familiar with the Arduino environment
    - Getting familiar with the Sketch structure
    - Basic digital & analog output control
    - Making LEDs glow and blink on command
    - Make sure you can compile, upload, and run 'Blink'.
- Your first Simulink model connected with Arduino
    - Getting familiar with the Arduino library in Simulink
    - Making LEDs glow and blink with MATLAB/Simulink environment

1. Your first Arduino sketch 'Blink'

- With reference to Canvas notes Lab slides – ArduinoInstallation (pages 1-12), install Arduino on your computer.

- Read through Canvas notes Lab slides – LED test (pages 1-3), make sure you are comfortable in using those mentioned components. Let the instructor know should you have any question.

- Read through Canvas notes Lab slides – IntrotoArduino, connect the LED, resistor to the Arduino board by referring to the Blink example at Arduino website https://www.arduino.cc/en/Tutorial/BuiltInExamples/Blink.

- In Arduino environment, open Blink.ino by following File -> Examples -> 01.Basics -> Blink. Compile, upload and run 'Blink' with reference to Canvas notes Lab slides – ArduinoInstallation (pages 13-16).

- Save Blink.ino as Blink13 in a folder called Blink13.

- With reference to Canvas notes Lab slides – LED test (pages 4-6), use breadboard to hook up an external LED to pin 13.

- Revise Blink13.ino according to the last page in Canvas notes Lab slides – IntrotoArduino. Compile, upload and run 'Blink13'. What happens?

- Save Blink13.ino as Blink9 in a folder called Blink9. Hook up an external LED to pin 9, change
  ```
  int led = 13;
  ```
  to

```
int led = 9;
```

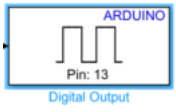
- Run the sketch 'Blink9' to make sure you can blink this LED.
- Replace everything in `loop()` with
  `analogWrite( led, value );`
- Change the value of `value` from 10 to 255; recompile and rerun the sketch each time you change `value`; note what happens to the LED brightness. What's going on here?

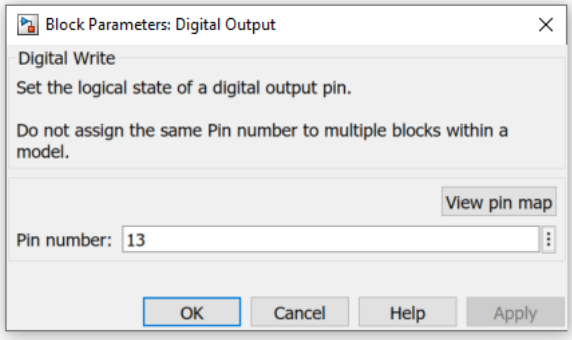
At this point your **notebook** should contain

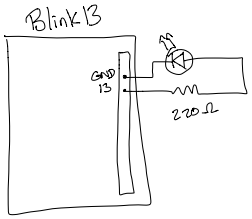
- A schematic of the circuit (include the resistor value).
- Your explanation about why the LED brightness changes.
- Regarding the LED brightness, what will happen if you change the resistor with a different resistance? Try 3 different resistors and write down your comparison.

2. Your first Simulink model connected with Arduino

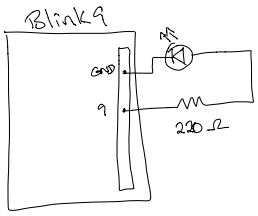
- With reference to Canvas notes Lab slides – SupportInstallation, install MATLAB/Simulink Support Package for Arduino on your computer.
- Have the Arduino board connected with your computer.
- In the MATLAB command window, type in `Simulink`, then create a Simulink model named Blink.slx.
- Find the block Digital Output from Arduino Common Blocks (shown below), and configure the pin to be 13 or 9. Correspondingly, your wiring on breadboard should match this configuration.







The external LED is flashing along w/ the built-in LED, so pin 13 is connected to the virtual pin LED_BUILTIN



- Switching to Pin 9 resulted in the LED flashing every second when connected to Pin 9. The built-in LED was not flashing this time

- Switching to analogWrite() allowed adjustments in the voltage of the circuit. Using a larger value results in the LED shining brighter.

- Switching the resistor to a higher resistance results in a dimmer LED at the same analogWrite() value, which agrees with our experience of how voltage, current, and resistance interact. ($V = IR$) The brightness of a LED is relative to the current flowing through it so a higher resistance will reduce the current through the LED, making the LED dimmer.

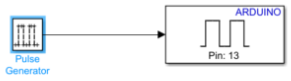
- Installing Matlab Support Package results in an error when testing the connection to the board after programming the board in the previous steps. Many reboots of Matlab, my desktop, and reinstalls of the package have not changed this result

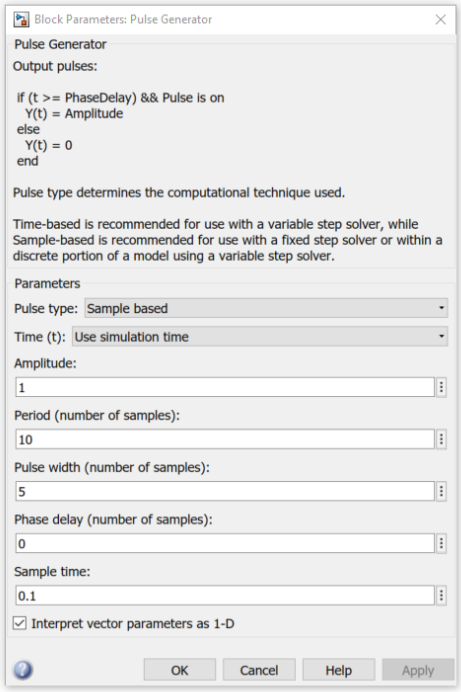
- The Simulink Support Package installed successfully the first time.

- Trying to run the Simulink model originally did nothing. I later found the Hardware tab, where I selected the "Mega 2560" board and was able to compile, build, and send the model to the board. I don't think this is how the lab wants to run the model, but it works!

- After researching on Google, I have found a forum that suggested altering a delayValue in arduino.m value to allow more time for the board to reset and establish a connection. After this change, the Matlab support package successfully makes a connection with the board. I also found that the Simulink support package's help information says to use the "Hardware -> Build, Deploy, and Start" function to get the model working on the board, so maybe the lab instructions are inaccurate? Or this is a separate feature that hasn't been discussed yet.

- Feed the Digital Output block with a pulse generator (shown below from Simulink Sources library). Keep the default configuration.





Tried changing amplitude to change voltage i.e. change current?

So having an amplitude of 0.4 or lower results in the LED not lighting up at all

Are pin 9 and 13 being used as digital pins?

- Run the file Blink.slx. Does it work as you expected? Is there any method to change the brightness of LED? Describe your efforts in the notebook.

Submit to Canvas your one-page summary lab report by 10am Wednesday Oct. 21. The summary has no need to include any code, diagrams, or screen captures. Description of your learning, lab experience, discussion on your observation, comparison of running results are contents expected to see in your summary.

TCES 455, Autumn 2020

Laboratory 3: First Order System (RC circuit) study using Arduino/Simulink

Prior to lab:

- Make sure you have completed Lab 2 and in particular, have installed both MATLAB and Simulink hardware support package for Arduino.

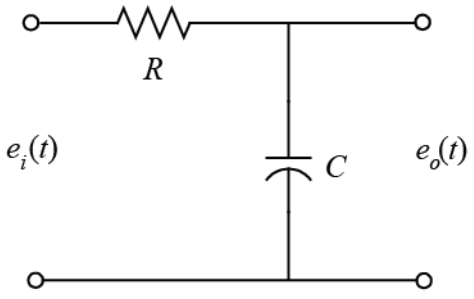
Purpose of this lab:

- to demonstrate how to model a simple electrical system. Specifically, a first principles approach based on the underlying physics of the circuit and a black-box approach based on recorded data will be employed.
- to demonstrate the accuracy of the resulting models by using the black-box approach, i.e., to verify the effectiveness of the approach.
- to provides a physical example of the common class of first-order systems.

1. Mathematic Modeling of System (the first principles approach)

We will first employ our understanding of the underlying physics of the RC circuit to derive the system model – we call this as the first principles approach. In the following RC circuit, variables include:

R - resistance of the resistor
C - capacitance of the capacitor
ei - input voltage
eo - output voltage



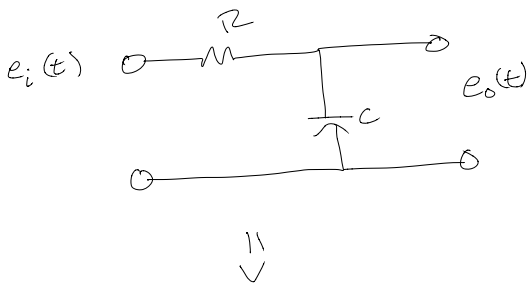
Find the transfer function $G(s) = \frac{E_o(s)}{E_i(s)}$. It should be a first order system with the standard format $\frac{K}{\tau s+1}$, so determine the value of gain *K* and time constant τ for the circuit above.

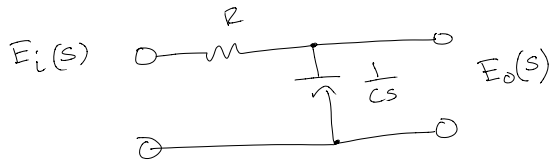
At this point your notebook should contain

- Derivation of the transfer function $G(s) = \frac{E_o(s)}{E_i(s)}$
- Value of *K* and time constant τ for the RC circuit

$R = 10k\Omega$

$C = 100\mu F$





$$E_o(s) = \frac{\frac{1}{Cs}}{R + \frac{1}{Cs}} E_i(s) = \frac{1}{RCs+1} E_i(s)$$

$$G(s) = \frac{E_o(s)}{E_i(s)} = \frac{1}{RCs+1}$$

$\tau = RC$

$K = 1$

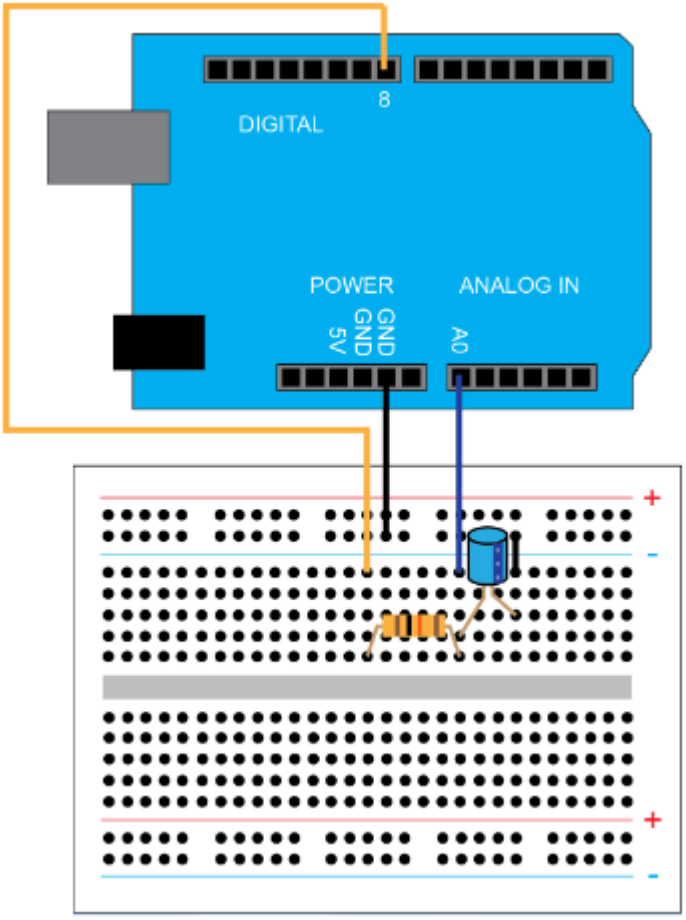
$a = \frac{1}{\tau} = 1\,Hz$

2. System Identification of the RC Circuit

In this part we will record the output voltage of the RC circuit for a step in input voltage. Based on the resulting time response of the output voltage, we will fit a model to the data. This approach is sometimes referred to as black-box modeling or data-driven modeling. After we have generated such a model, we will compare it to the first principles derived model we created previously.

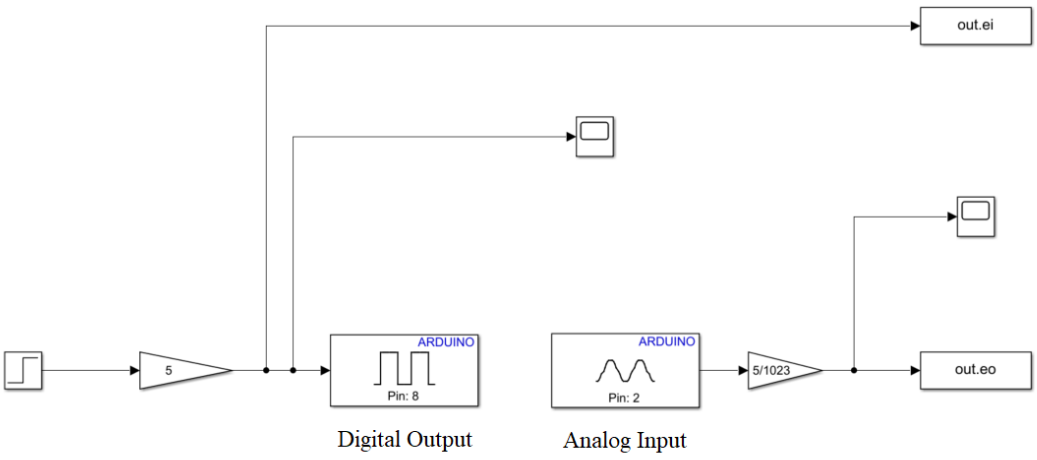
Hardware setup

The Arduino board is employed to receive the input command from Simulink and to apply the input voltage to the circuit (via a Digital Output). The board also acquires the output voltage data from the circuit (via an Analog Input) and communicates the data to Simulink.



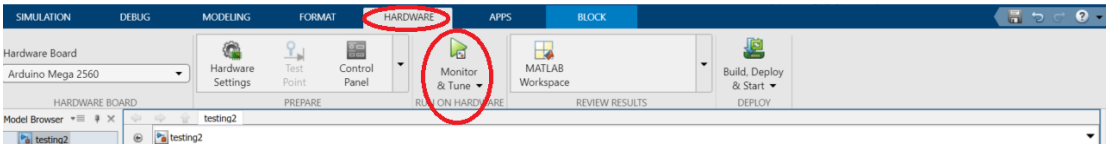
Software setup

We will employ Simulink to read the data from the board and to plot the data in real time. In particular, we will employ the Simulink Hardware Support Package for Arduino from the MathWorks. The Simulink model we will use can refer to the one shown below:

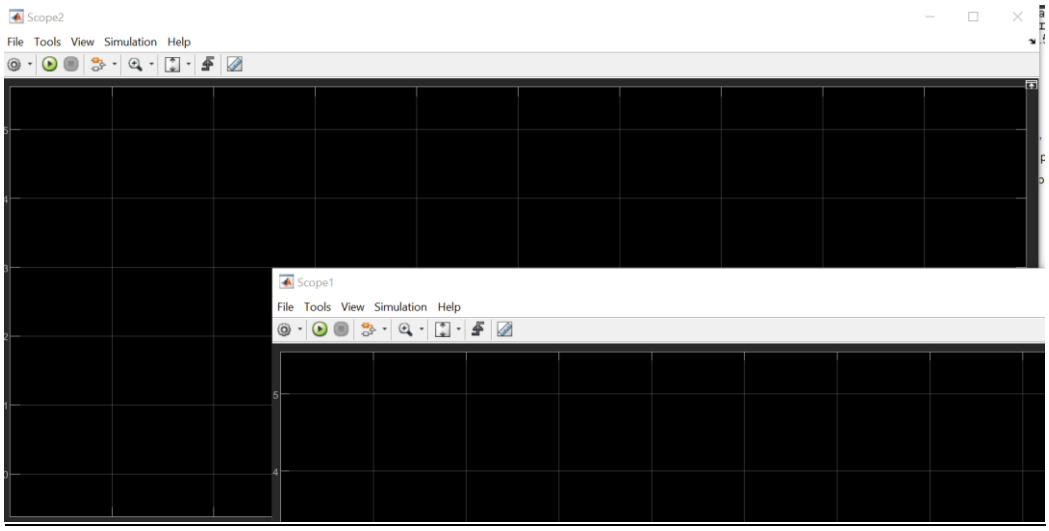


The Arduino Analog Read block reads the output voltage data via the **Analog Input** A2 on the board. Notice that Arduino Mega 2560 has 16 Analog Inputs from A0 to A15 available to use here. Double-clicking on the block allows us to set the Pin to 0 from the dropdown menu. We also will set the Sample Time to "0.1". This is considering the circuit's time constant and hence is sufficiently fast. The other blocks in the model can also be set to have a Sample Time of "0.1" (or left as "-1"). The Gain block connected with the Analog Input is included to convert the data into units of Volts (by multiplying the data by 5/1023). This conversion can be understood by recognizing that the Arduino Board employs a 10-bit analog-to-digital converter, which means (for the default) that an Analog Input channel reads a voltage between 0 and 5 V and slices that range into 2^{10} pieces. Therefore, 0 corresponds to 0 V and 1023 corresponds to 5 V. The Gain block connected with the Digital Output is here to amplifying the unit step input functioning as a voltage source. The step change is set to output 0 for the first 5 seconds of the run in order to discharge the capacitor completely (in case it had built up charge) before stepping to 1 for the last 5 seconds of the run (take the default simulation running time 10 seconds).

Note that once the Simulink model is created, pull down on the menu 'HARDWARE' and click on 'Monitor and Tune' as shown below.



Make sure you have the scopes open when the software is doing 'Monitor and Tune' (as below), so that you will see real-time curves be drawn on the scope.



Parameter identification

During the running of Simulink model, the input voltage and output voltage data get saved to the workspace. Following code can be used as a reference to generate the graph helping you identify the first order model's parameter *K* and τ. Note the code plots only the last five seconds of the run, thereby excluding any discharging of the capacitor that may have taken place.

```
plot(0:0.1:5,ei(51:101),'r--');
hold on
plot(0:0.1:5,eo(51:101),'b*-');
```

At this point your notebook should contain

- The exact MATLAB code you used to plot *ei* and *eo* together, and identify model parameters K and τ.
- Transfer function with identified model parameters K and τ.
- Comments on comparison of the model you identified here with the one by applying the first principles approach.

Model Validation – Free Response

In the previous sections we identified the model of an RC circuit from its step response. That case examined the system's response for a forcing input of a 5-Volt step in input voltage and zero initial conditions (no initial charge on the capacitor). In this section we will investigate how well the model can be predicted by using the RC circuit's free response. This case will assume no forcing input, i.e., *ei = 0*, and an initial charge on the capacitor. For our existing Simulink model, we could run the model for 15 seconds where for the first 5 seconds the input is 0 to discharge the capacitor completely, for the next 5 seconds the input is 1 in order to charge the capacitor completely (i.e., *eo* is 5Volts), and the final 5 seconds represents the free response as the input drops back to 0 (you need to do a little design here at the source, maybe combining 2 step sources, maybe use a pulse source). You can change the simulation run time in the toolbar of the window of your Simulink model, or via the drop-down menu Simulation > Model Configuration Parameters under the Solver menu. Running the existing model under these conditions and executing

At time $t = 10s$,

$e_o = 4.9462$

Assuming for $t > 10s$,

$e_{o,ss} \approx 5V$

$e_{i,ss} = 5V$

$\frac{K}{\tau s + 1}$, $\quad K = \frac{e_{o,ss}}{e_{i,ss}} = 1$

$5 \times (1 - e^{-1}) = 3.16$

@ $t = 1$, $e_o = 3.0792$

@ $t = 1.1$, $e_o = 3.2551$

Ave = 3.167

We'll approximate $\tau = 1.046$

$3.2551 - 3.0792 = 0.1759$

$0.1759 / 100 = 0.001759$

$3.16 = 3.0792 + x(0.001759)$

$x = 45.935 \approx 46$

similar (but modified) MATLAB commands as before to generate a figure helping you to verify the constant τ.

At this point your notebook should contain

- The exact MATLAB code you used to plot *eo*, and identify the time constant τ.
- Comments on the constant τ you obtained here with the one by investigating the step response.

Submit to Canvas your one-page summary lab report by 10am Wednesday Oct. 28. The summary has no need to include any code, diagrams, or screen captures unless you want to have discussion on them. Description of your learning, lab experience, discussion on your observation, comparison of running results are contents expected to see in your summary.

e^{-1} of 5V $\approx$ 1.84V

$t = 6s,\ e_0 = 1.789$

$t = 5.9,\ e_0 = 1.97$

AVE = 1.8795

$\frac{6 + 5.95}{2} = 5.975$

$\tau = 5.975s - 5s = 0.975$

$1.97 - 1.789 = 0.181$

$1.84 = 1.789 + x(0.00181)$

$x(0.00181) = 0.051$

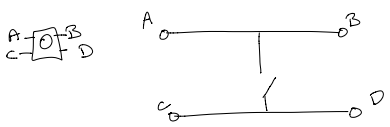
$x = 28.18 \approx 28$

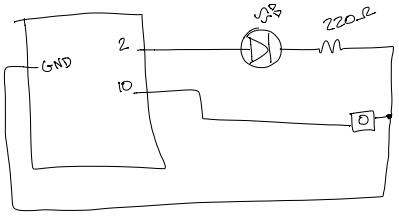
Fade

① × fadeAmount changes frequency of light
× Changing initial brightness value and lower bound of brightness check (that flips the fadeAmount value)
× Changing led changes the active pin.
× Creating three sets of variable (led, brightness, fadeAmount) let me have configurations of three LEDs w/ different starting brightnesses, brightness bounds, and fading frequency.

---

② Button Circuitry





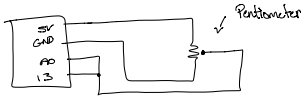
A LOW signal on pin 10 will toggle the "state" variable between 0 and 1. When 0, the LED will blink with a period of 0.5s. When 1, the LED will fade from 0 to 255, w/ a fadeAmount of 8.

---

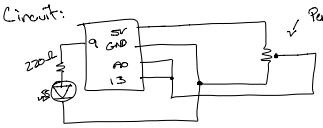
③ Analog Read Serial

- AnalogRead() outputs values between 0 - 1023
- AnalogWrite() takes value between 0 and 255
- Taking AnalogRead() output and multiplying by $\frac{255}{1023}$ converts the AnalogRead() value to its corresponding AnalogWrite() value.
- int scaledSensor = ((double) 255 / 1023) * sensorValue;

First Circuit:



Second Circuit:



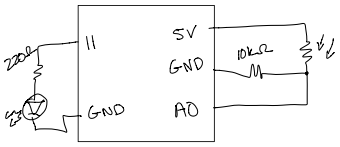
---

④ RGB_LED

- RGB_LED.ino cycles through a "rainbow" of colors by incrementing three pairs of colors.
  1. It starts with a solid red.
  2. Red fades out and green fades in.
  3. Solid green
  4. Green fades out and blue fades in.
  5. Solid blue
  6. Blue fades out and red fades in.

- Each increment is seperated by a 10ms delay.
- This value is stored in delayTime.
  - Changing delayTime value changes the transition speed of the colors.

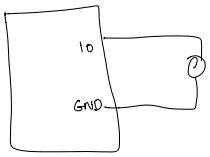
⑤ Photoresistor



- photoresistor changes resistance based off amount of light hitting it.
- The more light, the less resistance. So, if more light hits the photoresistor, the voltage at Pin A0 will be higher.
- I converted the AnalogRead() value to AnalogWrite() with $\frac{255}{1023}$.
- So, the more light hitting the photoresistor = the brighter the LED.

---

⑥ Active Buzzer & Passive Buzzer



- High and low signals seperated by delay(), which changes tone of buzzer. Lower frequency = lower sound
- Sketch -> Include Library -> Add .ZIP Library to add pitches

---

⑦ Serial Monitor

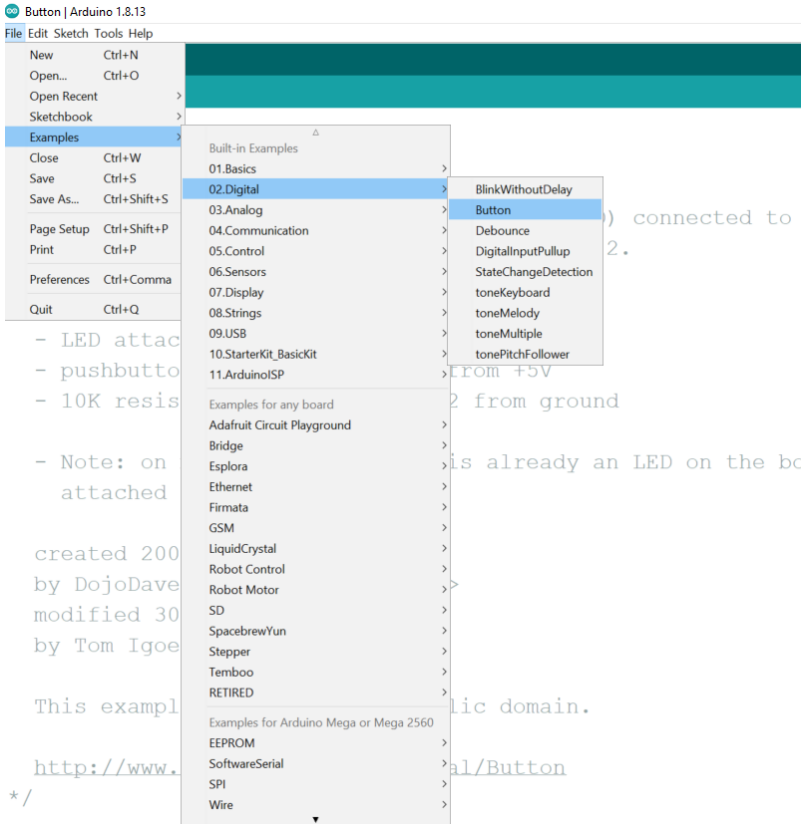
- Similar to print functions in C, C++, Java, etc.

Riley Rudeman

TCES 455, Autumn 2020

Laboratory 5: Arduino Lab Continued - Interrupts

1. As a continuation of Lab4, open, compile, and upload the built-in example 'Button':



Make sure the circuit is connected by following the highlighted instruction.

On your notebook,

- Sketch the connection.
- Explain what state you will read from pin2 when the button is not pressed. Low or high? Is the 10K resistor a pull-up resistor or a pull down resistor?
- Describe what will happen to the LED if you press the button really really fast.

2. Continuing the 'Button' example

- Read the following tutorial https://www.electronics-lab.com/project/arduino-button-debounce-tutorial/
  Note: the Arduino project mentioned in the article is optional.
- Open, compile and run the built-in example 'Debounce' (right below the example 'Button')

On your notebook,

- Briefly explain what is button/switch bounce and debounce.
- Briefly explain the method used in the built-in example 'Debounce' so the press/release of a button can be accurately detected.

3. Now let's start the study of Arduino interrupts

- Read the Arduino interrupt tutorial here.
- Use the example code given in the tutorial to build a project which uses interrupt to control the blinking of LEDs.

On your notebook, answer the following questions:

- Is **HIGH** a predefined valid value on Arduino Mega 2560 to trigger the interrupt whenever the pin is high?
- What digital pins on Arduino Mega 2560 are available and can be used for Interrupt purpose?
- Are digital pins 3 & 18 & 19 able to be programmed to control the blinking of LEDs? How about pins 20 & 21? (*hint:* for pins 20 & 21, useful information can be found on the discussion forum below https://forum.arduino.cc/index.php?topic=317091.0)

4. LEDs can also blink at preset frequencies using interrupts, and in particular, timer generated interrupts.

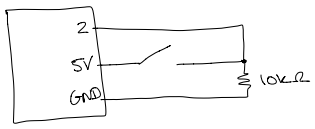
- Read this article on Arduino Timers and Interrupts https://www.robotshop.com/community/forum/t/arduino-101-timers-and-interrupts/13072
- Use the example codes given in the tutorial (minor changes may be needed) to build TWO projects which use Timer generated interrupts to control the blinking of LEDs: one uses the compare match interrupt, the other uses the timer overflow interrupt.

On your notebook, answer the following questions:

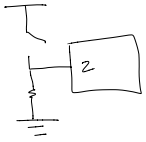
- What values should be written to registers OCR1A (used in timer generated compare match interrupt) and TCNT1 (used in timer generated overflow interrupt), respectively, if the LED is required to blink at a frequency of 5Hz? Show the step-by-step calculation in the notebook.
- Repeat the same calculation for the case when the LED is expected to blink at a frequency of 1Hz. Show the step-by-step calculation in the notebook.

Submit to Canvas your one-page summary lab report by 11:59pm Thursday Nov 12. The summary has no need to include any code, diagrams, or screen captures unless you want to have discussion on them. Description of your learning, lab experience, discussion on your observation, comparison of running results are contents expected to see in your summary.

(1)



When pressed: Pin 2 is 5V (High)
When not pressed: Pin 2 is 0V | GND (LOW)



Pull-down Resistor, since when the switch is not pressed, pin 2 is connected to GND.

* When the button is pressed quickly, a button press may not register, or the LED might flicker.

(2) - Bounce is when the switch output is changing from high to low or low to high, creating a period of noise where the output is not definitely high or low, which may cause problems in deciding which value to interpret during that window of time.

- Debounce is an implementation designed to remove bounce. One way to debounce is to wait an appropriate amount of time after a button press has been registered to verify the actual state of the button and use that verified state.

- Debounce.ino uses ↑ to incorporate debounce.

(3) - HIGH is not predefined on the Mega 2560 to trigger the interrupt.
- Pins 2, 3, 21, 20, 19, 18
- 3 ✓
  18 ✓
  19 ✓
  20 ✓, if the pull-down resistor is a lower resistance (1 kΩ or less)
  21 ✓, same as pin 20

* Pins 20 and 21 have built-in 10kΩ pull-up resistors.

> When the button was connected to GND (w/out pull-down resistor), pushing the button will turn off the board. Is that to protect itself? UPDATE: Switching source to 3.3V fixes the issue. Pins 20 and 21 can work w/ no external resistors at a source of 3.3V.
> ↳ This toggles LED sometimes when button is pressed.

(4) System clock of most Arduino boards: 16 MHz

5Hz - OCR1A = System Clock Frequency / Prescaler / Desired Freq

= 16 MHz | 256 | 5 Hz
= 12500

5Hz - TCNT1 = Size of timer register (bits) - [System clock freq. / Prescaler / Desired Freq.]

= 65536 - [16 MHz | 256 | 5 Hz]
= 53036

1Hz – OCR1A = 16MHz/256/1Hz = 62500

– TCNT1 = 65536 - 16MHz/256/1Hz = 3036

Riley Buckman

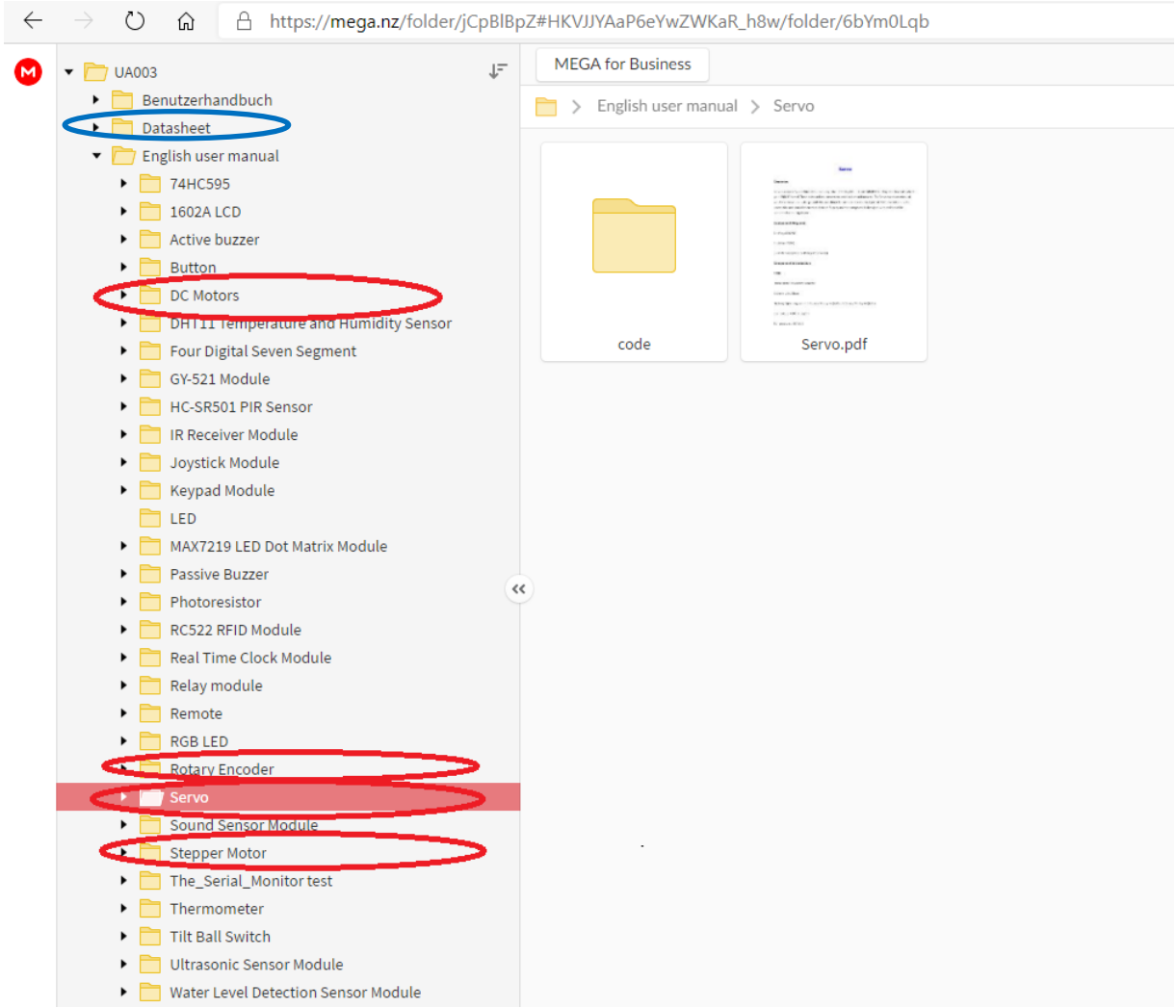
TCES 455, Autumn 2020

Laboratory 5: Arduino Lab Continued – Actuators/Motors

Prior to the lab, get the following components prepared from the UA003 Starter Kit:

- Servo motor (sg90) ✓
- 3v Servo motor (DC motor)
- IC L293D chip ✓
- Breadboard power supply module ✓
- Stepper Motor ✓
- ULN2003 Stepper Motor Driver Board ✓
- Rotary encoder module ✓

Also find from the CD, or, go to the link : https://mega.nz/folder/jCpBlBpZ#HKVJJYAaP6eYwZWKaR_h8w/folder/uCpizKCZ
find relevant folders containing detailed description of these components and sample codes, as circled in red below. Relevant datasheet can be found, too, in the folder "Datasheet" circled in blue.



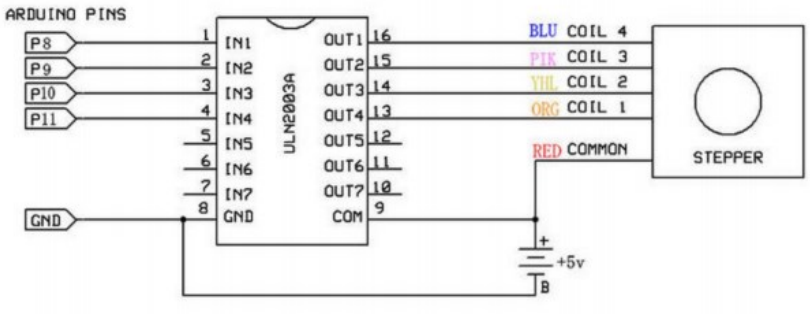
1. **Open the UA003 Starter kit document folder "Servo", read the pdf file 'Servo'.**
   - A library file (Servo.zip) is provided to be added to run the servo motor. Read the manual on Canvas for details regarding adding libraries (pages 12-16), and in particular, pages 15-16 for this lab.
   - There is a typo regarding the number of Arduino digital pin connected to the motor's pulse wire. Both the wiring diagram and the code use Pin #11, although it said Pin #9 at the beginning of the pdf file. Don't get confused by the pin number.
   - Open, compile and upload the sample sketch 'servo.ino'.

   On your notebook,
   - Explain how a servo motor works. Can a servo motor rotate continuously to, say, 360°?
   - Briefly describe what you observed when the sample code got executed.

2. **Open the UA003 Starter kit document folder "DC Motors", read the pdf file 'DC Motors'.**
   - Have components connected by following the schematic and wiring diagram given in the pdf file.
   - Notice that there is a white switch button next to the power connector of the breadboard power supply module. Only when you press it and see the red LED on, then the power gets delivered.
   - Open, compile and upload the sample sketch 'DC_Motor.ino'.
   - Keep the breadboard power supply module for next lab task.

   On your notebook,
   - Explain why the IC L293 chip is used here.
   - Explain how you change the direction of motor's rotation.
   - Explain how you change the speed of motor's rotation.
   - What do you observe when the function in the sample code delay() takes different values?
   - Compare and comment on the commonness and difference between a DC motor and a servo motor.

3. **Open the UA003 Starter kit document folder "Stepper Motor", read the pdf file 'Stepper Motor'.**
   - Have components connected as the schematic given below. Other wiring diagrams in the pdf file have messed up connection between Arduino pins and stepper motor driver pins.



- A library file (Stepper.zip) is provided to be added to run the stepper motor. Read the manual on Canvas for details regarding adding libraries (pages 12-16), and in particular, pages 15-16 for this lab.
- Again, notice that there is a white switch button next to the power connector of the breadboard power supply module. Only when you press it and see the red LED on, then the power gets delivered.
- Open, compile and upload the sample sketch 'stepper_oneRevolution.ino'.
- Keep all the components and connections for next lab task.

On your notebook,

- Describe what you observed when the code got executed.
- Compare and comment on the commonness and difference among a DC motor, a servo motor, and a stepper motor.

4. **Open the UA003 Starter kit document folder "Rotary Encoder", read the pdf file 'Controlling Stepper Motor with Rotary Encoder'.**
   - Keep the components and connection from last lab task, add the rotary encoder by following the schematic and wiring diagram given in the pdf file.
   - Make sure the white switch button (next to the power connector of the breadboard power supply module) is pressed and the red LED is on.
   - Open, compile and upload the sample sketch 'With_Encoder.ino'.

   On your notebook,

   - Describe what you observed when the code got executed.
   - Explain what the usage is of the rotary encoder in this lab task. Read the tutorial on rotary encoder included in its datasheet can help you understand it thoroughly.
   - Explain what the usage is of the interrupt in this lab task. What happens when the interrupt gets detected?

Submit to Canvas your one-page summary lab report by 11:59pm Sunday Nov 29. The summary has no need to include any code, diagrams, or screen captures unless you want to have discussion on them. Description of your learning, lab experience, discussion on your observation, comparison of running results are contents expected to see in your summary.

On your notebook,

- Explain how a servo motor works. Can a servo motor rotate continuously to, say, 360°?
- Briefly describe what you observed when the sample code got executed.

Servo Motor

- A servo motor is a DC motor, a gearbox that provides a good amount of torque. A servo motor cannot go past 180°, 90° each way from the starting position.
- When the code is executed, the motor steps through its full range of motion in one direction and then the other direction.

On your notebook,

- Explain why the IC L293 chip is used here.
- Explain how you change the direction of motor's rotation.
- Explain how you change the speed of motor's rotation.
- What do you observe when the function in the sample code delay() takes different values? ?
- Compare and comment on the commonness and difference between a DC motor and a servo motor.

DC Motor

Needed to connect Arduino GND to common GND.

- The IC L293 chip is used to control the behavior of the DC motor. An enable pin and two direction pins allow a motor to be spun in either direction (clockwise or counter-clockwise) and with a variable speed.
- Two direction pins, pins 2 and 7, change the direction of the motor. To enable a direction, one is set to HIGH and the other is set to LOW.
- The enable pin, pin 1, specifies the amount of current allowed through the motor.
- If delay() takes larger values, it allows more time for the IC to reach its "desired" state. When delay() takes smaller values, the inputs change more often, so the IC never reaches its "desired" state.

- DC and servo motors are used in a multitude of different applications. DC motors are desired for variable, continuous circular motion like a car or turbine. A servo motor has limited movement but is able to makes finer changes to its rotation, and its arm. This allows uses in applications that require precise movements, like manufacturing.
  - DC motor needs a driver IC.

On your notebook,

- Describe what you observed when the code got executed.
- Compare and comment on the commonness and difference among a DC motor, a servo motor, and a stepper motor.

Stepper Motor

- When the code executed, the stepper motor rotated clockwise and then clockwise continuously. When rotating, the LEDs on the driver board are all on. When switching between rotation directions, only LEDs A and B are on. During each rotation, about 1.5 revolutions occur. The motor is warm and vibrates

- The stepper motor seems to be a combination of the DC and servo motor, where it's able to be precisely rotated while providing full 360° motion. The stepper motor needs, at least, its own driver IC and appropriate driver circuits to operate successfully. It needs 4 pins its behavior, instead of 3 pins for the DC and servo motor.

On your notebook,

- Describe what you observed when the code got executed.
- Explain what the usage is of the rotary encoder in this lab task. Read the tutorial on rotary encoder included in its datasheet can help you understand it thoroughly.
- Explain what the usage is of the interrupt in this lab task. What happens when the interrupt gets detected?

- Nothing occurs until the the rotary encoder is rotated. When the rotary encoder is rotated, the stepper motor rotates about 10° in the opposite direction of rotation. It takes around 30-32 rotations of the encoder to complete a rotation of the stepper motor. Spinning the rotary encoder quickly "confuses" the motor: the motor will rotate back and forth "in place" a couple times, which results in little total rotation occuring. Pressing the rotary encoder like a button makes the rotary encoder return to the position it was when the program started.

- The rotary encoder is used to operate the stepper motor by rotating the motor step-by-step, and returning it to its original position. The CLK pin is used to send an interrupt signal to the Arduino. The DT pin sends the direction of the latest rotation. The SW pin sends a signal when the rotary encoder is pressed.

When the encoder is used.

- The use of the interrupt is to provide responsiveness when the rotary encoder is used. When the interrupt is detected, a delay() first occurs for Debouncing. Next, the CLK signal is checked to be HIGH or LOW. If HIGH, the rotationDirection is set to the DT pin. If LOW, it's set to the NOT of the DT pin. TurnDetected is then set to true to allow the code to rotate the stepper motor with the current variable values.