Question 1 (Hardware Setup - 75 points)**.** The goal of this question is to get your hardware setup up and running. We will start with the hardware.

A Hardware

A.1 Materials

1. DCMotor (Pololu 19:1 Metal Gearmotor 37Dx68L mm 12V with 64 CPR Encoder)
2. Microcontroller (Arduino Mega 2560)
3. Motor Shield (DFRobot DRI0009 OR Arduino A000079)
4. 12V 3A DC Power Supply (Universal AC Adapter with 2.1mm x 5.5mm Male Connector)
5. 8 Wires (20cm Male To Male Jumper Wire)
6. 3-D Printed Base
7. 3-D Printed Load
8. 3-D Printed Motor Clamp A
9. 3-D Printed Motor Clamp B
10. 4 - #4-40 1/4" Screws
11. 4 - #4-40 3/8" Screws
12. 2 - #5-40 3/8" Screws
13. 4 - #4-40 Hex Nuts
14. Female Barrel Jack (2.1mm x 5.5mm Female Power Jack Adapter)
15. Sticky Tack
16. USB B to A Converter Cable (USB 2.0 A-Male to B-Male Cable)

A.2 Hardware setup

1. Jumper the pins of the DRI0009 as a PWM input. Figure 2 shows the correct setup of DRI0009 with jumpers on the eight top right pins (labeled as M2, E2, E1, and M1 from left to right). These jumpers ensure that DRI0009 is configured to accept PWM signals to drive the DC motor.
2. Additionally, DRI0009 has 6 pins in the bottom left-hand corner of the board, which are also shown in Figure 2. Jumper the pins to accept an external power supply in order to power the DC motor. This requires that the left four pins out of the six pins are jumpered, with the top-left two pins jumpered together and the bottom-left two pins jumpered together.
3. Take the #4-40 **1/4"** screws and a screw the Arduino board onto the 3D printed base. See Figure 3.
4. Take the motor shield and plug it into the Arduino board. Make sure that the pin labeled `SCL` on the motor shield is plugged into the `SCL1` pin on the Arduino. Additionally, make sure

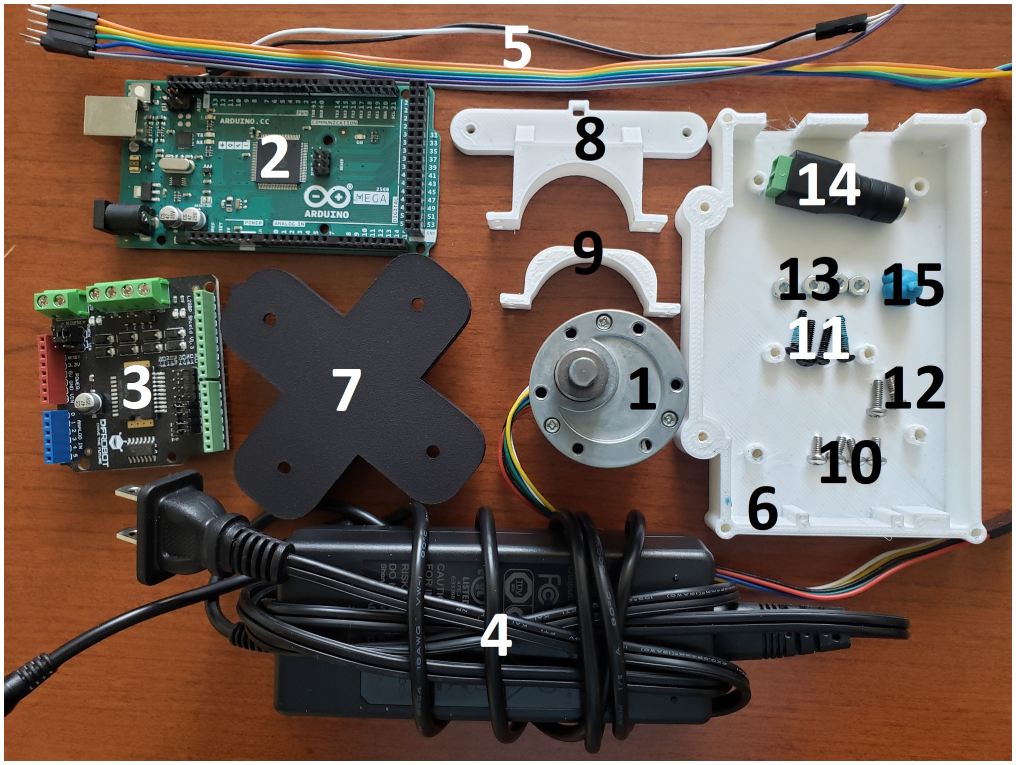


Figure 1: Hardware required for the experimental portions of this class

that the `IOREF` pin is aligned with the `IOREF` pin on the Arduino. This will ensure that the motor shield is connected properly to the Arduino. See Figure 3.

5. Next, take the female barrel jack and screw on a red wire to the positive and a black wire to the negative end. Each connector port has screw terminals, so clamp them down on the wires by using a small screw driver. Connect the other end to the **screw terminals** on the motor shield, the positive wire goes to `PWRIN +` on DRI0009 (`Vin` on A000079) and the negative wire goes to `PWRIN -` on DRI0009 (`GND` on A000079). See Figure 3.

6. Then, take the two **#5-40** 3/8" screws and attach motor clamp A to the base as shown in Figure 4.

7. Clamp the DC motor between motor clamps A and B using the four **#4-40** 3/8" screws and the four #4-40 hex nuts as shown in Figure 4. The clamp is plastic, it will break if you over tighten the nuts. Just tighten them enough to make sure that the motor does not rotate. Check that they are tight periodically throughout the semester.

8. Connect the remaining 6 jumper wires into the motor wiring harness as shown in Figure 5.

9. Motor power connections: the jumper wire connected to the red wire coming from the motor is connected to `M1+` on DRI0009 (`B/-` on A000079) and the jumper wire connected to the black wire coming from the motor is connected to `M1-` on DRI0009 (`B/+` on A000079). See Figure 5.

10. Encoder power connections: the jumper wire connected to the blue wire coming from the motor is connected to a `5V` pin on the motor shield and the jumper wire connected to the green wire coming from the motor is connected to a `GND` pin on the motor shield (Figure 6).

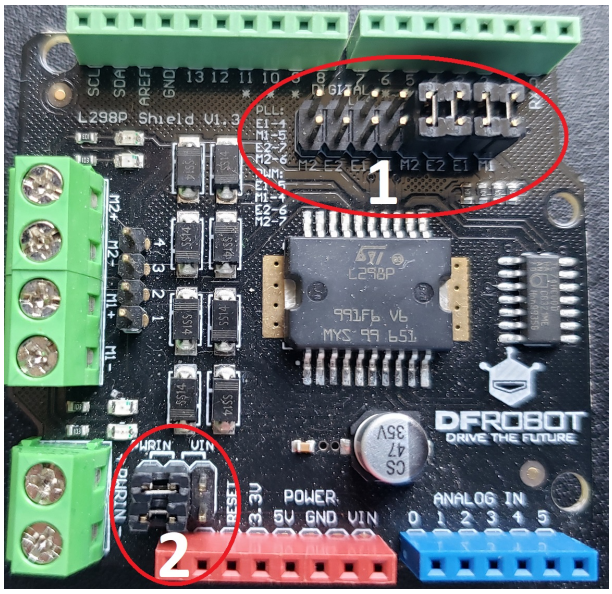


Figure 2: Motor shield jumper configuration.

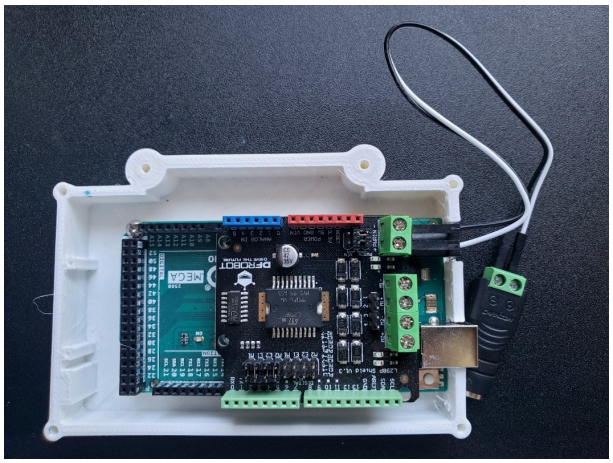


Figure 3: Motor shield and Arduino mounted on the base with the barrel jack attached.

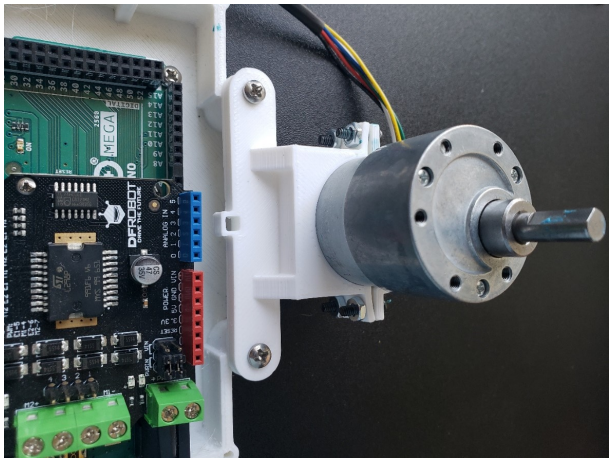


Figure 4: Motor mounted on the base.

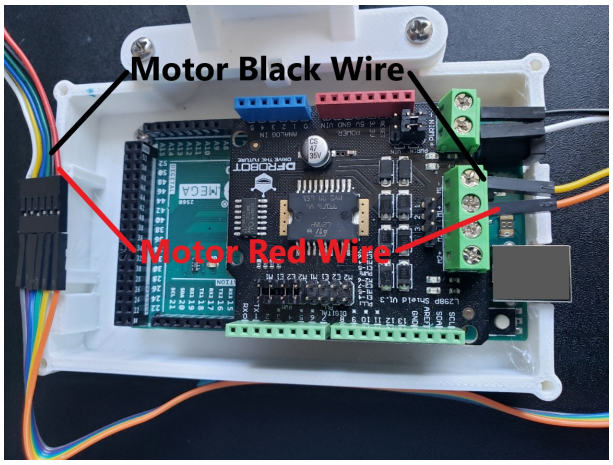


Figure 5: Motor power connections.

11. Encoder signal connections: the jumper wire connected to encoder A (the yellow wire coming from the motor) is connected to Pin 2 on the motor shield and the jumper wire connected to encoder B (the white wire coming from the motor) is connected to Pin 3 on the motor shield (Figure 7).

12. Insert the D-shaft of the motor into the D-shaped receptacle on the load. If it is lose, add a small amount of sticky tack into the receptacle.

13. To stop the base from moving, put a small sphere of sticky tack in each corner of the base and then press the base onto a flat surface. Once you are done, your setup should look like Figure 8

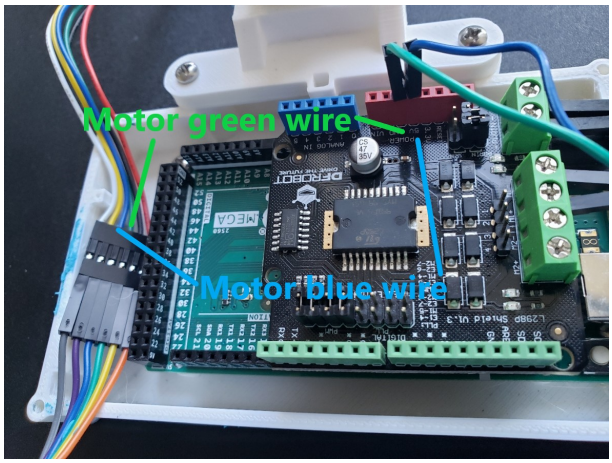


Figure 6: Encoder power connections.

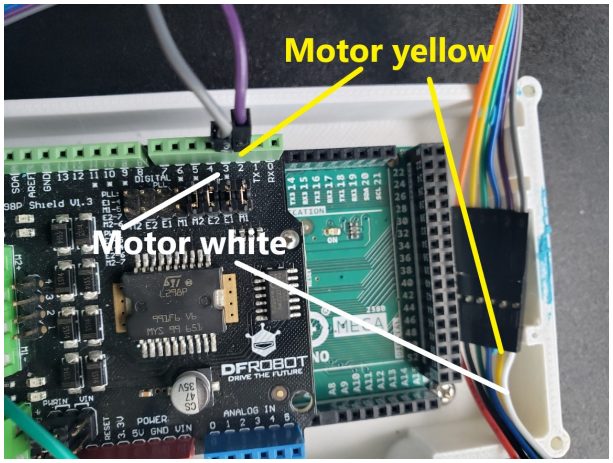


Figure 7: Encoder data connections.

B Software

B.1 Software needed

- Matlab/Simulink R2020a
- Arduino Support from Simulink
- Take home labs Simulink libraries.

B.2 Software setup

This section will describe how to set up the Simulink diagram used to run the simple DC motor, as well as how to setup the drivers for the Arduino. MATLAB 2020a with Simulink should be installed before beginning this section. (Other versions of MATLAB may work, but the 2020a version has been tested.)

1. Install the Arduino IDE by following this guide. Keep all the setup options at their default values.
2. When the install finishes, connect the USB cable from the Arduino to your computer. The green LED on the Arduino should turn on, indicating the board is being powered.
3. Wait for your computer to search for and install the device driver software.
4. Navigate to the *Device Manager* on the computer, and check that the Arduino shows under *Ports(COM & LPT)*. The Arduino Mega should be listed as *ArduinoMega 2560 (COMx)* where x is the communication port number being used. Write down the communication port number, we will need that later.
5. Install the Arduino Support from Simulink add-on using instructions provided on this web-page. Click on the *System requirements and installation options* link under the *Ready to install?* banner for installation instructions and options.

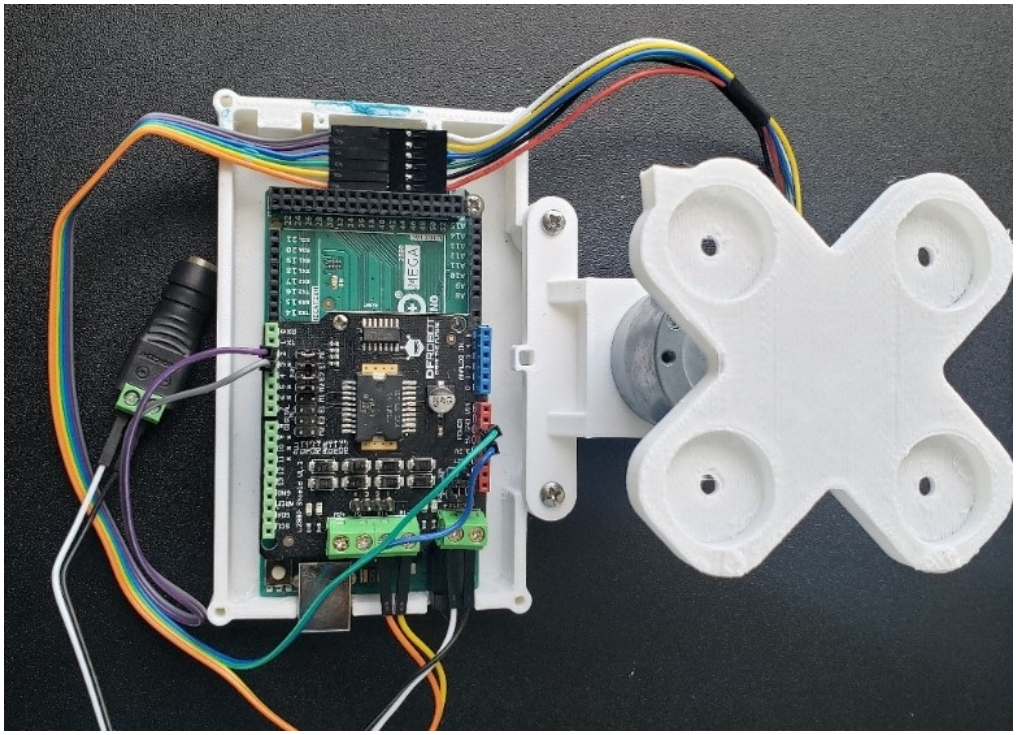


Figure 8: Complete hardware setup.

6. Once the installation is complete, type *simulink* in the MATLAB command window to open Simulink. A *Simulink Start Page* will open up. Click on *BlankModel*.

7. Now find the *Library Browser* button on the menu bar and click on it to open the Library Browser.

8. At this point, the *Simulink Support Package for Arduino Hardware* should be visible as one of the options on the left half side of the Library Browser (the portion with a scroll bar).

B.3 Basic motor speed manipulation

We have a 12V adapter, with no way to change its voltage output. Given that, if we want to apply 6V to the motor to make it run slower, we will have to rely on *Pulse Width Modulation*, or PWM. PWM is comprised of a series of ON/OFF pulses of constant magnitude, sent at a constant frequency. The percent of time that the pulse is ON (called the duty cycle) determines the effective voltage seen by a DC motor. A PWM signal is defined by its magnitude, frequency, and duty cycle.

In our case, since we are going to have a 12V power adapter connected to our motor shield, our PWM signal will be comprised of ON/OFF pulses of 12V magnitude, sent (under default settings) at a frequency of 490.2 Hz. At 50% duty cycle, our motor will run as if we were using a 6V input.

1. Click on *Simulink Support Package for Arduino Hardware* to see its contents in the right hand pane, and double click on *Common*. Various Arduino blocks should now be visible. Click, hold, and drag the block labeled PWM and drop it onto the blank Simulink model window.

2. Double-click the PWM block and input 5 as the Pin number if you have DRI0009 (pin 11 if you have A000079). Pin 5 on DRI0009 (11 on A000079) is used to specify the duty cycle

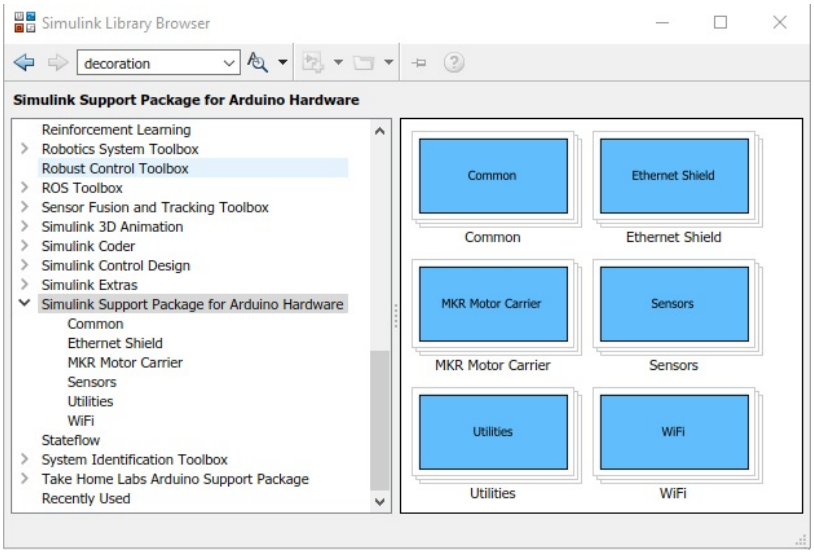


Figure 9: Simulink Library Browser.

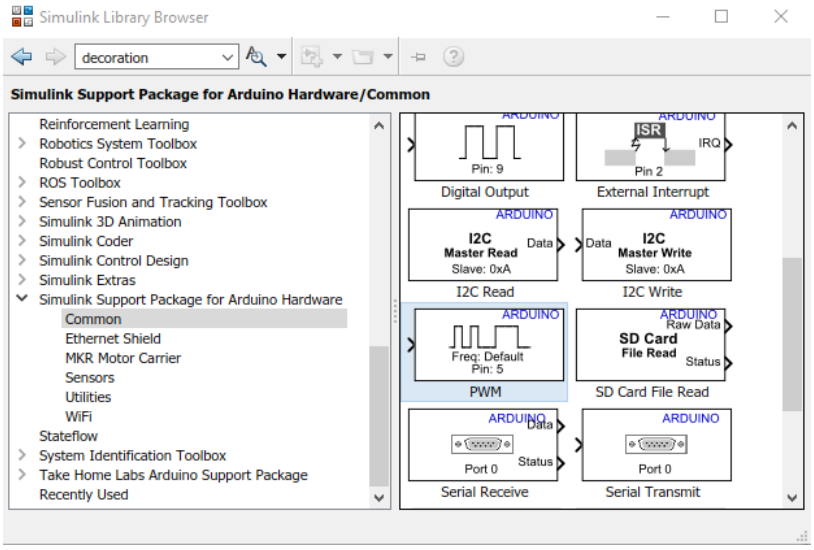


Figure 10: PWM block from Simulink Support Package for Arduino Hardware.

of the voltage being applied to the DC motor. It accepts a value between 0 and 255, which varies the duty cycle from 0% to 100% percent in a PWM wave.

3. Go back to the Library Browser and click, hold, and drag the block labeled *Digital Output* and drop it on to the Simulink model window. Double click on it and set it to Pin 4 if you have DRI0009 (Pin 13 if you have A000079). Pin 4 on DRI0009 (13 on A000079) is used to specify the direction of rotation for the DC motor.

4. For control applications, it is useful to be able to directly send a desired effective voltage command, between 0V and 12V, to the motor, as opposed to sending a duty cycle command between 0 and 255. To do that, we will use *Commonly Used Blocks* from the Simulink library browser to create the Simulink model in Figure 11

5. Before continuing make sure that the Arduino board is connected to your computer via USB.

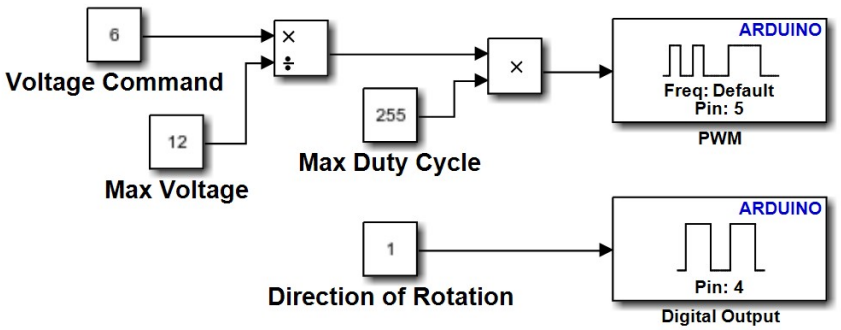


Figure 11: Simulink model for sending voltage commands to the motor via DRI0009.

6. In Simulink, go to the *MODELING* tab and select *Model Settings*.
7. Set the solver type to *Fixed-step*, the solver to *ode1*, and the *Stop time* to *Inf*.
8. Expand the *Solver details* dialog and set the fixed-step size to 0.01. See Figure 12.
9. In the left hand panel, click on *Hardware Implementation* and set the hardware board to *Arduino Mega 2560*.
10. Expand the *Target hardware resources* dialog box and make sure that under *Host-board connection*, the *Set host COM port* option is set to *Automatically*.
11. Click *Apply* and return to the Simulink model.
12. To power the motor, plug the DC power supply into an outlet, and connect the male end of the supply to the female barrel jack that is connected to the motor shield. Note: it is recommended to unplug the DC power supply from the motor every time something is being uploaded to the board.
13. Make sure that the load on the motor has space to rotate without hitting things.
14. In Simulink, go to the *Hardware* tab and click on *Build, Deploy, & Start*.
15. Change the Voltage Command to 2, 6, and 12 and make sure that the speed of the motor changes accordingly. Change the Direction of rotation from 1 to 0 and make sure that the motor changes its direction of rotation as expected. **Every time you change anything, for it to take effect, you have to click on *Build, Deploy, & Start*. Also, in this mode, the code runs on the Arduino, and as such, there is no way to stop it. If you want to stop it, send a zero voltage command.**
16. To avoid having to click on *Build, Deploy, & Start* every time you want to change a parameter, you can use *Simulink External Mode*. To switch to external mode, click on *Monitor & Tune* under the *Hardware* tab. In this mode, the Arduino acts as a conduit between your computer and the motor, and it is your computer that is actually controlling the motor. Now, you can change the voltage and the direction *on the fly* without having to click on *Build, Deploy, & Start*. Try a few different voltage and direction commands to make sure everything works as expected. The downside here is that since the code is running on your computer and not on

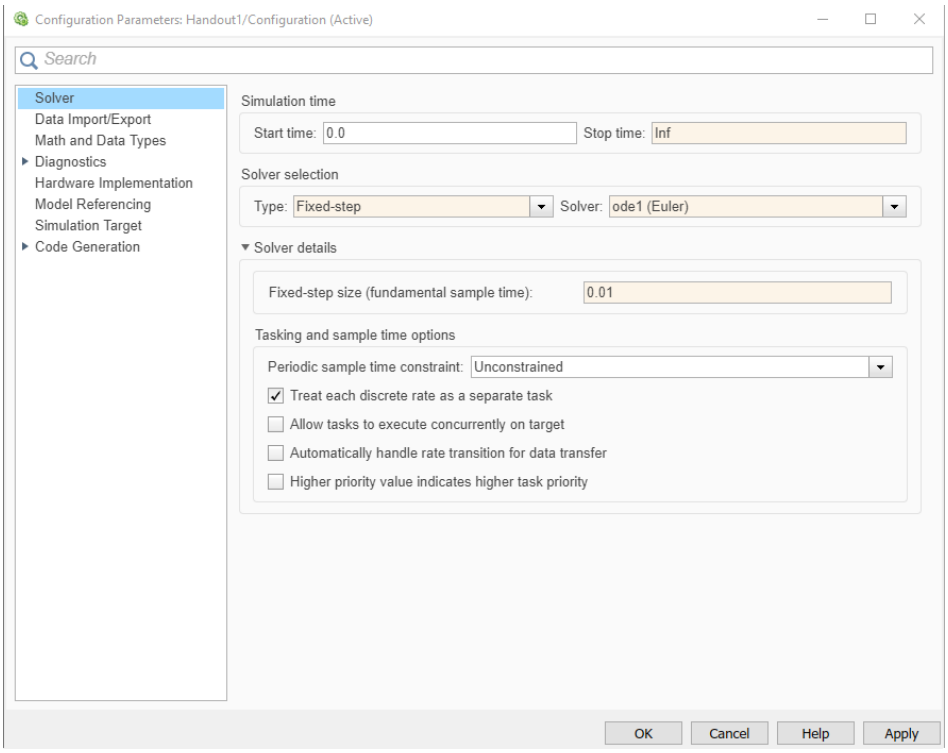


Figure 12: Solver settings.

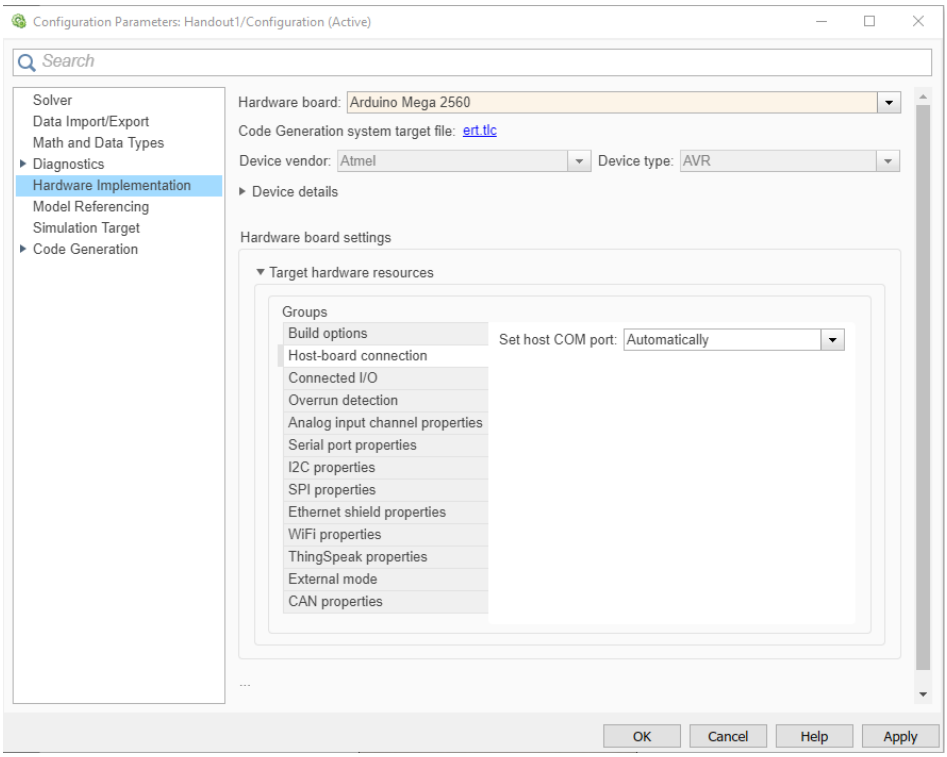
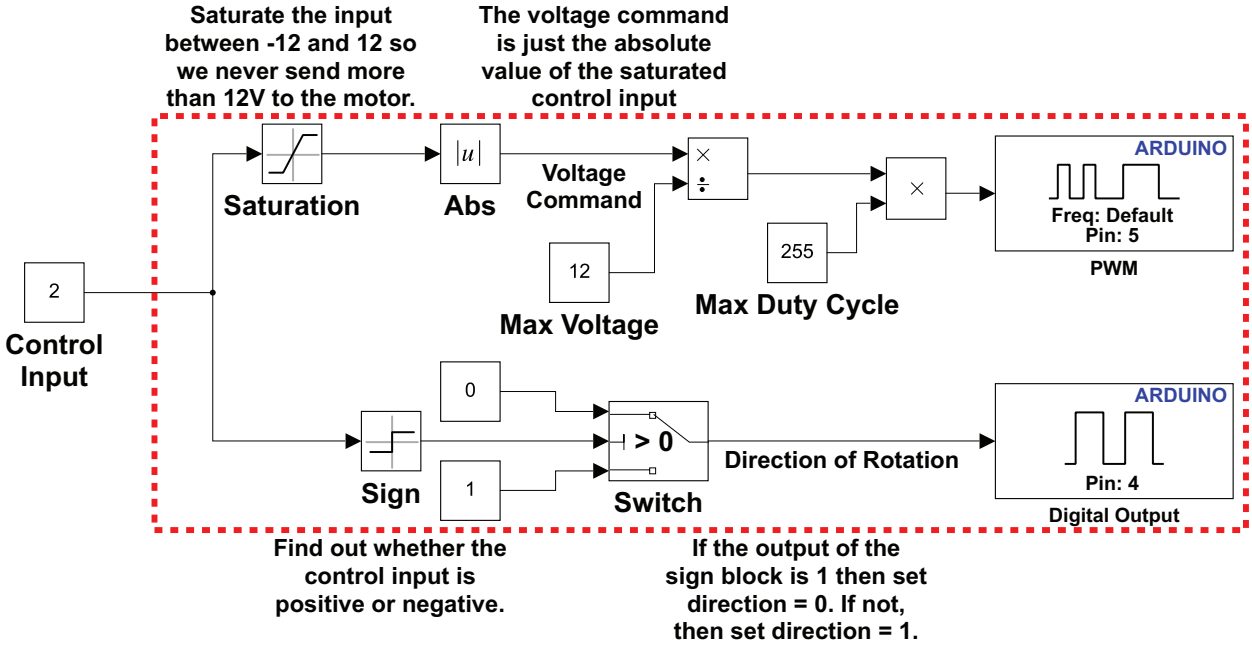


Figure 13: Hardware implementation settings.

the Arduino, **any communication delays between your computer and the Arduino will affect the performance of your experiment.** In this mode, you can click *Stop* to stop the experiment, but it is good practice to send a zero voltage command before hitting *Stop*.

17. When we design controllers, we will get a *control input* for the motor, which will be a number between -12 and 12. We will have to convert this into *Voltage* and *Direction* to be able to use the model we created above. To do that, you can do something like this:



Do not forget to open the *Saturation* block and change the limits to -12 and 12. Otherwise, you will just hear a 490 Hz sound, but the motor will not move. You can select everything in the dashed red box, right click and select *Create Subsystem from Selection* to create a subsystem. Save the resulting simulation in a directory on your computer.

18. Send a few positive and negative control inputs and make sure that the motor behaves as expected.

B.4 Motor angular position sensing

Now we will learn how to measure the angular position of the motor. For this part, you need a custom-made *Simulink* library.

1. Download *Take Home Labs support package for Arduino* (*Files ->Software for take home experiment ->THLlib.zip* on Canvas). Extract the contents of the zip archive to another directory on your computer. Once you do this, you will have a file

   ```
   C:\YourPath\THLlib\THLlib.slx
   ```

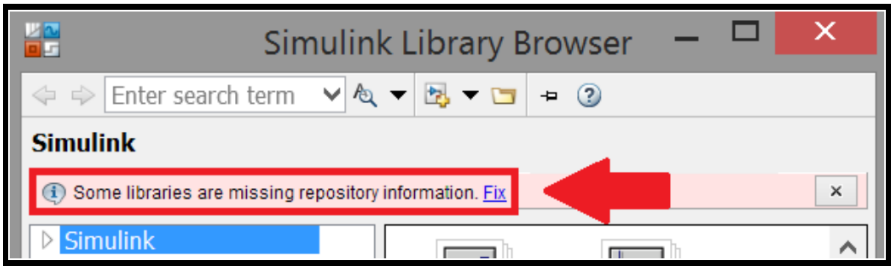
   somewhere on your computer, where YourPath is the path where you extracted the zip file.

2. Go to *MATLAB* and run the command

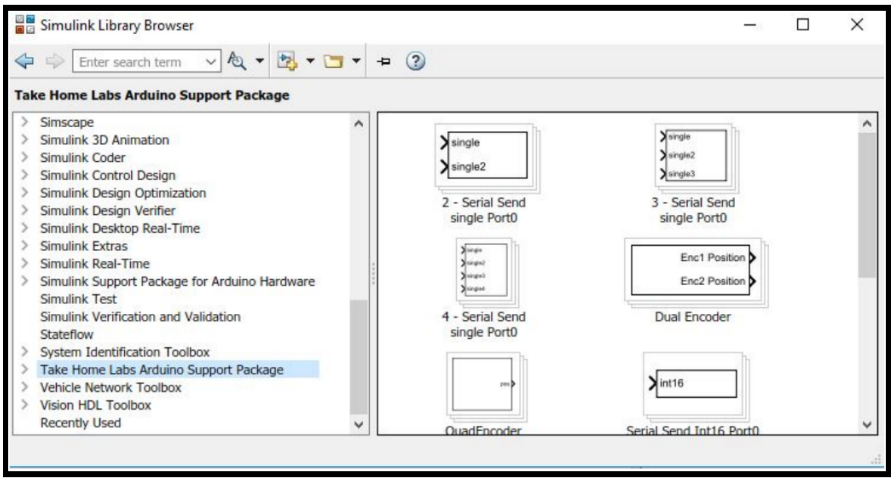
```
addpath('C:\YourPath\THLlib');
savepath;
```

This will ensure that the files in the *THLlib* directory are always available for use in *MATLAB*.

3. Now, open your *Simulink* model and open the *Simulink Library Browser*. Right click somewhere inside the left hand pane and click *Refresh Library Browser*. After the library refreshes, the message *"Some libraries are missing repository information. Fix"* will appear at the top of the window.

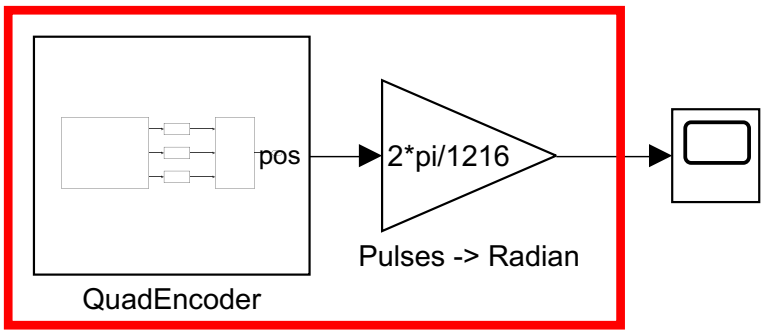


Press *Fix*, select "*Resave libraries in SLX file format*" on the pop-up that appears and press OK. Now, you will see a library by the name *Take Home Labs Arduino Support Package* in the left hand pane.

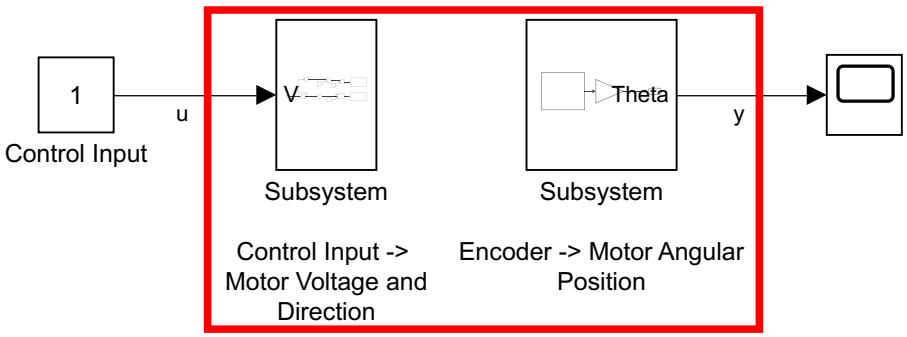


4. Click on that and find a block called *QuadEncoder* and add it to your model. This block will read the encoder attached to your motor and output the number of encoder pulses, which corresponds to the angular position of the motor.

5. The encoder sends 64 pulses per revolution of the motor rotor, and the motor has a gearbox with a 18.75:1 gear ratio. So, at the D-shaft, we get $64 \times 18.75 = 1216$ pulses per revolution. Thus, to get the angular position of the motor shaft, we can do something like this:



6. You can select everything in the dashed red box, right click and select *Create Subsystem from Selection* to create a subsystem. Save the resulting simulation in a directory on your computer. The simulation now looks like an input-output system:



   You can select the two subsystems, right click and select *Create Subsystem from Selection* to create a subsystem named *System*.

7. **IMPORTANT:** Using *External Mode*, make sure that when you send in a **positive** control input, the direction of motor rotation is such that your encoder reading **increases**, and when you send in a **negative** control input, your encoder reading **decreases**. If you get the opposite behavior, switch the connections for the red and the black wire coming from the motor.

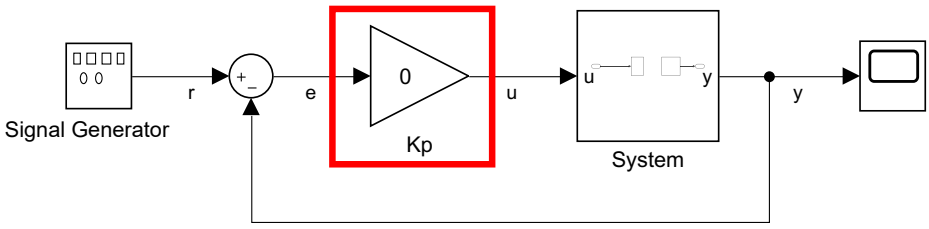
C Our very first feedback controller

All we have to do now is to take the output y, do some computations, and figure out what the input u should be to achieve the desired tasks. Let us assume that we want our motor shaft to track a square wave of amplitude 1 rad.

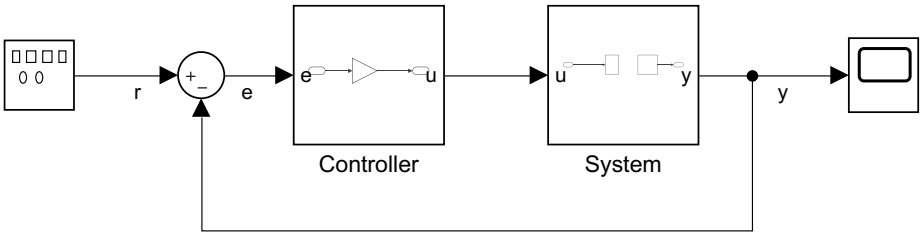
1. Add a *Signal Generator* block to your simulation, and set it to produce a square wave at 1 Hz frequency. This will be our reference signal r.

2. Subtract the output y from the reference r to get the error signal e.

3. For now, we will just use a proportional controller, i.e., $u = K_p e$. Add a *Gain* block to your simulation and connect it to generate the signal $K_p e$, and connect it to the input port of the system. Your simulation should now look like this:



4. You can select the *Gain* block, right click and select *Create Subsystem from Selection* to create a subsystem named *Controller* to get:

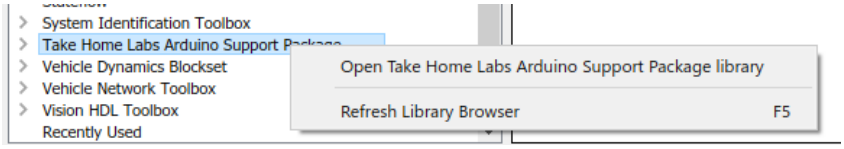


This is an implementation of a proportional controller. To implement more complex controllers, we will change the contents of the *Controller* subsystem.

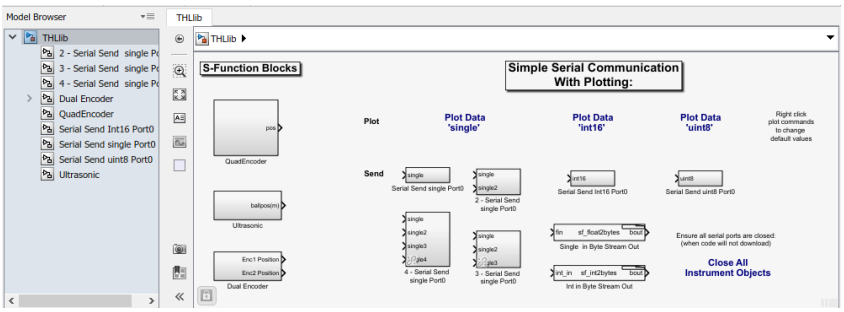
5. Further instructions for *Normal Mode* (skip if you plan to use *External Mode*):

   (a) In *Normal Mode*, i.e., when you press *Build, Deploy & Start*, there is no communication between MATLAB and Arduino once the code execution starts. So, naturally, **the scope will not show anything**. To get the encoder readings out of the Arduino when running in Normal Mode, we need serial communication. To do this, add the *2 - Serial Send single Port0* block to your model from the *Take Home Labs Arduino Support Package* library. Connect the output y of the system and the reference r to this block.

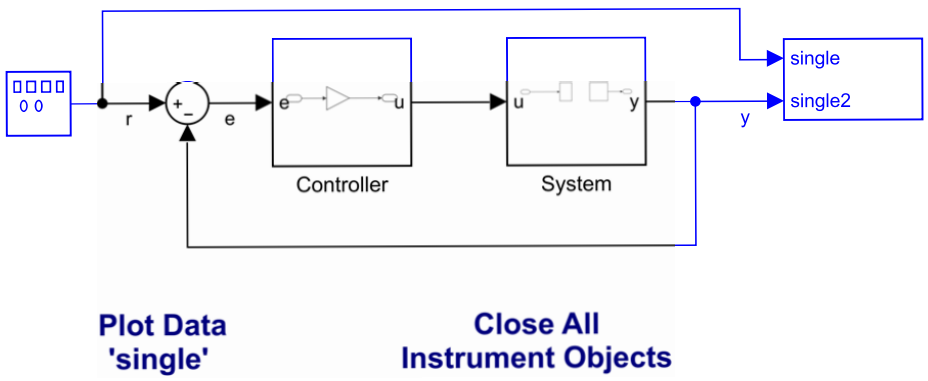
   (b) Then, right click on the *Take Home Labs Arduino Support Package* entry



   and select *Open Take Home Labs Arduino Support Package library.*



   (c) Once open, drag the text blocks *Plot Data 'single'* and *Close All Instrumented Objects* to your model.



To know more about sampling and data acquisition see the *Sampling and Data Acquisition* handout on the THL webpage.

6. For $K_p = 0.5$, 1, 1.5, 2, 3, and 5, repeat the following:

   (a) Instructions for Normal Mode, skip if you are using external mode

      i. Set the value of K_p and save the model. connect the Arduino to the PC, and **if you are running in Normal Mode**, deploy the model onto the Arduino. Remember that in Normal Mode, once you deploy the model to the Arduino, the code will begin to run automatically. Therefore, do not connect the DC power supply to the motor shield just yet.

      ii. **If you are running in Normal Mode,** click on *Plot Data 'single'* and change the COM port to the one your Arduino is using to. A figure should open up that will start plotting the serial data. It may take a few seconds to open.

      iii. To stop the experiment in *Normal Mode*, press the *Stop* button located at the bottom left of the plot window, and **NOT the Simulink Stop button**. Disconnect the DC power supply from the motor shield.

      iv. Once the plot is stopped, the "*WindowData*" variable, available in the *MATLAB* workspace is the data from the serial plot window, recorded at the rate of one data point every 0.01 seconds. Remember, we are using **serial** communication. As a result, the *WindowData* variable will be a single long vector that contains alternating values both r and y, i.e., $WindowData = [r_1, y_1, r_2, y_2, r_3, y_3, \cdots]$. Separate the values of r and y to generate a time versus r plot and a time versus y plot.

   (b) Before plugging in the power, make sure that the load is attached to the motor shaft securely, and that your hands and any other objects are away from the spinning load, to avoid potential injuries.

   (c) Plug in the power supply and if you are using *External Mode*, click on *Monitor & Tune*. **Caution:** in *Normal Mode*, the motor will start to spin the attached load when you connect the power supply.

   (d) To stop the experiment in *External Mode*, just use the Simulink *Stop* button. Disconnect the DC power supply from the motor shield.

   (e) If you are using *External Mode*, use standard Simulink techniques to save the data from the scope to MATLAB workspace. See the instructions here. **Do NOT submit screenshots of Simulink scopes**. Save the data in MATLAB and use the `plot` command to generate easily readable plots, with axis labels, a title, and a legend.

7. Once you generate the plots, add the value of K_p you used to the title of the plots and save them. You can also select everything in the workspace, right click, and save it for a backup.

8. After you have saved your figures, in the *MATLAB* command window, type *close all* and press enter to close all figures. Type *clear all* and press enter to clear the workspace of all variables. Then type *clc* and press enter to clear the command window.

D Submission

Submit the following:

- All the plots.
- A discussion on whether the performance (how well the output tracked the square wave) of the control system had any relationship with the value of K_p.

Question 2 (25 points)**.** The purpose of this question is to demonstrate the concept of feedback. Here we will experiment with open-loop vs feedback amplification of sound. Imagine that you are building a radio to reproduce a song. The signal you received from your antenna has traveled great distances and in the process, has picked up noise and is also greatly attenuated. You want to amplify this signal so that you can play it on your radio. You are doing this to listen to President Franklin D. Roosevelt's speeches in 1935, so the only amplifiers you have access to are bulky vacuum tube amplifiers that can have a fluctuating amplification ratio anywhere between $\pm 50\%$ of the nominal value.

Step 1: Experiment

1. Download the file *Oklahoma.wav* from the homework assignment page and move it to your current MATLAB directory.

2. Load the song into MATLAB using the command

   ```
   [r,Fs] = audioread('Oklahoma.wav');
   ```

   To play the song, use the command

   ```
   sound(r,Fs);
   ```

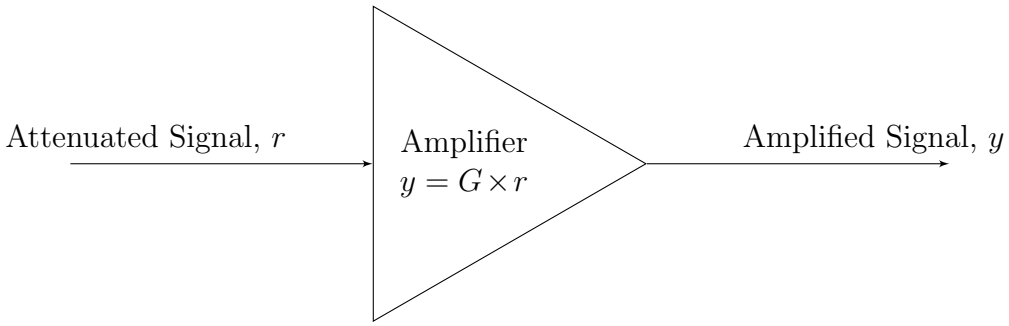
   and to stop playback, use the command

   ```
   clear sound;
   ```

3. Reduce volume to simulate transmission loss

   ```
   r=0.0000001*r;
   sound(r,Fs);
   ```

4. Perform open-loop amplification with desired nominal amplification gain $G = 500000000$

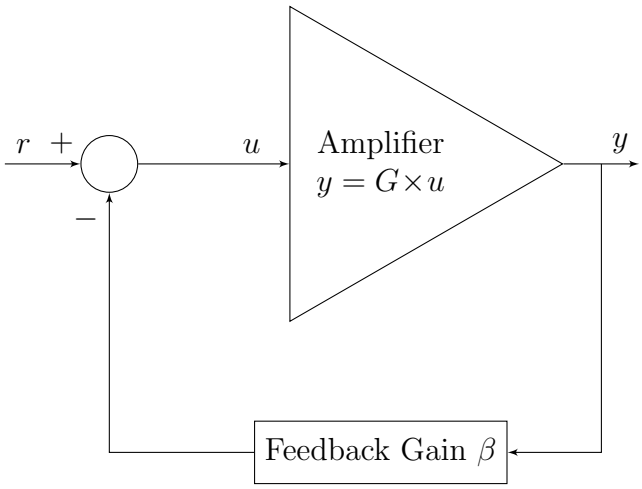


   but, the value of G has a 50% added noise. You can do so using the commands:

   ```
   G=500000000;
   NoisyG = G.*(0.5 + rand(size(r)));
   rOLAmp = r.*NoisyG;
   sound(uOLAmp,Fs);
   ```

Use *clear sound* to stop playback.

5. Now, perform closed-loop amplification. To do so, we will use the following feedback architecture



With this architecture, prove that the output is given by

$$y = \frac{G}{1+\beta G} \times r$$

6. Do the feedback amplification in *MATLAB* and listen to the amplified song:

```
beta = 0.0000001;
rCLAmp = r.*NoisyG./(1+beta.*NoisyG);
sound(rCLAmp,Fs);
```

Step 2:Analysis

1. Which amplification sounded better to you? Open loop or closed loop?
2. Repeat the experiment with ($G = 5000000$, $\beta = 0.00000001$) and another pair of values of your choosing. Which amplification sounds better in these other experiments?