Filtering and FFT

LAB 3 ECE 3400 SPRING 2021

31 MARCH 2021

QUIZ (REPORT): THREE SUBMISSIONS – SEE NEXT PAGE

Objectives

In this Lab 3, you will implement and test passive and active filters using the hardware that you have on hand, combined with your computer and MATLAB, and compare them to what is theoretically predicted. You will also implement and test a bandpass filter that will be used on your robot in Lab 4. The step 0 of this lab is about reviewing some of the Arduino material covered in class.

This is a long lab, it requires thought and time. It does not involve your robot: only the Arduino and circuitry, and MATLAB. The robot will not be moving in this lab.

The Table of Contents on the next page summarizes well what you will be doing in this lab.

This lab is divided into 3 sections, each section being due on Canvas every week. Consult the Table of Contents on the next page to see each STRICT deadline.

The table on the next page shows the contents of this Lab 3 handout along with the due dates of each grouping of sections. All material due must be uploaded to Canvas by 11PM on the specified date. There are no extensions.

Contents		Due Date on Canvas
Objectives	1	**Sections 0-3 incl.** **<u>06 April</u>**
Materials	3	
Procedure	3	
0. Questions on Canvas	3	
Learn Some LTSpice Basics	3	
1.A. Start LTSpice	3	
1. 1.B. Draw a low pass RC circuit	4	
1.C. Setup the simulation parameters and run the simulation	9	
Build the Microphone Circuit	12	
2. Code the Arduino and MATLAB to Characterize Circuits	13	
3. 4. Improve the Microphone Circuit	18	**Sections 4-6 incl.** **<u>13 April</u>**
5. Test your Low Pass and High Pass Circuits	19	
6. Bandpass Filter	21	
7. 8. FFT on Arduino	22	**Sections 7-9 incl.** **<u>20 April</u>**
9. Report (Quiz) & Wiki	26	
Wrap-up	27	

Contents of this Lab 3 handout along with the due dates of each grouping of sections. All material due must be uploaded to Canvas by 11PM on the specified date. There are no extensions.

Notes

- You CANNOT use the Arduino function `analogRead()`
- Disconnect the batteries: they will not be needed in this lab
- Make sure that you followed the directives on the home page of the Canvas site to install MATLAB.
- Install LTSpice on your computer.
- Ground yourself!
- Make sure that you disconnect the USB cable when you are finished.

Materials

- Capacitors and resistors
- Jumper wires
- Breadboard
- Op-Amps
- Your computer that has MATLAB with speakers
- Jumper wires (they can be separated out into single wires by pulling/peeling them)

Procedure

0. Questions on Canvas

A. Canvas submission: Class material review

Answer questions 0.1-0.3 on Canvas.

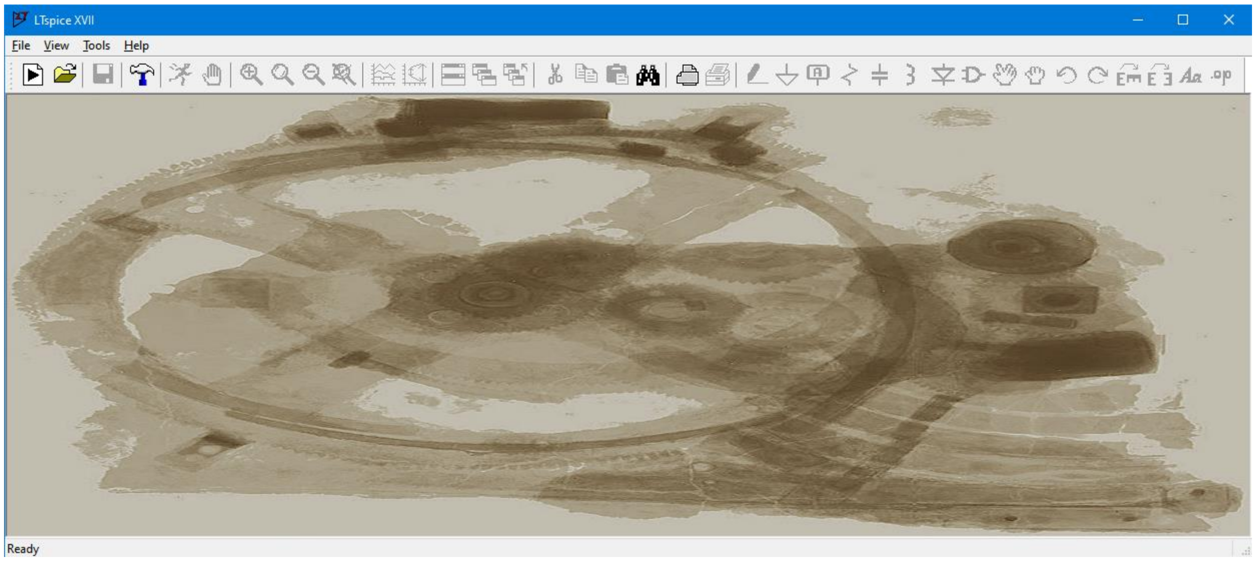
1. Learn Some LTSpice Basics

In order to simulate the behavior of the filters that you will implement in your robot, you will use LTSpice. This software is free, and is pretty straightforward to use. If you are not familiar with it, once you will have installed the software on your computer, follow these steps to get you acquainted with building your first circuit so that you can simulate it. Below are screenshots for the Windows platform but they are very similar on a Mac. Many of the instructions below are provided by right-clicking in the main window to access the various context menu items. You don't have to right-click in the main window: everything is also accessible via the menu under File, View, Tools.

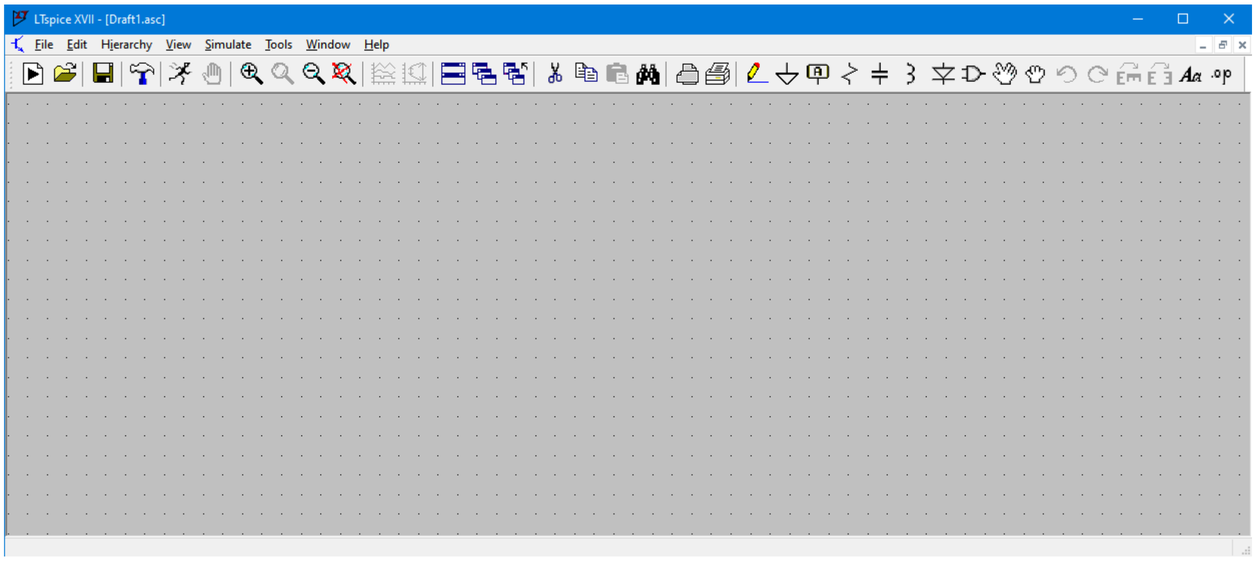
Note that you can click and drag anytime in your schematic gridded area to move the entire schematic at once around. You can also right-click in the main window and select View, Zoom to Fit to have a zoom in view with all of the components shown.

1.A. Start LTSpice

It should look as in the picture on top of the next page. Make sure that you make the window wide enough so that you can see all of the tools in the toolbar (it probably is the case that the program starts with a narrow window, preventing you from seeing all of the tools). It's a good idea to save your schematic – click on File, Save As, save it in an appropriate folder with an appropriate name (e.g., RC_Low pass_test1 – or whatever you want).

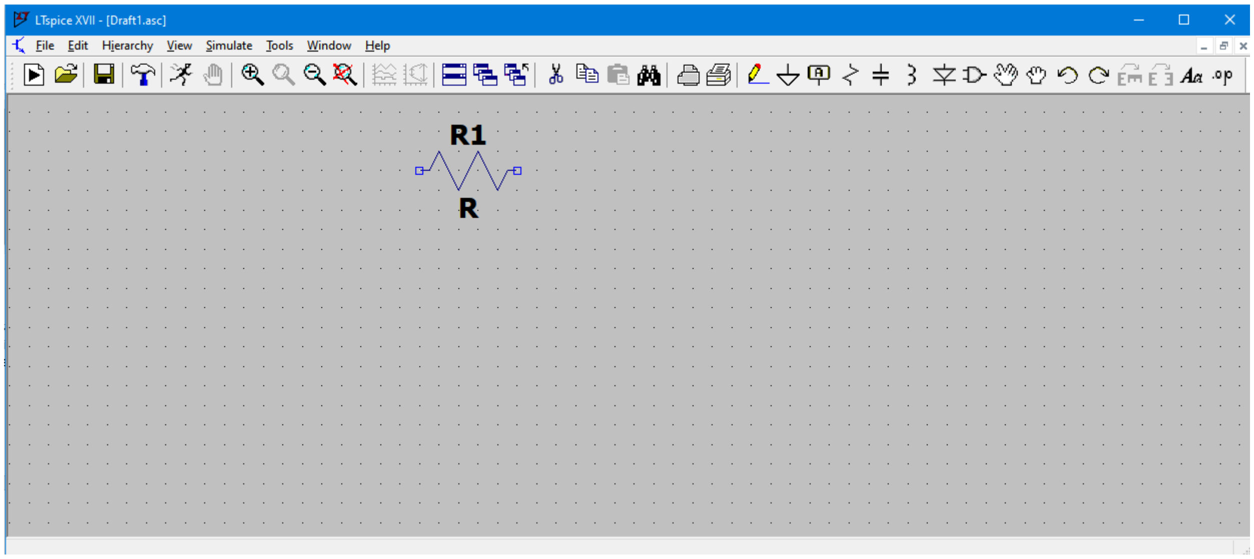


Right-click in the main window and select New Schematic. It should be that you will see the grid points. If not, right-click and select View, Show Grid, to get a window that looks like the following:

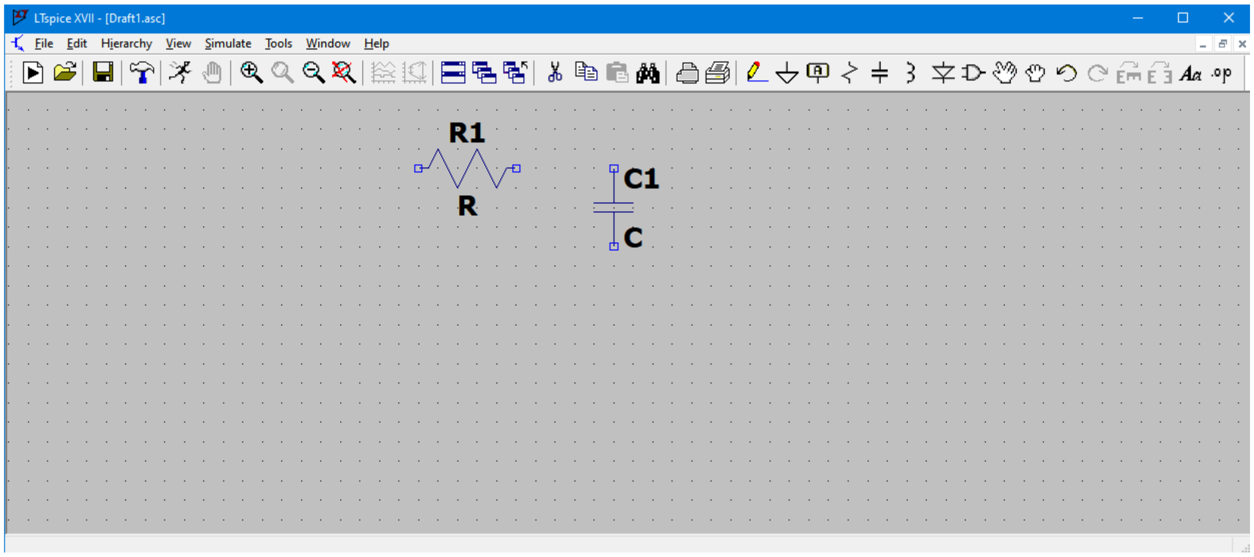

LTspice XVII - [Draft1.asc]
File Edit Hierarchy View Simulate Tools Window Help

1.B. Draw a low pass RC circuit

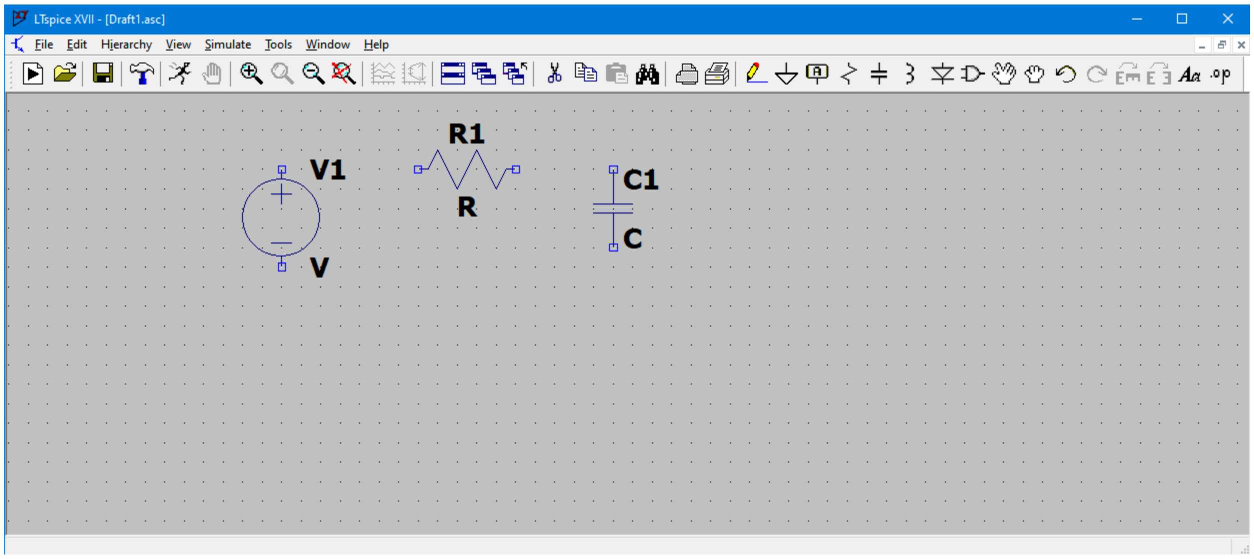
Click on the resistor symbol in the toolbar (you can also access devices by right-clicking in the gridded window and selecting Draft, Component, then type “resistor” (without the quotes) in the search window in the Select Component Symbol window that pops up, “res” will highlight and when you click OK the resistor becomes available in the main window) and move the mouse over the gridded window: the resistor will appear. Before clicking anywhere to insert the resistor, hit CTRL-R on your keyboard to rotate the resistor: it will then be horizontal. Click anywhere in the grid to insert the resistor, and when you do so, a second resistor R2 immediately becomes available: hit ESC on your keyboard to return to the normal mouse pointer. You should have something that looks like the picture below.


LTspice XVII - [Draft1.asc]
File Edit Hierarchy View Simulate Tools Window Help
R1
R

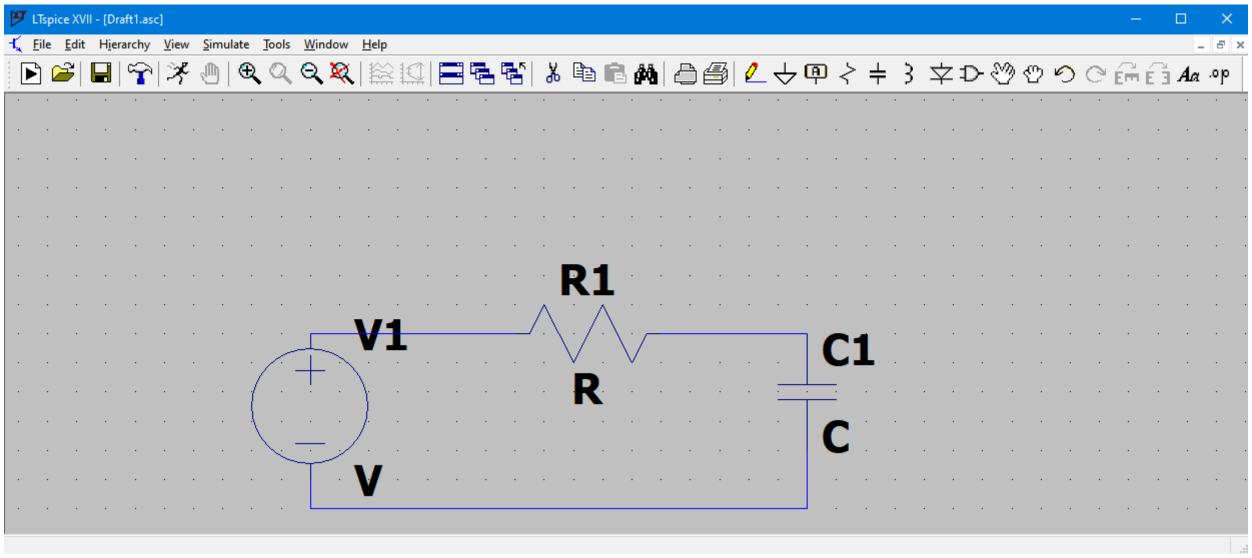
Now you will insert a capacitor to the right of the resistor. Select the capacitor tool on the toolbar and place a capacitor to the right of the resistor, as shown below. Notice how the right connector of R1 lines up with the top connector of C1 (they don't have to but things will look neater).


LTspice XVII - [Draft1.asc]
File Edit Hierarchy View Simulate Tools Window Help
R1
R
C1
C

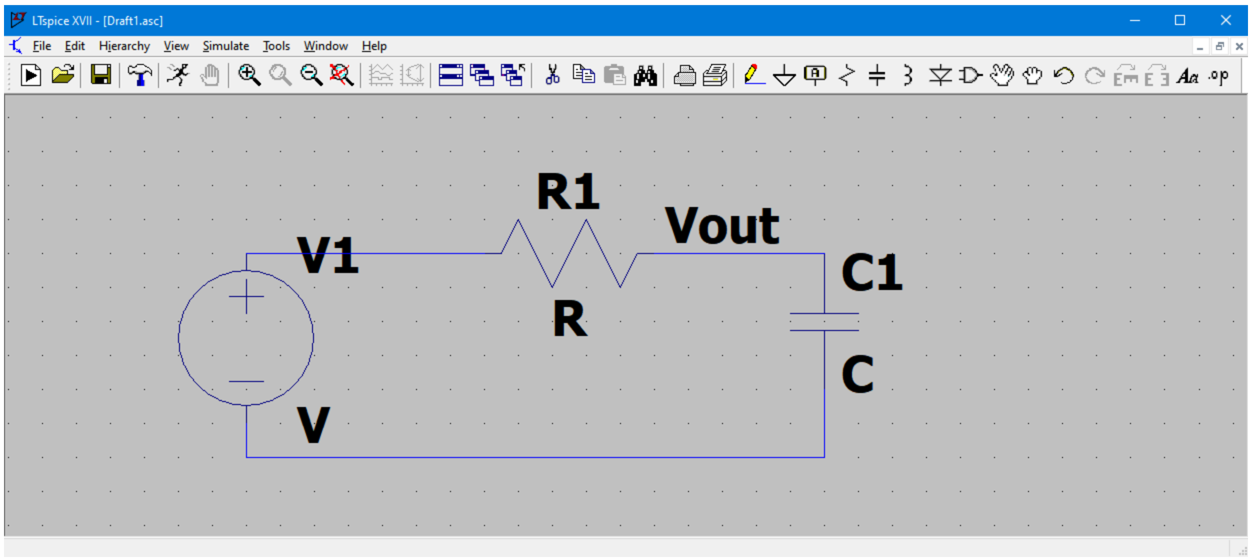
Now you will insert the voltage source to the left of the resistor. Right-click in the grid area, the Draft, Component (note that you can use instead F2 to do the same thing, or click on the component symbol on the toolbar) and type "voltage" in the search bar, then OK, then place the voltage source to the left of the resistor to get something that looks like the following picture:


LTspice XVII - [Draft1.asc]
File Edit Hierarchy View Simulate Tools Window Help
V1
V
R1
R
C1
C

Now let’s connect all of the components with wires. Right-click and select Draft, Draw Wire (or click on the pencil yellow icon). A cross hair will appear and move around with your mouse. Position the cross hair on the + terminal of the voltage source, click once and draw a line between the + terminal and the left connector of R1 (by clicking again once the cross hair is over the R1 connector). Repeat this process to connect everything together, to obtain something like shown below. When you are finished adding (a) wire(s), hit ESC on your keyboard.

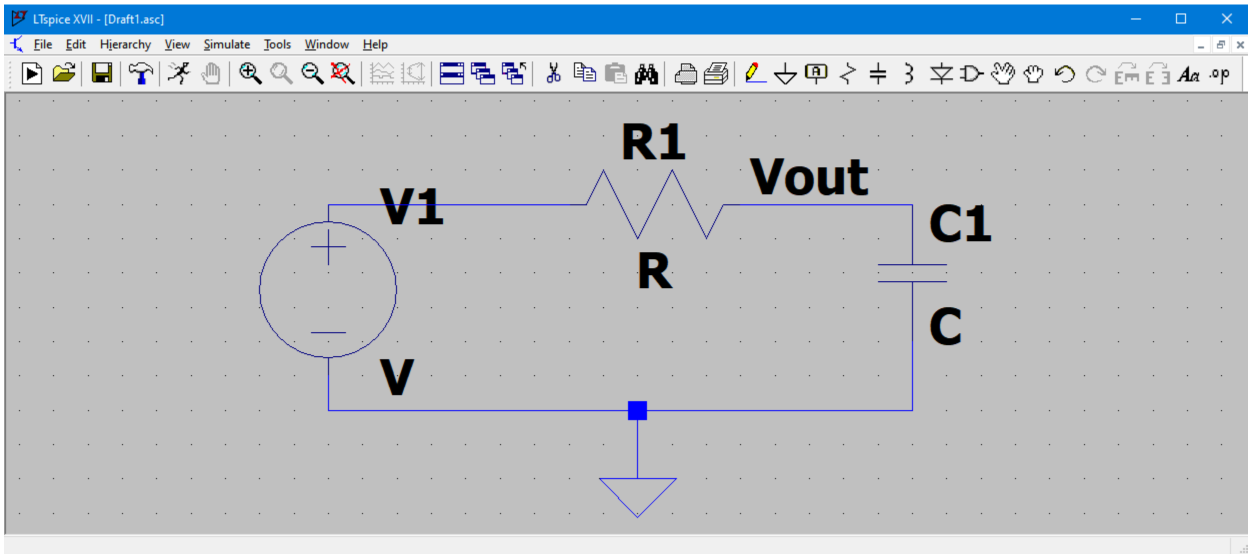

LTspice XVII - [Draft1.asc]
File Edit Hierarchy View Simulate Tools Window Help
V1
V
R1
R
C1
C

Now you will add names to important parts of the circuit (called net names). This is called naming the (important) nodes so that arbitrary names aren’t generated by the program (for ease of use). Right click somewhere in the gridded area, select Draft, Label Net.

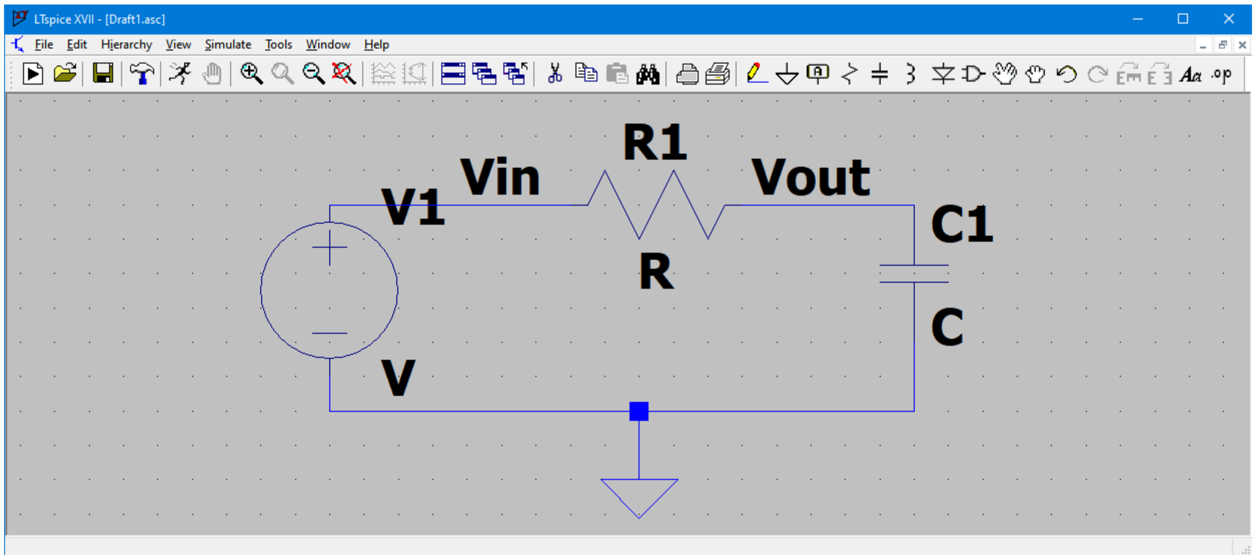

LTspice XVII - [Draft1.asc]
File Edit Hierarchy View Simulate Tools Window Help
R1
Vout
V1
C1
R
C
V

Enter “Vout” (without the quotes) in the white box (don’t click on either of the two options above the white box), then click OK. Position the square over the wire to the right of R1 and click on it, then hit ESC. You should have something that looks like the figure below.

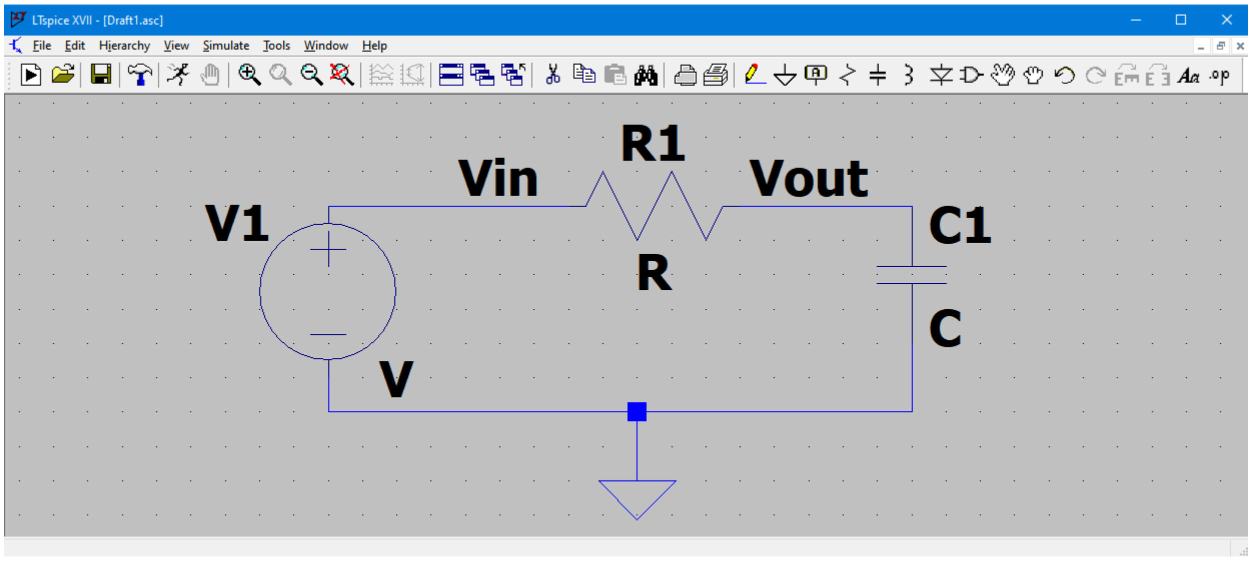
You will now add the ground. Right-click again in the gridded area, Draft, Label Net. Click the GND button, click OK and position the ground symbol a couple of grid points underneath the lowest wire (that connects V- to C). Then connect the ground to V- with a wire. You should get something as shown below.


LTspice XVII - [Draft1.asc]
File Edit Hierarchy View Simulate Tools Window Help
R1
Vout
V1
C1
R
C
V

Add now a third Label called Vin, that will be attached to the wire to the left of R1, as shown on the next figure:

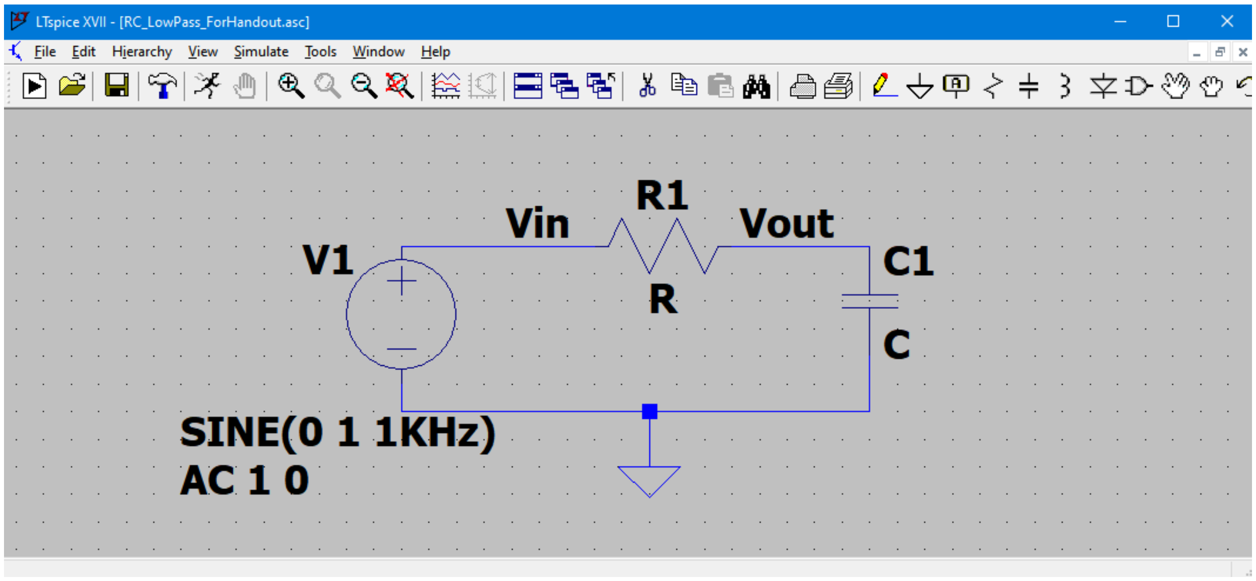

LTspice XVII - [Draft1.asc]
File Edit Hierarchy View Simulate Tools Window Help
V1
Vin
R1
Vout
C1
R
C
V

You can reposition things like the labels to have a neater looking circuit. Click on the Move icon (or right click in the main window and choose Edit, Move), click the hand-looking cursor on V1 to reposition the V1 label to the left of the voltage source. Repeat as you see fit with other labels. You will get something like shown below after moving the label V1.

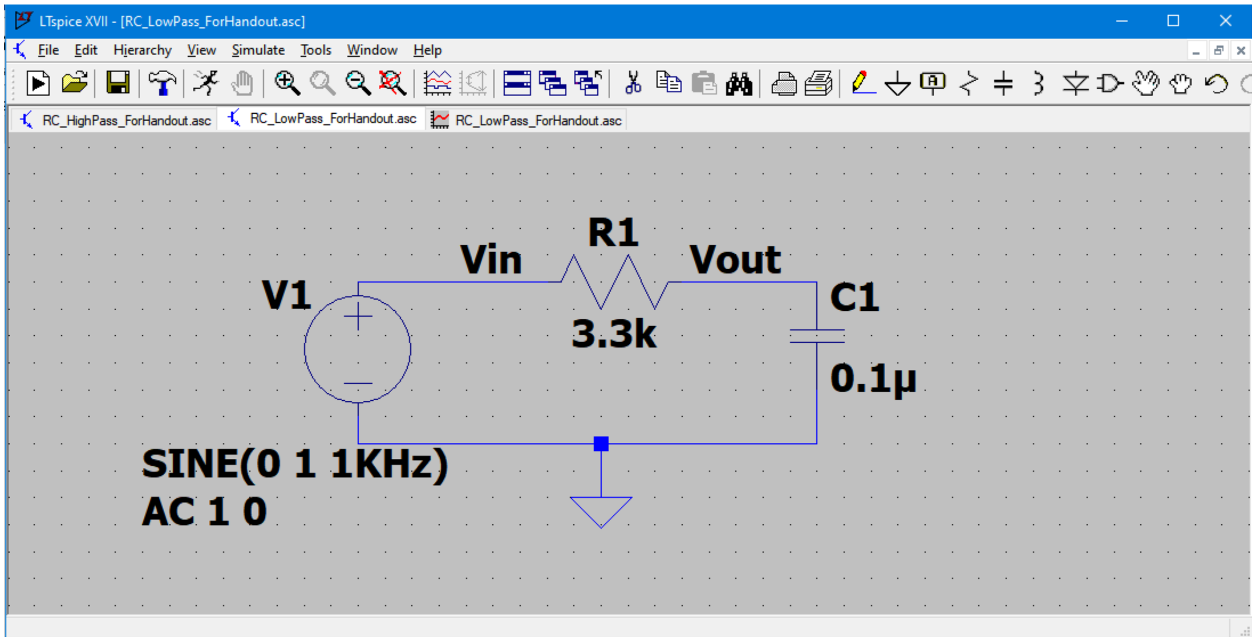

LTspice XVII - [Draft1.asc]
File Edit Hierarchy View Simulate Tools Window Help
V1
Vin
R1
Vout
C1
R
C
V

Since you are interested in getting the frequency response of your circuit, you need an AC voltage source whose frequency must vary. To set this, position your cursor over V1 (it will turn into a left-pointing finger) and right click on it. In the menu that pops up, click on Advanced, select SINE under Functions, enter a value of 0 for DC offset, 1 for Amplitude, and “1KHz” (without the quotes) for the Freq, and in the “Small signal AC analysis” section on the right enter an AC Amplitude of 1 and an AC Phase of 0, then click on OK. After moving the text

“SINE(0 1 1KHz)” from where it first appears, you will get something that looks like the figure on top of the next page.


LTspice XVII - [RC_LowPass_ForHandout.asc]
File Edit Hierarchy View Simulate Tools Window Help
Vin
R1
Vout
V1
C1
R
C
SINE(0 1 1KHz)
AC 1 0

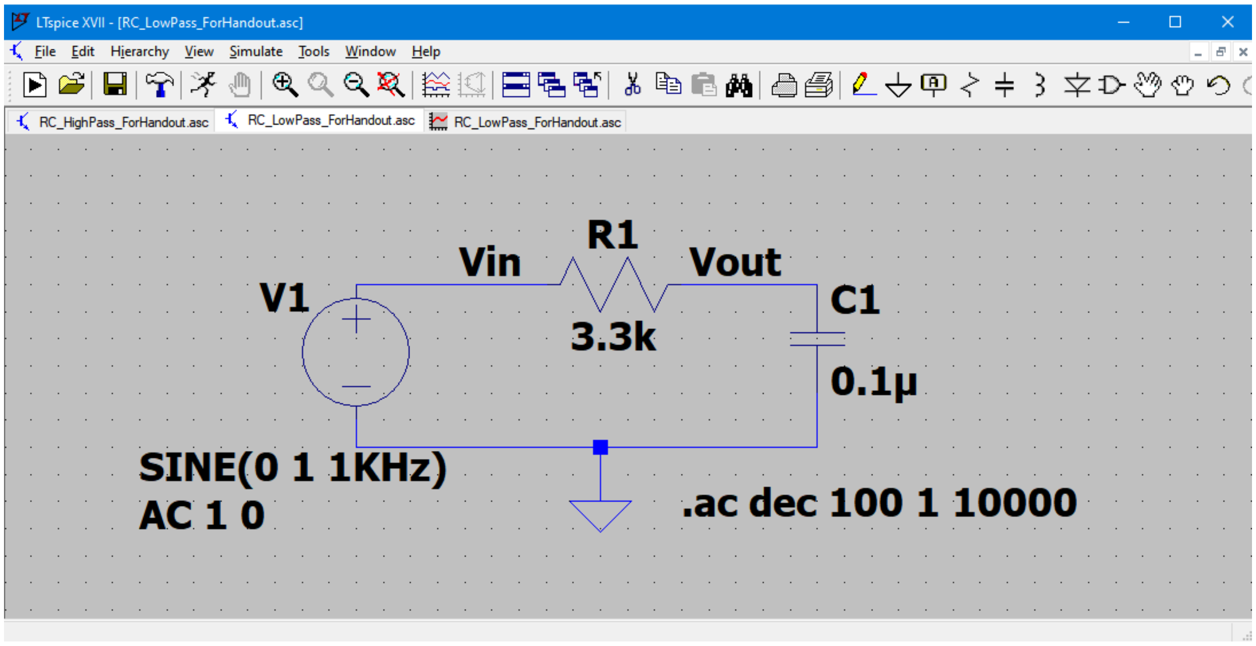
Now let’s set the values of R1 and C1. Right click on R1, and enter a value of “3.3k” (without the quotes) in Resistance and click OK. Similarly, set the value of C1 to “0.1u” (for a value of 0.1 μF, without the quotes). You will have something that looks like the following figure.


LTspice XVII - [RC_LowPass_ForHandout.asc]
File Edit Hierarchy View Simulate Tools Window Help
RC_HighPass_ForHandout.asc
RC_LowPass_ForHandout.asc
RC_LowPass_ForHandout.asc
Vin
R1
Vout
V1
C1
3.3k
0.1μ
SINE(0 1 1KHz)
AC 1 0

1.C. Setup the simulation parameters and run the simulation

Now to set the parameters of the simulation. Click on Simulate in the menu bar, then Edit Simulation Cmd. In the window that pops up, select the AC Analysis tab, select Decade for the

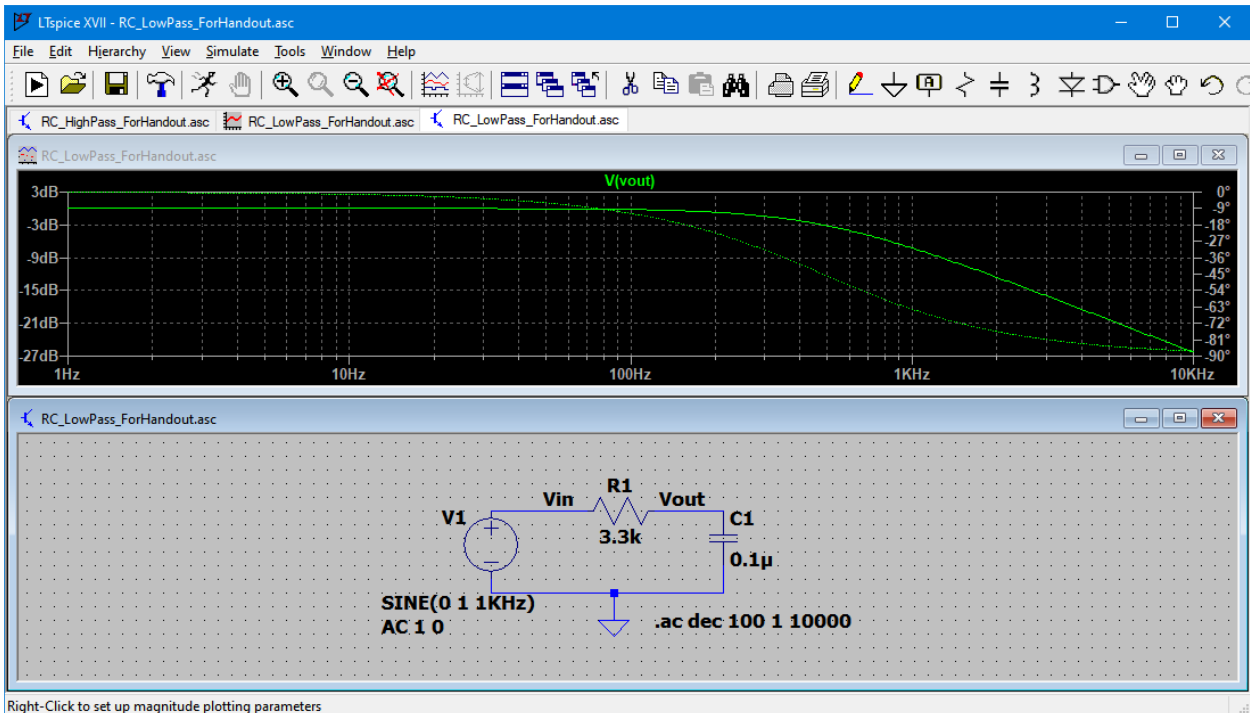
Type of sweep, enter 100 in Number of points, enter 1 for the Start frequency, and enter 10000 for the Stop frequency. Notice in the white bar at the bottom of the window the text commands that appear – they will show up in your grid when you click OK. Click OK, and position the command text (.ac dec 100 1 10000) somewhere on the side or above or below your schematic. You will have something that looks like the figure on top of the next page.


LTspice XVII - [RC_LowPass_ForHandout.asc]
File Edit Hierarchy View Simulate Tools Window Help
RC_HighPass_ForHandout.asc
RC_LowPass_ForHandout.asc
RC_LowPass_ForHandout.asc
V1
Vin
R1
3.3k
Vout
C1
0.1µ

```
SINE(0 1 1KHz)
AC 1 0
.ac dec 100 1 10000
```

Now you can run the simulation. Right click in the gridded area and select Run. When it is finished, the word Ready will appear in the lower-left corner of your LTSpice window. You will notice that a new window appeared inside of LTSpice and a .raw suffix – that is where your frequency response will appear. We will call this $\hat{H}_{Low\ pass}(\Omega)_{theory}$.

In your schematic (the .asc window), move your cursor to the wire to the right of Vout until a red probe appears. Click on the Vout wire with the tip of the probe: this tells the simulator where you want to perform the measurement for the frequency response. When you click on the wire, the frequency response appears in the .raw window. You should get something that looks like the figure on top of the next page.


LTspice XVII - RC_LowPass_ForHandout.asc
File Edit Hierarchy View Simulate Tools Window Help
RC_HighPass_ForHandout.asc
RC_LowPass_ForHandout.asc
RC_LowPass_ForHandout.asc
V(vout)
3dB
-3dB
-9dB
-15dB
-21dB
-27dB
0°
-9°
-18°
-27°
-36°
-45°
-54°
-63°
-72°
-81°
-90°
1Hz
10Hz
100Hz
1KHz
10KHz
RC_LowPass_ForHandout.asc
V1
Vin
R1
Vout
C1
3.3k
0.1µ
SINE(0 1 1KHz)
AC 1 0
.ac dec 100 1 10000
Right-Click to set up magnitude plotting parameters

You can maximize the curve (the .raw graph) window to have a better view. The solid line is the frequency response, and it is as expected: the schematic is that of a low-pass filter, and that is what the solid line curve reveals. You can improve/modify the curve by right-clicking on the graph, selecting View then Grid to view grid points.

Explore the various menu options to see what the software can do. You can, for example, export the data for analysis in another software later. You can zoom in a point of interest to read its coordinates. You can annotate graphs from the Plot Settings menu. You can right click on the graph and select File, Export data as text to export the graph's data so that you can use it in MATLAB or other software.

1. Canvas submissions

1. **Simulate a low pass filter using R=1.2kΩ and C=0.1µF. Upload a picture of the frequency response graph zoomed-in around the cutoff frequency ± 1 Hz and around the dB cutoff value ±0.01dB. Clearly identify the cutoff frequency on the graph with an arrow and write the cutoff frequency (with units!) on the graph using annotations.**
2. **Simulate a high pass filter using the same values for R and C as the low pass filter. Upload a picture of the frequency response graph zoomed-in around the cutoff frequency ± 1 Hz and around the dB cutoff value ±0.01dB. Clearly identify the cutoff frequency on the graph with an arrow and write the cutoff frequency (with units!) on the graph using annotations.**

1.D. Add the LM358 Op-Amp Model to LTSpice

Now that you know the basics, it is an easy task for you to draw different and more complex circuits. Eventually, you will need to place the LM358 op amp that you will use to design your active filters. This component is not available by default in LTspice. To add it:

- Go to TI's LM358's page here
- About halfway down the page, look for the Simulation Model: LMx58_LM2904 PSPICE Model (Rev. A), and download it
- Open the downloaded zip file and place the .CIR file in the same directory where you save your .ASC LTspice schematic
- To add the LM358 op amp to your schematic, right click on the gridded area and select Draft, Component, then search for opamp2 and click OK. Place opamp2 in your schematic.
- Change the name **opamp2** of the placed op amp to **LMX58_LM2904**
- Right click on the gridded area and select Draft, Spice Directive, and enter in the white box ".lib LMx58_LM2904.CIR" (without the quotes), click OK, and position the directive somewhere near your schematic (like you did for the AC Analysis).

2. Build the Microphone Circuit

Before you start: disconnect all battery wires from your breadboard/motors and secure the wires so that they don't accidentally hit the breadboard.

The microphone will be used to collect sound and the Arduino will eventually process this sound on your robot. But before you can directly implement this on your robot, you must characterize everything. One of the components is the microphone circuit.

2.A. Microphone Circuit without Amplification

We saw in class some material on the microphone that you will use on your robot. To get a feel for how electret microphones work and how not great they are by themselves, build the circuit shown in Fig. 1 (with the Arduino powered off), using R1 = 3.3kΩ and C1 = 10μF. It is **critical** that you take note of the polarity of the microphone as explained in class (also in Fig. 1 below: the right of Fig. 1 shows how to identify the ground pin – you can insert the microphone directly into the breadboard) **and** the polarity of the capacitor (examine your capacitor and determine which pin is **–** and which pin is **+**). Choose one analog pin into which your circuit connects.

You will notice that the breadboard is starting to have many things on it. It is suggested that you put your microphone on the breadboard as close as possible to the Arduino and leave as much room as possible between its circuit and the L293D which is at the tail end of the robot (the L293D should be at the very back edge of the robot).

You will test this circuit (not quite yet), and you will also add on to this circuit later in the lab and test that new circuit.

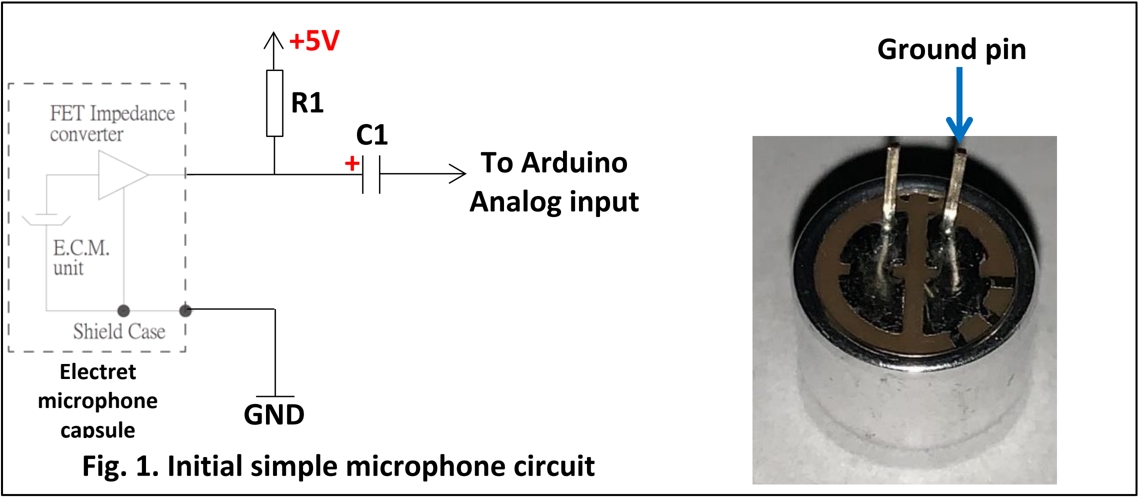


Fig. 1. Initial simple microphone circuit

2. Canvas submissions

1. **Upload a photo of your basic circuit above, showing the microphone, resistor and capacitor connected to one another. Your picture must show clearly the capacitor's polarity (by showing the negative sign marking on its side).**

3. Code the Arduino and MATLAB to Characterize Circuits

In this part, you will write the code that will be used by your Nano and code that will be used by MATLAB so that you can test and compare the physical implementation of the low pass and high pass circuits simulated in Section 1, along with the microphone circuit, and also test and compare the physical implementation of the bandpass filter that you will use on your robot in Lab 4.

So that you can write the appropriate code, you need to understand what will happen. In the final build of your robot in Lab 4, the microphone on your robot will pick up ambient sound while your robot is at rest. When this sound will have a given frequency (tone) in it (one of possibly several), your robot will do one particular operation (e.g., turn around in circles). This means that your robot (the Arduino) must be able to discern specific frequencies in sound that it picks up and act according to those frequencies. One key word appears here and it is "frequency" which means that Fourier analysis is involved.

To understand what happens in the frequency domain, you will use MATLAB to better and more accurately analyze the sounds and compare your results to how the Arduino will perform using frequency analysis.

3.A. Arduino Code

The first Arduino code that you will write is so that your Nano can collect sound from your microphone circuit that you built above and will be connected to your Arduino, and print it to the serial port so that it can be read by MATLAB for analysis. You CANNOT use the Arduino function `analogRead()`: it is too slow. You will manually code the ADC which shouldn't be difficult since we covered much of that material in class. The ADC will be setup so that it is in Free Running mode (read the ATmega4809 datasheet about this).

Also, referring to the “Getting started with ADC” PDF on Canvas, read section 4 (ADC Free Running mode) and the code corresponding to that section in Appendix of the document which is Example 9-2. You will adapt that code into a sketch. Create a new .INO sketch (name it `freeRunADC_MATLAB.ino`) and add the following, commenting clearly and well your sketch:

- In **`void setup()`**
    - Add `Serial.begin(115200);`
    - From “ADC Free Running Code Example,” copy and paste all of the code from Example 9-2 that is inside the `ADC0_init` function, changing the appropriate command to use an ADC prescaler value of 16, and changing the four entries that indicate which Arduino analog pin your microphone circuit is connected to (the analog pin you chose in the previous section). You also need to change `ADC_REFSEL_INTREF_gc` to `ADC_REFSEL_VDDREF_gc`.
    - Add the command to start the ADC conversion.
- Create a function **`bool ADC0_conversionDone(void)`** that contains the same command from the function of the same name as in Example 9-2.
- Create another function `int` **`ADC0_read(void)`** that contains the same two statements from the function of the same name as in Example 9-2
- In **`loop()`**
    - Read the ADC value by calling the function **`ADC0_read();`** (i.e., store it in a properly defined variable) *only when* the ADC is ready (i.e., when **`ADC0_conversionDone`** returns TRUE) – you need to code this
    - Convert the read ADC value to a signed 16 bit number.
    - Print this value to the serial port using `Serial.println`.

Note above the baud rate used. This is to ensure that the Arduino writes things fast to the serial port so that MATLAB can fetch a value whenever it (MATLAB) requests one.

One tricky point above is the second to last statement, “Convert the read ADC value to a signed 16 bit number.” As you know, even when using the function `analogRead(),` when reading an analog input connect to the Arduino, the value returned is between 0 and 1023: the Arduino cannot read negative values. In your basic circuit from Fig. 1, the +5V supply offsets the analog signal that the microphone picks up so that the oscillating waveform that makes up sound only has positive values (instead of having oscillations about the y-axis). The “Convert the read ADC value to a signed 16 bit number” statement above (that you have to code) therefore asks you to convert the read ADC values into values that are those the original audio sound, meaning having positive AND negative values. Figure 2 on the next page shows the sequence of events that happen:

- Image 1 shows an original sound signal. Notice how the signals oscillates about the x-axis (it has positive and negative values).
- Image 2 shows this same sound signal shifted up (it has a positive DC offset) so that all data values are positive (something that the Arduino can handle). Note that, and this is important, the offset must be sufficient so that all values are shifted enough so that they are all positive.

- Image 3 shows in red circles the analog-to-digitally converted points that will be sampled from image 2 by the Nano (the sampling period is only illustrative).
- Image 4 shows the analog-to-digitally converted points: they are all positive, and importantly, they must be between 0 and 1023.

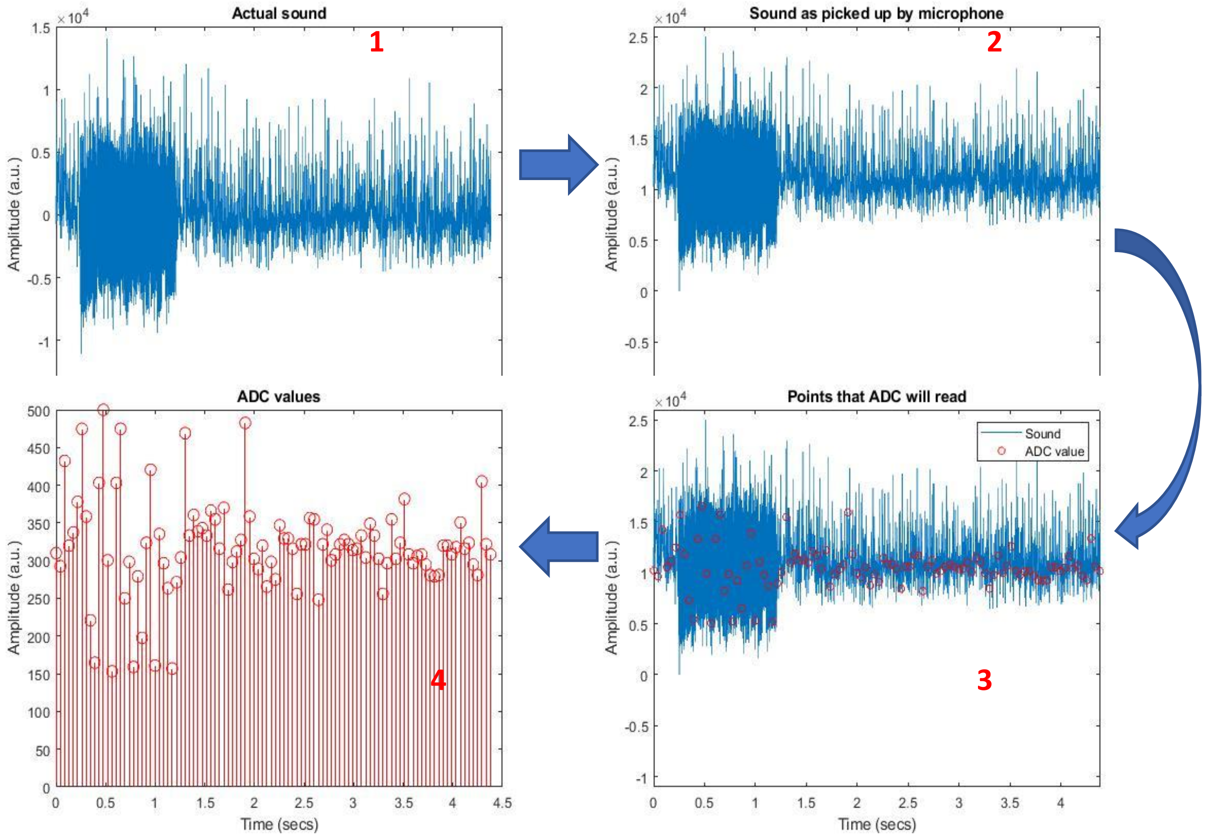


Fig. 2. Sequence of events when the Arduino reads a sound signal

So the statement “Convert the read ADC value to a signed 16 bit number” means that you must manipulate the ADC values read by the Arduino into signed positive and negative numbers. This should involve only two lines of code: one to shift the read ADC values down by some appropriate value, and the second to scale the ADC values such that you obtain a 16 bit number.

Once your code is ready (remember to name your file `freeRunADC_MATLAB.ino`), you can upload it to your Arduino. Connect your microphone circuit from Fig. 1 to the Arduino’s analog pin you chose (and for which you adequately modified the code to reflect which pin you are using), open the Serial Monitor to observe that the data makes sense (you need to change the Monitor’s baud rate so that it matches that of your code). It will be that the values just stream too fast on the Monitor: you can add a `delay()` statement in your .INO file**, *but comment it out for what comes next.***

Close the Serial Monitor before continuing and disconnect the USB cable from the Nano.

3.A Canvas submission: MATLAB Code

1. **Upload your well-commented `freeRunADC_MATLAB.ino` sketch to Canvas.**

3.B. MATLAB Code to Fetch Data from the Arduino

You now have Arduino code that does ADC on sound picked up by the microphone, but we can't tell what is going on. You will use MATLAB to analyze the data generated by the Arduino and visualize what is going on. You could copy & paste date from the Serial Monitor, but that gets old very quickly when you do it over and over again.

Download the file readData_INT_Canvas.m from Canvas (Modules, Lab 3) and rename it to `readData_INT_Canvas_netID.m` (where `netID` is your netID) from Canvas in the Lab 3 module. Read the entire file and understand what it does – you must do this in order to continue. Some things that you may/will change include (do not change them for this part of the lab)

- initialFreq = … from line 19
- finalFreq = … from line 20
- soundDuration = … from line 26
- countRead = … from line 33
- serialComm = … from line 40
- dataChar = … from line 56
- Calculate and plot the FFT/spectrum – see lines 74 to 103.

The above MATLAB file's purpose is to: **1.** Play a sound of increasing frequency over your computer's speakers; **2.** Collect with MATLAB the sound with your Nano by doing analog-to-digital conversion while the sound is being played; **3.** Perform Fourier analysis in MATLAB on the data so that you can analyze what is going on.

Of importance in this file is understanding what sampling frequency will be used: it is the frequency at which MATLAB fetches data from the Arduino via the USB cable with the command read. It is *that* frequency that is the sampling frequency because it is much smaller than the sampling frequency of the Free Running ADC on the Nano. This is quite different than the sampling frequency of the Arduino when the Arduino does ADC, as you will study shortly. The variable `durationOfDataRead` captures the time that MATLAB took to read all of the `countRead` samples, so you therefore know the sampling time, hence the sampling frequency.

Before you can use the MATLAB code, you need to add code to it. This is in the section of lines 74 to 103 (you can insert as many lines as you need): this is where the FFT of the acquired signal will be calculated and where you will plot it. MATLAB stores the acquired time signal from the Nano in the variable `dataDouble` and corresponding time axis contained in `timeAxis` (so if you issue the command `plot(timeAxis, dataDouble)`, this will plot the time domain signal as acquired by MATLAB – I just gave you a hint!). Now you need to perform Fourier analysis on the time signal contained in `dataDouble` so that you can plot its spectrum. Write this code on line 85+. You will lastly write commands in the section "Plot the results" so that the time domain signal and the spectrum can be plotted as described in the .m file. Make sure that the plots make sense, the axes labels large enough and readable, the plot symbols are clear (the time domain and the frequency domain signals are discrete time signals, NOT continuous time signals, and so the plots must be symbols and not continuous lines) and that your graphs have titles. Features of interest must be clearly visible and obvious (i.e., scale things appropriately).

With your circuit from Fig. 1 connected to your Arduino and your code from section 3.A uploaded to your Arduino, position your circuit "close" to your computer speakers (find a relatively undisturbed spot to which you can reliably put your circuit back at the same position relative to your speakers), connect your Arduino to your computer with the USB cable (you will only use your breadboard for this), and execute the MATLAB code **(make sure that the Serial Monitor is closed when running this code: the Nano will write to the serial port and MATLAB will collect the data from the serial port)**. Do not change the values anywhere in the code, except for initialFreq=500, finalFreq=500.1, soundDuration=2, and an appropriate countRead. Apart from those, your FFT and plotting code go after line 76 Run the MATLAB code, positioning yoru circuit close to your speaker(s) set to a rather high volume, until a figure is plotted (it takes a few seconds)[1]. Zoom-in the spectrum around the frequency of 500 Hz: you should observe a (small) peak that represents the presence of a ~500Hz signal that was picked up by your microphone circuit. If you don't see a peak, rerun the MATLAB code with your speakers set at a higher volume and/or your microphone closer to your speakers. Assuming your Arduino code is correct, you should see the peak. It may be useful here to not plot the 0 frequency component (i.e., the DC component of the spectrum).

It can happen that MATLAB cannot communicate with the Arduino. When that happens, make sure that the Serial Monitor is closed, and if necessary, unplug and reconnect your Arduino. Also the command `clear serialComm` in MATLAB to close all serial ports that may be open may help. It can happen that your IDE cannot communicate/program your Arduino. When that happens, type the command `clear serialComm` in MATLAB to close all serial ports that may be open.

3.B Canvas submissions: Code & Plots from Basic Microphone Circuit

1. **Upload your completed MATLAB file `readDataFromArduino_withSound_netID.m` `to Canvas.`**
2. **Upload the graph output by MATLAB (in MATLAB Figure window, click on File, Save As, choose JPEG) that shows in the top part the time domain signal, and the spectrum in the bottom. Unreadable, questionable, badly labeled graphs will lose points.**

This is the end of what must be completed in the first week of the release of this lab, 31 March – 06 April 11PM Ithaca time, Section 0 to Section 3 inclusively.

----

[1] Look at the time signal that you plot to make sure that it doesn't saturate. If it does, reduce the volume of your speakers.

4. Improve the Microphone Circuit

As you probably observed above, the spectrum's value at the frequency of ~500 Hz is relatively weak. That is because your microphone circuit is not really good (or rather, because the microphone is not very good at picking up sound). What you need to do is to amplify the sound that is picked up by the microphone. We discussed this in class, and the circuit that you will implement is shown in Fig. 3.

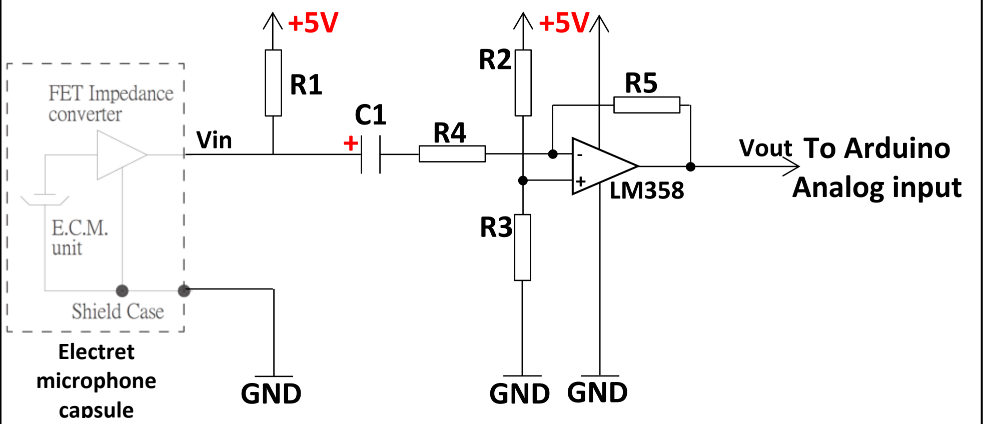


Fig. 3. Improved microphone circuit (not that overlapping wires do not connect to one another unless there is a dot •)

The values of R1 and C1 are the same as before. Use the following for the other components:

- R2, R3: 10 kΩ
- R4: 3.3 kΩ
- R5: 511 kΩ

For this circuit, the gain is given by: $V_{out} = -V_{in}\frac{R_5}{R_4} + \frac{V_{cc}}{2}$.

With this circuit implemented, position your circuit close to your computer speakers as before and execute the MATLAB code (same as in the previous section) to get the plots. You should observe a much larger peak near the ~500Hz signal frequency due to the amplification.

4. Canvas submissions: Picture & Plots from Amplified Microphone Circuit

1. **Upload a clear picture of your circuit.**
2. **Upload the picture that is output by MATLAB (in the MATLAB Figure window, click on File, Save As, and choose JPEG.**
3. **From the spectra that you obtained, what is $V_{out}(\omega)/V_{in}(\omega)$ (i.e., the frequency response) as a function of frequency provided by your amplifying circuit?**

5. Test your Low Pass and High Pass Circuits

Now you will test your passive low pass and high pass filters that you simulated in Section 1 with components from your kit. Firstly, build your low pass filter on the breadboard (be careful with the components: you will need them for the rest of the course), and connect the output of your microphone amplifier circuit into the input of your low pass filter (this means that the output of your microphone amplifier circuit takes the role of V1 in your LTspice circuit – it is then Vin), and connect the output of your low pass filter into the appropriate analog pin of your Nano. What you are doing now is low pass filter the amplified signal coming out of your microphone circuit.

We are now interested in experimentally characterizing your filter's frequency response. LTspice's Bode plot is your filter's frequency response, and you will compare the experimental frequency response to the simulated one. To obtain your low pass (or any!) filter's frequency response, apply the knowledge that you know from circuit analysis:

$$\widehat{H}(\Omega) = \frac{\widehat{Y}(\Omega)}{\widehat{X}(\Omega)}$$

that is, the system's frequency response $\widehat{H}(\Omega)$ is the ratio between your system's output signal's spectrum $\widehat{Y}(\Omega)$ divided by your system's input signal's spectrum $\widehat{X}(\Omega)$ – note here that Ω is the frequency, with $\Omega \in \mathbb{R}$. What you will do is compare the system's frequency response $\widehat{H}_{Low\ pass}(\Omega)_{theory}$ that you simulated in Section 1 to your experimental system's frequency response $\widehat{H}_{Low\ pass}(\Omega)_{experiment}$ that you will measure with the Nano and characterize with MATLAB. This means that when you will measure the signal coming out of the low pass filter, that be $\widehat{Y}_{Low\ pass}(\Omega)_{experiment}$ and to obtain $\widehat{H}_{Low\ pass}(\Omega)_{experiment}$ you need $\widehat{X}_{Low\ pass}(\Omega)_{experiment}$.

What is $\widehat{X}_{Low\ pass}(\Omega)_{experiment}$? It is the spectrum of the signal that is fed into the low pass filter. So you then need to do the following:

1. Measure the signal coming out of the low pass filter, and obtain $\widehat{Y}_{Low\ pass}(\Omega)_{experiment}$;
2. Measure the signal that is *fed* into your low pass filter to obtain $\widehat{X}_{Low\ pass}(\Omega)_{experiment}$. To do this: without moving your setup from its position from your speakers **AND with the USB cable disconnected**, simply move your wires around in your circuit so that it is the output of the amplified, unfiltered signal that goes into the Arduino, then measure that signal. You must understand that the conditions must be the same when measuring the filtered and unfiltered signal as much as possible so that you can claim that the unfiltered signal is indeed an, *the*, input into the filter.
3. Calculate $\widehat{H}_{Low\ pass}(\Omega)_{experiment}$.

Note that in MATLAB, you need to set initialFreq=100, finalFreq=2000, soundDuration=10, and an appropriate value for countRead[2]).

For step 3 above, you will have noticed that your graphs for the spectra $\hat{X}_{Low\ pass}(\Omega)_{experiment}$ and $\hat{Y}_{Low\ pass}(\Omega)_{experiment}$ are very noisy: when you look at them you get a good impression of what is going on, but to use them to do calculations results in messy signals. In order to reduce the noise, you can process (smooth) the spectra with the function `smoothdata` in MATLAB. You will notice that you need a rather large window to get appreciable smoothing (I used `smoothdata(A,'sgolay',2000)`, for example, where A is the spectrum you want to smooth – go and try it as written here, you are welcome to play with the method and window width). After smoothing both $\hat{X}_{Low\ pass}(\Omega)_{experiment}$ and $\hat{Y}_{Low\ pass}(\Omega)_{experiment}$, you can calculate $\hat{H}_{Low\ pass}(\Omega)_{experiment}$ in step 3 above.

Think about what you are doing instead of blindly running code. In step 1 above, once the code is finished running, save the spectrum in a new variable. In step 2, when you run the code again, save the spectrum in yet another variable. That way, you will have both spectra that you can now use for your calculations.

Note that it may happen that when you want to calculate $\hat{H}_{Low\ pass}(\Omega)_{experiment}$ (i.e., when calculating the ratio $\hat{Y}_{Low\ pass}(\Omega)_{experiment}/\hat{X}_{Low\ pass}(\Omega)_{experiment}$), the latter two vectors may not be of the same length (i.e., one may have more entries than the other). If that happens, and this is due to the serial connection between MATLAB and the Nano, you can modify the `fft` function in MATLAB by using `fft(X,n)` (If X is a vector and the length of X is greater than n, then X is truncated to length n) to ensure that both spectra have the same number of elements.

Now that you have $\hat{H}_{Low\ pass}(\Omega)_{experiment}$ (smoothed), superimpose on its graph the simulated low pass filter's spectrum $\hat{H}_{Low\ pass}(\Omega)_{theory}$ that you obtained from LTspice(being watchful of taking logs or not, as appropriate) to compare theory to experiment. On another graph, plot the difference between the simulated and experimental filter responses (i.e., $\hat{H}_{Low\ pass}(\Omega)_{theory}$-$\hat{H}_{Low\ pass}(\Omega)_{experimental}$). Make sure your graphs are clear, the axes fonts are large with units, and your graphs have titles. Points will be deduced for unclear plots.

Repeat the process for the high pass filter.

[2] It is important to understand that MATLAB needs to read a sufficient number of samples such that they are for *at least* the duration of the sound being played. You could do a simple characterization on your computer by changing the value of countRead and noting how long the signal is in MATLAB, i.e., the last value of timeAxis. For Carl's computer, I got the approximate relationship: $countRead > 6200 \cdot timeDuration$, for $timeDuration > .05$ secs. It's ok to read data values for a longer time than the sound is played: assuming it's quiet in your environment, when the sound is not/finished playing, MATLAB will only record quietness.

5. Canvas submissions: Frequency Responses with Low Pass and High Pass Filters

1. **Upload the graph that shows the superposition of the smoothed $\hat{H}_{Low\ pass}(\Omega)_{experiment}$ and $\hat{\mathrm{H}}_{\mathrm{LowPass}}(\Omega)_{\mathrm{theory}}$ obtained from your experimental measurements and simulations, respectively.**
2. **Upload the graph that shows the plot of the difference between the simulated and experimental filter responses, $\hat{H}_{Low\ pass}(\Omega)_{theory}$-$\hat{H}_{Low\ pass}(\Omega)_{experimental}$.**
3. **Comment on points 1 and 2 above (i.e., what you observe, why theory and experiment are different, how good is the attenuation beyond the cutoff frequency, etc.)**
4. **Upload the graph that shows the superposition of the smoothed $\hat{H}_{Highpass}(\Omega)_{experiment}$ and $\hat{\mathrm{H}}_{\mathrm{HighPass}}(\Omega)_{\mathrm{theory}}$ obtained from your experimental measurements and simulations, respectively.**
5. **Upload the graph that shows the plot of the difference between the simulated and experimental filter responses, $\hat{H}_{Highpass}(\Omega)_{theory}$-$\hat{H}_{Highpass}(\Omega)_{experimental}$.**
6. **Comment on points 4 and 5 above (i.e., what you observe, why theory and experiment are different, how good is the attenuation below the cutoff frequency, etc.)**

6. Bandpass Filter

The point of filtering the audio signal that the Arduino picks up is to eliminate noise and interference. The reason why you are working on audio signals in the course is to eventually program your robot so that it will respond to audio signals. In order to eliminate undesirable audio noise and interference (e.g., from harmonics), you need to filter the audio signal to attenuate/eliminate noise and interference in our audio range of interest which here is set to be approximately between 500 and 900 Hz (the trigger frequencies in Lab 4 will be in this range). It is better, in our case, to bandpass filter the signal to attenuate/eliminate low frequency undesirable signals (60 Hz hum, 1/f noise aka flicker noise, and other low frequency signals that could generate harmonics in our range of interest) and also attenuate/eliminate high frequency undesirable signals ("high frequency" here meaning above our range of interest).

We discussed in class the Butterworth 4-pole bandpass filter that you will implement. With the USB cable disconnected, remove the low pass or high pass filter components that you have on your breadboard to make room for and build the bandpass filter.

You should implement your circuitry as shown in Fig. 4, but you are welcome to switch the order of the microphone's amplification stage and put it after the bandpass filter to try it out.

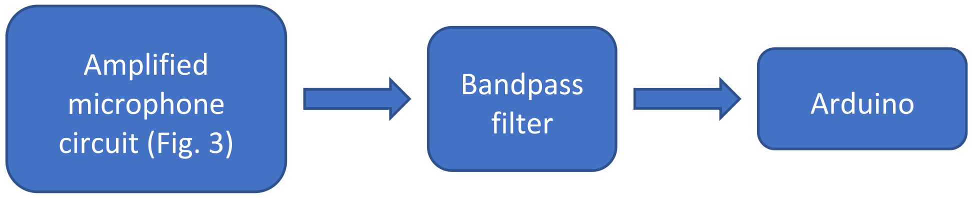


Fig. 4. Sequence of connection.

With your circuit in Fig. 4 implemented, you will test it with MATLAB as you did at the end of Section 5 to determine your bandpass filter's response compared to the theory (LTspice).

6. Canvas submissions: Frequency Responses with Bandpass Filter

1. **Upload the graph that shows the superposition of the smoothed $\widehat{H}_{Bandpass}(\Omega)_{experiment}$ and $\widehat{\mathrm{H}}_{\mathrm{Bandpass}}(\Omega)_{\mathrm{theory}}$ (the latter obtained by simulating the bandpass filter in LTspice) obtained from your experimental measurements and simulations, respectively.**
2. **Upload the graph that shows the plot of the difference between the simulated and experimental filter responses, $\widehat{H}_{Bandpass}(\Omega)_{theory}$-$\widehat{H}_{Bandpass}(\Omega)_{experimental}$.**
3. **Comment on points 1 and 2 above (i.e., what you observe, why theory and experiment are different, how good is the attenuation outside the bandpass region, etc.)**

This is the end of what must be completed in the second week of the release of this lab, 07 April – 13 April 11PM Ithaca time, Section 4 to Section 6 inclusively.

----

7. FFT on Arduino

You developed a good sense of the signal processing aspect of your robot, especially after doing much Fourier analysis. But for your robot, MATLAB will not be used to perform Fourier analysis: it must be performed on the Arduino. While the Nano's capabilities are not as powerful as your computer's, it still can perform Fourier analysis. For this, you will use a library.

One such library that works well is available from this site – we covered in some detail this library in class. Download the Arduino FFT library (V3.0) from Canvas (ArduinoFFT3.zip in Labs Module, Lab 3) and install it on your computer (scroll down about halfway down the site's webpage for the instructions on how to install the library). Read that site's main page too to get some understanding of what it does, but here's a summary of what the FFT process will look like in your sketch:

- The Nano will accumulate ADC sampled values (257 such values**: you will acquire 257 samples but will ignore the first value and the remaining 256 will be used for calculations**. Here on out, I will write "the 256 values" so keep in mind that in your code you will acquire 257) of the amplified ~~(and filtered)~~ sound signal at very specific time intervals of 0.41667ms (corresponding to a sampling frequency of what?) using interrupts (in fact, with code very similar to the one I went over in class).
    - This will ensure that this accumulation of sampled values takes place at precise time intervals for uniformity.
    - This sampling interval of 0.41667ms is much larger than the sampling interval of the ADC that runs in free-running mode, so it is an almost 100% accurate assumption that the time interval between each accumulated values is 0.41667ms.
- The 256 sampled values will be manipulated to convert them from a value between 0 and 1023 that the Nano provides to a signed integer with negative and positive values (because the 256 sampled values are a sampled version of the continuous time sound signal picked up by the microphone that, in reality, oscillates about the x-axis).
- The rescaled 256 sampled values will be processed by the FFT library on the Nano by invoking a series of commands from the library resulting in an FFT spectrum in 256 bins[3] (essentially, as explained in class, an FFT spectrum with 256 points but due to the symmetry of the FFT, only the first (or last) 128 spectrum values of the FFT are useful).
- For testing and characterization purposes, you can print the resulting 128 bins to the Serial Monitor and plot the result to visualize the spectrum obtained by the Nano (simply copy the values that are output to the Serial Monitor so that you can manipulate them in MATLAB).

Then, based on what will be in the spectrum that the Nano will calculate, your robot (in Lab 4) will make a decision as to what to do. What will happen in your robot is that it will "listen" to sound (via your amplified and filtered microphone circuit), a sound of a given frequency will be played by you eventually, and the Nano will perform Fourier analysis on the signal to determine the frequency of the played sound so that your robot can take appropriate action. This means that you need to code the Arduino so that it can do this.

You already have code from Section 3 in this lab that does the ADC in free running mode, you need now to add the FFT part from the library you just installed. The coding is a bit more tricky and involves you to think about what needs to happen in what part of the sketch. Here is what happens:

- The ADC will operate in free-running mode like before. This is very fast, much faster than we need, but we will keep it that way to learn and apply other things that we covered in class.
- Because the ADC in free-running mode samples the sound signal too fast (the sampling rate is so small that the this affects the binning of the spectrum in a way that it becomes

[3] A bin is nothing more than a grouping of frequencies: each bin "contains" a range of frequencies. See previous lectures on FFT.

unusable), you will code the Nano to only take a few of the values spit out by the free-running ADC at an interval of 0.41667ms between each. This will be done using interrupts on TCA.

This means that your code from Section 3 will have to be modified and will contain the following functions/commands named the way they are below (name your file `freeRunADC_ISR_netID.ino`):

- It will not contain any printing to the Serial monitor except possibly at one location (see below)
- It will contain the appropriate declarations of variables (e.g., volatile for those used in the ISR, arrays for the ADC values, etc.)
- It will contain the inclusion of the FFT library and the definition of constants used for and by the FFT library – write the following three statements as written here:
  - `#define LOG_OUT 1 // use the log output function`
  - `#define FFT_N 256 // set to 256 point fft`
  - `#include <FFT.h> // include the library`
- **`void setup()`** section that will have
  - `Serial.begin(9600);`
  - The code to setup the free-running mode of the ADC (except now that you should change the Baud rate to 9600 – we will not use MATLAB now)
  - Three statements to capture the state of the CTRLC, CTRLA and MUXPOS registers of the ADC *before* the free-running parameters are defined (this is so that once the capture of the 257 samples is complete, you will restore the ADC to its original state so that when you use `analogRead()` in Lab 4, the ADC will behave as expected)
  - The statements to setup the interrupt on TCA for normal mode, overflow interrupt, and an interrupt interval equal to 0.41667ms
  - The rest of the ADC and interrupts statements
- **`int ADC0_read(void)`** function as you had it before
- **`ISR(TCA0_OVF_vect)`** interrupt routine that will contain three statements:
  - one to read the ADC value written as `adcVal[counter] = ADC0_read();`[4] where `counter` is a counting integer global volatile variable that counts up from 0 to 256 to capture the 257 time signals.
  - one that will increment `counter` by 1
  - one that will clear the interrupt flag[5]
- **`void loop()`** which will be an adequately coded section (it requires thought) that will only be executed if there is a sufficient number of samples collected (257), in which case the following is executed:

[4] Note here that adcVal[counter] is an array – see the Arduino online reference for arrays.

[5] The alert reader will notice that there are numerous things to be executed by the ISR which is *not* the way that ISR routines should be written. In this case, as you will see in Lab 4, this ISR routing will be used only in the beginning when your robot is motionless and is waiting to receive a command which will be a sound signal. After that, this ISR routine will be ignored.

    - The TCA is disabled
    - The ADC registers whose default values were saved in the **`setup`** section are restored
    - The 256 (ignore the first one of your 257 read values, `adcVal[0]`) ADC values are converted into signed (i.e., positive and negative) 16 bit numbers
    - The converted values are then stored in the even numbered (0, 2, 4, …, 512) indexed `fft_input` array (the odd numbered (from 1, 3, 5, …, 511) indexed `fft_input` array will contain 0) that the FFT library will use to calculate and process the FFT:
        - `fft_input[even number]=` converted ADC value[6];
        - `fft_input[odd number]=` 0;
    - The following commands are executed as written here and in this order (see the FFT site's for a description)
        - `fft_window(); // window the data for better frequency response`
        - `fft_reorder(); // reorder the data before doing the fft`
        - `fft_run(); // process the data in the fft`
        - `fft_mag_log(); // take the output of the fft`
    - You can use `Serial.println()` to display the values of the FFT bins (which you can copy/paste into MATLAB or other to plot and visualize). The values of the FFT (i.e., the spectrum) are contained in the array `fft_log_out[]` which is an array and contains 128 values.
    - And for the purpose of this section and this lab only, you can include a very long delay/infinite loop so that your code doesn't try to run again (e.g., you can use `while(1) {}`) (after all, it is in the loop section!).
- It will not contain the function **`bool ADC0_conversionDone(void)`** from earlier (comment it out).

So once everything is coded, your .ino sketch will read 257 ADC values at specific time intervals that will be called by an interrupt routine that will be set to be triggered every 0.41667ms. Once those 257 ADC values are read, you will discard the first one and process the 256 remaining values so that they can be stored into `fft_input.`Then the FFT library will be invoked to calculate the spectrum from the data.

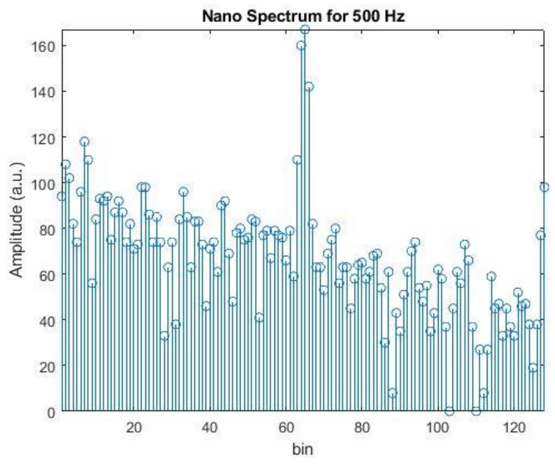
Upload the code to your Nano.

With this code running on the Arduino, what you will do is play a sound as you did before (i.e., with MATLAB: note that MATLAB will not be used to communicate with the Arduino) and while the sound is playing, your above Arduino code will capture the data from the played sound and output the resulting spectrum in the Serial Monitor. To play sound only, download the file `playSoundOnly.m` from Canvas (you will recognize it). In that file, you can change the sound duration, the initial and final frequencies. So here is what you will do:

[6] This means that fft_input[0] will contain the first converted ADC value, fft_input[2] will contain the second converted ADC value, fft_input[4] will contain the third converted ADC value, etc. Also, fft_input[1]=0, fft_input[3]=0, fft_input[5]=0, etc.

A. Modify the `playSoundOnly.m` file to play a sound with initial frequency of 500Hz, final frequency of 500.1 Hz, and a duration of 10 seconds. Run this file and sound will be played.
B. Open the Serial Monitor, press and hold RESET on your Arduino, click on “Clear output” in Serial Monitor to clear the display.
C. While the sound is playing, release the reset button on your Arduino and the code will run, acquire the data, perform Fourier analysis on the data, then output the data on the Serial Monitor.
D. Copy the data (which is the spectrum that the Nano calculated) from the Serial Monitor and use MATLAB (or other) to plot the data. You can use the function `stem` in MATLAB for a plot like this one. You should observe a nice peak in/near the bin that corresponds to the frequency that was played (here, near 500 Hz).

You should in this case get something that looks like the figure below. Note that in the figure below, Carl obtained it using his bandpass filter. You do not need to implement your BP filter so your signal may be more noisy. You can observe a nice strong signal that really stands out and its corresponding bin(s) should correspond to the frequency of interest. You will repeat the above for 700Hz and 900Hz to ensure that your Arduino, microphone and amplifier circuits and your code respond well to this range of frequencies.



7. Canvas submissions: Code and Final Results

1. Upload your file `freeRunADC_ISR_netID.ino`
2. Upload the spectrum obtained with the Nano for a sound frequency of 500 Hz
8. **3. Upload the spectrum obtained with the Nano for a sound frequency of 700 Hz**
4. Upload the spectrum obtained with the Nano for a sound frequency of 900 Hz

Report (Quiz) & Wiki

Enter all required material and answers on Canvas as described in the handout. Update your wiki nicely and professionally, including with pictures of your progress so far with everything that you did in this Lab. Do NOT upload code to your wiki. Include graphs obtained herein along with explanations of what you are working on. Grading for this lab involves grading the Lab 3 quiz, as well as your wiki. At the end of the semester, points for your wiki will be given for the overall look/feel/ease of navigation of your wiki.

9. Wrap-up

Disconnect the USB cable from the Arduino, and disconnect all battery wires from the breadboard. Leave the microphone and amplifier and filter circuits on: you will need them in Lab 4. Put your robot frame away securely to make sure that it will not fall on the floor. *It is fragile!*

This is the end of what must be completed in the third week of the release of this lab, 14 April – 20 April 11PM Ithaca time,
Section 7 to Section 9 inclusively.
