Lab 4

Digital Filtering I

Authors:

Jonas Lussi, Tarik Rifai, Naveen Shamsudhin, Franziska Ullrich, Kathrin E. Peyer, Dimitris Felekis, Bradley E. Kratochvil, Chauncey F. Graetzel, and Prof. Bradley J. Nelson

Institute of Robotics and Intelligent Systems

Date: 2019

Version: 2.0

Summary: These weeks you will design and implement two different first order FIR (Finite Impulse Response) low pass filters. You will learn how to determine the experimental frequency response of each filter. During the lab session, you will measure the frequency response of the filters and compare the results with the analytical solution. You will learn:

- Offline and online FIR filtering in **C**
- Digital filter design and analysis with MATLAB

4.1 Background

4.1.1 Digital Filtering

The background information for this lab is given in the lecture entitled **Digital Filtering**.

4.1.2 Sleeping

Often times, when running a control program we do not require it to run at the maximum speed capable of the computer. We must then instruct our program to give up its processing time and go to *sleep*. You can do this with a command such as

```
usleep(0);
```

This function theoretically allows the process to sleep for a prescribed number of **microseconds**. Unfortunately, this function is limited by the resolution of the program in charge of managing processes (the scheduler). We are using a non-realtime version of Linux, and are thus limited to the resolution of the scheduler which can be determined experimentally by having the process sleep for the minimum amount of time 0 microseconds. On some of the lab machines, e.g. the process runs with approximately 15 kHz. 1 KHz is sufficient for experiments we run in the lab. However, if you run several commands, particularly when communicating with hardware, this can add processing time and a usleep(1000), which in theory creates a 1ms delay , does not necessarily result in a process frequency of the desired 1 kHz. Thus in the rest of the labs, when you are asked to sample data at 1 KHz you must first test the communication frequency of the loop of your code (see description below) and adjust the value of usleep() accordingly.

As a related point, since this is not a realtime operating system it is **not guaranteed** that the process will return after the allotted time. Thus, when you sleep it may be longer than intended. This can be observed in this lab while sampling the input signal and outputting it to the oscilloscope. For example, if you click on the menu bar of a window and drag it around while sampling, this will cause the windowing process to use more time and your filtering program to get less.

4.1.3 Testing the communication frequency

The easiest way to get the communication frequency is by measuring the time it takes for a *for* loop (e.g. n = 5000), in which the data is sampled, and then dividing the time by the number of samples. This is more accurate than trying to measure how long only a single loop takes. There is already a function implemented that measures the time it takes from `tic()` to `toc(0)` in seconds. In order to use this function, the library `<smaract_util.h>` has to be included. The description of the function is copy-pasted below:

```
// tic and toc functions work together to measure elapsed time.
// tic saves the current time that TOC uses later to measure
// the elapsed time. The sequence of commands:
//
//          tic();
//          operations
//          toc(0);
//
inline void tic()
{
  gettimeofday(&start_time, NULL);
}

inline double toc(int restart)
{
  struct timeval now;
  gettimeofday(&now, NULL);
  double elapsed_time = now.tv_sec - start_time.tv_sec +
                        (now.tv_usec - start_time.tv_usec)/1e6;
  if (0 != restart)
    tic();
  return elapsed_time;
}
```

4.2 Prelab Procedure

Note: Pelab assignments must be done before reporting to the lab and must be turned in to the lab instructor at the beginning of your lab session. Additionally, you also have to upload your solution as a single PDF-file to moodle. Make sure to upload the file before your lab session starts, late submissions will not be corrected. The prelab needs to be handed in as a group. All the prelab tasks are marked with **PreLab Qx**

For the two algorithms we need in this lab, we would like to be able to sample the signals indefinitely and in real time. This makes the programming a bit more challenging due to having to keep track of the last M samples or develop a recursive implementation. In the lab procedure, you will first run the program over a finite series of pre-sampled measurements. This is useful to test if you understood the Blackman filter and have chosen sensible parameters. Afterwards, you will program your algorithms to filter indefinitely in real time.

1. Write a C function that implements a smoothing filter to average data over the previous N samples. We want this function to be easily used by different programs in the future. Therefore integrate this function in the `digital_filter.c` file skeleton that was given to you. The function has already been declared and is called "smoothing_filter". The specifics are described in the `digital_filter.h` (declaration) file. **(PreLab Q1)**

2. Design a first order FIR filter using a Blackman windowed sinc with a cutoff frequency of 100Hz and a transition band of 20Hz. Assume a sampling frequency of 1000 Hz. You don't have to write any code, but just note on your prelab report the M and f_c you found. **(PreLab Q2)**

3. Write C functions that implement a windowed sinc filter with Blackman window. Integrate these functions in the `digital_filter.h` and `digital_filter.c` files. You will have to write two functions "blackman_coefs" and "blackman_filter" to implement the filter. Refer to the `digital_filter.h` file for details. You will need to include `math.h` to have access to the `sin()` function to calculate the kernel. **(PreLab Q3)**

4. Study the files of this week's lab from the course website

5. Print out your Prelab solutions (Code from Prelab Q1, Q3 and answer from Prelab Q2 and turn them in to the lab instructor at the beginning of your lab session. Upload your Prelab as a single pdf to the moodle platform.

4.3 Lab procedure

The skeleton code is available on the moodle platform. Download the files on your Udoo and copy them into a folder called `irm/lab04`.

4.3.1 Digital Filtering

This laboratory assignment involves implementing a first order digital filter in **C** and comparing the results to the analytical frequency response that you determined in the prelab.

1. Download a function generator app (e.g. "Audio Funktionsgenerator" for Iphone and "Signal Generator (XYZ Apps)" for Android) in your cell phone. Connect your phone to the oscilloscope with the provided cable and set the function generator to output a 20 Hz sine wave with a magnitude of about 200mV RMS (565.6 mVpp). This sine wave will have a DC offset of 1.3V from the cable. The reason for that is a circuit in the cable converting the output voltage of microphone from the range (-2V to 2V) to the range (0V to 2V). Arduino only accepts the voltage range from 0V to 3.3V. The voltage converter circuit uses a 3.3 V power supply (red cable). Make sure to connect the cables correctly - the document with the voltage converter will help you understand the circuit.

2. Now, connect your phone to the ADC0 channel (A0) and connect channel 2 of the oscilloscope to the DAC1 channel according to the pinout detailed in Arduino.

3. Write a function in Arduino to sample the ADC and output the **same** signal to the DAC. Make sure you are sampling at 1 kHz by adjusting delay() value (no need to measure exact communication frequency here). Show your working system to your assistant.**(Postlab Q1)**

   Hint: The DAC of Arduino can only output the voltage from 1/6 to 5/6 Vcc (Vcc=3.3V). Therefore, the output signal from DAC would have a difference in DC offset of 1/6*3.3V = 0.55V and a rescaled amplitude of 2.2V/3.3V = 2/3 folds. Recover the output signal by reversing the DC offset and scale value. Use the functions analogReadResolution(12) and analogWriteResolution(12) to set the resolution to 12 bits and analogRead() and analogWrite() to output the voltage. Run your program for some seconds and observe the output on the oscilloscope. You may be able to note how the signal output by the DAC is discretized along with the time delay in the signal.

4. In Geany, complete the file `RMS.c` (provided as skeleton) where you implement a function `RMS()` that calculates the RMS value of the sampled data. For discrete distributions, the RMS value can be described as: $R(x) = \sqrt{\frac{\sum_{i=1}^{n} x_i^2}{n}}$ Where, x_i is your data at interval i. **(Postlab Q2)**

5. Write a C program `lab04_test.c` (provided as skeleton) that samples the signal (read 5000 samples from the Arduino) and saves it in an array (check that you are sampling at 1 kHz by measuring the communication frequency and adjusting usleep() accordingly). Now take the sampled data, convert it to voltage values and run separately the moving average (N = 10) and Blackman windowed sinc (cutoff frequency 100Hz and bandwidth 20Hz) over it and take the RMS() afterwards of the sampled data. Also take the RMS() of

unfiltered data which should always be around 200mV. Repeat this for the following frequencies (see list below) and save the values into a .dat or .txt file. Then plot the data, e.g. in Matlab (label the axes: y= RMS; x = frequency as fraction of sampling frequency 1000 Hz). Comment on the cutoff frequency. **(Postlab Q3)**

Frequency (Hz)	Unfiltered Data (RMS value)	moving average (RMS value)	windowed sinc (RMS value)
20	____	____	____
25	____	____	____
30	____	____	____
35	____	____	____
40	____	____	____
45	____	____	____
50	____	____	____
60	____	____	____
70	____	____	____
80	____	____	____
90	____	____	____
100	____	____	____
110	____	____	____
120	____	____	____
130	____	____	____
140	____	____	____
150	____	____	____
175	____	____	____
200	____	____	____
225	____	____	____
250	____	____	____

4.4 Postlab and lab report

Please hand in an organized report before your next lab. Upload a single PDF-file with your solutions to the moodle platform that includes:

- The plots and comments from task 5 (Postlab Q3)
- All the source code you wrote from tasks 3, 4 and 5 (Postlab Q1-Q3).
- After you upload the PDF-file on moodle, print it out and hand it in to the assistant at the beginning of the next lab session.

4.5 Literature Reference

[1] Smith, Steven W., "The Scientist and Engineers Guide to Digital Signal Processing." California Technical Publishing, Sand Diego, CA 1997-1999.