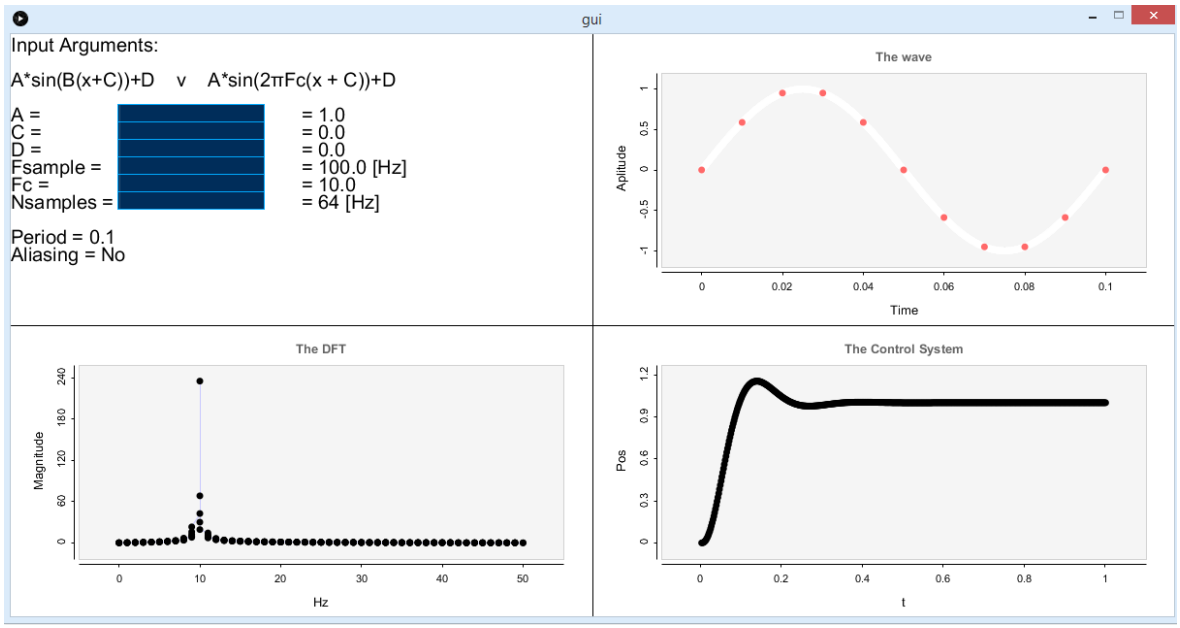
5LIU0 Project



Name:	Ömer Yildiz
Student ID:	1644300
Canvas group:	6
Date:	29-01-2020
Course:	5LIU0
Lecturer 1:	Dr. Alexios Balatsoukas Stimming
Lecturer 2:	Dr. Dip Goswami
Version:	1.0

Index

Introduction

For the 5LIU0 course, the students could define their own project related to signal processing or feedback control. During my previous education (HBO Embedded Systems, HBO from now on) both subjects were touched upon, control systems much more than signals. The idea was to combine a project related to signals with a control related project.

On the HBO, I struggled with understanding the practical side of the signals course. For a group project, my partners wrote a Discrete Fourier Analysis (DFT) in C, while I was programming an FPGA for the game we were developing. I wanted to get some experience in programming my own DFT.

As for the control part, we were only taught in simulations during the HBO courses. We once wrote a PID controller, but not from the ground up. I would like to write my own controller just to see if I could do it and what challenges I would face.

This report details my journey through this project.

In the first chapter the assignment is discussed in detail. The goals, deliverables and criteria are discussed. This project consists of multiple parts and so, chapter 2 is divided up into sections. Each section details how that part of the project was developed, which design considerations were made and how they were tested. Finally, a conclusion is given along with a look back at the project.

1 Assignment

This chapter details how and why the project changed after the project proposal. The goals and deliverables are updated. Criteria for each part of the project are given.

1.1 Assignment description

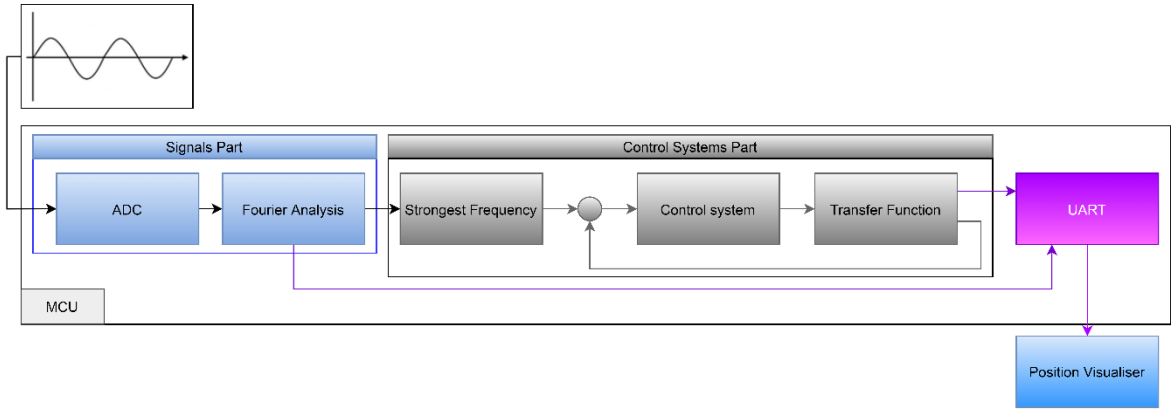


Figure 1 Block Diagram of the Project Proposal

The goal for this project was to combine Signals and Control systems projects into one project, see Figure 1 In my previous education (HBO Embedded Systems, HBO from now on) both concepts were touched upon separately (control systems much more than signals).

For the signals part, the first two proposed signals related projects in from the 5LIU0 Study Guide (Stimming & Goswami, 2020) were implemented:

- A selected, real-time, signal processing algorithm, for instance filtering, or feature extraction, implemented in software on a pc, or on a microprocessor board.
- Signal visualization. Extract and visualize relevant features from an input signal. For instance, related to energy or spectral content of the signal.

Originally a microcontroller would sample an analog signal using its build-in ADC, after which a Fourier analysis (DFT) would be done to find the strongest frequency present in the sampled signal.

$$H(s) = \frac{6.988 \cdot 10^5}{s(s+85)(s+200)}$$

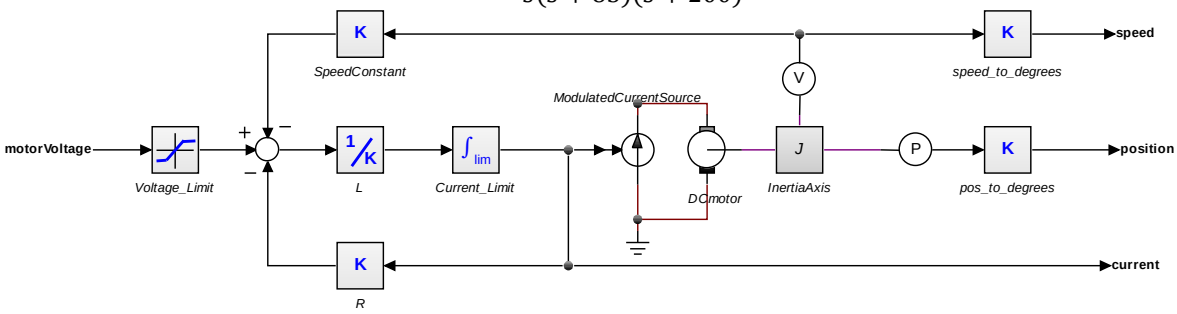


Figure 2 Block Diagram of the robot arm and the transfer function

This strongest frequency, a number, would be used as the reference to the control part of this project. Either a PID or Full-State Feedback controller would be written in C for the microcontroller. The plant is a transfer function of an assignment from HBO, Figure 2.
We worked with a simulated robot arm to determine if a PID or Full-State Feedback controller (FSFB) would be best for that case. The robot arm simulation model is only used for its transfer function. The position of the robot arm is determined by the input frequency.

Using the UART of the microcontroller information would be sent to the pc to visualize the Fourier Analysis and the control system output.

Shortly after the start of the project, it was decided to not program an embedded system. The function generator used for generating the sinusoidal signal either had bad spectral purity or there was something wrong with the microcontroller. Either way, it could not be determined as due to corona virus measures the campus (HU) was closed. I had no access to the labs to determine what the problem was. It was decided to write all the programs that would run on the microcontroller on Windows in C. If it needed porting to a microcontroller it could easily be done.

1.2 Goals and deliverables

Table 1 shows the goals of this project. The project is broken up into manageable pieces, shown in Figure 3 .

Table 1 Goals and Deliverables

Goals	
1. Writing a Fourier Analysis in C. 2. Writing my own PID or Full-State Feedback controller in C.	
Deliverables	
Must	**Could**
1. A demonstration of the Signals part with the visualizer. 2. The software with detailed documentation.	1. A demonstration of the Control part with the visualizer 2. A demonstration of the Signal and Control part working together with the visualizer.

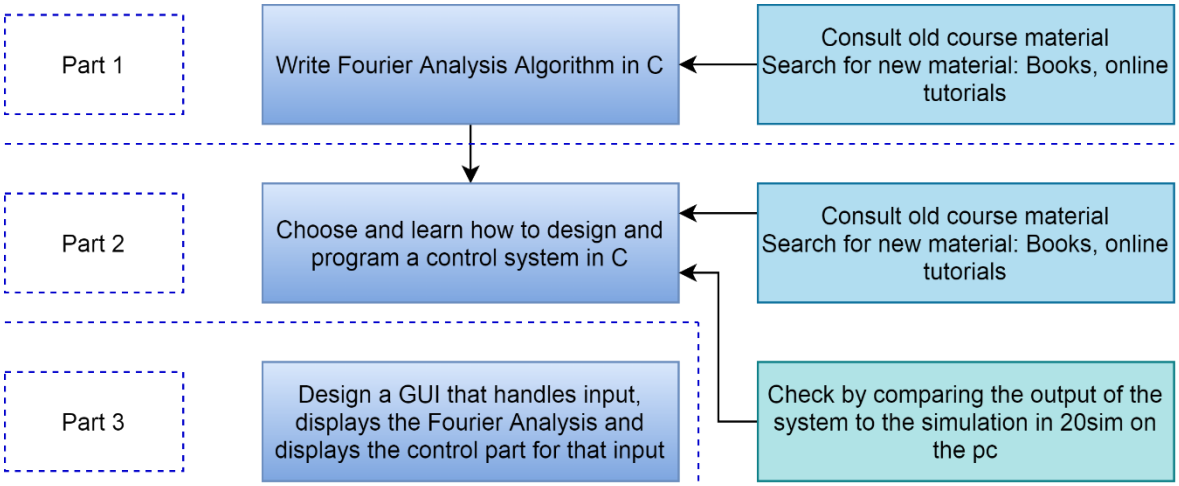


Figure 3 Flow chart project progression

1.3 Criteria

As Figure 3 suggests, this project consists of three parts:

1. Visualizer
2. DFT program
3. FSFB program

An unwritten but possible use case for this project was that it could be used for educational purposes for my previous education and for the TUE. Therefor it would perhaps need to run on different computers. It was decided to have the visualizer be an executable file, for Windows, that did not rely on other programs installed on the computer.

The DFT and FSFB programs have no hard criteria. However, they are compared to (professional) tools during development. Their output should match these tools. It does not have to be exact, but any deviation that cannot be classified as a rounding error is too much. This would be apparent in the output and this was kept in mind during development.

2 Realization

This chapter details how each part of the project was developed. The design considerations and choices during development are discussed for each part.
First the DFT part is discussed which is followed by the FSFB and the Visualizer parts.

2.1 DFT

During the HBO, we used “The Scientist and Engineer’s Guide to Digital Signal Processing” book to learn about DSP (Smith, 1999). Chapter 8 goes explains how the Discrete Fourier Analysis works. Simply put: it describes how an individual sample in the frequency domain is affected by all of the samples in the time domain (Smith, 1999).
The chapter also includes a Discrete Fourier Analysis written in BASIC.

This paragraph details the creation and verification of the DFT program written in C, based on the BASIC program. The name of the program is DFT.c

2.1.1 Baseline

Before writing the C code, Excel and an online tool were prepared to set expectations for the final C code. Matlab would also have used, but at the time the written Matlab code did not give a satisfactory output.

Excel was used for:

1. Generating the sine values for the online tool
2. Performing its build-in FFT function on the same sine values.

A sine wave of 100Hz with a sample frequency of 1kHz was created in Excel. The output of Excels FFT function is given in Figure 4 and the output of the online tool (using the same sine values) is given in Figure 5. Note that Excel cuts of at half the sampling frequency while to online tool goes up to the sampling frequency. This is intentional, as the frequencies above halve the sampling frequency are not used.

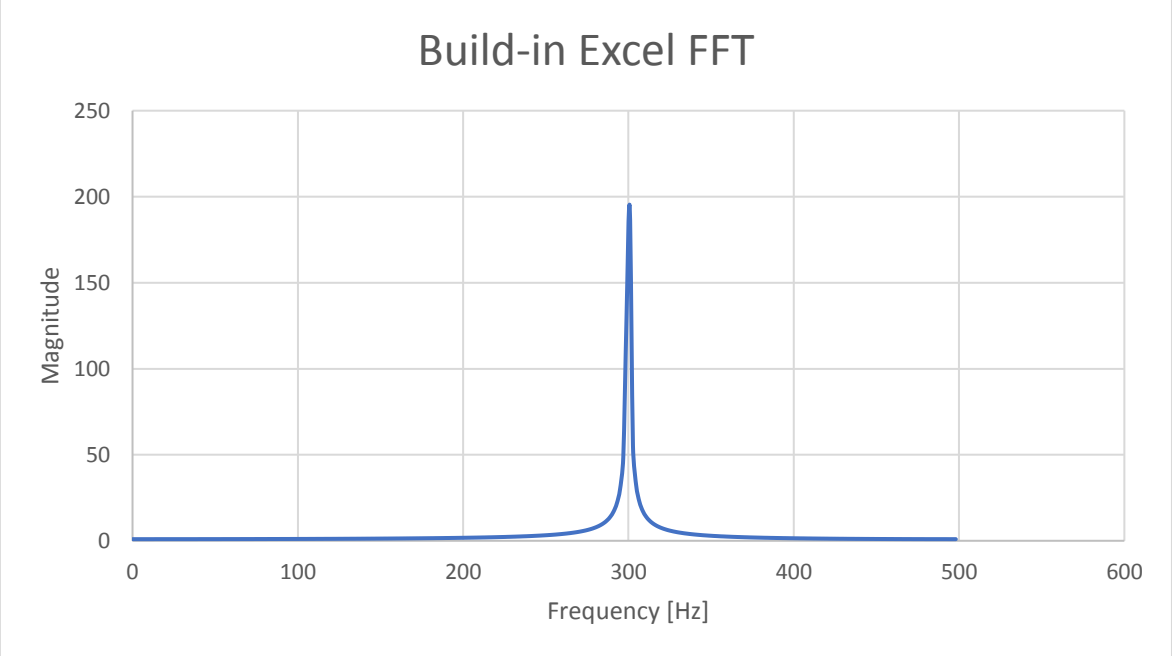


Figure 4 Excel DFT. Fsample = 1000Hz, Fsignal = 300Hz

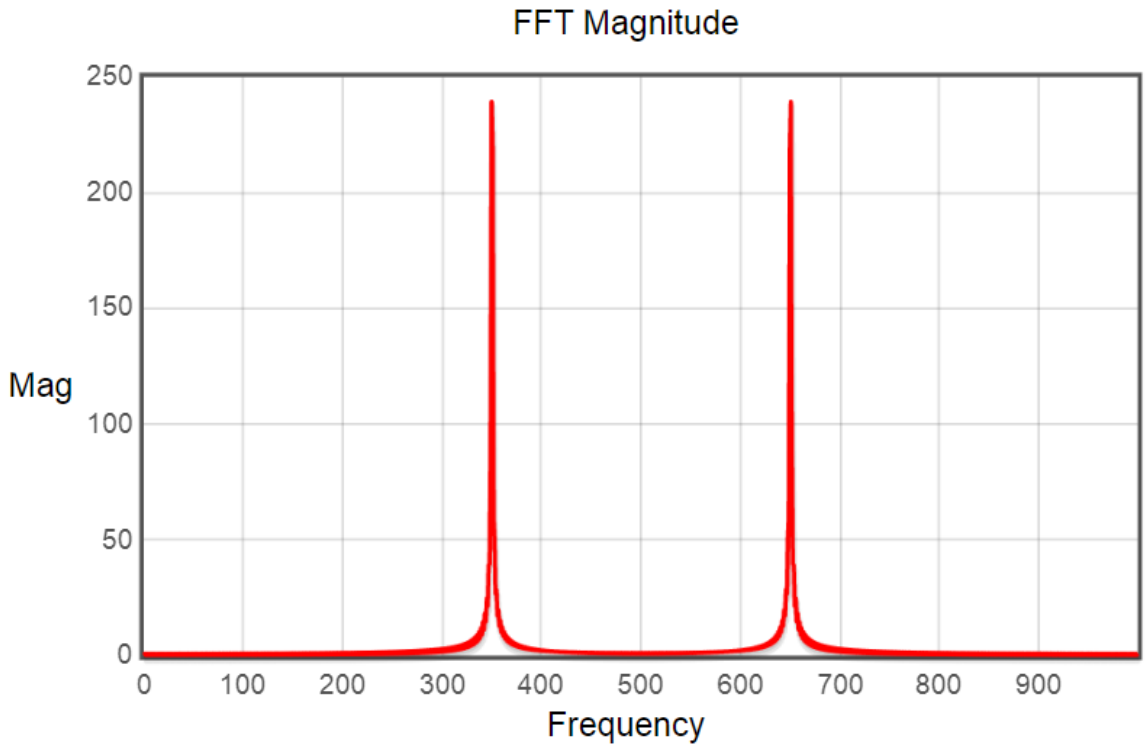


Figure 5 FFT Magnitude plot from the online tool (Ahmed, 2013)

2.1.2 C code

A high-level overview of the written program is given in Figure 6. The program generates its own sine values. The reasoning was, if this would be run on an embedded system one would only need to send some parameters instead of a lot of sine values. The sine values are stored in a text file for later use.
The actual DFT code is given in Code Snippet 1. Two arrays are used to store the real and imaginary outputs of the DFT. These are stored in their own text file. Excel is used to plot the output.

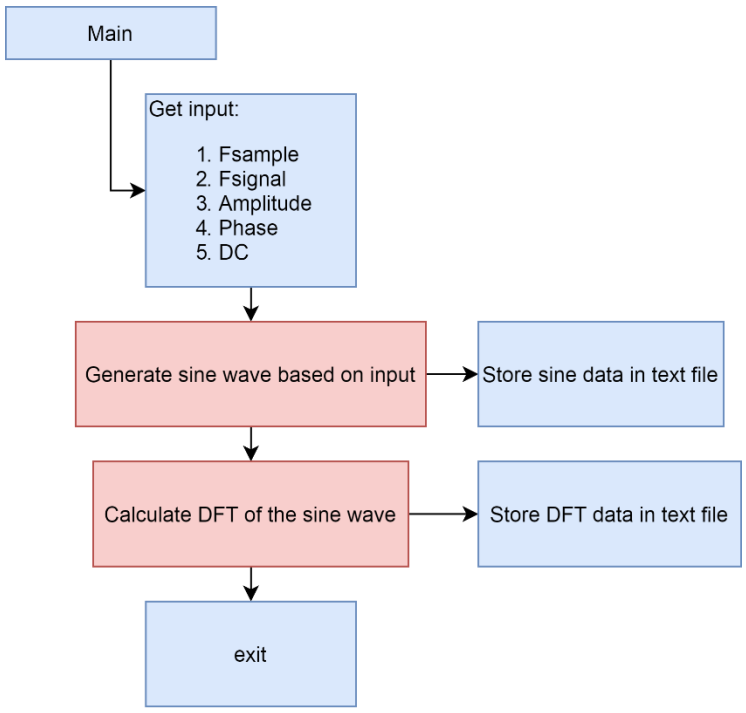


Figure 6 High-level overiew DFT.c

```c
for(k = 0; k < N/2; k++)
{
    for(i = 0; i < N-1; i++)
    {
        ReX[k] = ReX[k] + input[i]*cos(2*PI*k*i/N);
        ImX[k] = ImX[k] - input[i]*sin(2*PI*k*i/N);
    }
}
```

Code Snippet 1 DFT in C

2.1.3 Verification

The output of the C code will be compared against Excels FFT, the online tool and Matlab. After writing the DFT program I managed to put together Matlab code to plot the FFT of given sine values.

A sinewave of 1600Hz with a sampling frequency of 5kHz was generated.
Figure 7 gives the output of the DFT C program. Figure 8 gives the Excel output, Figure 9 the online tool and Figure 10 gives the Malab output. The Matlab code is shown in Code Snippet 2. It requires that a vector x containing the sine values and scalar Fsample have already been defined in the Matlab command window.

```
close all

adft = fft(x); % calculate the FFT of the sine values
xdft = adft(1:length(x)/2+1); % Ignore the upper half of the values
length(x) % Print the lenght of the vector x

DF = Fs/length(x) % Frequency increment
freqvec = 0:DF:Fs/2; % Frequency vector, it is the x-axis

plot(freqvec,abs(xdft)) % Plot the absolute values of the FFT
```

Code Snippet 2 Matlab code

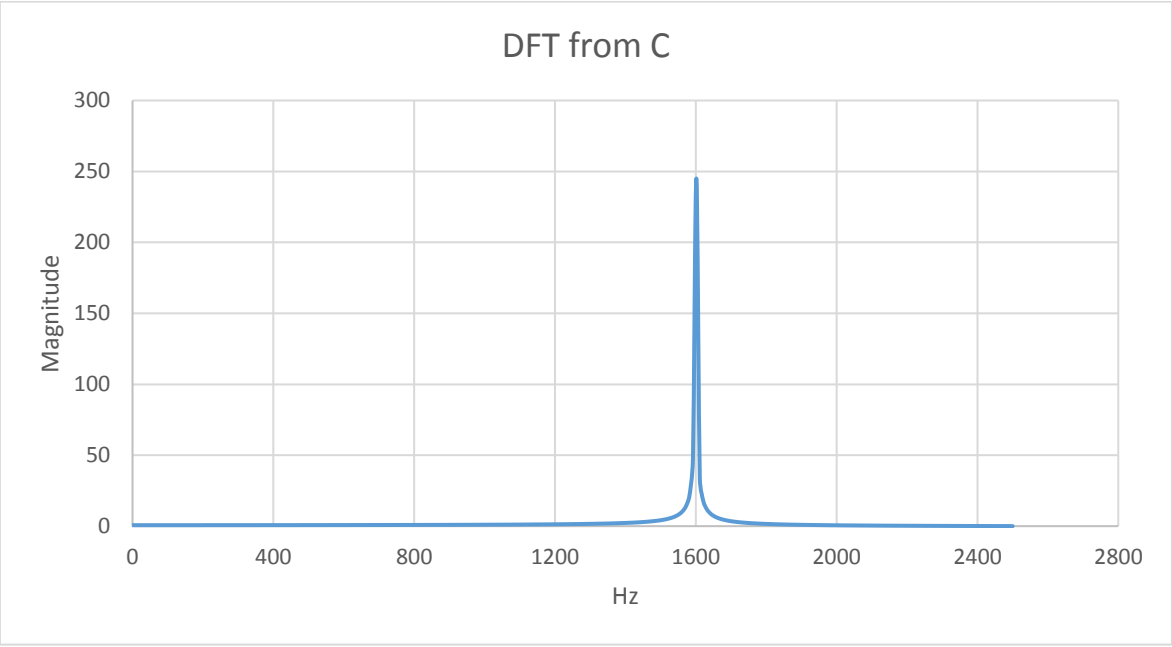


Figure 7 DFT calculated by C code, Fsample = 5000Hz, Fsignal = 1600Hz

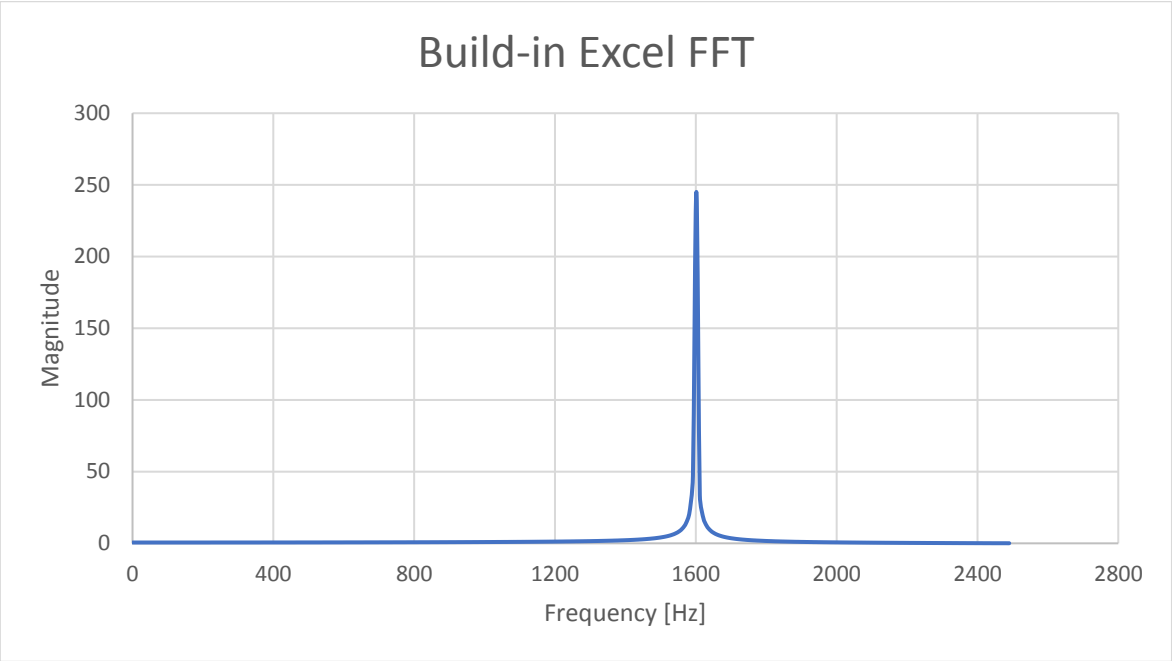


Figure 8 Excel FFT, Fsample = 5000Hz, Fsignal = 1600Hz

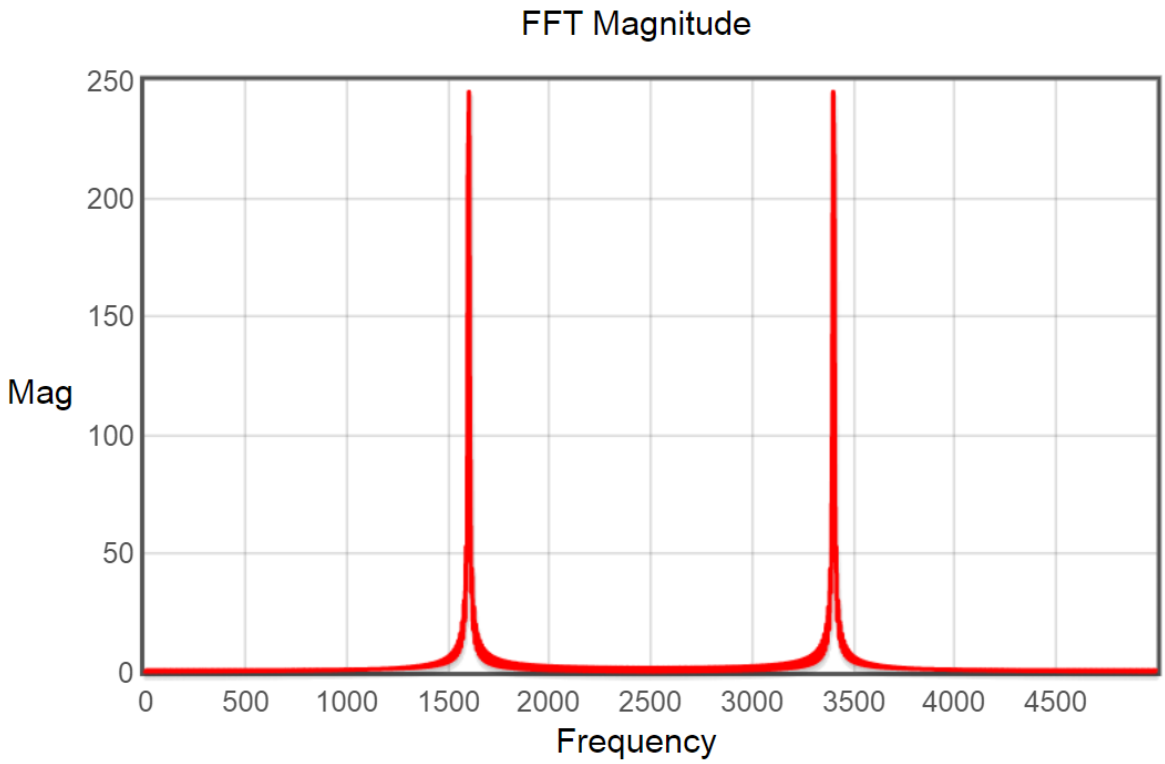


Figure 9 Online tool output, Fsample = 5000Hz, Fsignal = 1600Hz

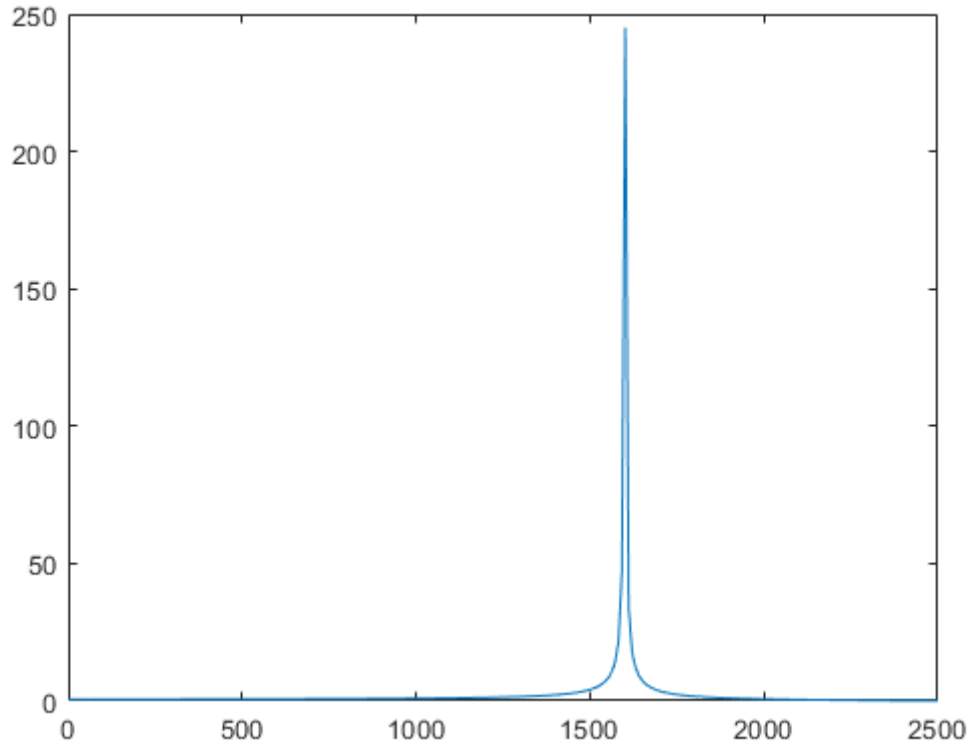


Figure 10 Matlab output, Fsample = 5000Hz, Fsignal = 1600Hz

2.1.4 Tricky x axis

After calculating the DFT, the horizontal axis needs to be homologized to the frequency range. If this would not be, Figure 8 will look like Figure 11. (Alexios Balatsoukas Stimming, Personal Communication, 22-01-2021). This can be done by remapping the input range [0, bin size] to the frequency range [0, Fsample].

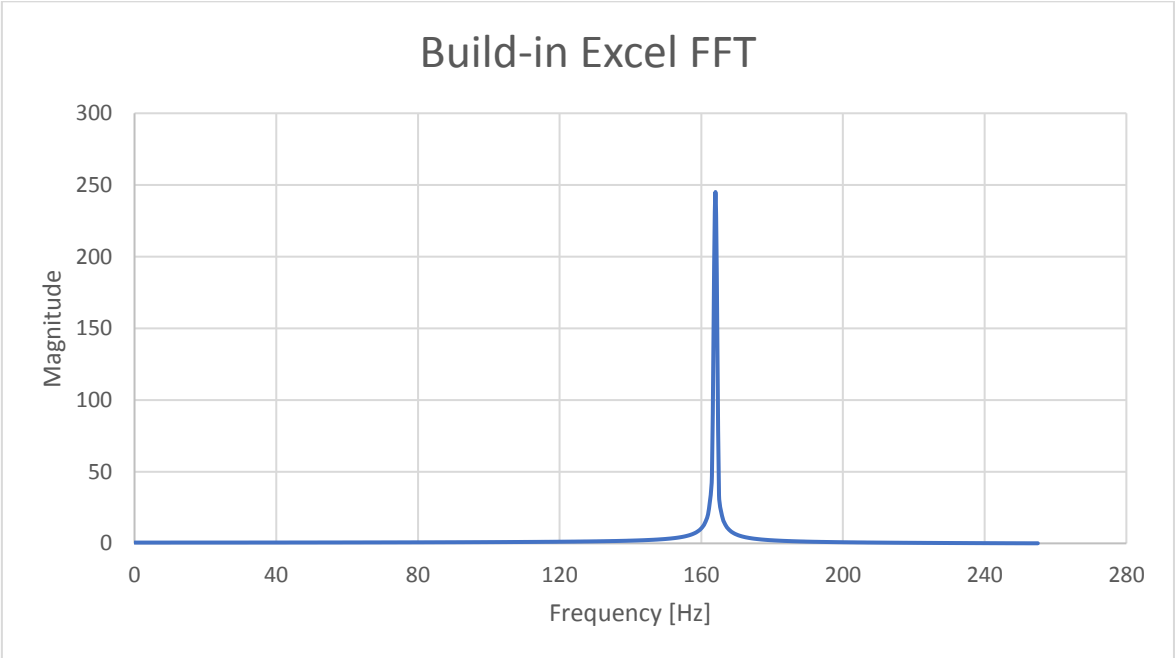


Figure 11 No frequency homologation version of Figure 8

2.1.5 DFT conclusion

A Discrete Fourier Analysis program was written by translating a BASIC program to C. It was compared to Excel's build-in FFT function, Matlab and an online tool. It performs identically as the platforms it was compared to.

2.2 FSB

20Sim is a professional software package that was used for my HBO courses. The license for version 4.8 was obtained by my HBO teachers. It runs Matlab under the GUI.

This paragraph details how the C code was iteratively developed. A model/feature is first explained using 20Sim to then be implemented in C code. The resulting output is compared. The C program is called FSB.c.

2.2.1 The plant

Figure 12 shows the base 20Sim model. The RobotArm block is the single block representation of the block diagram shown in Figure 2. It has 3 outputs:

1. Position of the robot arm, in degrees
2. Speed of the robot arm, in radians/second
3. Current drawn by the motor arm, in Ampère

Linearizing the model results in equation 1, the transfer function. Equation 2 is the factored version of equation 1. Figure 13 shows the Root Locus plot of the model. The poles are roughly at 0, -85 and -200.

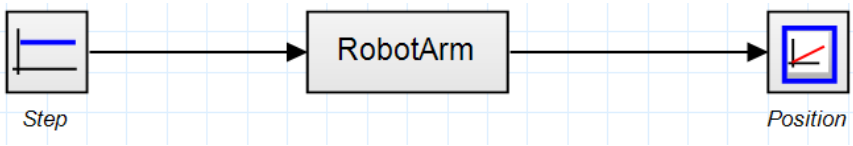


Figure 12 20Sim model

$$H(s) = \frac{6.988 \cdot 10^5}{s^3 + 285.7s^2 + 1.707 \cdot 10^4 s} \quad (1)$$

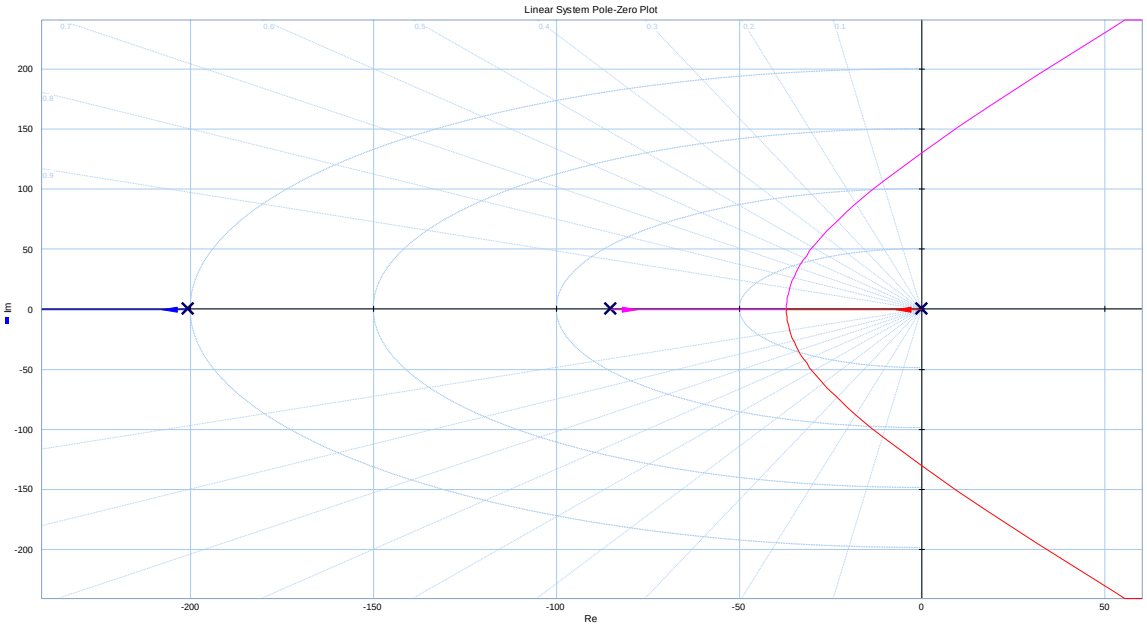


Figure 13 Root Locus plot plant

2.2.2 Plant without FSB in C

One of the assignments for an HBO course was to implement a second order transfer function on an arduino. The base code was written by one of the teachers (Moes, 2015). This code was extended to work with third order transfer functions, like in equation 1.

$$H(s) = \frac{a}{bs^3 + cs^2 + ds + e} \tag{2}$$

$$\dot{x}_1 = x_2 \tag{3}$$

$$\dot{x}_2 = x_3 \tag{4}$$

$$\dot{x}_3 = \frac{1}{b}(-cx_3 - dx_2 - ex_1 + au) \tag{5}$$

The transfer function is first rewritten into a state-space representation.
Example: equation 2 is broken up into equations 3, 4 and 5. These functions are converted to C code. The input is u and, in this case, x_1 is the output. The output is subtracted from the input at the beginning of each loop. A high-level overview of the code is given in Figure 14.

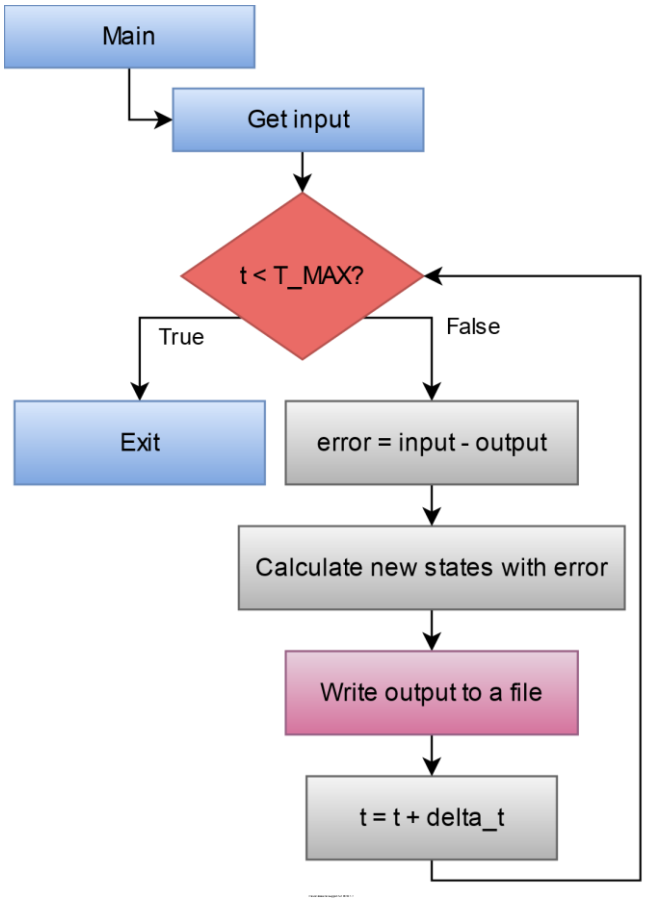


Figure 14 High Level overview FSB.c

Unit feedback was added to the model in Figure 12 in 20Sim, resulting in Figure 15, and a step response was generated . The output of FSB.c for a unit step response and unit is given Figure 16 and the output of 20Sim in Figure 17. Both give the same output.

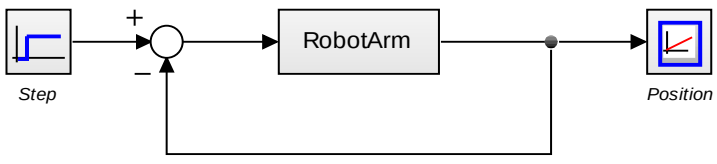


Figure 15 20Sim model with unit feedback

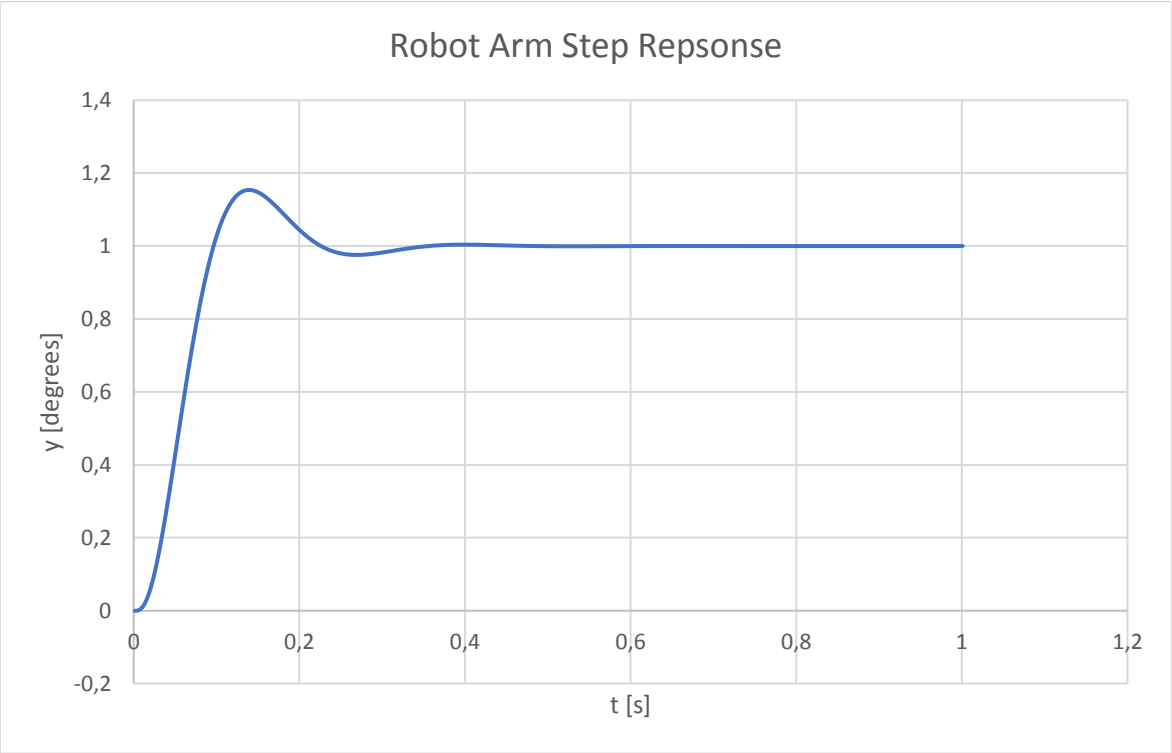


Figure 16 C code output on Step Response

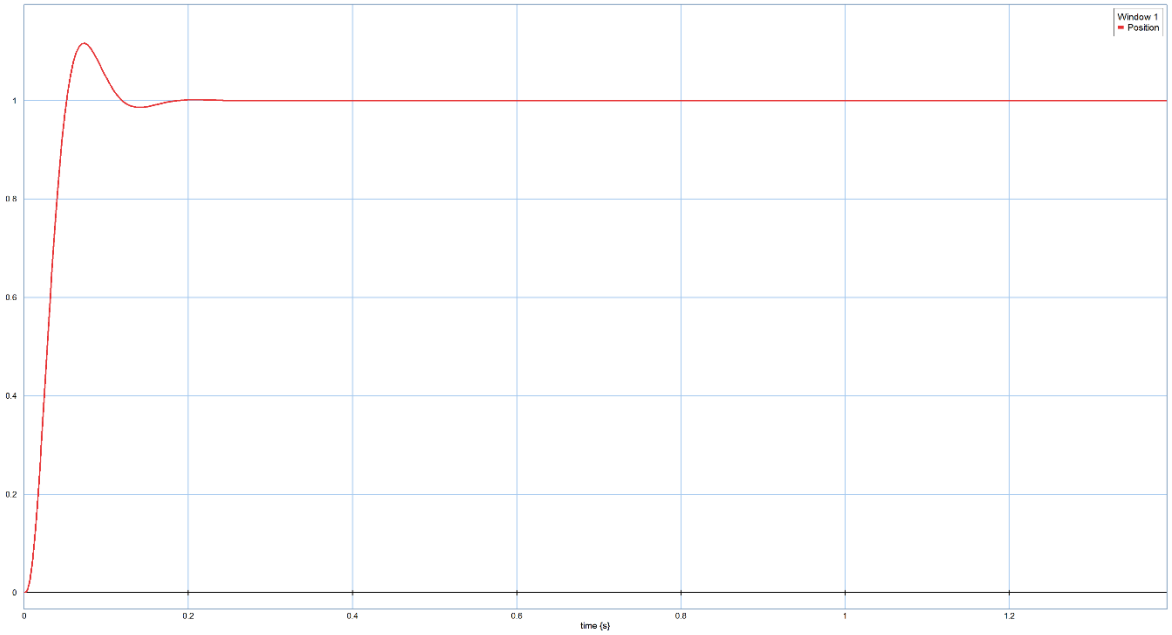


Figure 17 20Sim unit step response output

2.2.3 Plant with FSB

As mentioned before, the Robot Arm model was used in a previous assignment. We implemented Full-State Feedback Control on the same plant. The resulting 20sim model is given in Figure 18. The blocks K1, K2 and K3 connect to the internal states of the RobotArm model. Their values were calculated using an Excel sheet provided by the teachers. Given a matrix describing a system and the desired pole locations, it uses the Ackerman formulas to calculate the corresponding K values (Hamberg & Moes, 2014).
The K values could alternatively be calculated by hand, as we did during the lectures. The attenuate blocks correct for internet gains in the RobotArm model.

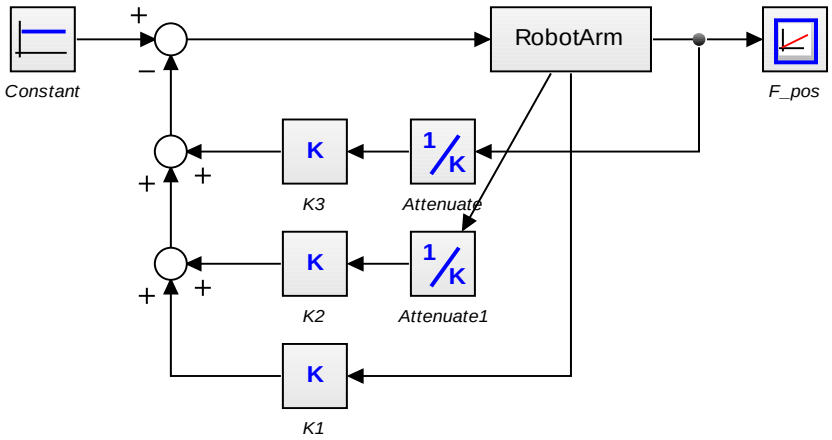


Figure 18 20Sim RobotArm model with Full-State Feedback Control

Placing all three poles at -400 results in the step response shown in Figure 19. The system reaches steady state in 5ms instead of 600ms. The K values: K1 = 32, K2 = 37,96 and K3 = 5248.

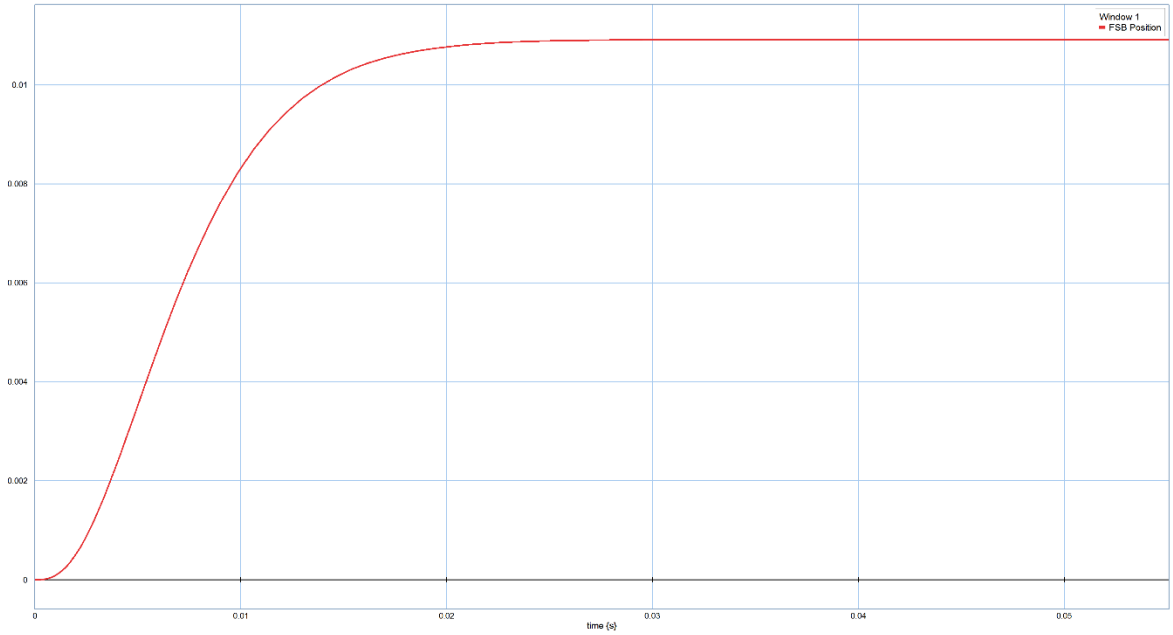


Figure 19 20Sim step response Robot Arm with Full-State Feedback control

2.2.4 FSB in C

To implement FSB in C, two possible methods we tried.

1. Take the base plant, multiple states with K values each loop.
2. Calculate new plant with FSB in 20Sim and implement that instead.

Originally, method 1 would be used, because of personal preference. By plotting the outputs of the states (both in 20Sim and in C) the variables corresponding to the states were identified.
K3 is connected to x_1, k2 to x_2 and k1 to x_3. Adding K2 and K3 to the C program worked fine. The output matched the 20sim simulation. Sadly, when adding K1 to the C program, the output went to infinity. The only reasonable explanation I can think of is because the plant is a continuous time plant, and the K values only work with discrete time plants, as we saw during the lectures.

Method 1 was dropped in favor of method 2. Linearizing the model given in Figure 18 results in equation 7. The step response of the C program is given in Figure 20, note that in the beginning the graph dips. This has to do with how Excel graphs smooth lines. The dip is not present in the output data of FSB.c.

$$H(s) = \frac{6.988 * 10^5}{s^3 + 1200s^2 + 4.8 * 10^5 s + 6.4 * 10^7} \quad (7)$$

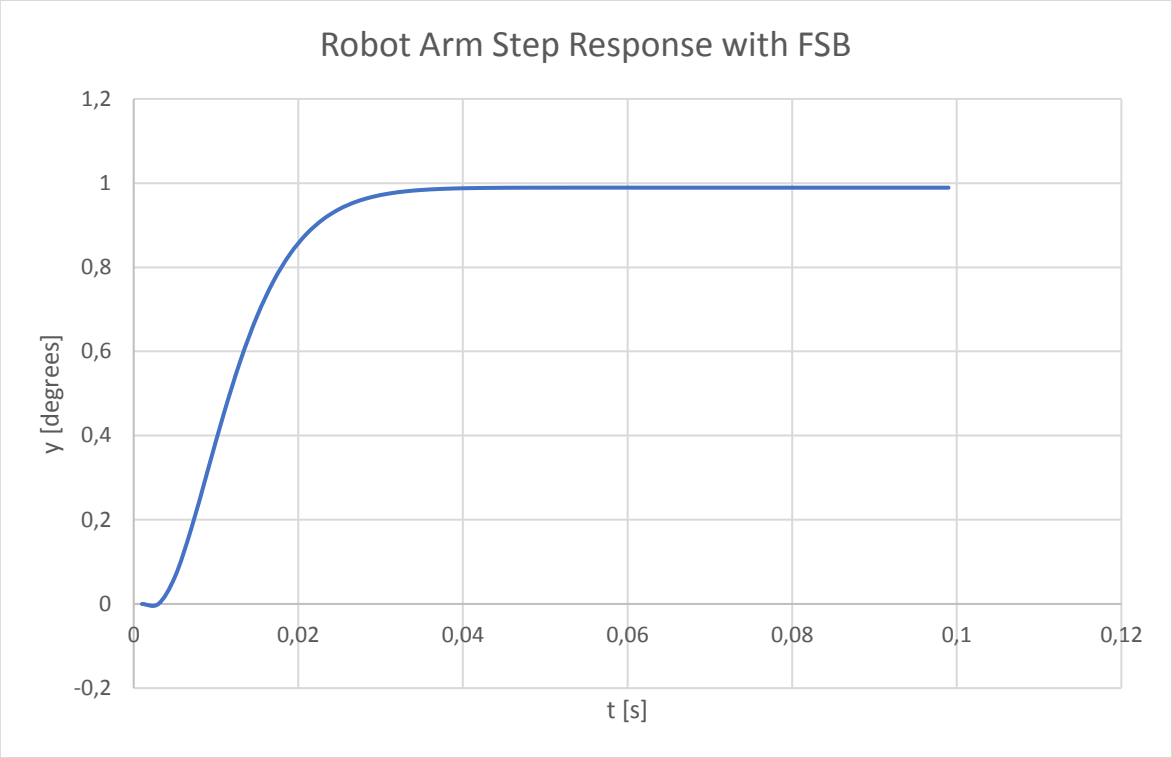


Figure 20 C code output on step response with FSB

2.2.5 FSB Conclusion

A C program was written that reads an input value, runs it through a plant and writes the output to a text file. 20Sim was used to calculate the transfer function of the robot arm plant where the poles were placed using Ackerman theory.
20Sim was also used to validate the output of the C program.

2.3 Visualization

The visualizer combines the programs of the previous sections and adds a user interface where the user can input data.

The end goal of the visualizer is summed up in Figure 21. There would be a window divided into 4 quarters, each with its own function:

- Q1 is used to get input from the user such as sample frequency, sine frequency and phase.
- Q2 plots one period of the sine wave the user desires, super imposed with the sampled version of it.
- Q3 calls the DFT program and plots its output.
- Q4 calls the FSFB program and plots its output.

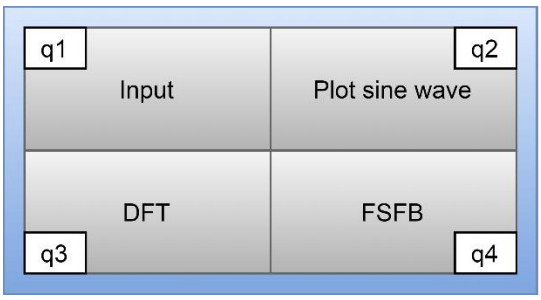


Figure 21 Template of the visualizer

This paragraph details what choices were made during development of the visualizer.

2.3.1 Choice of backend

My background is not in Graphical Design or Human Machine Interfaces. There were only two tools that I was comfortable using: Python and Processing.

Processing was used during HBO to make simple graphical user interfaces for our embedded projects. Even though programs must be written in Java, there is a build-in function to export the project to an executable file. Sadly, the graphs in Processing did not look as good as the graphs generated by Matplotlib in Python, so I initially chose Python.

Python has a module called Tkinter which can be used to create GUIs (TkInter, 2020). It was used to create the input field widow in Figure 22. The graph in Figure 22 was made using Matplotlib (Matplotlib, 2020).

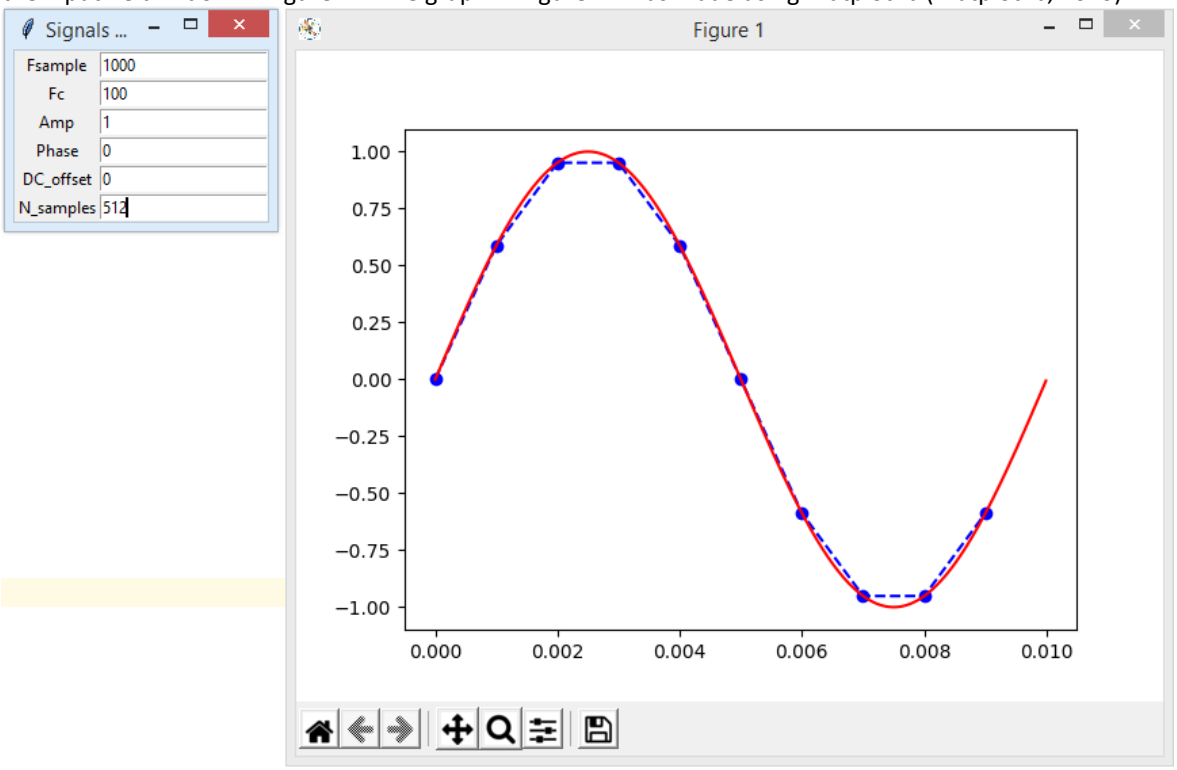


Figure 22: Python with Input and Graph windows

To convert python code to an executable, pyinstaller was used. It is run in the command line and generates its output in a specified directory (PyInstaller Manual, 2020).
The reason that Figure 22 does not include the DFT and FSFB visualizers is because the python program would need to execute the executables of the corresponding programs. The issue being that, after generating the executable of the python program, Windows Defender kicks in and flags it as a Trojan Virus, as shown in Figure 23. This happens on both Windows 8.1 and Windows 10.

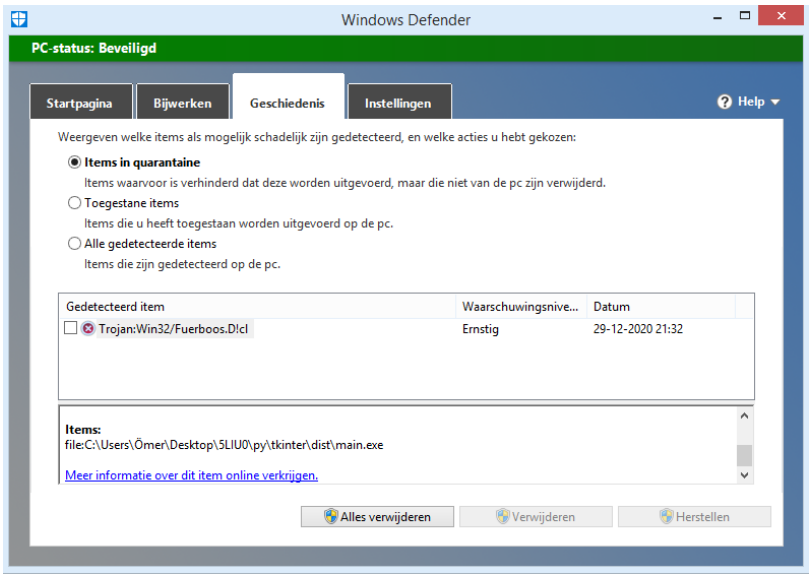


Figure 23 Windows Defender blocking the executable from running

Rewriting the programs into python would defeat the purpose as they were meant to run on embedded systems in C code.

There was an option to import the C programs as a module directly into Python. This was not done as Windows could still flag it as a virus. Python was dropped in favor of Processing. In the next paragraph the result of the Processing implementation will be discussed.

2.3.2 Final visualizer

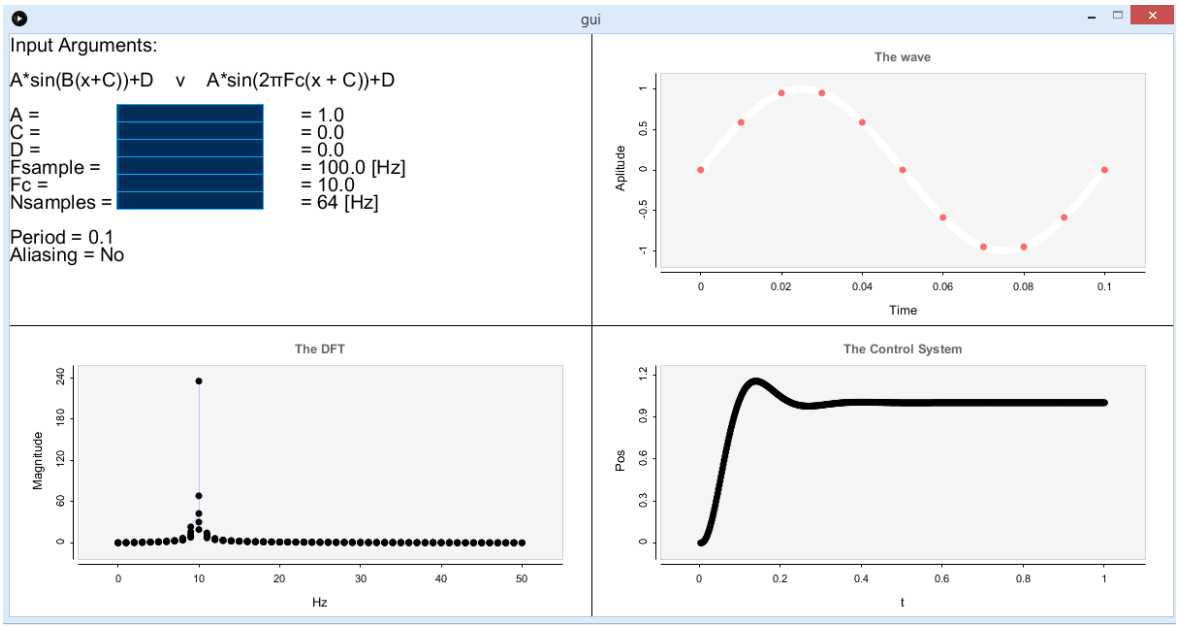


Figure 24 Final Visualizer

The template shown in Figure 21 is implemented in Processing and the result is given in Figure 24.

The blue boxes are the input fields. When the enter key is pressed in one of the blue boxes:

1. The input values are read.
2. The sine wave plot in the upper right corner is drawn.
3. The values are put in a text file and DFT.exe is called.
4. DFT.exe reads the values from the text file, performs the DFT and stores its output in a text file.
5. Processing reads those values and draws the magnitude plot of the DFT.

6. The maximal frequency is found and stored in a text file for FSFB.exe.
7. FSFB.exe is called, it generates its output into a text file.
8. Processing reads the text file and plots the values.

The code is straight forward and does not need to be discussed here.

2.3.3 Bugs

Two bugs were found in the visualizer:

1. Filling more than one blue box and pressing enter results in only the last entered field being updated.
2. The graphs in quarter 3 and 4 do not always show the latest values.

Both bugs could not be solved in time. A workaround for bug 2 was found. After putting in the desired values, send the same argument three times to any text field and the graphs will update.

3 Conclusion and reflection

3.1 Conclusion

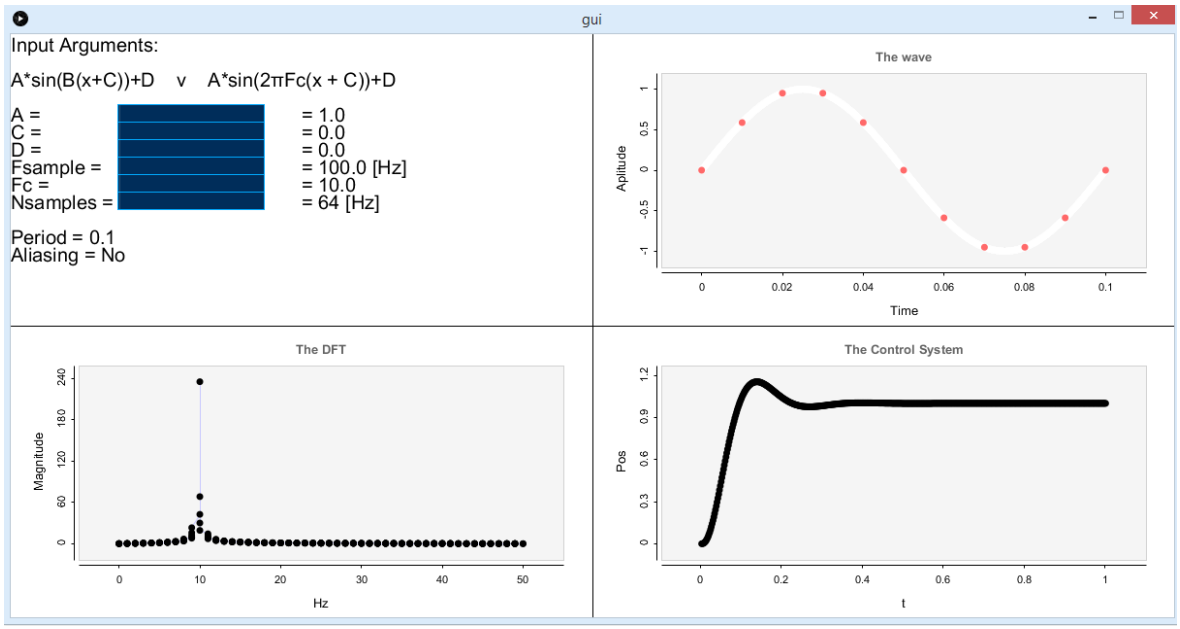


Figure 25 Final artifact

A Discrete Fourier Transform program was written in C with the help of a DFT program written in BASIC. It was compared to Excel's build-in FFT function, Matlab and an online tool. It performed identically to those. The output is stored in a file.

A C program was written that runs its input through a plant and stores its output in a file. The plant is a third order transfer function. The 20Sim software package was used to calculate the transfer function of the robot arm plant where the poles were placed using Ackerman theory. 20Sim was also used to validate the output of the C program. It performed identically to 20Sim.

Processing was used to create the visualizer. It takes user input and directs the DFT and FSFB program. The output of the programs is graphed.
Figure 25 illustrates the final artifact.

3.2 Reflection

Writing the DFT was fun. Even though I knew the basics, as I was thought those on HBO and was reminded of them during this course, it was surprising how much insight it gave me implementing this knowledge in a program. I looked up supplementary information (YouTube videos) on the theory behind the Fourier Analysis and even enjoyed solving the numerous bugs during development. As for improvements, the bin size is static and next time I would like to spend more time to make this user configurable. A lookup table instead of the sine and cosine functions would speed up the DFT program. Also, implementing a Fast Fourier Transform would be a cool next challenge.

Now, the plant in the FSFB program is static. Making it user configurable would not be hard, it would just take time. A lot of time was wasted trying to fix the K value issue, detailed in section 2.2.4, which could have been avoided if I had made the link to the Control theory lectures.

The visualizer works, but it is bodged together. For the improved version I would solve the bugs listed in section 2.3.3 and improve the interface. Or I would write a C program which would use the Win32 API to ensure everything works in Windows. This would however take considerably more time.

Bibliography

Ahmed, A. (Ed.). (2013, December 13). *FFT calculator*. Retrieved January 29, 2021, from https://scistatcalc.blogspot.com/: https://scistatcalc.blogspot.com/2013/12/fft-calculator.html

Hamberg, R., & Moes, A. (2014, December). Ackerman.xlsx. Hogeschool Utrecht Institute for Engineering & Design.

Matplotlib. (2020, November 12). Retrieved from https://matplotlib.org/: https://matplotlib.org/

Moes, A. (2015, October 3). Sysdyn_tweede_orde. Hogeschool Utrecht Institute for Engineering & Design.

PyInstaller Manual. (2020, August 9). Retrieved from https://pyinstaller.readthedocs.io: https://pyinstaller.readthedocs.io/en/stable/

Smith, S. W. (1999). *The Scientists and Engineer's Guide to Digital Signal Processing.* US: California Technical Publishing. Retrieved December 13, 2020, from http://www.dspguide.com/pdfbook.htm

Stimming, A. B., & Goswami, D. (2020, November 10). *5LIR0 / 5LIU0: Linear Systems, Signals and Control,Design-Based Learning.* Retrieved from https://canvas.tue.nl: https://canvas.tue.nl/courses/16596/files/2758530?module_item_id=224724

TkInter. (2020, June 23). Retrieved January 29, 2021, from https://wiki.python.org: https://wiki.python.org/moin/TkInter