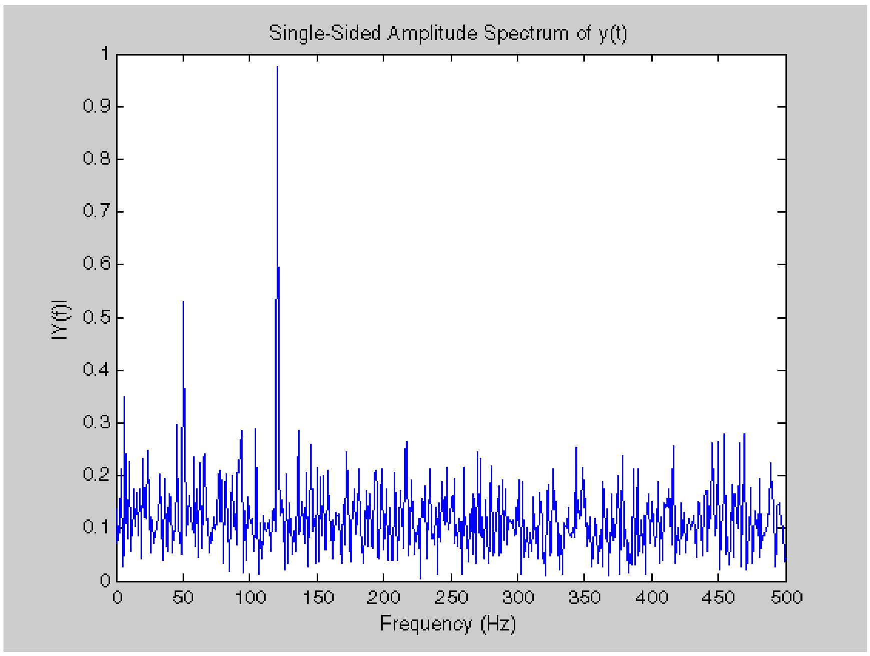
Student: Filip Andrei - 1405B

Tema: Frequency detection using the FFT



The aim of the project is to make a Raspberry Pi device to detect in real-time the fundamental frequency of the input audio file. Such a device can be used in instrument tuning, but also for other pitch-tracking applications

Steps required:

1. Read enough data to fill the FFT
2. Low-pass the data
3. Apply a window to the data
4. Transform the data using the FFT
5. Find the peak value in the transformed data
6. Compute the peak frequency from from the index of the peak value in the transformed data

For the audio data processed we will be using PortAudio as an API wrapper. Because we want to process fast we will use an 8 kHz sample rate instead of the usual 44.1 kHz one.

Some instruments have component frequencies called harmonics that are more powerful than the “fundamental” frequencies, and usually we are interested in the fundamental frequencies. Filtering, therefore, can improve the reliability of the rest of the pitch tracker significantly. Without filtering, some noise might appear to be the dominant pitch, or, more likely, the dominant pitch might appear to be a harmonic of the actual fundamental frequency.

A good choice for the filter is a low-pass filter with a center frequency around or a little above the highest pitch you expect to detect. For a guitar tuner, this might be the high E string, or about 330 Hz

Hardware required:

- Raspberry Pi board
- XMC 2 Go
- Digital display
- 2 LEDs
- 3 1k resistors
- Cables
- Microphone

Documentation:

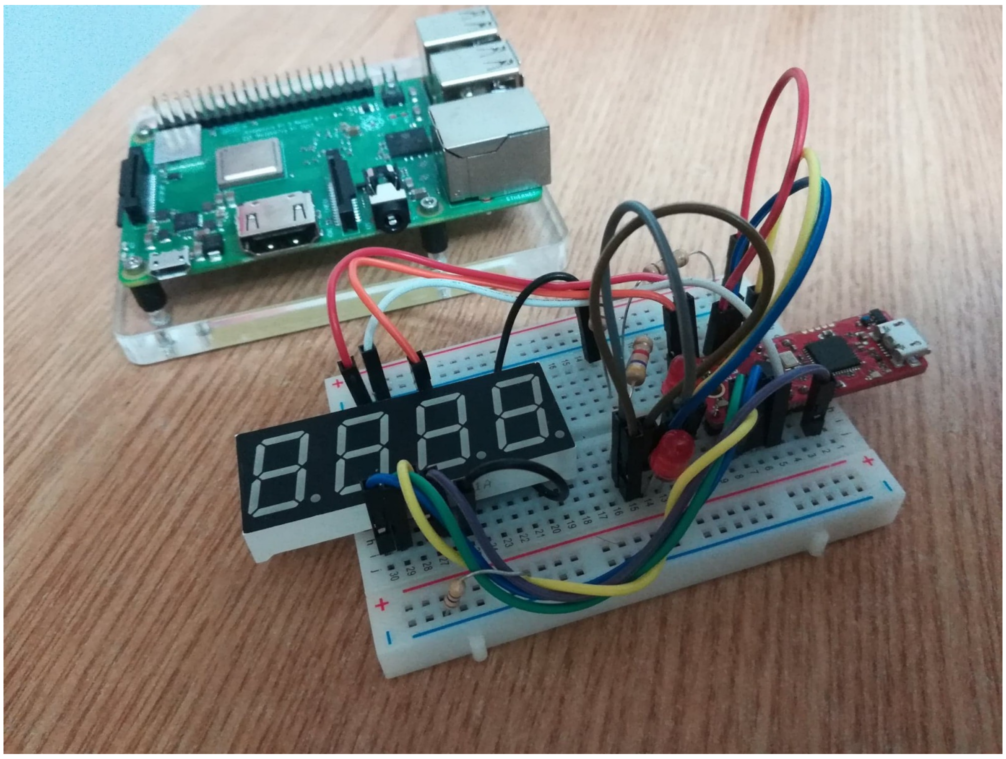
- http://blog.bjornroche.com/2012/07/frequency-detection-using-fft-aka-pitch.html?m=1&fbclid=IwAR3rexuc--5pn0r-cCJZH-Jm0nX8m1Encn98RsoTgSWJvGLyJ54oirTVwEQ

Ciorna solutii:

- Raspberry pi
- PortAudio driver
- USB microphone

The frequency detection part of the project is done by the RPi board, which we control from a PuTTY terminal through SSH. After computing the fundamental frequency, the nearest note is written in a file. A python script reads from that file and sends the read value to the XMC board through UART.

The XMC 2 Go board uses two threads. One for reading from the UART and another one to write the read value to the digital display. The two LEDs used are to check if a thread stops (if they blink, the microcontroller works as expected).



```
#include <DAVE.h>
uint8_t received_note;

void serial_read()
{
DAVE_Init();
DIGITAL_IO_Init(&DIGITAL_IO_0); // rosu - dreapta ( spre placuta )

while(1)
{
DIGITAL_IO_SetOutputHigh(&DIGITAL_IO_0);
UART_Receive(&UART_0, &received_note,1);// carcater receptionat
UART_Transmit(&UART_0, &received_note, 1);
osDelay(50);
DIGITAL_IO_SetOutputLow(&DIGITAL_IO_0);
osDelay(50);
}

DIGITAL_IO_SetOutputHigh(&DIGITAL_IO_0);
}
```

```
osThreadDef(serial_read, osPriorityNormal, 1, 0);

void digital_display()
{
BUS_IO_Init(&BUS_IO_0);
BUS_IO_Write(&BUS_IO_0, 0b0001001);

DIGITAL_IO_Init(&DIGITAL_IO_1); // rosu - stanga
DIGITAL_IO_SetOutputLow(&DIGITAL_IO_1);

while(1)
{
DIGITAL_IO_SetOutputHigh(&DIGITAL_IO_1);
switch(received_note)
{
case 'C':  BUS_IO_Write(&BUS_IO_0, 0b0111001); break;
case 'D':  BUS_IO_Write(&BUS_IO_0, 0b0111111); break;
case 'E':  BUS_IO_Write(&BUS_IO_0, 0b1111001); break;
case 'F':  BUS_IO_Write(&BUS_IO_0, 0b1110001); break;
case 'G':  BUS_IO_Write(&BUS_IO_0, 0b1111101); break;
case 'A':  BUS_IO_Write(&BUS_IO_0, 0b1110111); break;
case 'B':  BUS_IO_Write(&BUS_IO_0, 0b1111111); break;
default:
BUS_IO_Write(&BUS_IO_0, 0b0001001);
}
osDelay(50);
DIGITAL_IO_SetOutputLow(&DIGITAL_IO_1);
osDelay(50);
}
DIGITAL_IO_SetOutputHigh(&DIGITAL_IO_1);
}
osThreadDef(digital_display, osPriorityNormal, 1, 0);

int main(void)
{
 DAVE_STATUS_t status;
 received_note = 0;
 status = DAVE_Init();           /* Initialization of DAVE APPs */
osKernelInitialize();                   // initialize RTOS kernel
osThreadCreate(osThread(serial_read), NULL);
osThreadCreate(osThread(digital_display), NULL);
osKernelStart();                        // start KernelRTOS
while(1U)
{

}
}
```