**I. Capa**

**II. Agradecimentos**

Sendo este um projeto que requer muito trabalho e tempo, a sua elaboração e precisão não seria possível sem a ajuda de algumas pessoas e organizações.

Indiretamente à direção da escola, pela abertura do curso profissional e diretamente a todos os professores de todas as disciplinas, profissionais e não profissionais, ao longo dos três anos.

Aos dois formadores, Ricardo Pires, diretor do curso, proporcionador dos estágios e professor de SDAC e CD e ao Professor de IMEI, Vitor Gonçalves.

À minha mãe e ao meu pai que sempre me apoiaram neste percurso de 3 anos e na transição do Ensino Secundário em Ciências noutra escola para o Ensino Profissional numa escola num Municipio diferente.

**III. Resumo**

A prova de aptidão profissional (PAP) surge no âmbito do 3º e final ano do Curso de Técnico de Gestão de Equipamentos Informáticos. Com a ajuda de um ou mais formadores, no meu caso, Ricardo Pires, professor de SDAC e CD e diretor do curso e Vítor Gonçalves, professor de IMEI, a PAP consiste num projeto desenvolvido individualmente pelos alunos do curso em contexto das devidas aprendizagens que o mesmo fornece.

É requisitado, não só este projeto, mas também um relatório, com objetivo de contextualizar todos os passos do processo de construção do projeto. Após a realização de ambos, será realizada a defesa e apresentação do projeto, perante um júri, do devido projeto realizado, com objetivo de demonstração das competências e saberes adquiridos ao longo dos 3 anos do curso.

Sendo assim, passo a apresentar a minha proposta da Prova de Aptidão Profissional.

Tenho como principal objetivo a implementação de um programa que me permita afinar a minha guitarra. Porém, isto já é algo que existe e, certamente, com melhor software e acessibilidade, com um nível de complexidade muito mais elevado do que eu consigo desenvolver. Pessoalmente, para afinar a minha guitarra, utilizado sempre o GuitarTuna, uma aplicação com afinação em tempo real com uma acessibilidade muito boa, porém, tem um defeito.

Esta aplicação, assim como a maioria das que já testei, não têm diversidade das afinações necessárias, portanto foquei-me em criar um programa que, primeiramente, funcionasse e depois, implementar alguns “tunings” utilizados na maioria das músicas que oiço e que gosto de tocar. Decidi então implementar 4 afinações, E Standard, Eb Standard, Drop D e Drop C#.

Para a realização deste projeto foram necessárias algumas tecnologias, sendo elas o Visual Studio Code com o Python instalado. Como auxílio, para além dos dois formadores, foi utilizado do Auto-Pilot do VisualStudioCode, ferramentas de Inteligência Artificial e um projeto já existente semelhante ao que eu queria concretizar, o ‘HPS Tuner’, destacado na Webgrafia.

Para a execução do mesmo é necessário o Terminal do Sistema Operativo do computador onde o programa estará devidamente instalado.

**IV. Índice**

Teste .........................................................................................................................................

?

Anexos........ ..................................................................................................................................

?

?

Boot do Afinador.......................................................................................................................

12

Processo de Callback................................................................................................................

11

Identificação Musical...............................................................................................................

8

Variáveis e os seus Cálculos.....................................................................................................

8

Configurações do Afinador.......................................................................................................

7

Bibliotecas ...............................................................................................................................

?

Referências e Pesquisa..................................................................................................................

?

Conclusão.........................................................................................................................................

10

Menu........................................................................................................................................

6

Código.............................................................................................................................................

5

Diagrama de Gantt........................................................................................................................

4

Introdução..........................................................................................................................................................

3

Índice...............................................................................................................................................

2

Resumo............................................................................................................................................

1

Capa................................................................................................................................................

Agradecimentos..............................................................................................................................

1

**V. Introdução**

**VI. Diagrama de Gantt**

Inspirado na ideia de Karol Adamiecki, o americano Henry Gantt desenvolveu o famoso “Diagrama de Gantt” para o acompanhamento do desenvolvimento de trabalho. É uma ferramenta muito útil para a realização de projetos como a PAP. O mesmo é um gráfico usado para ilustrar o desenvolvimento e os processos do projeto que estamos a implementar.

Numa fila é descrita os diferentes processos (Captação da Ideia, Estudo da Ideia etc…) e noutra os meses do ano para que na tabela, selecionemos as células que definam assim qual tarefa foi realizada em qual mês.

Considerando isso, desenvolvi assim o meu, apresentado aqui em baixo:

****

**VII. Código**

**I. Bibliotecas**

Sendo assim, passamos então para a parte do código.

Começamos pela importação (*import* ...) das bibliotecas necessárias, neste caso sendo as mesmas:  
1. *copy* – biblioteca para a duplicação de arrays (estrutura que armazena um conjunto de elementos) e criação de cópias profundas das estruturas mais complexas do código.

2. *os* – biblioteca para possibilitar o uso de comandos de interação com o sistema, no meu caso, o ‘clear’.

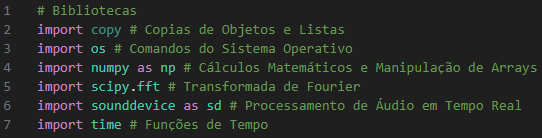
3. *numpy* – biblioteca para os cálculos matemáticos e manipulação de arrays, neste caso, de áudio.

4. *scipy* – biblioteca que traz os pacotes da Transformada e Fourier, utilizado para a conversão do domínio do tempo para o domínio da frequência.

5. *sounddevice* – biblioteca para captura e processamento de áudio em tempo real.

6. *time* – biblioteca para a execução do programa em relação ao tempo.

Importante anotar que é necessário realizar um ‘*pip install’* no terminal do VisualStudioCode para a instalação das bibliotecas *numpy*, *scipy* e *sounddevice*. Aplico nestas bibliotecas também uma nomenclatura para escrita mais facilitada no código, como na biblioteca ‘*import numpy as np*’. As três restantes bibliotecas ‘*time*’, ‘*copy*’ e ‘os’ já vêm com a extensão do Python, que tem de ser instalada também.

****

**Fig. 1 - Bibliotecas**

**II. Configurações do Afinador**

De seguida, definimos os valores de frequência para cada corda.

É um passo relativamente simples, introduzimos a variável ‘*TUNINGS*’ e atribuímos os valores da seguinte maneira:

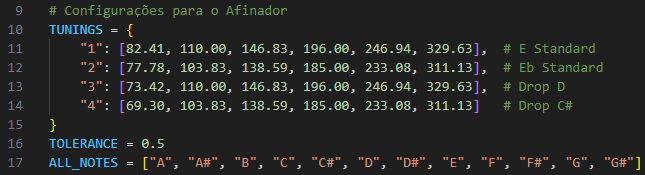
*"1": [82.41, 110.00, 146.83, 196.00, 246.94, 329.63],*

*"2": [77.78, 103.83, 138.59, 185.00, 233.08, 311.13],*

*"3": [73.42, 110.00, 146.83, 196.00, 246.94, 329.63],*

*"4": [69.30, 103.83, 138.59, 185.00, 233.08, 311.13]*

Adicionamos também uma variável de 0.5Hz de tolerância de frequência para considerar a corda afinada e colocamos as todas as notas disponíveis para layout (*ALL\_NOTES*).

****

**Fig. 2 – Tuning e Tolerância**

**III. Variáveis e Cálculos**

Asseguir, aplicamos algumas variáveis importantes e os cálculos nelas envolvidas:

**I. Variáveis**

1. *SAMPLE\_FREQ = 48000*: Define a quantidade de amostras de áudio por segundo. Neste caso 48kHz.

2. *WINDOW\_SIZE = 48000*: Define o número das amostras que serão analisadas futuramente em cada janela de tempo. Neste caso 48000 amostras.

3. *WINDOW\_STEP = 12000*: Define a análise entre janelas consecutivamente, neste caso 12000 para ser um valor inferior ao ‘*WINDOW SIZE*’ para haver sobreposição de janelas.

4. *NUM\_HPS = 5*: Define os espectros harmônicos do método HPS, com finalidade de melhorar a precisão na deteção.

5. *POWER\_THRESH**= 1^-6*: Define o valor mínimo da potência do sinal que o programa deteta. Todo o sinal abaixo deste valor é ignorado.  
6. *CONCERT\_PITCH = 440*: Define a frequência de referência como a nota ‘Lá’ (440Hz) para cálculos das restantes notas musicais.

7. *WHITE\_NOISE\_THRESH = 0.2*: Define um parâmetro usado para filtrar ruído branco do áudio recebido.

8. *HANN\_WINDOW = np.hanning(WINDOW\_SIZE)***:** Método para o processamento e suavização de ruídos.

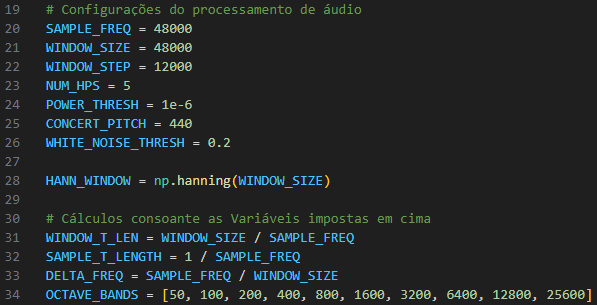
**II. Cálculos das Variáveis em questão**

1. *WINDOW\_T\_LEN = WINDOW\_SIZE / SAMPLE\_FREQ*: Considerando que ‘*WINDOW\_SIZE*’ representam o número de amostrar e ‘*SAMPLE\_FREQ*’ representa amostrar por segundo, e têm valores iguais, ´*WINDOW\_T\_LEN*´, que representa o número de análises de janela por segundo, terá um valor de 1.

2. *SAMPLE\_T\_LENGTH = 1 / SAMPLE\_FREQ*: Calcula o intervalo de tempo entre cada amostra, consecutivamente. Considerando que o ‘SAMPLE\_FREQ’ é de 48 kHz, ‘*SAMPLE\_T\_LENGTH*’ representa 0,0000208 segundos (20.8 microssegundos).

3. *DELTA\_FREQ = SAMPLE\_FREQ / WINDOW\_SIZE*: Calcula a resolução da “Transformada de Fourier”, isto é, a capacidade de distinção da frequência, ora vejamos. A conta será feita inversamente da ‘*WINDOW\_T\_LEN*’, dividindo a quantidade de amostras por segundo pelo número de amostrar por janela, o que dá 1Hz, equivalente à capacidade de distinção das frequências.

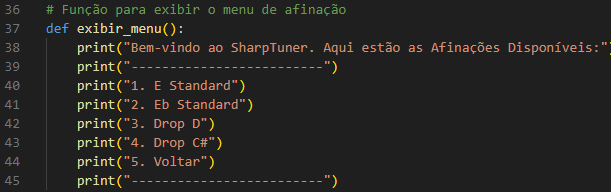
4. *OCTAVE\_BANDS* *= [50, 100, 200, 400, 800, 1600, 3200, 6400, 12800, 25600]*:Define os limites das oitavas, isto é, as faixas de frequências que contêm as notas mais importantes.

****

**Fig. 3 – Variáveis e Cálculos**

**IV. Menu de Layout**

Vamos agora desenvolver o menu de layout para o utilizador do programa. Implementamos uma função “*def exibir\_menu():*” e dentro da mesma implementar o menu desta forma:

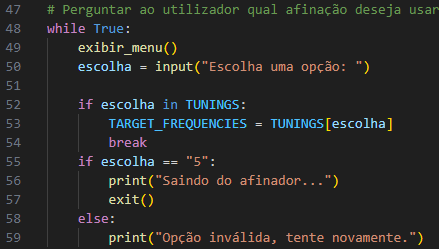


**Fig.4 – Escrita do Menu**

O “*print*” é utilizado para a escrita do layout, logo, o que estiver dentro dos parêntesis será escrita ao executar o programa. Depois, consoante o que o utilizador escolher, será o que a próxima página apresentará.

Para o utilizador selecionar o que pretende, implementamos o que fará com que esse processo aconteça.

Criamos então este processo:



**Fig.5 – Implementação das Escolhas**

Por linguaguem corrente o que isto significa é:

Aplica o loop infinito (*while true*) que exibe o menu (*exibir\_menu()*). Dentro deste loop, é proposta uma escolha ao utilizador, para inserir o a opção que pretende (*escolha = input("Escolha uma opção: ")*).

Para direcionar o utilizador para a sua escolha é realizado um ‘*if*’:

Se a escolha do utilizador estiver dentro dos ‘*TARGET\_FREQUENCIES*’, o programa redireciona essa mesma escolha para os ‘*TUNINGS*’. Ou seja, se escolher a opção 1 (*"1. E Standard"*), o mesmo vai buscar os conteúdos dos ‘*TUNINGS*’, neste caso:

(*"1": [82.41, 110.00, 146.83, 196.00, 246.94, 329.63]*)

Sendo assim, as opções 1 a 4, serão chamadas as devidas afinações.

Quando acaba esta etapa, esta parte para e passa para a parte de afinação, que já vamos abordar mais à frente.

Porém ainda não acabou esta fase, temos de implementar a opção para o utilizador para se desejar, sair do programa:

Se a escolha for “*5*”, é escrito no ecrã “*Saindo do afinador…*” e sai do mesmo (*exit()*)

Finalmente, implementamos um ‘*else*’ para, caso as opções que o utilizador selecione sejam qualquer tipo de número ou caracter, para além de 1, 2, 3, 4 ou 5, apresentar no ecrã, “*Opção inválida, tente novamente .*”

**V. Identificação Musical**

Após a escrita do menu, passamos realizamos a função para a identificação da nota músical.

Implemento a função “*def find\_closest\_note(pitch)*:”, dentro da mesma estão:

***I. “i = int(np.round(np.log2(pitch / CONCERT\_PITCH) \* 12))”:***

1.1 Converte a frequência (*pitch*) para semitons relativos à nota da referência (*CONCERT\_PITCH*)

1.2 Calcula a distância (*np.log2*), em semitons entre a frequência (*pitch*) e a nota de referência (*CONCERT\_PITCH*)

1.3 Arredonda esse valor para o semitom mais próximo (*np.round*)

1.4 Converte esse mesmo semitom para um número inteiro (*int*) para, depois, ser exibido na tela consoante a nota tocada.

***II. “closest\_note = ALL\_NOTES[i % 12] + str(4 + (i + 9) // 12)”***

2.1 Obtém o nome da nota musical correspondente (*ALL\_NOTES[i % 12]*)

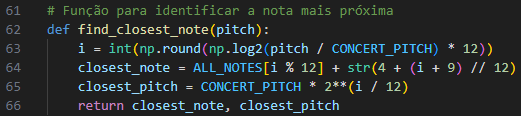
2.2 Calcula a oitava da nota musical recebida (*(i + 9) // 12*)

***III. “closest\_pitch = CONCERT\_PITCH \* 2\*\*(i / 12)”***

3.1 Calcula a frequência exata (*closest\_pitch*) da nota musical tocada calculando a nota de referência (*CONCERT\_PITCH*) e a devida fórmula (*2\*\*(i / 12*)\*\*)

***IV. “return closest\_note, closest\_pitch”***

4.1 Depois deste processo todo, devolve a nota musical recebida e a sua frequência real já processada.



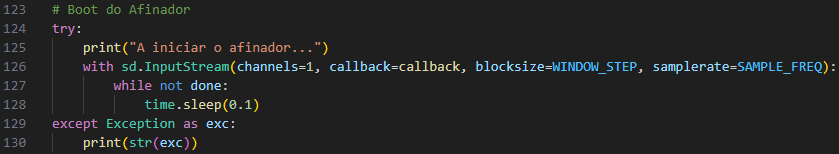
**Fig.6 – Identificação da nota tocada**

**VI.Processo de Callback**

Passamos agora para o que faz realmente o programa funcionar. Agora, como todas as funções, variáveis e os seus cálculos, criamos a função que realiza o processo de callback, o mais importante num programa em tempo real.

**VII. Boot do Afinador**

Chegamos então ao capítulo final, depois de abordarmos tudo o que está “dentro do programa”, passamos ao passo final e abordamos o que o faz arrancar.

****Implementamos este bloco:

**Fig.1- Boot do Programa**

O que está dentro deste bloco é o seguinte:

O ‘*try*’ tenta executar o código dentro dele e executá-lo. Ao executá-lo (ENTER depois da seleção da opção de 1 a 5) escreve no ecrã “A iniciar o afinador…”.

Em “*with sd.InputStream(channels=1, callback=callback, blocksize=WINDOW\_STEP, samplerate=SAMPLE\_FREQ):*” o ‘sd.InputStream’ começa a capturer o áudio em tempo real no primeiro canal disponível (channels=1). É aplicado a função (callback=callback) para a atualização constante de novas frequências a serem tocadas na guitarra. Por final, é definido o tamanho de blocos de áudio processados (*blocksize=WINDOW\_STEP*) e a taxa de amostragem de áudio (*samplerate=SAMPLE\_FREQ*).

Enquanto não estiver afinado (*while not done:*) o programa espera 0,1 segundos até verificar a frequência novamente (*time.sleep(0.1)*).

Depois é aplicado uma espécie de tratamento de erros. Caso o programa ‘crashe’, o erro é exibido no ecrã em vez de encerrar o programa. Isto é possível devido ao “*except Exception as exc:*”, que escreve no ecrã o devido ecrã com o *print(str(exc))*.

**VIII. Teste**

**VIII. Conclusão**

Não esperava ser este o resultado final da PAP. Comecei com várias ideias complexas e algumas até impossíveis. Eventualmente depois de algumas semanas de investigação encontrei um projeto em Python no GitHub então dei download e adaptei-o para o que queria implementar.

Com a Inteligência Artificial e o AutoPilot do Python não foi muito complexo realizar o código, porém evitei copiar e colar o que pedia à IA e fui pedindo ajuda à mesma consoante o que queria implementar e alguns exemplos de exercícios em Python que já tinha feito em aula.

Considerando que a área musical é o que mais me desperta interesse, depois de entender realmente o que queria fazer, foi árduo, porém de fácil realizar o relatório e o código. Espero que no futuro consiga vincular as duas áreas profissionalmente, a que mais me desperta interesse a nível académico e a que mais me dá gosto pessoalmente.

O Balanço Final é positivo e sinto que, se quiser, consigo realizar algo mais complexo e útil não só para mim, mas também para terceiros, implementando uma aplicação, uma interface gráfica por exemplo e afinador para outros instrumentos de cordas e sopro.

**IX. Referências e Pesquisa**

<https://www.w3schools.com/python/>

<https://chatgpt.com/>

<https://github.com/not-chciken/guitar_tuner>

**X. Anexos**

# Bibliotecas

import copy # Copias de Objetos e Listas

import os # Comandos do Sistema Operativo

import numpy as np # Cálculos Matemáticos e Manipulação de Arrays

import scipy.fft # Transformada de Fourier

import sounddevice as sd # Processamento de Áudio em Tempo Real

import time # Funções de Tempo

# Configurações para o Afinador

TUNINGS = {

    "1": [82.41, 110.00, 146.83, 196.00, 246.94, 329.63],  # E Standard

    "2": [77.78, 103.83, 138.59, 185.00, 233.08, 311.13],  # Eb Standard

    "3": [73.42, 110.00, 146.83, 196.00, 246.94, 329.63],  # Drop D

    "4": [69.30, 103.83, 138.59, 185.00, 233.08, 311.13]   # Drop C#

}

TOLERANCE = 0.5

ALL\_NOTES = ["A", "A#", "B", "C", "C#", "D", "D#", "E", "F", "F#", "G", "G#"]

# Configurações do processamento de áudio

SAMPLE\_FREQ = 48000

WINDOW\_SIZE = 48000

WINDOW\_STEP = 12000

NUM\_HPS = 5

POWER\_THRESH = 1e-6

CONCERT\_PITCH = 440

WHITE\_NOISE\_THRESH = 0.2

HANN\_WINDOW = np.hanning(WINDOW\_SIZE)

# Cálculos consoante as Variáveis impostas em cima

WINDOW\_T\_LEN = WINDOW\_SIZE / SAMPLE\_FREQ

SAMPLE\_T\_LENGTH = 1 / SAMPLE\_FREQ

DELTA\_FREQ = SAMPLE\_FREQ / WINDOW\_SIZE

OCTAVE\_BANDS = [50, 100, 200, 400, 800, 1600, 3200, 6400, 12800, 25600]

# Função para exibir o menu de afinação

def exibir\_menu():

    print("Bem-vindo ao SharpTuner. Aqui estão as Afinações Disponíveis:")

    print("-------------------------")

    print("1. E Standard")

    print("2. Eb Standard")

    print("3. Drop D")

    print("4. Drop C#")

    print("5. Voltar")

    print("-------------------------")

# Perguntar ao utilizador qual afinação deseja usar

while True:

    exibir\_menu()

    escolha = input("Escolha uma opção: ")

    if escolha in TUNINGS:

        TARGET\_FREQUENCIES = TUNINGS[escolha]

        break

    if escolha == "5":

        print("Saindo do afinador...")

        exit()

    else:

        print("Opção inválida, tente novamente.")

# Função para identificar a nota mais próxima

def find\_closest\_note(pitch):

    i = int(np.round(np.log2(pitch / CONCERT\_PITCH) \* 12))

    closest\_note = ALL\_NOTES[i % 12] + str(4 + (i + 9) // 12)

    closest\_pitch = CONCERT\_PITCH \* 2\*\*(i / 12)

    return closest\_note, closest\_pitch

# Callback para processar o áudio em tempo real

current\_string = 0

done = False

def callback(indata, frames, time, status):

    global current\_string, done, TARGET\_FREQUENCIES

    static\_text = f"Toca a {current\_string + 1}ª corda ({TARGET\_FREQUENCIES[current\_string]} Hz)."

    if status:

        if "overflow" in str(status):

            return

        print(status)

    if any(indata):

        if not hasattr(callback, "window\_samples"):

            callback.window\_samples = [0 for \_ in range(WINDOW\_SIZE)]

        if not hasattr(callback, "last\_displayed\_note"):

            callback.last\_displayed\_note = None

        callback.window\_samples = np.concatenate((callback.window\_samples, indata[:, 0]))

        callback.window\_samples = callback.window\_samples[len(indata[:, 0]):]

        signal\_power = (np.linalg.norm(callback.window\_samples, ord=2, axis=0)\*\*2) / len(callback.window\_samples)

        if signal\_power < POWER\_THRESH:

            os.system('cls' if os.name == 'nt' else 'clear')

            print(static\_text)

            print("Nota mais próxima: ...")

            return

        hann\_samples = callback.window\_samples \* HANN\_WINDOW

        magnitude\_spec = abs(scipy.fft.fft(hann\_samples)[:len(hann\_samples)//2])

        for i in range(int(62 / DELTA\_FREQ)):

            magnitude\_spec[i] = 0

        max\_ind = np.argmax(magnitude\_spec)

        max\_freq = max\_ind \* (SAMPLE\_FREQ / WINDOW\_SIZE)

        closest\_note, closest\_pitch = find\_closest\_note(max\_freq)

        os.system('cls' if os.name == 'nt' else 'clear')

        print(static\_text)

        print(f"Nota mais próxima: {closest\_note} ({max\_freq} Hz).")

        if abs(max\_freq - TARGET\_FREQUENCIES[current\_string]) <= TOLERANCE:

            escolha = input("Corda afinada! (Enter para avançar, 'b' para voltar): ").strip().lower()

            if escolha == "b":

                current\_string = max(0, current\_string - 1)

            else:

                current\_string += 1

                if current\_string >= len(TARGET\_FREQUENCIES):

                    print("Parabéns! Todas as cordas estão afinadas!")

                    done = True

# Boot do Afinador

try:

    print("A iniciar o afinador...")

    with sd.InputStream(channels=1, callback=callback, blocksize=WINDOW\_STEP, samplerate=SAMPLE\_FREQ):

        while not done:

            time.sleep(0.1)

except Exception as exc:

    print(str(exc))