

Visão Computacional com OpenCV e MediaPipe

Vamos Conversar e Codar

Passo 1

```
1  #importação do opencv-python
2  import cv2
3  import mediapipe as mp # importação do mediapipe para usar o facemesh
```

Passo 2

```
4  #criar uma variável para camera
5  cap = cv2.VideoCapture(0)
6  # usando uma solução de desenho
7  mp_drawing = mp.solutions.drawing_utils
8  # usando uma solução para o Face Mesh Detection
9  mp_face_mesh = mp.solutions.face_mesh
```

Passo 3

```
10 #liberação automática
11 with mp_face_mesh.FaceMesh(min_detection_confidence=0.5, min_tracking_confidence=0.5) as facemesh:
12     # enquanto a camera estiver aberta
13     while cap.isOpened():
14         # sucesso-booleana (verificar se o frame esta vazio)
15         # frame - captura
16         sucesso, frame = cap.read()
17         # realizar a verificação
18         # sucesso = 1    fracasso = 0
19         if not sucesso:
20             print("ignorando o frame vazio da câmera")
21             continue
```

Passo 4

```
# transformando de BGR para RGB
frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
# criar uma variável      dados processados (ex.pontos do rosto)
saida_facemesh = facemesh.process(frame)
# O OpenCV - entende BGR
frame = cv2.cvtColor(frame, cv2.COLOR_RGB2BGR)
```

Passo 5

```
# vamos desenhar?
# 1 - Fizemos a detecção do rosto com facemesh.process(frame)
# 2 - Agora temos que mostrar essa detecção
# 3 - Vamos usar o for que é especie de while compacto
# 4 - Vamos usar multi_face_landmarks : x,y,z de cada ponto que MediaPipe encontrar no rosto
for face_landmarks in saida_facemesh.multi_face_landmarks:
    # desenhando
    # 1 - frame : representa o frame de vídeo
    # 2 - face_landmarks: os landmarks detectados - pontos específicos
    # 3 - FACEMESH_CONTOURS - é uma constante que representa os contornos da face na malha facial.
    mp_drawing.draw_landmarks(frame,face_landmarks,mp_face_mesh.FACEMESH_CONTOURS)
```

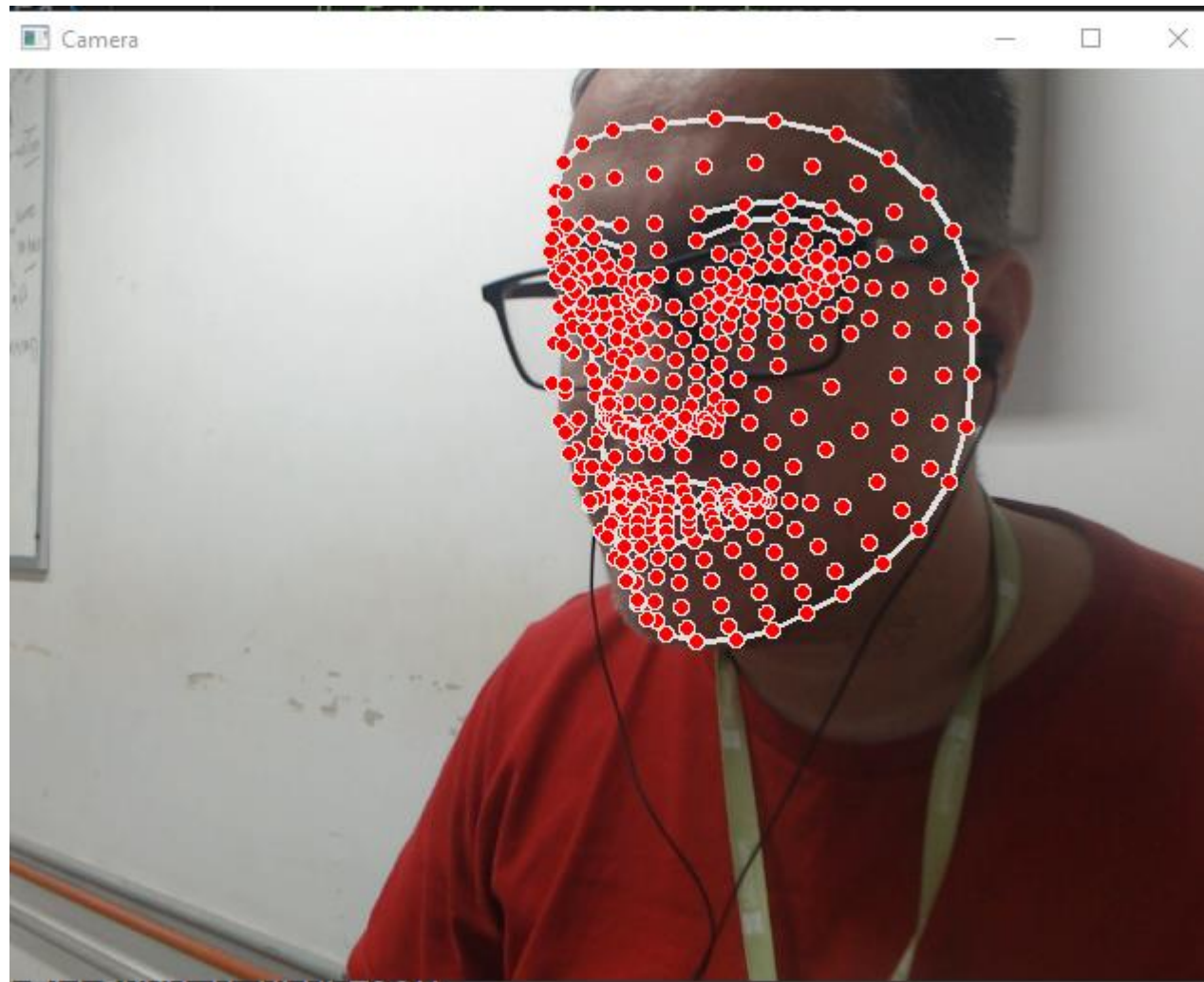
Passo 6

```
try: ←
    #mostrar os pontos, mostrar a detecção que o MediaPipe fez
    # vou criar uma variável face_landmarks - que são as coordenadas da nossa face
    # Ele vai ser atribuído ao nosso conjunto de coordenadas
    # saida_facemesh é o nosso conjunto de coordenadas
    # o multi_face_landmarks vai retornar as coordenadas
    # após isso ele deve desenhar esses pontos no nosso rosto
    for face_landmarks in saida_facemesh.multi_face_landmarks:
        # desenhar
        # uso o método draw_landmarks para pontuar os desenhos
        # dentro dos parenteses
        # o nosso (frame)
        # as coordenadas - (face_landmarks)
        # Especificar os nossos pontos : FACEMESH_CONTOURS
        mp_drawing.draw_landmarks(frame, face_landmarks, mp_face_mesh.FACEMESH_CONTOURS)
except: ←
    print("algo deu errado")
finally: ←
    print("encerrando o processo")
```

Passo 7

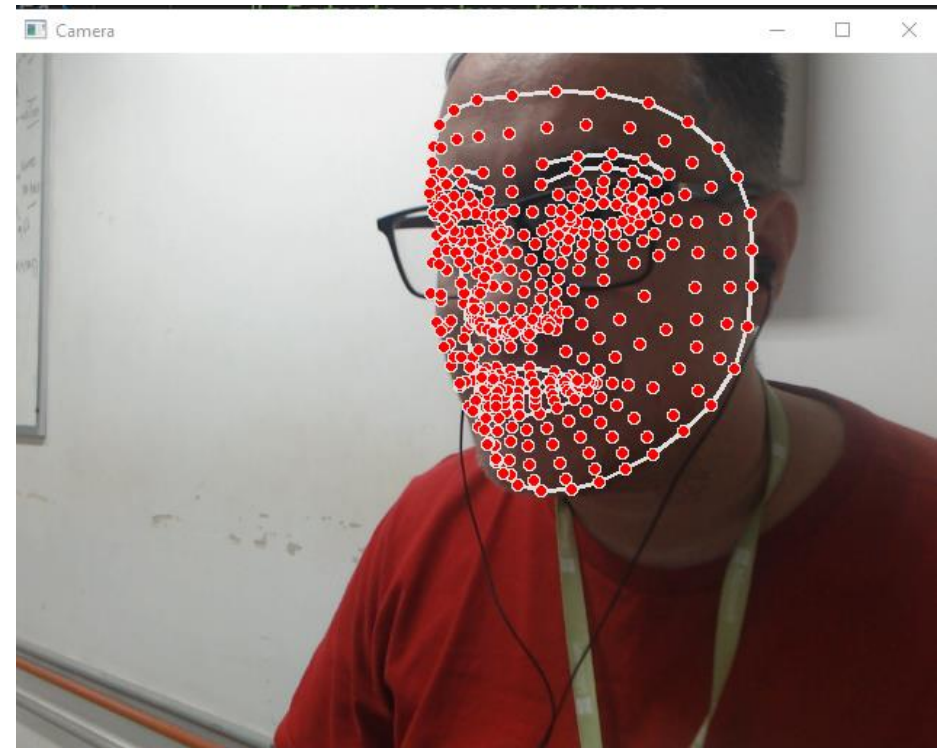
```
58 # Libera o recurso de captura de vídeo
59 cap.release()
60 # Esse método fecha todas as janelas abertas pelo OpenCV.
61 cv2.destroyAllWindows()
```


Resultado final



Passo 8

- Ao rodar novamente nosso código de visualização do MediaPipe.
- Mais uma vez, uma janela de vídeo foi aberta com o meu rosto detectado.
- É possível observar uma série de pontos.
- Eles são vermelhos
- Envolta, há uma linha/moldura branca.
- Isso dificulta a visualização.



Passo 9

- Em algumas áreas, existe uma maior concentração de pontos.
- Por exemplo, na região dos olhos, no nariz e na boca.
- Nestas áreas, quase não dá para enxergá-los.
- Você concorda que essa visualização não é tão interessante para uma análise mais profunda dos pontos, isto é, um entendimento melhor sobre o que o MediaPipe está coletando?
- Precisamos melhorar essa visualização para alcançarmos uma análise mais assertiva sobre o que está acontecendo com os pontos na nossa face.
- Então, vamos **alterar os pontos e como são mostrados**.
- Podemos apertar "c" para fechar a janela.
- Agora, vamos começar as melhorias na visualização dos pontos.
- Nosso primeiro passo é retornar à linha onde criamos o desenho das nossas coordenadas.

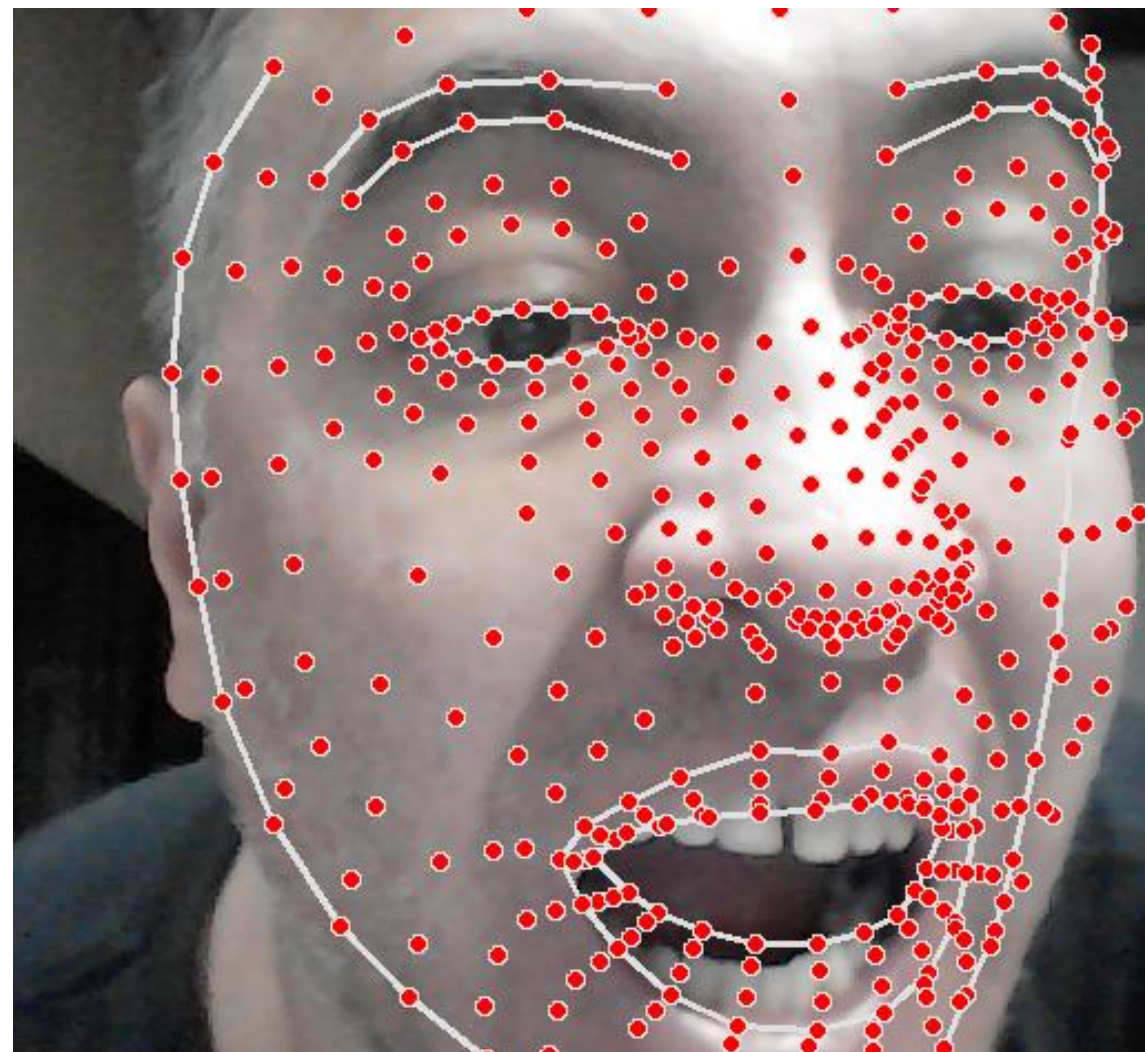
Passo 10

- Nosso primeiro passo é retornar à linha onde criamos o desenho das nossas coordenadas.

```
52  
53     → mp_drawing.draw_landmarks(frame,  
54                                     face_landmar
```

Passo 11

- Existe uma opção no MediaPipe
- Que nos permite especificar a **cor**.
- A **grossura**
 - do tamanho do círculo
- De cada ponto do nosso rosto.
- Bem como a linha branca
- Que está em volta da nossa face
- Vamos rodar o código de novo
- E observá-la com mais atenção.



Passo 12

- A linha em branco conecta todos os pontos da nossa face e forma um círculo em volta do rosto.
- Além dela, existe uma linha branca que contorna as sobrancelhas
- E outra que contorna os olhos
- Mas não conseguimos ver, porque estão cobertas pelos pontos vermelhos.

Passo 13

- Nossa primeira alteração será nos pontos em vermelho.
 - Podemos deixá-los menores e preenchidos com outra cor.
 - Para isso, usaremos o parâmetro `landmark_drawing_spec`, que será adicionado ao parâmetro `drawn_landmarks()`.
- Ao final dos parâmetros que já estão no método, passaremos vírgula e apertaremos "Enter".
- Também apertaremos "TAB" para que o comando não fique na mesma indentação.
- Finalmente, adicionaremos o `landmark_drawing_spec`

Passo 14

```
mp_face_mesh.FACEMESH_CONTOURS,  
Landmark_drawing_spec = mp_drawing
```


Passo 15

- Esse é o primeiro parâmetro com o qual trabalharemos para alterar a visualização dos pontos no nosso rosto.
- Após isso, vamos chamar novamente o objeto mp_drawing e, em seguida, usaremos o método DrawingSpec().
- Nos parênteses, podemos especificar a cor de cada ponto.
- Para isso, adicionaremos o parâmetro de cor: color.

```
mp_drawing.DrawingSpec(color=(255,102,102))
```

Passo 16

- Depois de color, vamos usar o sinal de igual, abrir parênteses e especificar as cores em BGR, seguindo os tons de Blue (azul), Green (verde) e Red (vermelho).
- Para os pontos da face, talvez seja interessante buscar por uma tonalidade mais azulada.
- Vamos usar a seguinte estrutura de cor: 255 para o azul, 102 para verde e 102 para o vermelho.

```
mp_drawing.DrawingSpec(color=(255,102,102).
```

Passo 17

- Fora dos parênteses de cores, vamos especificar o segundo parâmetro, referente ao tamanho do ponto, sua grossura.
- Por isso, alteraremos o parâmetro de thickness (grossura).
- Vamos especificá-lo com valor 1, porque queremos pontos bem pequenos: thickness=1.

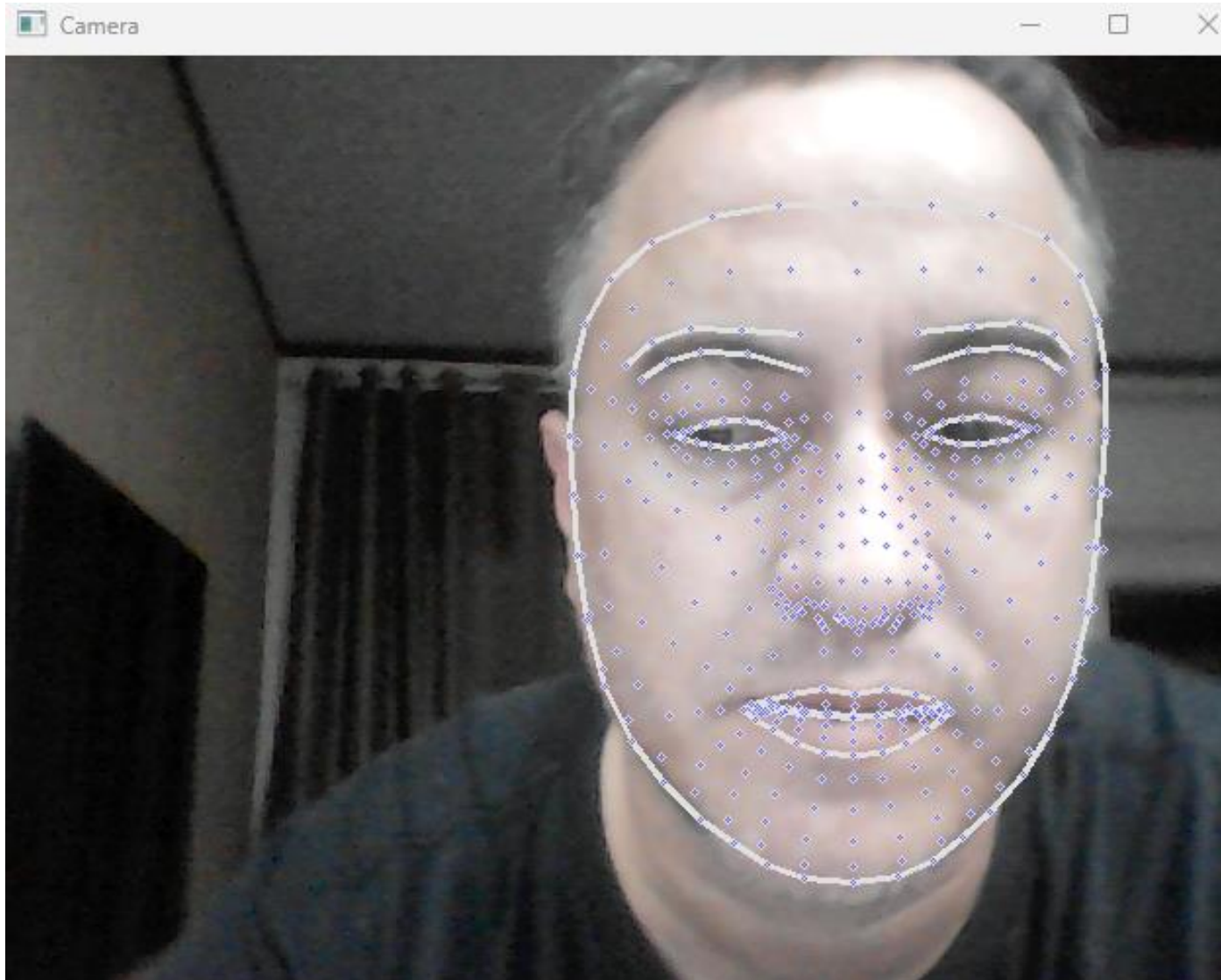
```
thickness=1,
```

Passo 18

- O terceiro e último parâmetro é o raio da circunferência dos pontos. Para isso, usaremos o parâmetro `circle_radius=1`.

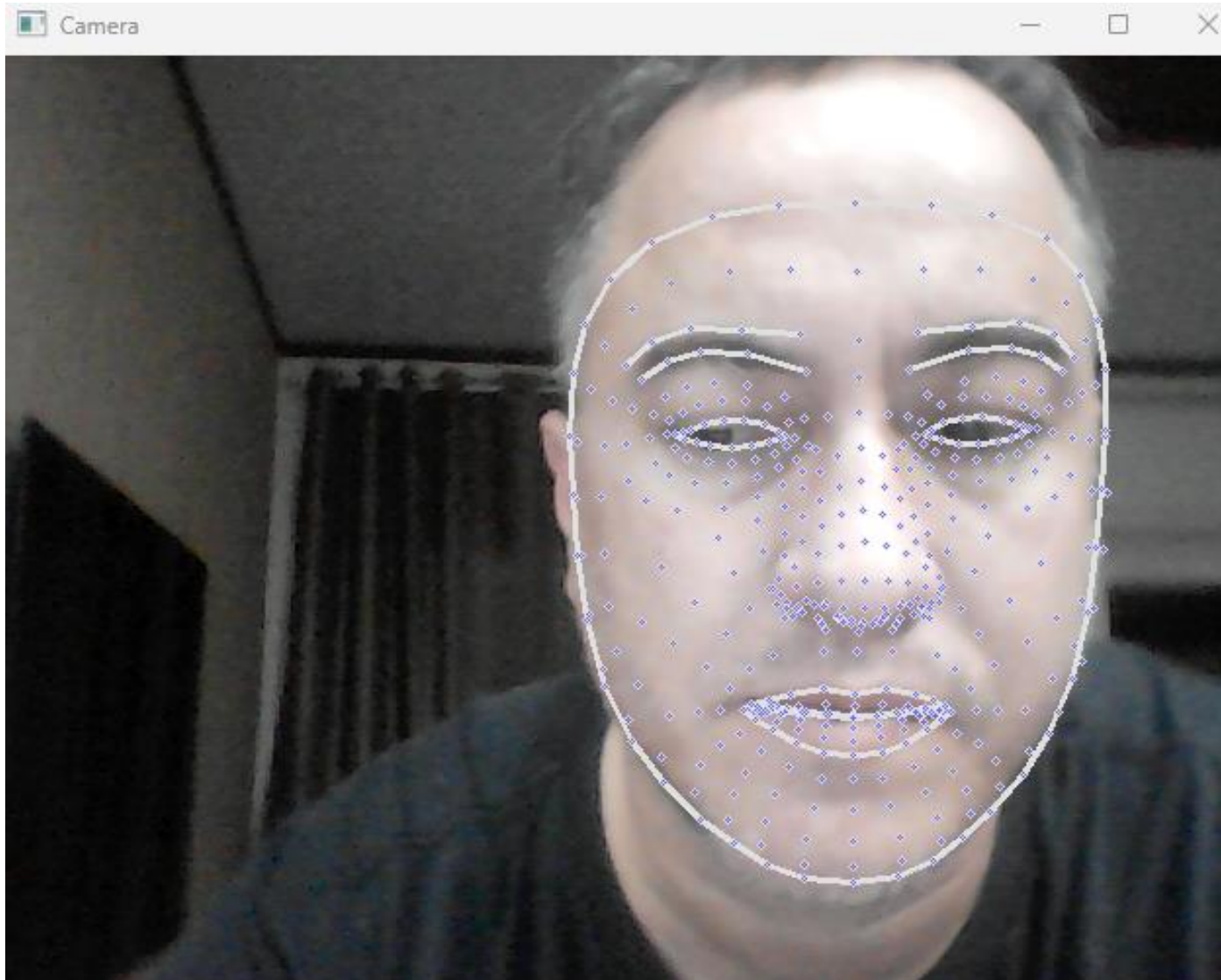
```
c(color=(255,102,102),thickness=1,circle_radius=1),
```

Resultado final



- Abrimos a janela de vídeo.
- Já é possível ver as linhas de contorno e perceber que os pontos estão bem pequenos e brancos.
- Isso permite uma melhor visualização do rosto, sem um acúmulo de pontos.

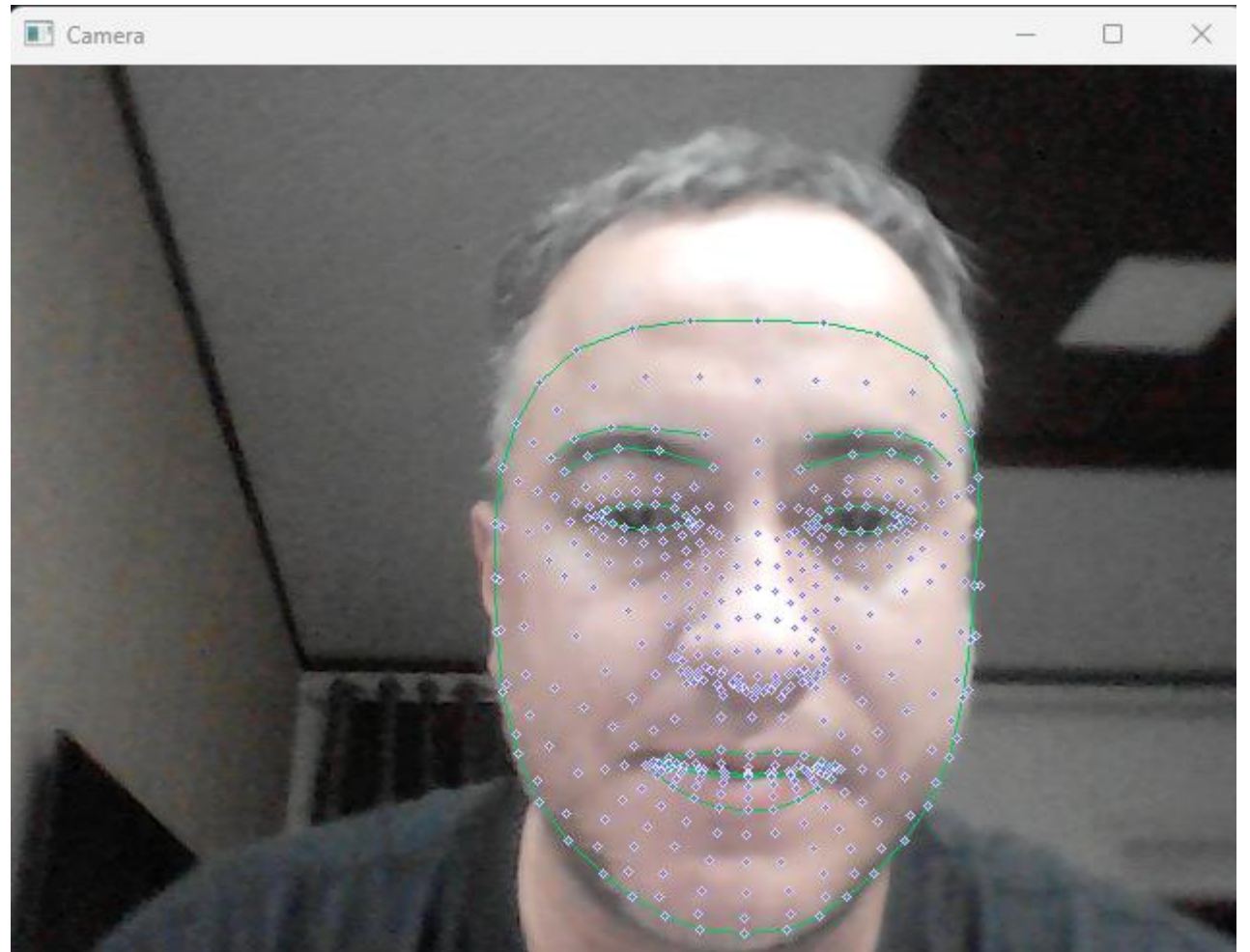
Resultado final



- A grossura das linhas de conexão ainda incomoda e a cor branca também.
- Podemos alterar os parâmetros de conexão.
- Vamos apertar "c", fechar o vídeo e realizar a alteração.
- Para isso, usaremos outro parâmetro, bastante similar ao `landmark_drawing_spec`, o `connection_drawing_spec`

Passo 19

- connection_drawing_spec

[illegible]

Passo 20

- vamos alterar os valores do parâmetro das cores, porque não queremos que as linhas sejam azuis, mas, sim, uma tonalidade mais esverdeada, por isso, precisamos diminuir o valor da cor azul para 102 e deixar o valor de verde mais forte, com 204. Por fim, vamos zerar a cor vermelha.

Passo 21

- A grossura da linha continuará 1 e o raio da circunferência, tanto faz o valor, porque não há circunferência na linha.
- Vamos apertar "Shift + Enter" mais uma vez.
- Nossa visualização está muito melhor.
- As conexões aparecem de forma mais simples, bem como os pontos. Isso nos permitirá realizar uma análise melhor dos pontos.
- Entendemos como fazer a conexão do MediaPipe, a amostragem dos pontos e a captura em tempo real.
- Falta construir um detector de sono. Te convido para descobrir isso na próxima aula!

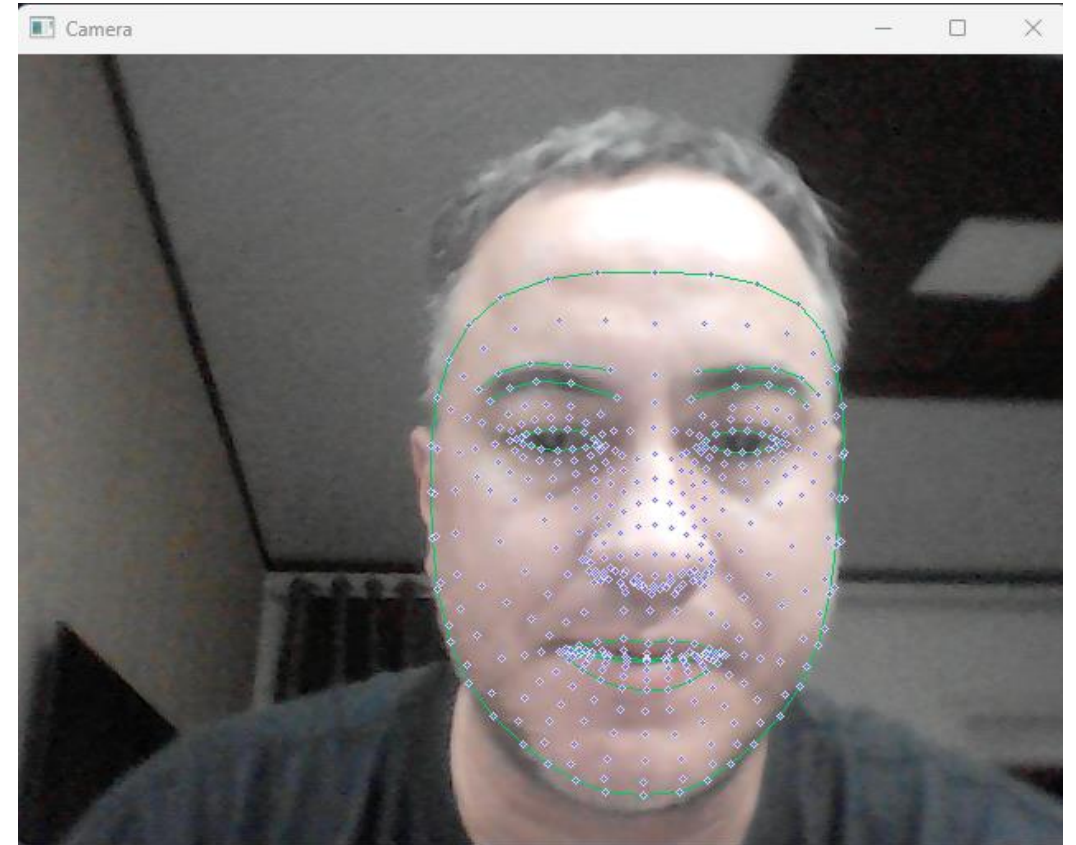
Nessa aula, você aprendeu a:

- Estruturar o ambiente para um projeto de visão computacional;
- Compreender como podemos fazer uma captura em tempo real com OpenCV;
- Processar um frame para detecção de face com MediaPipe Face Mesh; e Construir e melhorar uma visualização de pontos faciais

Mapeando os olhos

Passo 22 - Coordenadas

- Através da detecção do *MediaPipe*,
- conseguimos desenhar diversos
- pontos do nosso rosto que são lidos
- como coordenadas representadas
- por pequenos círculos desenhados
- no nosso frame.
- Mas como o MediaPipe tem
- acesso a elas?



Passo 23

Como o MediaPipe tem acesso às coordenadas?

- A gente processou o nosso frame

```
    continue
    frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
    saida_facemesh = facemesh.process(frame)
    frame = cv2.cvtColor(frame, cv2.COLOR_RGB2BGR)
```

- No momento que processamos nosso frame as coordenadas foram geradas.
- O trabalho da inteligência artificial do MediaPipe gerou essas coordenadas

Passo 23

Como o MediaPipe tem acesso às coordenadas?

- Para ter acesso as coordenadas, com o objeto da nossa saída chamamos o multi_face_landmarks

```
try:
    for face_landmarks in saida_facemesh.multi_face_landmarks:
        mp_drawing.draw_landmarks(frame, face_landmarks, mp_face
            landmark_drawing_spec = mp_drawing.DrawingSpec(color
            connection drawing spec = mp drawing.DrawingSpec(col
```

Passo 23 – Entendendo Coordenadas

- O retorno trata-se de uma lista com diversas coordenadas dos eixos x, y e z, então essa é a maneira que o MediaPipe armazena e disponibiliza as coordenadas.

Output exceeds the size limit. Open the full output data in a text editor

```
[landmark {  
  x: 0.57531583  
  y: 0.527997806  
  z: -0.037834693  
}  
landmark {  
  x: 0.575518  
  y: 0.4628154  
  z: -0.077181056  
}  
landmark {  
  x: 0.5743907  
  y: 0.48317027  
  z: -0.039508026  
}
```

Passo 23 – Entendendo Coordenadas

- É importante salientar que esses valores podem variar de acordo com a última posição de coordenadas detectada, além, é claro, das diferenças faciais entre uma pessoa e outra.

Output exceeds the size limit. Open the full output data in a text editor

```
[landmark {  
  x: 0.57531583  
  y: 0.527997806  
  z: -0.037834693  
}  
landmark {  
  x: 0.575518  
  y: 0.4628154  
  z: -0.077181056  
}  
landmark {  
  x: 0.5743907  
  y: 0.48317027  
  z: -0.039508026  
}
```


Passo 23 – Entendendo Coordenadas

- Construimos e acessamos essas coordenadas através de um loop for que itera por cada objeto contido na lista.

Output exceeds the size limit. Open the full output data in a text editor

```
[landmark {  
  x: 0.57531583  
  y: 0.527997806  
  z: -0.037834693  
}  
landmark {  
  x: 0.575518  
  y: 0.4628154  
  z: -0.077181056  
}  
landmark {  
  x: 0.5743907  
  y: 0.48317027  
  z: -0.039508026  
}
```

Passo 23 – Entendendo Coordenadas

- Construimos e acessamos essas coordenadas através de um loop for que itera por cada objeto contido na lista.

Output exceeds the size limit. Open the full output data in a text editor

```
[landmark {  
  x: 0.57531583  
  y: 0.527997806  
  z: -0.037834693  
}  
landmark {  
  x: 0.575518  
  y: 0.4628154  
  z: -0.077181056  
}  
landmark {  
  x: 0.5743907  
  y: 0.48317027  
  z: -0.039508026  
}
```

Passo 23 – Entendendo Coordenadas

- O retorno é semelhante ao anterior e traz coordenadas separadas.
- Embora se pareça com um dicionário, trata-se de um objeto do próprio MediaPipe para armazenar coordenadas.
- Lembre-se que o resultado de `saida_facemesh.multi_face_landmarks` só aparece porque executamos a célula anterior, na qual é feito o processamento do frame, portanto, se houver a execução da célula anterior, esta pode gerar algum erro..

```
for face_landmarks in saida_facemesh.multi_face_landmarks:  
    print(face_landmarks)
```

✓ 0.3s

Output exceeds the **size limit**. Open the full output data [in a text editor](#)

```
landmark {  
  x: 0.57531583  
  y: 0.52797806  
  z: -0.037834693  
}  
landmark {  
  x: 0.575518  
  y: 0.4628154  
  z: -0.077181056
```

Passo 23 – Entendendo Coordenadas

- O retorno é semelhante ao anterior e traz coordenadas separadas.
- Embora se pareça com um dicionário, trata-se de um objeto do próprio MediaPipe para armazenar coordenadas.
- Lembre-se que o resultado de `saida_facemesh.multi_face_landmarks` só aparece porque executamos a célula anterior, na qual é feito o processamento do frame, portanto, se houver a execução da célula anterior, esta pode gerar algum erro..

```
for face_landmarks in saida_facemesh.multi_face_landmarks:
    print(face_landmarks)
```

✓ 0.3s

Output exceeds the **size limit**. Open the full output data [in a text editor](#)

```
landmark {
  x: 0.57531583
  y: 0.52797806
  z: -0.037834693
}
landmark {
  x: 0.575518
  y: 0.4628154
  z: -0.077181056
```

Passo 24 – Acessando coordenadas

- Além de encontrar o objeto
- de landmarks,
- também conseguimos acessar
- diretamente
- as coordenadas x, y e z
- que estão armazenadas
- dentro deste retorno.
- É possível, ainda, encontrarmos o id correspondente a essas coordenadas, assim conseguimos saber a qual ponto correspondem.
- Para isso, podemos ler face_landmarks com outro loop for.

```
landmark {  
  x: 0.57531583  
  y: 0.52797806  
  z: -0.037834693  
}
```

Passo 25 – Acessando coordenadas

- Além de encontrar o objeto
- de landmarks,
- também conseguimos acessar
- diretamente
- as coordenadas x, y e z
- que estão armazenadas
- dentro deste retorno.
- É possível, ainda, encontrarmos o id correspondente a essas coordenadas, assim conseguimos saber a qual ponto correspondem.
- Para isso, podemos ler face_landmarks com outro loop for.

```
landmark {  
  x: 0.57531583  
  y: 0.52797806  
  z: -0.037834693  
}
```

Passo 26 – 468 pontos

- Que ponto é o que?
- Que coordenada corresponde a tal ponto?

```
for face_landmarks in saida_facemesh.multi_face_landmarks:  
    face = face_landmarks  
    for id_coord, coord_xyz in enumerate(face.landmark):  
        print(coord_xyz)
```


x: 0.57531583

y: 0.52797806

z: -0.037834693

x: 0.575518

y: 0.4628154

z: -0.077181056

x: 0.5743907

y: 0.48317027

z: -0.039508026

```
for face_landmarks in saida_facemesh.multi_face_landmarks:  
    face = face_landmarks  
    for id_coord, coord_xyz in enumerate(face.landmark):  
        print(id_coord)
```

Output exceeds the size limit. Open the full output data in a text editor

0

1

2

3

4

5

6

7

8

9

10

11

12

13

...

464

465

466

467

Passo 27 – Analisando os olhos

- Vamos começar o protótipo analisando os olhos, porque eles são bons indicadores de sono.
- Quando temos sono, nossos olhos ficam fechados por mais tempo e a piscada demora um pouco mais. Existem outros parâmetros que indicam a sonolência, mas os olhos talvez sejam os mais importantes.
- Para aprofundarmos nossos estudos sobre os olhos, descobrirmos o tempo que ficam fechados, enfim, investigarmos a abertura ocular, precisamos buscar algumas leituras. Os artigos, por exemplo, podem nos oferecer conhecimentos teóricos nessa área.

Passo 28 – Artigo (Papers)

- Artigo de detecção de piscadas usando coordenadas da face

*21st Computer Vision Winter Workshop
Luka Čehovin, Rok Mandeljc, Vitomir Štruc (eds.)
Rimske Toplice, Slovenia, February 3–5, 2016*

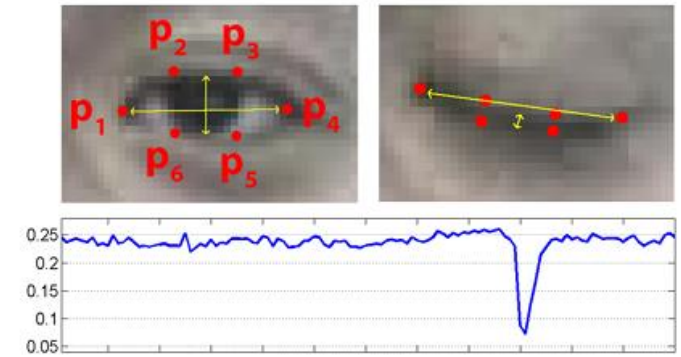
Real-Time Eye Blink Detection using Facial Landmarks

Tereza Soukupová and Jan Čech
Center for Machine Perception, Department of Cybernetics
Faculty of Electrical Engineering, Czech Technical University in Prague
`{soukuter, cechj}@cmp.felk.cvut.cz`

[05.pdf](#)

Passo 29 – Pontos dos Olhos

- Vamos começar o protótipo **analisando os olhos**, porque eles são bons indicadores de sono.
- Quando temos sono, nossos olhos ficam fechados por mais tempo e a piscada demora um pouco mais.
- Existem outros parâmetros que indicam a sonolência, mas os olhos talvez sejam os mais importantes.
- Para aprofundarmos nossos estudos sobre os olhos, descobrirmos o tempo que ficam fechados, enfim, investigarmos a abertura ocular, precisamos buscar algumas leituras.
- Os artigos, por exemplo, podem nos oferecer conhecimentos teóricos nessa área.

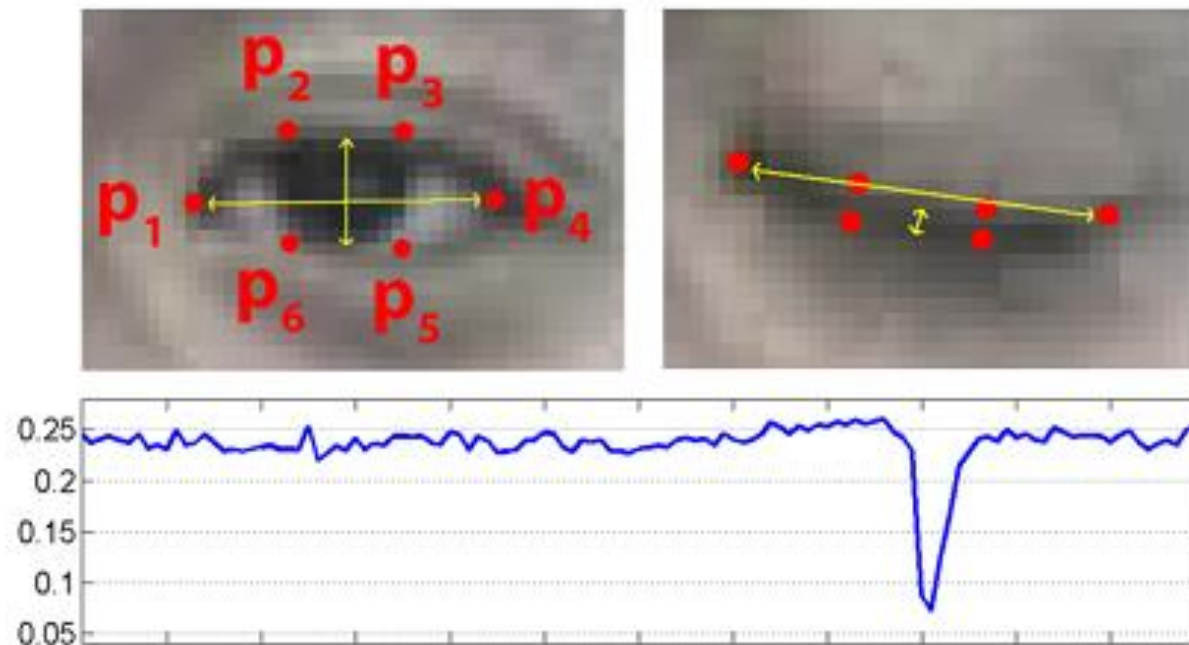


Passo 30 – O artigo

- artigo [Real-Time Eye Blink Detection using Facial Landmarks](#) (em tradução livre: Detecção de piscar de olhos em tempo real usando coordenadas faciais), escrito por **Tereza Soukupova** e **Jan Cechnos**, nos fornece duas informações principais

Passo 40 – 12 pontos nos olhos

- A primeira informação está na primeira página do artigo, é a detecção de 6 pontos a serem verificados em cada olho: um em cada extremidade dos olhos, dois na parte de cima e dois na parte de baixo. Considerando os dois olhos, são 12 pontos no total.



Passo 41 - EAR

- A segunda informação está na página 3 do artigo.
- É o valor de **EAR**, uma equação que utilizaremos para verificar se os olhos estão abertos ou fechados.
- **EAR** significa **Eye Aspect Ratio** (Proporção do olho) e se refere à distância dos olhos.

Passo 42 – A Fórmula

- O EAR é dado pelo valor da distância euclidiana entre o **ponto P2** e o **ponto P6**, mais o valor da distância euclidiana entre o **ponto P3** e o **ponto P5**.
- Isso tudo sobre duas vezes a distância euclidiana entre o **ponto P1** e o **ponto P4**.

$$\text{EAR} = \frac{\|p_2 - p_6\| + \|p_3 - p_5\|}{2\|p_1 - p_4\|},$$

Passo 43 – Distância Euclidiana



Passo 44 - EAR

2.1. Description of features

For every video frame, the eye landmarks are detected. The eye aspect ratio (EAR) between height and width of the eye is computed.

$$\text{EAR} = \frac{\|p_2 - p_6\| + \|p_3 - p_5\|}{2\|p_1 - p_4\|}, \quad (1)$$

where $p_1, p_2, p_3, p_4, p_5, p_6$ are the 2D landmark locations of

Passo 45 – Entendendo a fórmula

- A segunda informação está na página 3 do artigo.
- É o valor de **EAR**, uma equação que utilizaremos para verificar se os olhos estão abertos ou fechados.
- **EAR** significa **Eye Aspect Ratio** e se refere à distância dos olhos.
- O EAR é dado pelo valor da distância euclidiana entre o ponto P2 e o ponto P6, mais o valor da distância euclidiana entre o ponto P3 e o ponto P5. Isso tudo sobre duas vezes a distância euclidiana entre o ponto P1 e o ponto P4.

Passo 46 – A fórmula

$$\text{EAR} = \frac{\|p_2 - p_6\| + \|p_3 - p_5\|}{2\|p_1 - p_4\|},$$

EAR **Eye Aspect Ratio**

Passo 47 – Distância Euclidiana

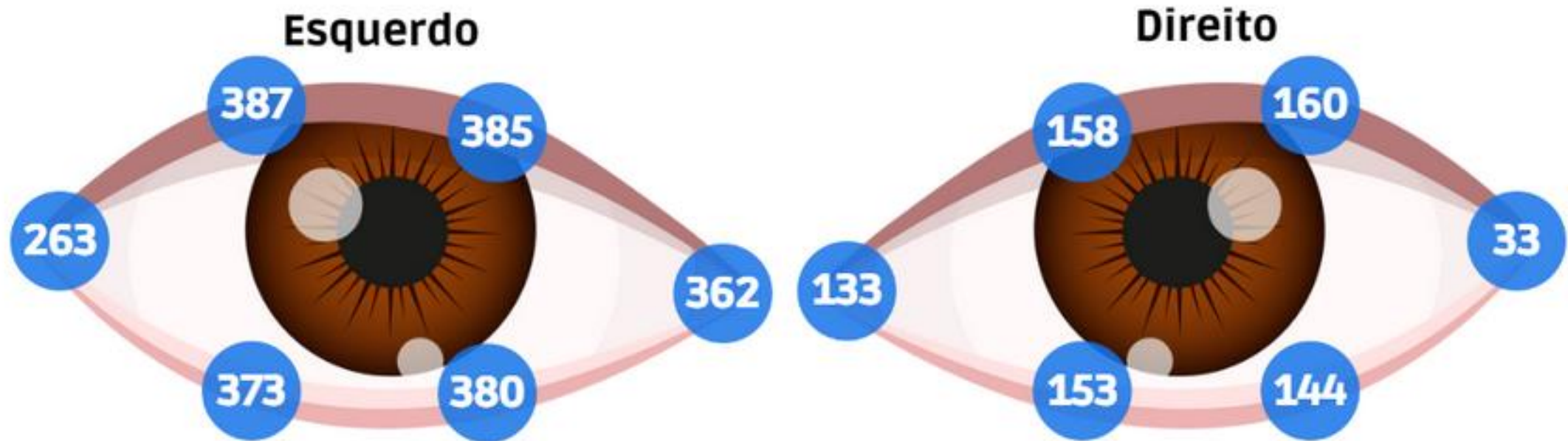
$$\underline{\text{EAR}} = \frac{\|p_2 - p_6\| + \|p_3 - p_5\|}{2\|p_1 - p_4\|},$$

- O EAR é dado pelo valor da distância euclidiana entre o ponto P2 e o ponto P6, mais o valor da distância euclidiana entre o ponto P3 e o ponto P5. Isso tudo sobre duas vezes a distância euclidiana entre o ponto P1 e o ponto P4.

Passo 48 – Os olhos

- Os 6 pontos que estão determinados nessa equação foram abordados na página 1, que acabamos de conferir.
- Esse artigo está disponível para download e recomendo que você faça a leitura, porque ele apresenta vários documentos e conclusões interessantes.

Passo 49 – Os pontos dos olhos



Passo 50 – Criando uma lista

```
p_olho_esq = [385, 380, 387, 373, 362, 263]  
p_olho_dir = [160, 144, 158, 153, 33, 133]
```

Passo 51 – Olhos

- O resultado é a concatenação de todos os valores dos pontos dos olhos.
- Criamos uma lista composta por todos os pontos dos nossos olhos. Com isso, é possível criar um código para visualização desses pontos com OpenCV.

```
5
6  p_olho_esq = [385, 380, 387, 373, 362, 263]
7  p_olho_dir = [160, 144, 158, 153, 33, 133]
8  p_olhos = p_olho_esq+p_olho_dir
9
10
11  cap = cv2.VideoCapture(0)
12
```

Passo 52 – Visualização no OpenCV

- Criamos as listas com todos os pontos dos nossos olhos. Falta criar uma visualização no OpenCV para, de fato, verificarmos essa transformação.
- Começaremos retomando o código do MediaPipe de verificação dos pontos que fizemos anteriormente.

Passo 53 - Adicionando código

- Nós adicionaremos um código abaixo da verificação de sucesso.
- Após o continue, vamos apertar "Enter" e, fora do bloco de condicional, nós coletaremos o **tamanho, isto é, a largura e comprimento do vídeo.**
- O motivo de fazer isso é porque precisamos de um parâmetro para transformar os pontos normalizados do MediaPipe em pontos de pixel

Passo 54 – shape do nosso frame

```
if not sucesso:  
    print('Ignorando o frame vazio da câmera.')  
    continue  
comprimento, largura, _ = frame.shape
```

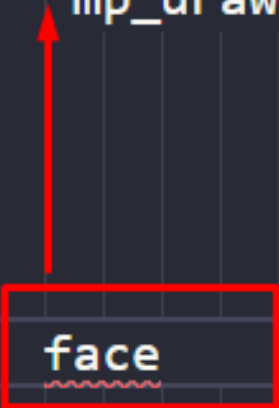
```
# pegando o shape do nosso frame
# O shape de um frame em OpenCV é uma propriedade da matriz (array)
# Que representa o frame e fornece as dimensões da imagem capturada.
comprimento, largura, _ = frame.shape
```

Passo 56

- Transformando pontos normalizados do MediaPipe em Pixels
- Para marcar esses pontos no rosto o MediaPipe precisa que seja Pixels

Passo 57 – Objeto face

```
57      Dentro dos parênteses, colocaremos o frame, que é o c
58      e o face_landmarks, que são as coordenadas de cada po
59      Ainda nos parênteses, utilizaremos o mp_face_mesh.FAC
60      os nossos pontos
61
62      """
63      mp_drawing.draw_landmarks(frame,
64                                face_landmarks,
65                                mp_face_mesh.FACEMESH_CONTOURS,
66                                landmark_drawing_spec = mp_drawing.DrawingSpec(landmark_size=3, landmark_color=mp_drawing.Color(255, 255, 255)),
67                                connection_drawing_spec = mp_drawing.DrawingSpec(connection_size=3, connection_color=mp_drawing.Color(255, 255, 255)))
68      face
69
```



Passo 58 – face recebe face_landmarks

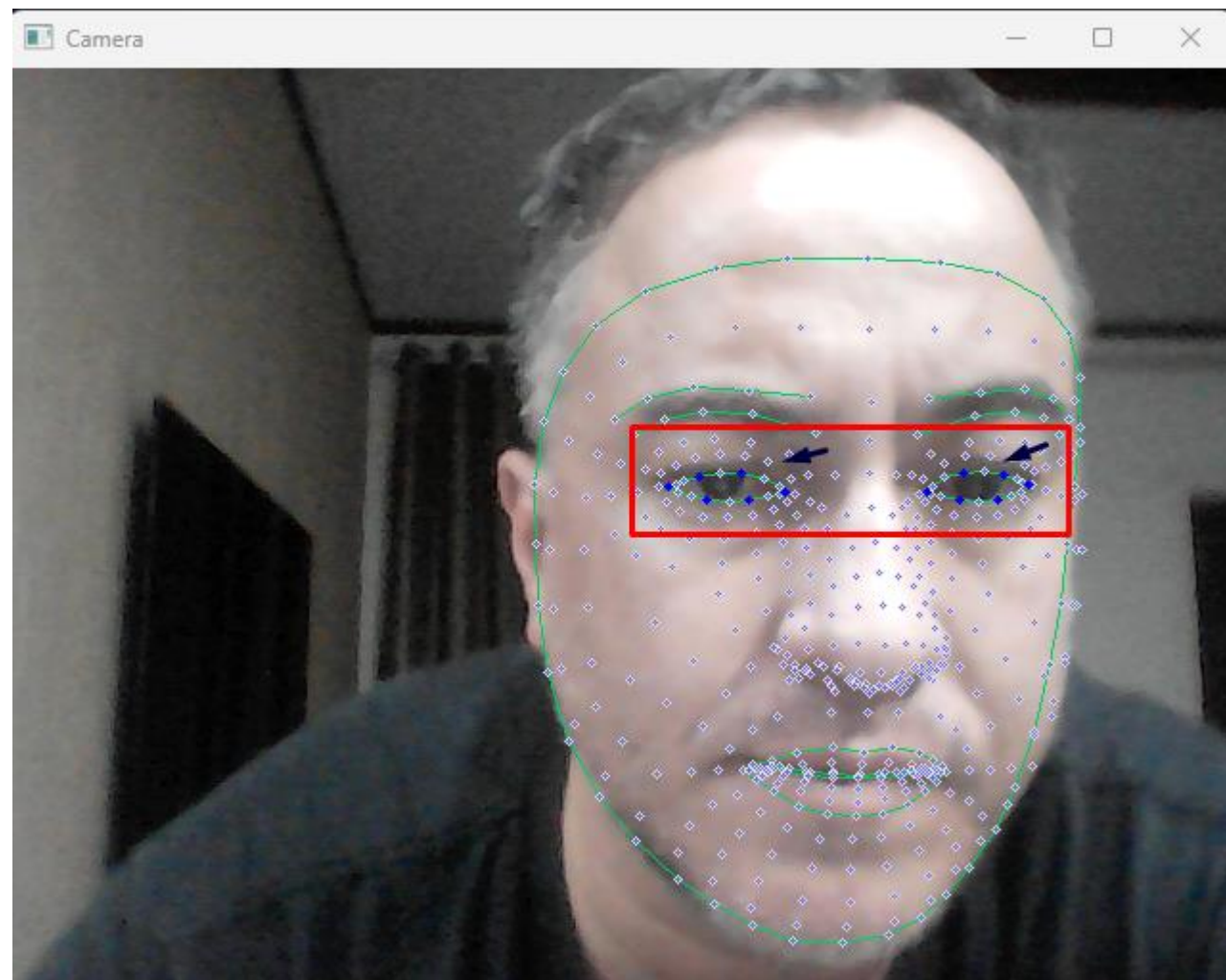
```
try:
    for face_landmarks in saida_facemesh.multi_face_landmarks:
        mp_drawing.draw_landmarks(frame, face_landmarks, mp_face_mesh.
            FACEMESH_CONTOURS,
            landmark_drawing_spec = mp_drawing.DrawingSpec(color=(255,102,102),
                thickness=1,circle_radius=1),
            connection_drawing_spec = mp_drawing.DrawingSpec(color=(102,204,0),
                thickness=1,circle_radius=1))
        face = face_landmarks
```

- face = **face_landmarks**: ele tem as coordenadas

Passo 59 - Normalizando

```
face = face_landmarks.landmark # garantir que as coordenadas sejam enviadas diretamente para o nosso laço for
for id_coord, coord_xyz in enumerate(face):
    if id_coord in p_olhos:
        coord_cv = mp_drawing._normalized_to_pixel_coordinates(coord_xyz.x, coord_xyz.y, largura, comprimento)
```

Passo 60



```
face = face_landmarks.landmark
for id_coord, coord_xyz in enumerate(face):
    if id_coord in p_olhos:
        coord_cv = mp_drawing._normalized_to_pixel_coordinates(coord_xyz.x, coord_xyz.y, largura, comprimento)
        cv2.circle(frame, coord_cv, 2, (255,0,0), -1)
```

Passo 61 – EAR dos olhos D e E

$$EAR_{dir} = \frac{\|p_{160} - p_{144}\| + \|p_{158} - p_{153}\|}{2 \cdot \|p_{33} - p_{133}\|}$$

$$EAR_{esq} = \frac{\|p_{385} - p_{380}\| + \|p_{387} - p_{373}\|}{2 \cdot \|p_{362} - p_{263}\|}$$

Passo 62 – NumPy

- Nosso primeiro passo nessa nova jornada do cálculo EAR
- É importar a biblioteca NumPy.
- Então vamos subir e importar o NumPY

```
1  import cv2
2  import mediapipe as mp
3  import numpy as np ←
```

Passo 63 – Criando a função

- Precisamos calcular o valor de EAR para o olho direito e para o olho esquerdo.
- O resultado dessa função será a média do valor entre esses dois EARs.

Passo 64 – Criando a função

- Então, vamos passar def para criarmos a função.
- Ela se chamará calculo_ear().
- Dentro dos parâmetros da função, vamos solicitar as coordenadas da face e os pontos referentes ao olho esquerdo e ao olho direito: **face, p_olho_dir, p_olho_esq**

Passo 65 – A função

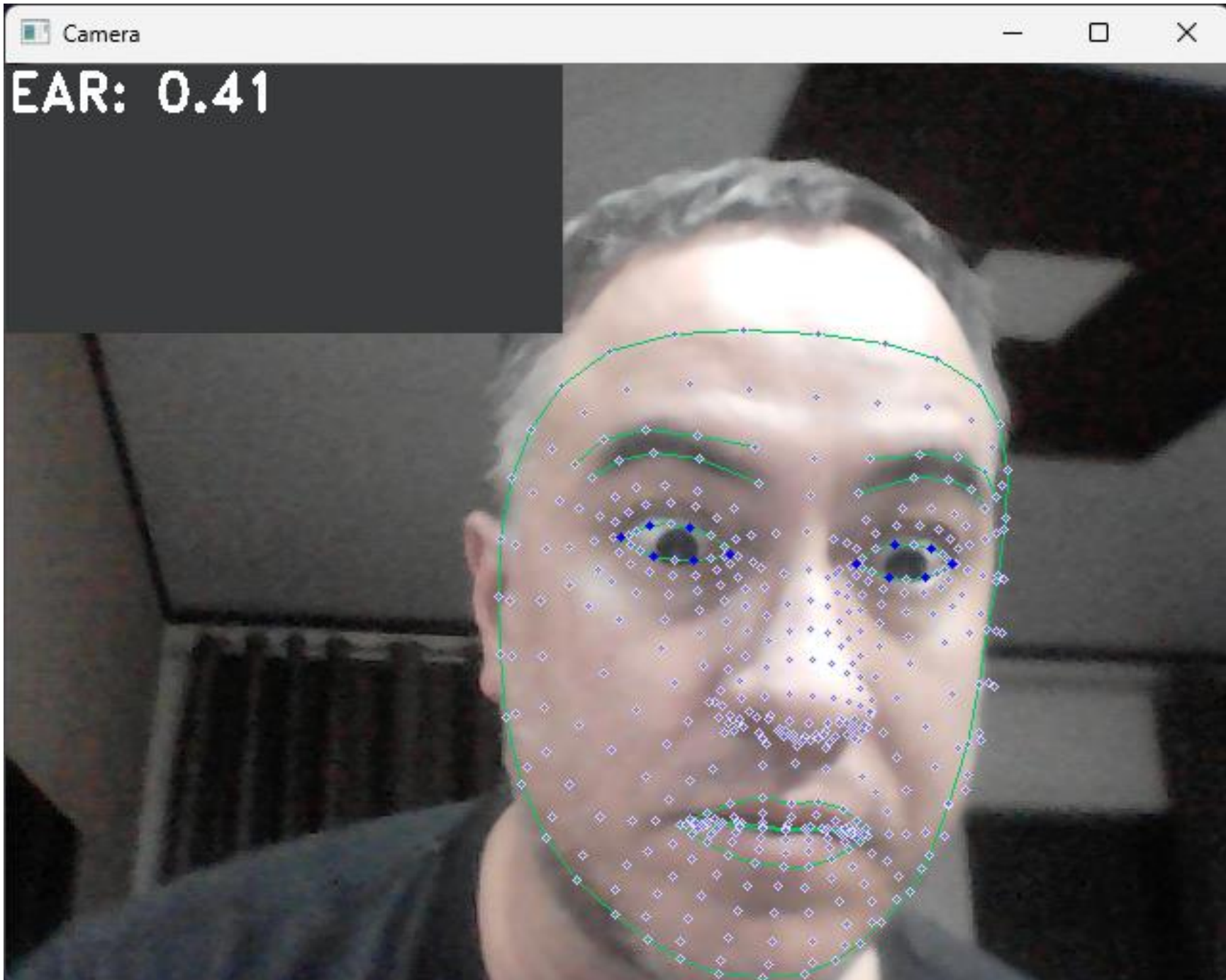
```
# função EAR

def calculo_ear(face, p_olho_dir, p_olho_esq):
    try:
        face = np.array([[coord.x, coord.y] for coord in face])
        face_esq = face[p_olho_esq, :]
        face_dir = face[p_olho_dir, :]

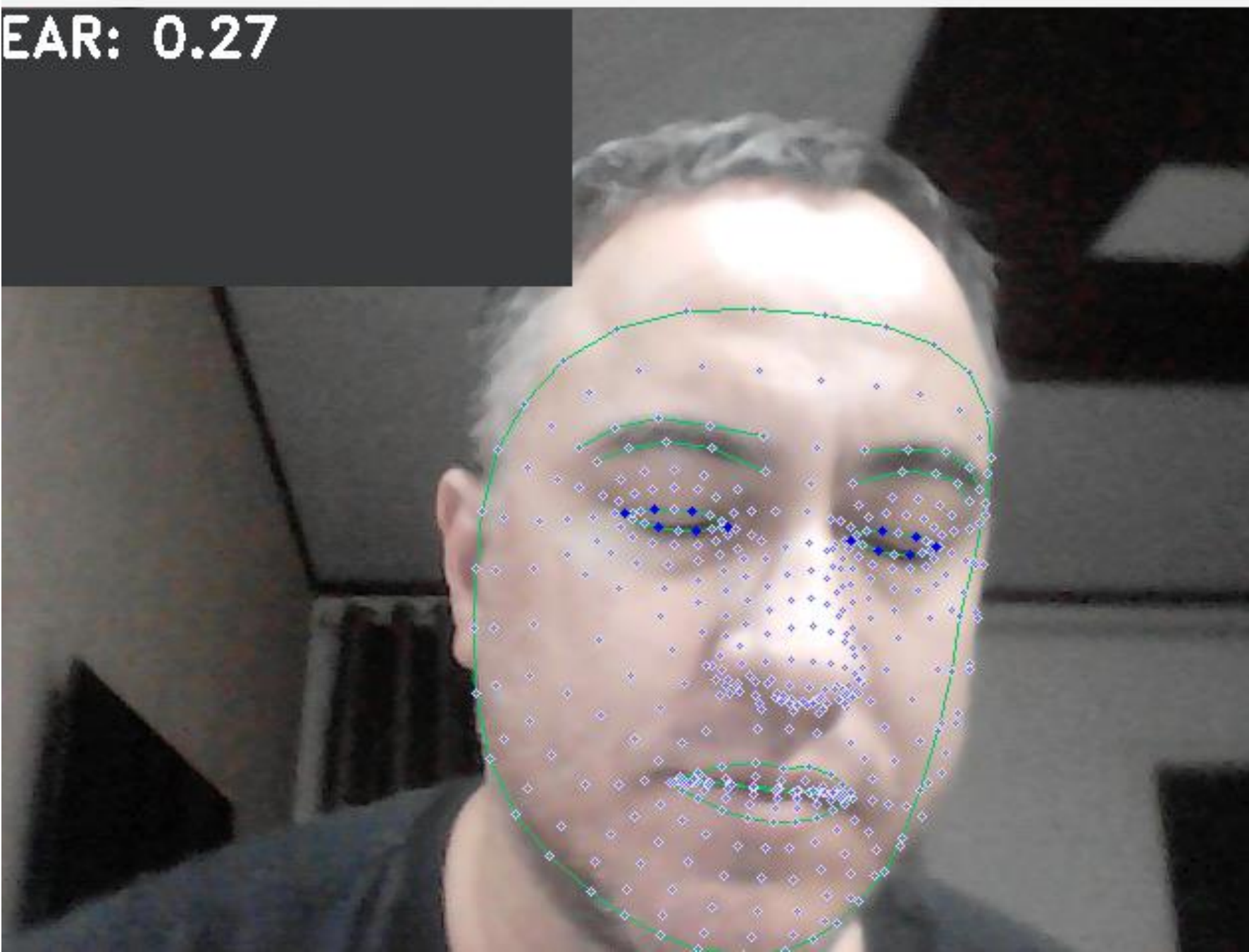
        ear_esq = (np.linalg.norm(face_esq[0]-face_esq[1])+np.linalg.norm(face_esq[2]-face_esq[3]))/(2*(np.linalg.norm(face_esq[4]-face_esq[5]))
        ear_dir = (np.linalg.norm(face_dir[0]-face_dir[1])+np.linalg.norm(face_dir[2]-face_dir[3]))/(2*(np.linalg.norm(face_dir[4]-face_dir[5]))
    except:
        ear_esq = 0.0
        ear_dir = 0.0
    media_ear = (ear_esq+ear_dir)/2
    return media_ear
```

Passo 66 – Calculo do EAR

- Assim, adicionamos o EAR no texto.
- Vamos apertar "Shift + Enter", esperar inicializar e, está pronto!
- Nossa janela de vídeo apareceu e mostra tanto a visualização da nossa face com os pontos, quanto o valor do EAR sendo calculado ao lado superior esquerdo da tela. Há uma variação no valor se estamos com o olho aberto ou fechado.



EAR: 0.27



Passo 67 – Calculo do Tempo

- Se estamos sonolentos, com vontade de dormir, tendemos a fechar os olhos por mais tempo ou a fechá-los de vez e dormir.
- Quando construímos um código de análise da sonolência através dos olhos, precisamos verificar **quanto tempo a pessoa que está sendo analisada passa de olhos fechados**.
- Nós podemos fazer isso, pois já sabemos calcular e verificar o valor de EAR.
- Vamos trabalhar com o tempo em que os olhos ficam fechados e o tempo em que o valor de EAR estará menor.
- Isso porque quando o EAR está alto, significa que o olho está aberto, mas quando ele diminui, quer dizer que o olho está fechado.

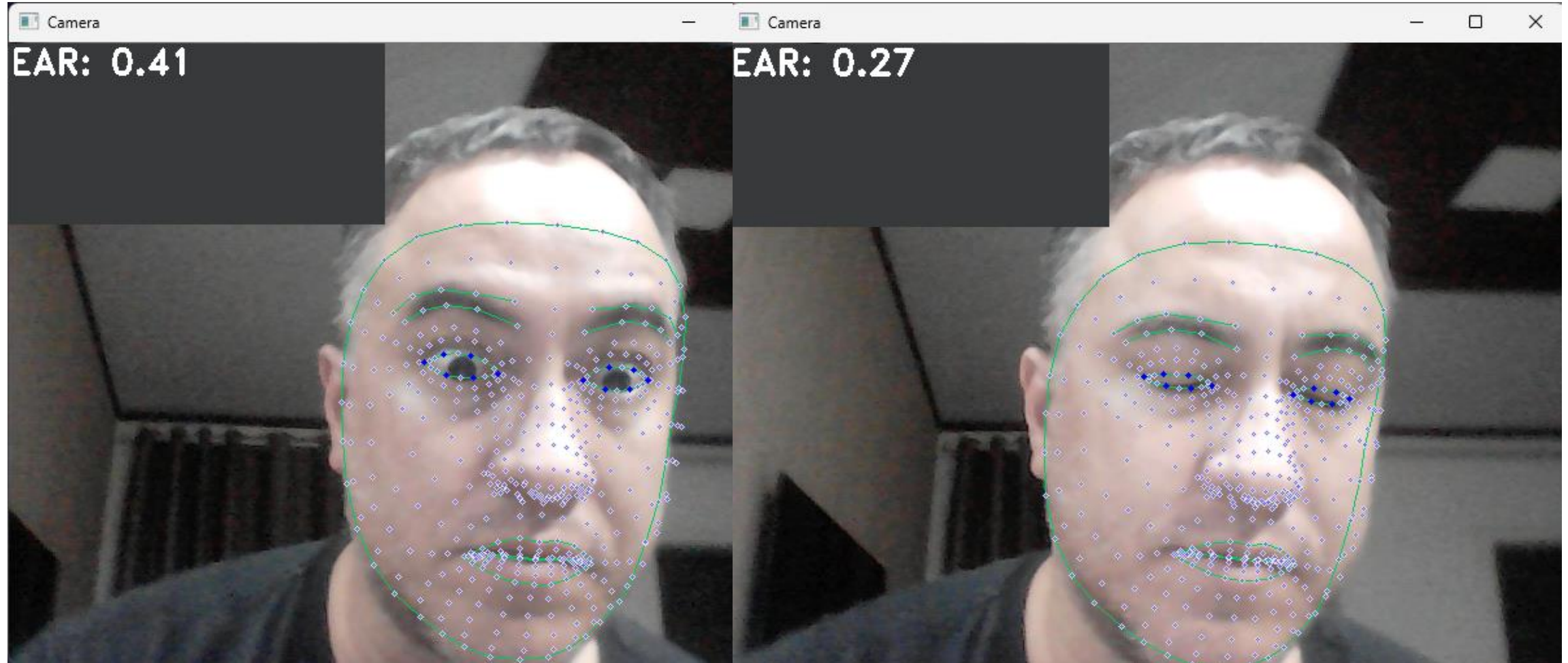
Passo 68 – Calculo do Tempo

- Cada olho tem suas particularidades.
- Existem aberturas oculares diferentes.
- Quando eu estou com o olho aberto, meu EAR é sempre um valor acima de 0.3, aproximadamente 0.4.
- Se fecho os olhos ou se os deixo semi cerrados, o valor de EAR é aproximadamente 0.22.

Passo 69 – Calculo do Tempo

- Então, podemos dizer que o valor 0.3 é o limiar entre o olho aberto e fechado.
- Eu vou memorizar esse valor.
- Recomendo que você também verifique seus valores: tanto para quando o olho está aberto, quanto para quando está fechado e memorize o limiar, ele será usado logo mais.

Passo 69 - Limiar



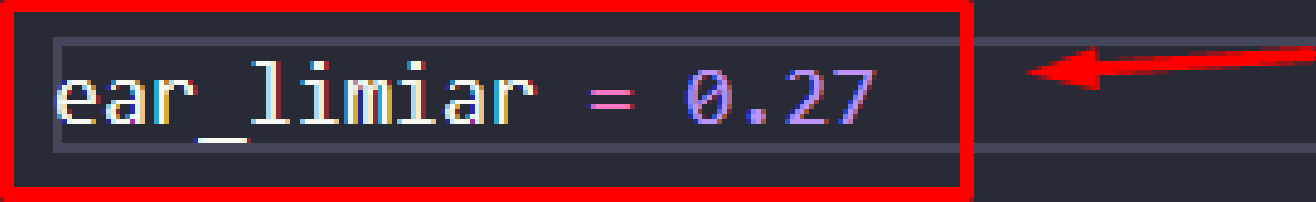
Passo 70 – biblioteca time

- Agora fecharemos a janela de vídeo e começaremos a trabalhar com o tempo.
- Para isso, vamos retornar à célula dos imports.
- Nós apertaremos "Enter" e adicionaremos a **biblioteca time**, que lida com o tempo.

```
1  import cv2
2  import mediapipe as mp
3  import numpy as np
4  import time
```

Passo 71 - Tempo

```
25  
26 ear_limiar = 0.27  
27  
28 cap = cv2.VideoCapture(0)  
29  
30 mp_drawing = mp.solutions.drawing
```



Passo 72 - Dormindo

```
26     ear_limiar = 0.27  
27     dormindo = 0
```

Passo 73 – Calculando o limiar

```
if ear < ear_limiar:  
    t_inicial = time.time() if dormindo == 0 else t_inicial  
    dormindo = 1  
if dormindo == 1 and ear >= ear_limiar:  
    dormindo = 0  
    t_final = time.time()
```

Passo 78 – Calculando o Tempo que o olho ficou fechado

- Agora podemos calcular o tempo geral, isto é, o tempo em que o olho ficou fechado.
- Para isso, criaremos uma variável chamada tempo.
- Esse tempo será igual ao tempo final (o tempo em "tempo real") menos o tempo inicial (o tempo marcado toda vez que fechamos os olhos).

EAR: 0.24

Tempo: 3.154



Muito tempo com olhos fechados!

Passo 79 – Revisando

- Durante a construção do projeto, conseguimos:
- Identificar os pontos dos olhos.
- Definir um índice para verificar se olho está aberto.
- Produzir uma mensagem de alerta se olho ficar fechado muito tempo

Passo 80 – Verificando problemas no projeto

- Algo recomendado para qualquer projeto é realizar uma análise: verificar possíveis problemas e pensar em ajustes para eles.

Passo 81 – Rodar novamente

- Vamos **rodar a célula** em que produzimos nosso código de **verificação de EAR** e fazer essa análise.
- Se eu fecho **meus olhos**, o tempo passa a ser contado e uma **mensagem de alerta** aparece caso permaneça com os olhos fechados por muito tempo.

Passo 82 –Ponto de inflexão (olhos)

- **Nessa aula, você aprendeu a:**

- Identificar as coordenadas e os pontos da solução Face Mesh;
- Localizar e exibir os pontos que representam os olhos direito e esquerdo;
- Interpretar as coordenadas do MediaPipe;
- Explorar o estado da arte para criação de um projeto;
- Formular o cálculo do índice de nível de abertura dos olhos; e
- Desenvolver o cálculo de EAR em código.

Ponto 83 – Verificando pontos da boca

- Durante a construção do projeto, conseguimos:
- Identificar os pontos dos olhos.
- Definir um índice para verificar se olho está aberto.
- Produzir uma mensagem de alerta se olho ficar fechado muito tempo.

Ponto 83 – Verificando pontos da boca

- Algo recomendado para qualquer projeto é realizar uma análise: verificar possíveis problemas e pensar em ajustes para eles.

Ponto 83 – Verificando pontos da boca

- Vamos rodar a célula em que produzimos nosso código de verificação de EAR e fazer essa análise.
- Se eu fecho meus olhos, o tempo passa a ser contado e uma mensagem de alerta aparece caso permaneça com os olhos fechados por muito tempo..

Ponto 83 – Verificando pontos da boca

- No entanto, se eu estiver sorrindo, dando uma gargalhada, meus olhos ficarão **semi cerrados**.
- Isso faz com que o nosso código entenda que os olhos estão fechados.
- Vou fazer o teste sorrindo.
- A mensagem apareceu.

EAR: 0.2

Tempo: 2.216



Muito tempo com olhos fechados!

Ponto 83 – Verificando pontos da boca

- Durante a construção do projeto, conseguimos:
- Identificar os pontos dos olhos.
- Definir um índice para verificar se olho está aberto.
- Produzir uma mensagem de alerta se olho ficar fechado muito tempo.

Ponto 83 – Verificando pontos da boca

- Durante a construção do projeto, conseguimos:
- Identificar os pontos dos olhos.
- Definir um índice para verificar se olho está aberto.
- Produzir uma mensagem de alerta se olho ficar fechado muito tempo.

Ponto 83 – Verificando pontos da boca

- Durante a construção do projeto, conseguimos:
- Identificar os pontos dos olhos.
- Definir um índice para verificar se olho está aberto.
- Produzir uma mensagem de alerta se olho ficar fechado muito tempo.

Ponto 83 – Verificando pontos da boca

- Durante a construção do projeto, conseguimos:
- Identificar os pontos dos olhos.
- Definir um índice para verificar se olho está aberto.
- Produzir uma mensagem de alerta se olho ficar fechado muito tempo.

Ponto 83 – Verificando pontos da boca

- Durante a construção do projeto, conseguimos:
- Identificar os pontos dos olhos.
- Definir um índice para verificar se olho está aberto.
- Produzir uma mensagem de alerta se olho ficar fechado muito tempo.

Ponto 83 – Verificando pontos da boca

- Durante a construção do projeto, conseguimos:
- Identificar os pontos dos olhos.
- Definir um índice para verificar se olho está aberto.
- Produzir uma mensagem de alerta se olho ficar fechado muito tempo.

Ponto 83 – Verificando pontos da boca

- Durante a construção do projeto, conseguimos:
- Identificar os pontos dos olhos.
- Definir um índice para verificar se olho está aberto.
- Produzir uma mensagem de alerta se olho ficar fechado muito tempo.

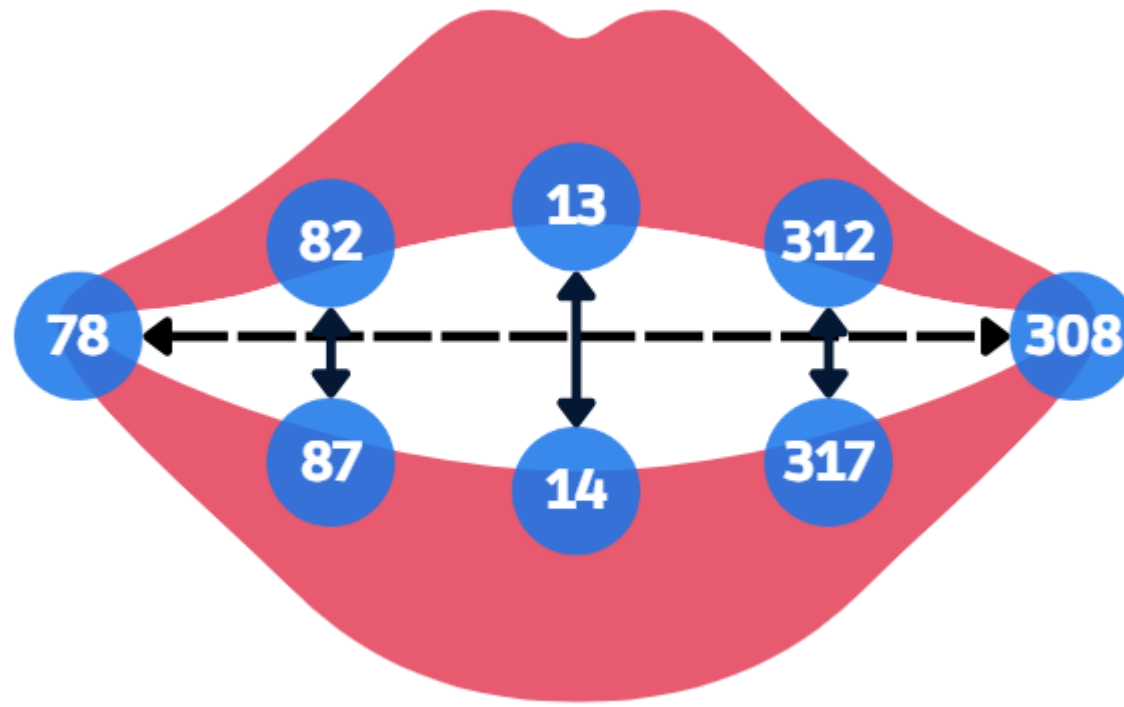
Ponto 83 – Verificando pontos da boca

- Durante a construção do projeto, conseguimos:
- Identificar os pontos dos olhos.
- Definir um índice para verificar se olho está aberto.
- Produzir uma mensagem de alerta se olho ficar fechado muito tempo.

Ponto 83 – Verificando pontos da boca

- Durante a construção do projeto, conseguimos:
- Identificar os pontos dos olhos.
- Definir um índice para verificar se olho está aberto.
- Produzir uma mensagem de alerta se olho ficar fechado muito tempo.

1. Precisamos verificar a abertura da boca, porque quase ninguém dorme sorrindo ou gargalhando.
2. Geralmente, as pessoas dormem com o rosto relaxado.
3. Agora fecharemos a janela de vídeo e calcularemos a abertura da boca.
4. Para isso, utilizamos os pontos da boca

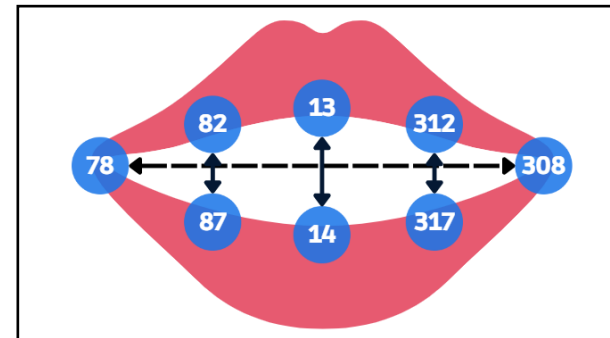


Passo 84

- Temos **8 pontos** especificamente na boca.
- Nós realizaremos um cálculo similar ao EAR, utilizando esses pontos. O **MAR** ou **Mouth Aspect Ratio** (Em tradução livre: Proporção da boca)
- Funciona exatamente como o EAR, mas para as coordenadas da boca.

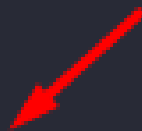
Passo 85

- Nosso primeiro passo é encontrar os pontos que estão indicados na imagem no MediaPipe.
- Então, temos que identificar esses pontos e mostrá-los na tela, assim como fizemos para o EAR.
- Então, criaremos um novo código.
- Dentro dela, faremos uma lista com os pontos da boca, exatamente os que são indicados na imagem : 82, 87, 13, 14, 312, 317, 308 e 78.



Passo 86

```
6  p_olho_esq = [385, 380, 387, 373, 362, 263]
7  p_olho_dir = [160, 144, 158, 153, 33, 133]
8  p_olhos = p_olho_esq + p_olho_dir
9
10 # variáveis da boca
11 p_boca = [82, 87, 13, 14, 312, 317, 78, 308]
```



Passo 87

- Nessa célula, temos o **p_boca**, variável que guardará a lista, igual aos valores dos pontos.
- Vamos rodar esse código para que a variável seja criada.
- Agora, adicionaremos esses pontos na nossa visualização.
- Nós já preparamos um código de verificação das coordenadas presentes nos pontos dos olhos.
- A partir desses valores, desnormalizamos o valor das coordenadas e colocamos um círculo em cada uma delas.
- Lembrando que os pontos da boca são diferentes dos pontos dos olhos. Vamos copiar o código da condicional de verificação do id da coordenada para o id dos olhos.

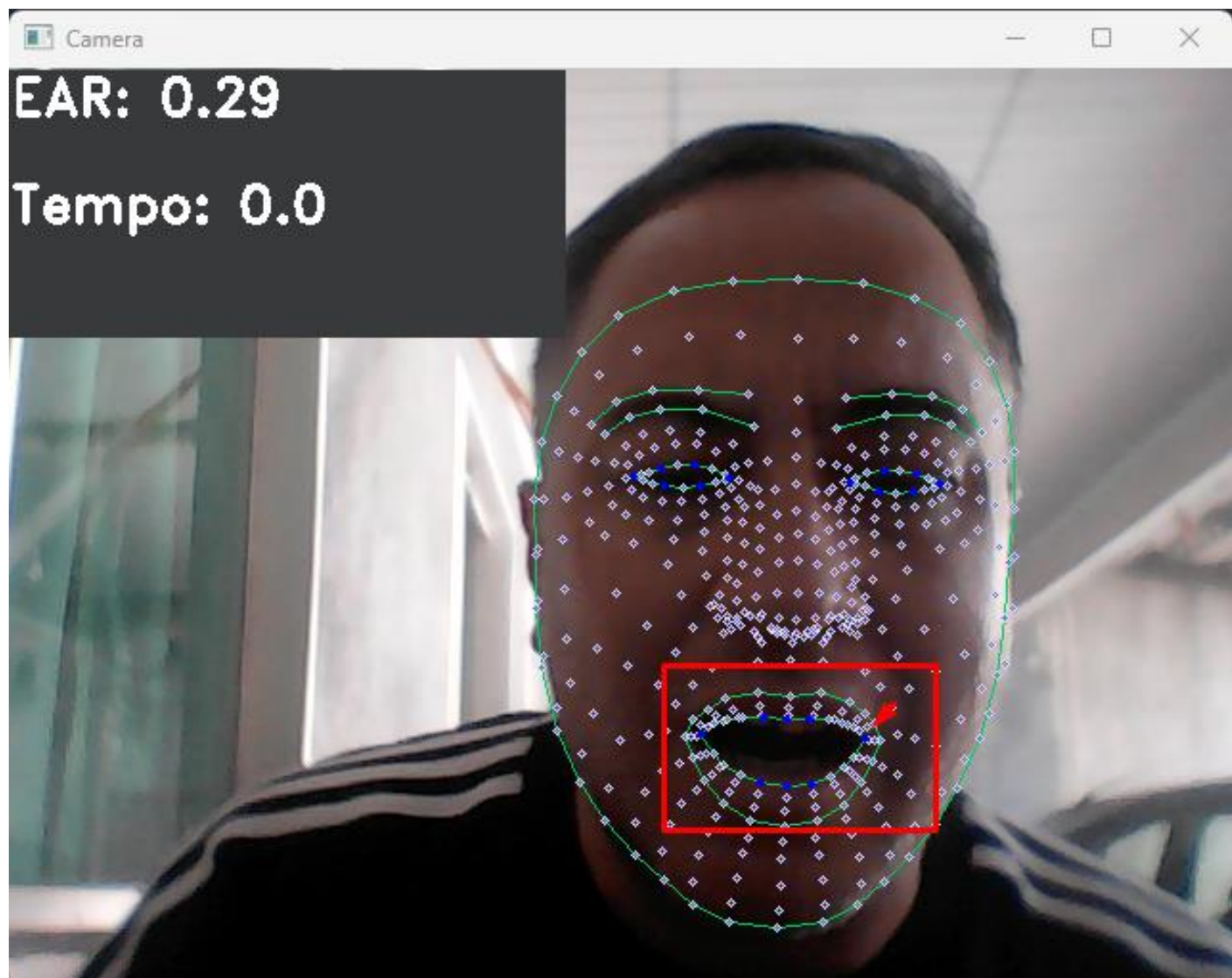
Passo – 88 (copie o if dos olhos)

```
60
61     face = face_landmarks.landmark
62     for id_coord, coord_xyz in enumerate(face):
63         if id_coord in p_olhos:
64             coord_cv = mp_drawing._normalized_to_pixel_coordinates(coord_xyz, frame.shape[:2])
65             cv2.circle(frame, coord_cv, 2, (255,0,0), -1)
66
```

Passo 99

```
_coord, coord_xyz in enumerate(face):  
    if id_coord in p_olhos:  
        coord_cv = mp_drawing._normalized_to_pixel_coordinates(coord_xyz.x, coord_xyz.y, largura, comprimento)  
        cv2.circle(frame, coord_cv, 2, (255,0,0), -1)  
    if id_coord in p_boca:  
        coord_cv = mp_drawing._normalized_to_pixel_coordinates(coord_xyz.x, coord_xyz.y, largura, comprimento)  
        cv2.circle(frame, coord_cv, 2, (255,0,0), -1)
```

Passo 100 – Verificar o resultado



Passo 101 - MAR

- Encontramos os pontos na nossa face e agora vamos definir o valor de MAR (Mouth Aspect Ratio) dentro do nosso código para aprimorá-lo cada vez mais.
- A seguir, temos a equação de MAR:

$$MAR = \frac{\|p_{82} - p_{87}\| + \|p_{13} - p_{14}\| + \|p_{312} - p_{317}\|}{2 \cdot \|p_{78} - p_{308}\|}$$

Passo - 102

- A equação é muito similar ao EAR.
- O cálculo dos pontos ligados verticalmente se dá pela distância euclidiana entre eles.
- São seis pontos: 82 e 87; 13 e 14; e 312 e 317.
- Somamos os valores das distâncias euclidianas e dividimos por duas vezes a distância euclidiana entre os pontos das extremidades: 78 e 308.

Passo 103 – Função para MAR

- Também faremos o cálculo de maneira bastante similar.
- Nós criaremos uma linha de código para a nossa função **def calculo_mar()**.
- O **calculo_mar()** receberá os pontos da face e da boca.

```
def calculo_mar(face,p_boca):  
    try:  
        face = np.array([[coord.x, coord.y] for coord in face])  
        face_boca = face[p_boca,:]  
  
        mar = (np.linalg.norm(face_boca[0]-face_boca[1])+np.linalg.norm(face_boca[2]-face_boca[3])+np.linalg.norm(face_boca[4]-face_boca[5]))/(2*(np.linalg.norm(face_boca[6]-face_boca[7])))  
    except:  
        mar = 0.0  
  
    return mar
```

```
def calculo_mar(face,p_boca):  
    try:  
        face = np.array([[coord.x, coord.y] for coord in face])  
        face_boca = face[p_boca,:]  
  
        mar = (np.linalg.norm(face_boca[0]-face_boca[1])+np.linalg.norm(face_boca[1]-face_boca[2]))/2  
    except:  
        mar = 0.0  
  
    return mar
```

Passo 104 - Testando

- Vamos rodar o nosso código.
- Abrimos a janela de vídeo e agora aparece o cálculo do valor de MAR integrado ao nosso projeto.
- Quando estou de boca fechada, o MAR é zero.
- Se abro a boca ou sorrio, o MAR aumenta.
- Falta integrarmos esse novo valor ao cálculo de tempo em que os olhos estão fechados.