Detector de Fadiga

Ponto de Controle IV

Felipe Costa Gomes, Vitória Bandeira Melo
Engenharia Eletrônica
Universidade de Brasília - Faculdade do Gama
Gama, Distrito Federal
E-mails: felipe-gomes.fg@aluno.unb.br , vitoria.bandeira@aluno.unb.br

I. JUSTIFICATIVA

Sono e cansaço estão entre os principais motivos pelos quais acontecem acidentes nas estradas brasileiras. A probabilidade de uma sinistralidade acontecer é maior se o condutor dorme menos do que o período recomendado. Basta um momento de cansaço ou distração para que o pior aconteça. Segundo uma pesquisa [1], 60% dos acidentes são causados por sono ou fadiga, o que mostra a seriedade do assunto. É por isso que, cada vez mais, as empresas e profissionais fazem uso do sensor de fadiga. Sendo assim, o sensor de fadiga é um importante instrumento que atua para identificar os comportamentos de risco e alertar o condutor em tempo real. Os alertas na cabine auxiliam os condutores a manterem o foco na estrada, reduzindo riscos e os conscientizando sobre a importância de uma condução segura.

Quando navegamos na internet, alguns projetos se assemelham ao que será produzido neste trabalho. O artigo [2] é bastante parecido em alguns aspectos, pois o projeto é desenvolvido em uma Raspberry Pi modelo 3. Inclusive, nesse artigo é exposto o fluxograma de funcionamento do algoritmo criado, como exposto na figura 1. As divergências estão relacionadas principalmente com a linguagem de programação a ser utilizada, no caso desse artigo, o python foi o escolhido, enquanto no projeto a ser desenvolvido a linguagem final escolhida será a C++.

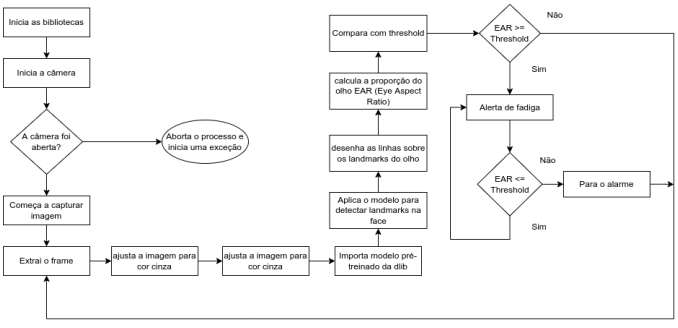

Figura 1. Algoritmo de detecção de fadiga.

Outro projeto que será utilizado como referência está exposto no site sigmoidal, desenvolvido por Carlos Melo. A forma como o autor aborda o algoritmo de detecção de fadiga está mais detalhado do que no artigo [2], citado anteriormente. Sendo assim, esses e outros artigos servirão de guia para o desenvolvimento do protótipo, com algumas alteração para que se encaixe no escopo da disciplina.

A partir do sucesso desse projeto, é possível que esse sensor de fadiga saia do âmbito de protótipo e evolua para um projeto mais robusto, com o objetivo de comercializar o produto.

II. OBJETIVOS

O dispositivo serve, antes de tudo, para proteção. Afinal, ao acompanhar o estado do condutor e emitir alerta quando necessário, preza pela integridade física dos motoristas. A proteção também é válida para o veículo e demais bens patrimoniais do condutor. evitando, assim, gastos desnecessários. Além disso, no ramo de transportes, os dados obtidos pelo sensor pode ajudar as empresas a detectar os perfis dos motoristas e, assim, indicá-los a rotas onde esses profissionais sentem mais fadiga, distração ou sono. Assim, age-se preditivamente na prevenção de acidentes e problemas associados.

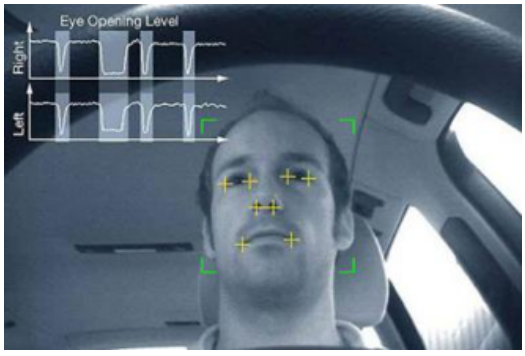

Figura 2. Exemplo abstrato de funcionamento da câmera de fadiga.

Como exposto ilustrativamente na imagem acima 2, o projeto visa avaliar o intervalo entre o level de olho do condutor e a partir disso avaliar se o condutor está com fadiga ou não, caso seja identificada a fadiga, o alarme seria ativado e despertaria o motorista. Além disso, é possível ir além da detecção da fadiga, é possível detectar quando o usuário está mexendo no celular, está bebendo e fumando.

III. REQUISITOS

O sensor de fadiga é um projeto que precisa do processamento disponível na Raspberry Pi, tendo em vista a captura e uso de imagens, as quais serão utilizadas para o

monitoramento e controle do motorista. A priori, para realizar o projeto será utilizado os seguintes equipamentos (passível de mudança durante o decorrer do projeto):

1) Placa Raspberry Pi 3 model B+

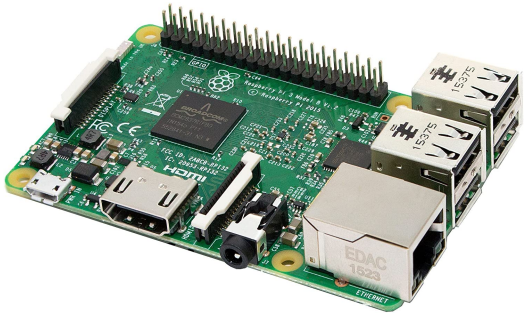

Figura 3. Placa Raspberry Pi que será utilizada no projeto.

2) Câmera

Figura 4. Câmera que será conectada na Raspberry para captar a imagem do condutor.

3) Buzzer (Alarme)

Figura 5. Buzzer que funcionará como alarme.

4) Caixa para armazenar os componentes

Essa caixa será projetada com o intuito de minimizar o espaço utilizado pelos componentes, além de ser uma forma mais bonita de vizualização do projeto.

A execução do protótipo, inicialmente, se dará baseado nas pesquisas feitas com as seguintes referências: [3] [4] [2] [5].

IV. Benefícios

Como exposto nas seções I e II, os benefícios do sensor de fadiga está relacionado com a prevenção de acidentes de trânsito e consequentemente evitar custos materiais e o mais importante salvar vidas. Pensando nisso, no âmbito de transportes, o sensor de fadiga para caminhões não deve ser encarado como uma despesa, mas como um investimento para evitar o pior.

V. Descrição de hardware

O projeto do detector de fadiga não demanda alta complexidade no âmbito do hardware, pois, de acordo com a figura 6, serão utilizados apenas a raspberry pi 3 model B+, a câmera e um buzzer. A maior complexidade está voltada pra parte do software.

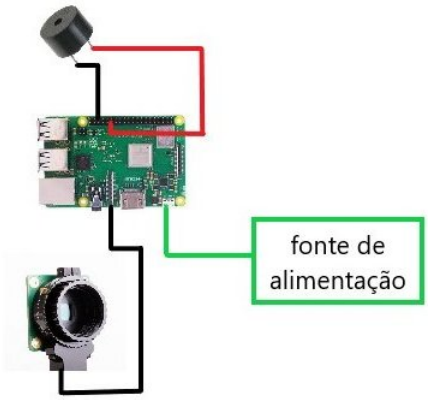

Figura 6. Montagem o projeto.

A raspberry pi é o cérebro do trabalho, responsável por processar todas as informações do projeto, além de gerenciar o funcionamento dos demais periféricos utilizados. A câmera possui a função de captar a imagem da face do condutor para que o sistema na raspberry analise se o motorista está ou não com sonolência, caso seja verdadeiro, será emitido um som pelo buzzer com o intuito de despertar o condutor.

VI. Descrição de software

Como exposto na seção V, a maior complexidade do projeto está na elaboração e execução do código que será implementado na raspberry. Primeiramente, o código foi elaborado na linguagem python devido a sua facilidade e ao grande conteúdo presente nos fóruns da internet. Entretanto, por ser uma linguagem de alto nível, os códigos em python não são rápidos e isso pode acabar afetando o bom funcionamento do projeto. Sendo assim, o código está em C++, visando reparar essa desvantagem do python e otimizar o processamento do sistema.

O algoritmo, por enquanto, foi realizado a partir do fluxograma apresentado na figura 1. Nele é demonstrado a sequência de instruções escritas para serem interpretadas pela raspberry com o objetivo de executar tarefas específicas. A parte de mandar mensagem para o condutor e subir as informações do carro para a nuvem não é relevante para o projeto em si, sendo assim, foi descartada.

VII. Testes

A. Pontos de referências faciais

Para fazer o reconhecimento facial, a imagem é usada como entrada e é utilizada a biblioteca opencv. O objetivo é detectar a estrutura facial no rosto usando métodos para prever as formas. O proceso de reconhecimento facial tem dois passos: o primeiro é localizar o rosto na imagem e depois se detecta as principais estruturas faciais, no caso desse trabalho são utilizados os olhos da pessoa.

O método de detecção de expressões faciais incluído na biblioteca dlib usa um conjunto de expressões faciais já especificados para rotular coordenadas específicas na imagem, manualmente, ao redor do rosto da pessoa. Com os dados da imagem, um conjunto de árvores de regressão estimam as posições das coordenadas usando como referência para isso os pixels da imagem. O detector de expressão facial que tem dentro da biblioteca dlib é treinado e usado para estimar a localização de 68 coordenadas x-y que mapeam as estruturas faciais do rosto da pessoa, como exposto na figura 7.

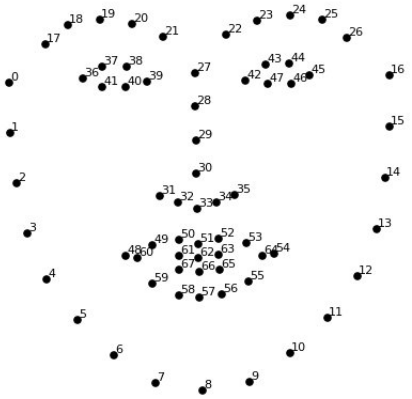

Figura 7. Visualização das 68 coordenadas de referência facial. Fonte: [4]

B. Detecção de sonolência pelo nível do olho

Primeiro é fixado a câmera em frente ao volante e a raspberry pi e o buzzer no lado direito do motorista, próximo a marcha do carro. Se o rosto do condutor for encontrado, é aplicado o reconhecimento facial e é obtida a região dos olhos da pessoa. Com ela pode ser computado a situação que os olhos estão, fechados ou abertos, com a taxa de proporção dos olhos(EAR - eye aspect ratio) e se caso eles estiverem fechados por uma quantidade de tempo considerável, detectando fadiga, o buzzer será acionado para acordar o motorista.

VIII. Resultados

O código atualmente detecta o rosto e desenha os olhos da pessoa através das bibliotecas OpenCV e dlib. Com isso, ele usa os pontos específicos, dados pela dlib, de cada olho e calcula o EAR (eye aspect ratio) de cada olho, que varia de acordo com a situação dos olhos, aberto ou fechado. E os EARs de cada olho vão ser usados para a ativação do buzzer, quando for necessária. Os testes foram feitos calculando

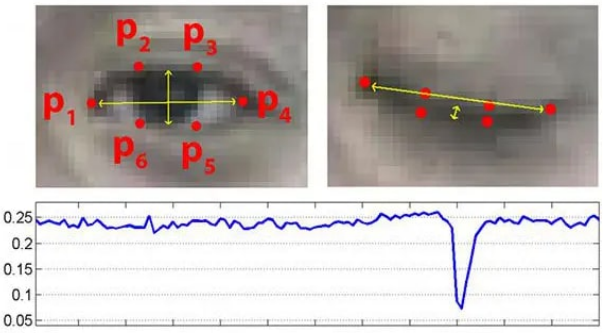

Figura 8. Canto superior esquerdo: Uma visualização de pontos de referência do olho quando o olho está aberto. Superior direito: pontos de referência do olho quando o olho está fechado. Inferior: Plotando a proporção do olho ao longo do tempo. A queda na proporção do olho indica um piscar. Fonte: [4].

somente o EAR do olho direito e foi visto que o EAR acima de 0,25 significa que o olho está aberto e o olho completamente fechado fica na faixa de 0,10 a 0,15.

Também foi testada a possibilidade de uma média móvel para o sistema e foi verificado que não é necessário a sua utilização, já que não agregava benefícios significativos ao projeto, por isso foi descartada a possibilidade.

- Usuário presente em um ambiente com boa iluminação e sem nada que prejudique a visibilidade da sua face. Nota-se que pela imagem 9 não houve problemas de detecção dos pontos.

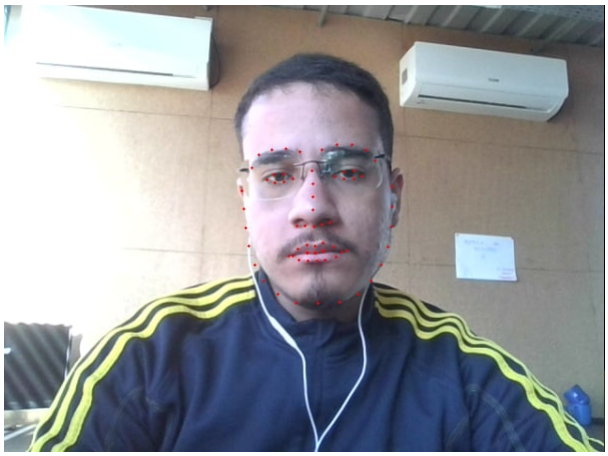

Figura 9. Reconhecimento dos pontos de referências faciais.

- Usuário presente em um ambiente com boa iluminação e com a utilização de máscara.

Atualmente é bastante comum os motoristas utilizarem máscaras enquanto dirigem, principalmente motoristas de aplicativos, como uber. Para isso, foram realizados testes para averiguar se o sistema satisfaria essa situação. Os resultados obtidos comprovam que o sistema atual não é satisfatório para quando o usuário estiver utilizando máscara.

Como demostrado na imagem 10, o sistema conseguiu reconhecer a face do usuário, porém logo em seguida, imagem

11, houve um falha no reconhecimento, comprometendo todo o sistema.

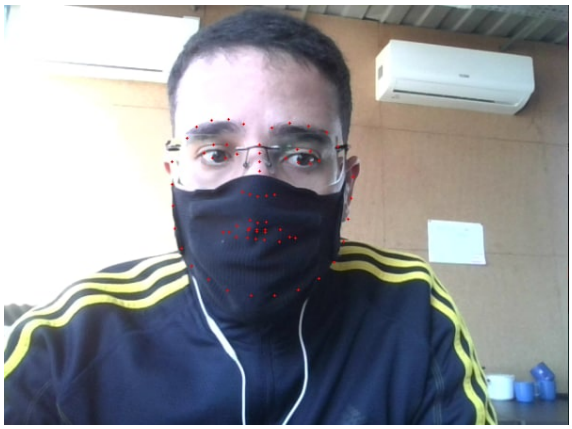

Figura 10. Correto reconhecimento dos pontos de referências faciais com a utilização de máscara.

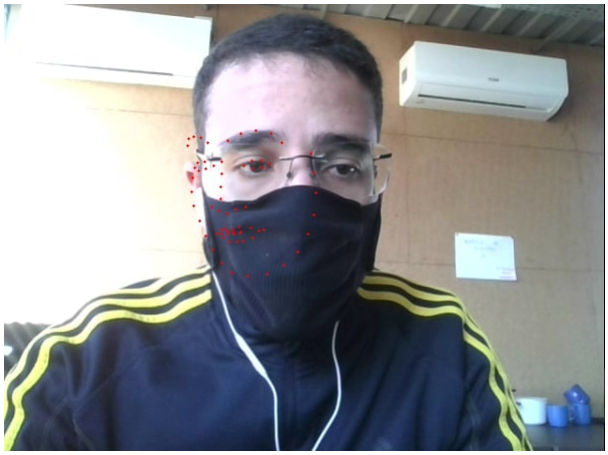

Figura 11. falha no reconhecimento dos pontos de referências faciais com a utilização de máscara.

- Usuário presente em um ambiente com boa iluminação e com a utilização de boné ou chapéu.

Através do teste, imagem 12, é possível afirmar que a utilização de bonés ou chapéus pelos condutores não acarretará em falhas no sistema, sendo assim, o sistema atende a essa condição.

Já na detecção de sonolência pelo nível do olho, os resultados se assemelham bastante com o que foi exposto anteriormente para a detecção de pontos de referências faciais. Dessa forma, os resultados foram satisfatórios, porém com algumas ressalvas, que serão tratadas nos itens abaixo.

- Usuário presente em um ambiente com boa iluminação e sem nada que prejudique a visibilidade de sua face.

Nota-se que pela figura 13 não houveram problemas na detecção e cálculo da proporção dos olhos (EAR - eye aspect ratio). Na figura 14 é detectado a sonolência pois o EAR está abaixo de 0.27, valor definido no código.

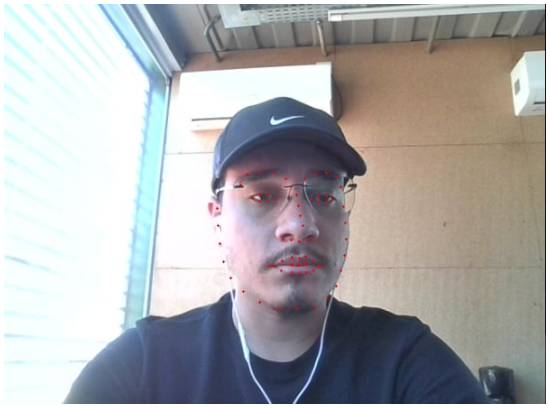

Figura 12. Correto reconhecimento facial com o condutor portando boné.

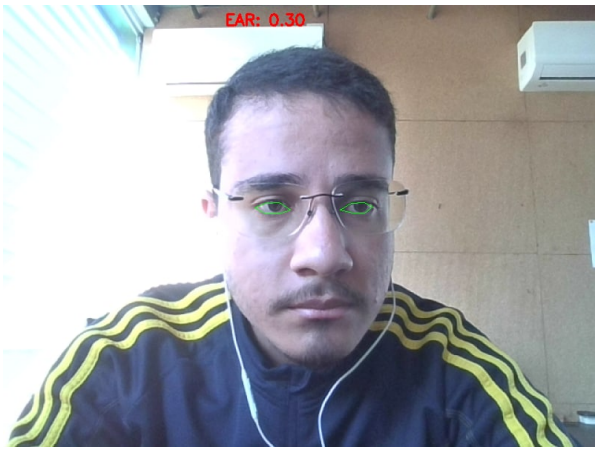

Figura 13. correta detecção de abertura dos olhos.

Figura 14. Correta detecção de sonolência.

- Usuário presente em um ambiente com boa iluminação e com a utilização de máscara.

Assim como na detecção dos pontos de referências faciais, o sistema não funciona de forma adequada, pois o seu correto funcionamento não é de forma constante. De acordo com o posicionamento do usuário, a detecção de sonolência pode ocorrer de forma correta, mas, como é exposto na figura 15, é facilmente possível que ocorra uma falha na detecção, comprometendo o objetivo do projeto.

Figura 15. falha na detecção de sonolência devido ao uso da máscara.

- Usuário presente em um ambiente com boa iluminação e com a utilização de boné ou chapéu

Assim como exposto anteriormente, o boné não atrapalha na detecção dos pontos de referências faciais, sendo assim, o mesmo ocorre na detecção de sonolência. O sistema funciona de forma satisfatória, como exposto na figura 16.

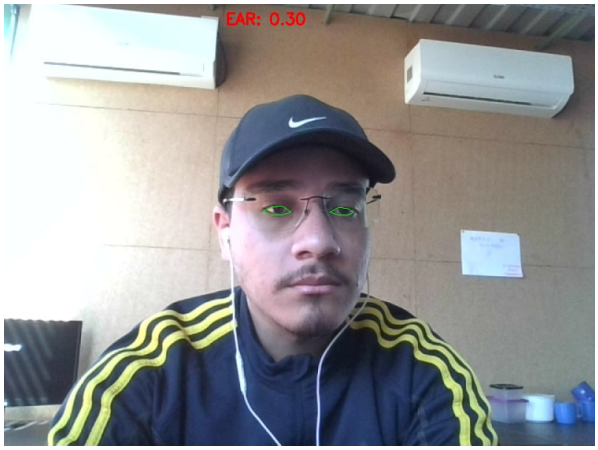

Figura 16. Correta detecção de sonolência com o condutor portando boné.

- Usuário presente em um ambiente com boa iluminação com apenas um olho aberto

Como demonstrado na figura 17, o código da forma que foi elaborado, identifica sonolência se o usuário estiver apenas com um olho aberto, isso pode se tornar um agravante no futuro pois isso não significa sonolência por parte do condutor e sim apenas uma limitação no projeto quanto a forma utilizada para detecção.

Dessa forma, uma alternativa viável é medir o EAR levando em conta somente um olho, pois anteriormente esse cálculo era feito com base nos dois olhos. Assim, mesmo que o usuário feche apenas um olho, o sistema não detectará fadiga, pois o cálculo da proporção do olho agora é feita de forma individual.

Figura 17. Falha na detecção de sonolência com apenas um olho aberto.

REFERÊNCIAS

[1] "Sensor de fadiga : O que É, como funciona e para que serve." Disponível em: https://greenroad.com.br/2020/05/28/sensor-de-fadiga-o-que-e-como-funciona-e-para-que-serve/. Acesso em: 23 de junho 2022.

[2] A. Chellappa, M. S. Reddy, R. Ezhilarasie, S. K. Suguna, and A. Umamakeswari, "Fatigue detection using raspberry pi 3," *International Journal of Engineering & Technology*, vol. 7, no. 2.24, pp. 29–32, 2018.

[3] K. Anjali, A. K. Thampi, A. Vijayaraman, M. F. Francis, N. J. James, and B. K. Rajan, "Real-time nonintrusive monitoring and detection of eye blinking in view of accident prevention due to drowsiness," in *2016 International Conference on Circuit, Power and Computing Technologies (ICCPCT)*. IEEE, 2016, pp. 1–6.

[4] J. Cech and T. Soukupova, "Real-time eye blink detection using facial landmarks," *Cent. Mach. Perception, Dep. Cybern. Fac. Electr. Eng. Czech Tech. Univ. Prague*, pp. 1–8, 2016.

[5] B. R. Ibrahim, F. M. Khalifa, S. R. Zeebaree, N. A. Othman, A. Alkhayyat, R. R. Zebari, and M. A. Sadeeq, "Embedded system for eye blink detection using machine learning technique," in *2021 1st Babylon International Conference on Information Technology and Science (BICITS)*. IEEE, 2021, pp. 58–62.

Códigos Python

```

# importar as bibliotecas
import cv2
import dlib
import time
import imutils
from imutils.video import VideoStream
from imutils import face_utils

# dlib detector
detector = dlib.get_frontal_face_detector()
predictor = dlib.shape_predictor('
    shape_predictor_68_face_landmarks.dat')
vs = VideoStream(src=0, usePiCamera=False,
    resolution=(
    320, 240), framerate=32).start()
time.sleep(2.0)

# video processing pipeline
while True:
    frame = vs.read()
    frame = imutils.resize(frame, width=600)
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    rects = detector(gray, 0)

    for rect in rects:
        shape = predictor(gray, rect)
        shape = face_utils.shape_to_np(shape)
        for (x, y) in shape:
            cv2.circle(frame, (x, y), 1, (0, 0, 255)
    , -1)

        cv2.imshow("Frame", frame)
        key = cv2.waitKey(1) & 0xFF

        if key == ord("q"):
            break

# clean
cv2.destroyAllWindows()
vs.stop()
```

Listing 1. Landmark - Reconhecimento Facial

```

from scipy.spatial import distance as dist
from imutils.video import VideoStream
from imutils import face_utils
from threading import Thread
import numpy as np
import playsound
import imutils
import time
import dlib
import cv2
import matplotlib.pyplot as plt

# definir constantes
ALARM = "alarm.wav"
WEBCAM = 0
EYE_AR_THRESH = 0.27
EYE_AR_CONSEC_FRAMES = 40
COUNTER = 0
ALARM_ON = False

def sound_alarm(path=ALARM):
    # play an alarm sound
    playsound.playsound(ALARM)

def eye_aspect_ratio(eye):
    # compute the euclidean distances between the
    two sets of
    # vertical eye landmarks (x, y)-coordinates
    A = dist.euclidean(eye[1], eye[5])
    B = dist.euclidean(eye[2], eye[4])

    # compute the euclidean distance between the
    horizontal
    # eye landmark (x, y)-coordinates
    C = dist.euclidean(eye[0], eye[3])

    # compute the eye aspect ratio
    ear = (A + B) / (2.0 * C)

    # return the eye aspect ratio
    return ear

# dlib's face detector (HOG-based)
print("[INFO] carregando o preditor de landmark...")
detector = dlib.get_frontal_face_detector()
predictor = dlib.shape_predictor("
    shape_predictor_68_face_landmarks.dat")

# pegar os  ndices  do previsor, para olhos esquerdo
     e direito
(lStart, lEnd) = face_utils.FACIAL_LANDMARKS_IDXS["
    left_eye"]
(rStart, rEnd) = face_utils.FACIAL_LANDMARKS_IDXS["
    right_eye"]

# inicializar v deo
print("[INFO] inicializando streaming de v deo...")
vs = VideoStream(src=WEBCAM, usePiCamera=False,
    resolution=(
    320, 240), framerate=32).start()
time.sleep(1.0)

# loop sobre os frames do v deo
while True:
    frame = vs.read()
    frame = imutils.resize(frame, width=800)
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    # detectar faces (grayscale)
    rects = detector(gray, 0)

    # loop nas detec  es de faces
    for rect in rects:
        shape = predictor(gray, rect)
        shape = face_utils.shape_to_np(shape)

        # extrair coordenadas dos olhos e calcular a
     propor  o de abertura
        leftEye = shape[lStart:lEnd]
        rightEye = shape[rStart:rEnd]
        leftEAR = eye_aspect_ratio(leftEye)
        rightEAR = eye_aspect_ratio(rightEye)

        # ratio m dia para os dois olhos
        ear = (leftEAR + rightEAR) / 2.0

        # convex hull para os olhos
        leftEyeHull = cv2.convexHull(leftEye)
        rightEyeHull = cv2.convexHull(rightEye)
        cv2.drawContours(frame, [leftEyeHull], -1,
    (0, 255, 0), 1)
        cv2.drawContours(frame, [rightEyeHull], -1,
    (0, 255, 0), 1)

        # checar ratio x threshold
        if ear < EYE_AR_THRESH:
            COUNTER += 1

            # dentro dos crit rios, soar o alarme
            if COUNTER >= EYE_AR_CONSEC_FRAMES:
                # ligar alarme
```

```
                if not ALARM_ON:
                    ALARM_ON = True
                    t = Thread(target=sound_alarm)
                    t.deamon = True
                    t.start()

                cv2.putText(frame, "[ALERTA] FADIGA!", (10, 30),
                            cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 255), 2)

        # caso acima do threshold, resetar o contador e desligar o alarme
        else:
            COUNTER = 0
            ALARM_ON = False

            # desenhar a propor  o de abertura dos olhos
        cv2.putText(frame, "EAR: {:.2f}".format(ear), (300, 30),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 255), 2)

    # show frame
    cv2.imshow("Frame", frame)
    key = cv2.waitKey(1) & 0xFF

    # tecla para sair do script "q"
    if key == ord("q"):
        break

# clean
cv2.destroyAllWindows()
vs.stop()
```

Listing 2. Detecção de Sonolência com python

Códigos C++

```

#include<dlib/image_processing/frontal_face_detector.h>
#include<dlib/image_processing.h>
#include<dlib/opencv.h>
#include<opencv2/imgproc.hpp>
#include<opencv2/highgui.hpp>
#include<iostream>

using namespace cv;
using namespace std;
using namespace dlib;

//Desenhar linha
void desenhalinha(cv::Mat &image, full_object_detection landmarks, int start, int end, bool isClosed=false){
    std::vector<cv::Point> points;
    for(int i=start; i<=end; i++){
        points.push_back(cv::Point(landmarks.part(i).x(), landmarks.part(i).y()));
    }
    cv::polylines(image, points, isClosed, cv::Scalar(0, 255, 255), 2, 16);
}

void desenhalinhas(cv::Mat &image, full_object_detection landmarks){
    desenhalinha(image, landmarks, 36, 41, true);  //olho esquerdo
    desenhalinha(image, landmarks, 42, 47, true);  //olho direito
}


//encontra os pontos da face e desenha as linhas sobre os pontos dos olhos
void encontraLandmarks(Mat &frame, frontal_face_detector faceDetector, shape_predictor landmarkDetector, std::vector<dlib::rectangle> &faces, float resizeScale, int skipFrames, int frameCounter){

    //armazenar imagem redimensionada
    Mat smallFrame;

    //redimensiona frame para imagem menor
    resize(frame, smallFrame, Size(), 1.0/resizeScale, 1.0/resizeScale);

    //muda para o formato da dlib
    cv_image<bgr_pixel> dlibImageSmall(smallFrame);
    cv_image<bgr_pixel> dlibImage(frame);

    //detecta faces no intervalo de quadros
    if(frameCounter % skipFrames == 0){
        faces = faceDetector(dlibImageSmall);
    }

    //loop sobre rostos
    for(int i=0; i<faces.size(); i++){

        //dimensione as coordenadas do ret ngulo como fizemos a detec  o de rosto em uma imagem menor redimensionada
        dlib::rectangle rect(int(faces[i].left() * resizeScale),
            int(faces[i].top() * resizeScale),
            int(faces[i].right() * resizeScale),
            int(faces[i].bottom() * resizeScale));

        //detec  o dos landmarks faciais
        full_object_detection faceLandmark = landmarkDetector(dlibImage, rect);

        //desenha as linhas sobre os landmarks
        desenhalinhas(frame, faceLandmark);

        //c lculo das distancia euclidianas
       double P37_41_x = faceLandmark.part(37).x() - faceLandmark.part(41).x();
        double P37_41_y=  faceLandmark.part(37).y() - faceLandmark.part(41).y() ;
        double p37_41_sqrt=sqrt((P37_41_x * P37_41_x) + (P37_41_y * P37_41_y));

        double P38_40_x = faceLandmark.part(38).x() - faceLandmark.part(40).x();
        double P38_40_y = faceLandmark.part(38).y() - faceLandmark.part(40).y();
        double p38_40_sqrt=sqrt((P38_40_x * P38_40_x) + (P38_40_y * P38_40_y));

        double P36_39_x = faceLandmark.part(36).x() - faceLandmark.part(39).x();
        double P36_39_y = faceLandmark.part(36).y() - faceLandmark.part(39).y();
        double p36_39_sqrt=sqrt((P36_39_x * P36_39_x) + (P36_39_y * P36_39_y));

        //Calculo do EAR
        double EAR = (p37_41_sqrt +  p38_40_sqrt)/(2* p36_39_sqrt);

        cout << "EAR value =  " << EAR << endl;
    }
```

```
}

int main(){

    //crie um objeto de captura de v deo para
    mostrar o v deo da webcam
    VideoCapture videoCapture(-1);

    //checar se o opencv conseguiu abrir a camera
    if(!videoCapture.isOpened()){
        cout<<"can not open webcam"<<endl;
        return 0;
    }

    //definir os landmarks do rosto
    frontal_face_detector faceDetector =
    get_frontal_face_detector();

    //define landmark detector
    shape_predictor landmarkDetector;

    //carregar o modelo treinado de landmarks
    deserialize("../dlibAndModel/
    shape_predictor_68_face_landmarks.dat") >>
    landmarkDetector;

    //redimensionar altura
    float resizeHeight = 480;

    //definir skip frames
    int skipFrames = 3;

    //Obter primeiro frame
    Mat frame;
    videoCapture >> frame;

    //redimensiona a escala
    float height = frame.rows;
    float resizeScale = height/resizeHeight;

    //inicia o tickCounter
    double tick = getTickCount();
    int frameCounter=0;

    //cria uma tela para mostrar os frames
    namedWindow("frame", WINDOW_NORMAL);

    //criar variavel que armazena o fps
    double fps = 30.0;

    std::vector<dlib::rectangle> faces;

    while (1){
        if(frameCounter == 0){
            tick = getTickCount();
        }

        // l   o frame
        videoCapture >> frame;

        encontraLandmarks(frame, faceDetector,
    landmarkDetector, faces, resizeScale, skipFrames
    , frameCounter);

        String fpss;
        fpss = to_string(fps);

        //mostra o frame
        imshow("frame", frame);

        //precionar o esc para sair da aplica  o
        char key = waitKey(1);
        if(key == 27){
            break;
        }

        //incrementa o frame counter
        frameCounter++;

        //calcula fps depois de todo 100 frames
        if(frameCounter == 100){
            tick = ((double)getTickCount() - tick)/
    getTickFrequency();
            fps = 100.0/tick;
            frameCounter = 0;
        }
    }

    //release captura de video
    videoCapture.release();

    //fecha todas as janelas abertas
    destroyAllWindows();

    return 0;
}
```

Listing 3. Detecção de Sonolência com c++