# Sistema de detecção de sonolência com Raspberry Pi

Gabriel Teixeira Brasil
*Universidade de Brasília*
*Engenharia Eletrônica - FGA*
Brasília, Brasil
1gabrielbrasil@gmail.com

Leticia Ribeiro de Maccedo
*Universidade de Brasília*
*Engenharia Eletrônica - FGA*
Brasília, Brasil
leticia.rmacedolm@gmail.com

***Resumo*—Este trabalho dedica-se a desenvolver um sistema de detecção de sinais de sonolência em motoristas, através do processador Raspberry Pi. A imagem do rosto do motorista é capturada através da câmera do sistema e analisada para detectar o movimento dos olhos do motorista. Quando o estado de sonolência é identificado, o alarme é acionado para alertar o usuário. Este projeto visa evitar acidentes causados por motoristas que sofrem por fadiga ao volante.**

***Index Terms*—Raspberry Pi, movimento dos olhos, sonolência**

## I. Introdução

O sono é uma função biológica natural, mas, combinado à direção, se torna um dos problemas mais subestimados por motoristas comuns e profissionais. A fadiga afeta a capacidade cognitiva do indivíduo, altera seu humor e diminui o tempo de resposta do condutor. Nesse sentido, uma pesquisa da Universidade Federal de São Paulo (UNIFESP) [1] apontou que, após 19 horas de privação de sono, o desempenho do condutor ao volante é semelhante àquele observado em motoristas com teor alcoólico no sangue de 0,7g/L.

Visando evitar acidentes envolvendo a sonolência dos motoristas, diversos sistemas de alerta foram desenvolvidos pelas principais montadoras. Entre eles estão o *Lane Keeping Sytem* desenvolvido pela ford, que detecta a posição do carro com relação à pista através de uma câmera a fim de detectar mudanças bruscas; o *Attention Assist* da Mercedes-Benz, que analisa o comportamento do condutor buscando perceber caracteristicas de desatenção; além de alguns aplicativos que fazem testes sonoros periódicos e determinam a atenção do motorista através da velocidade de resposta do condutor.

Existem ainda alguns projetos, em fase de prototipagem e avaliação, cuja acurácia científica e prática não foi determinada. PERCLOS, um complexo sistema de detecção baseado em imagem, apresenta performance primossora, mas exigem poder computacional para o processamento dos sinais [2]. Por outro lado, sistemas baseados em eletro-oculograma, como o apresentado em [3], despertam o interesse pela maior simpliciade de processamento, mas dependem de equipamento que pode atrapalhar o motorista.

Este trabalho busca aliar a simplicidade, o conforto do motorista e o processamento de imagens para desenvolver um sistema não invasivo que alerte o motorista de seu estado de fadiga e evite acidentes. Para isso, o sistema captura o video do rosto do motorista, em seguida processa a imagem para possibilitar a identificação do rosto. Dessa forma, o processador raspberry pi é capaz de acompanhar os movimentos dos olhos do usuário, caso não haja movimentos em um período de tempo superior ao programado, o projeto aciona o alarme.

Assim, O sistema aqui proposto é simples, não requer um hardware muito pesado ,tem baixo custo e alta eficiência. Este trabalho foi organizado em três seções: Desenvolvimento, resultados e conclusões. Na parte referente ao desenvolvimento, são delineados a solução proposta, a lógica programada e os materiais utilizados. Em seguida, a seção de resultados se dedica a apresentar os testes realizados para validação dos requisitos de projeto. Finalmente, a conclusão resume o trabalho realizado com um sumário da solução obtida.

## II. Desenvolvimento

A solução proposta apresenta a configuração geral apresentada na figura 1. A câmera utilizada opera a uma taxa de 30 frames por segundo com resolução de 640 por 470 pixels, essa resolução facilita o processamento sem inviabilizar o reconhecimento to rosto. Nesse contexto, o alarme é ativado quando a média móvel de três frames indicar que os olhos do motorista estão fechados.

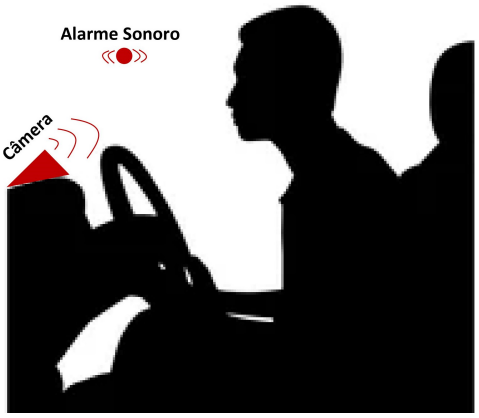


Figura 1. Visão geral da estrutura do projeto.

O processamento de imagem para identificação do rosto foi construído em linguagem C++ baseado nas funções da

biblioteca OpenCV e também em artigos, como: o CHELLAPPA(2018) [4], o do HASHMI(2021) [5] e o MUSALE(2017) [6]. Essas publicações apresentam sistemas de detecção de sonolência com a Raspberry Pi utilizando dois algoritmos principais: o *Haar Cascade*, cujo código é bastante difundido e tem o propósito de detectar face e olhos; e o EAR( Eye Aspect Ration), cuja função é calcular uma estimativa do quanto o olho está aberto por meio da razão entre pontos nas extremidades horizontal e vertical dos olhos.

### *A. Descrição do software*

Após a definição da estrutura do projeto, deteminaram-se os seguintes requisitos de software a serem alcançados pelo sistema: primeiramente coletar imagens do condutor; em seguida, processar os frames para identificar a condição de olhos fechados; finalmente disparar um alarme se não houver abertura dos olhos pelo tempo médio de três frames.

Para isso, foi instalado na placa raspberry Pi o sistema operacional Raspberry Pi OS com Desktop e softwares recomendados, versão 5.4. Além disso foram instaladas as bibliotecas dlib versão 19.18 e OpenCV versão 4.5.2. Também utilizou-se as funções Make e CMake nas versões 4.2 e 3.6 respectivamente.

Nesse contexto, o código criado para alcançar os requisitos mencionados seguiu a sequência lógica descrita na imagem 2. O código completo encontra-se no apêndice A juntamente com as diretrizes utilizadas para a aplicação CMake no apêndice B.

O sistema inicia fazendo uma checagem do funcionamento da câmera e definindo a saída textitGPIO que será utilizada para acionar o alarme. Caso não haja leitura de imagem o código aciona um aviso ao usuário para indicar que a câmera não foi encontrada.

Após a leitura bem sucedida da imagem, altera-se a resolução do frame para facilitar o processamento e converte-se o tipo de dados utilizado para possibilitar o uso da função de reconhecimento da biblioteca *dlib*. Em seguida, através da função $get_frontal_face_detector()$, calcula-se a quantidade de faces na imagem capturada. Caso não sejam identificadas faces na imagem, o programa aguarda o próximo frame chegar para testar novamente.

Uma vez identificados um ou mais rostos, o código seleciona apenas o primeiro identificado e destaca-o com um retângulo. Então é feita uma varredura da matriz obtida e o resultado é comparado ao modelo de topologia facial incluído ao sistema. A partir disso, obtém-se pontos que destacam a posição dos olhos na imagem. Esses pontos foram utilizados no cálculo do EAR (*Razão de aspecto dos olhos*). Esse cálculo relaciona a distância vertical e horizontal de abertura dos olhos e fornece um parâmetro para reconhecimento da posição dos olhos do motorista na imagem. Com isso, definiu-se a condição de que se essa razão de aspecto for menor que o limite de 0,2 os olhos estão se fechando. Na sequência o código realiza a média dos três ultimos frames para identificar quantas vezes essa condição foi identificada. Caso seja observada essa condição no intervalo de três frames o alarme é acionado.

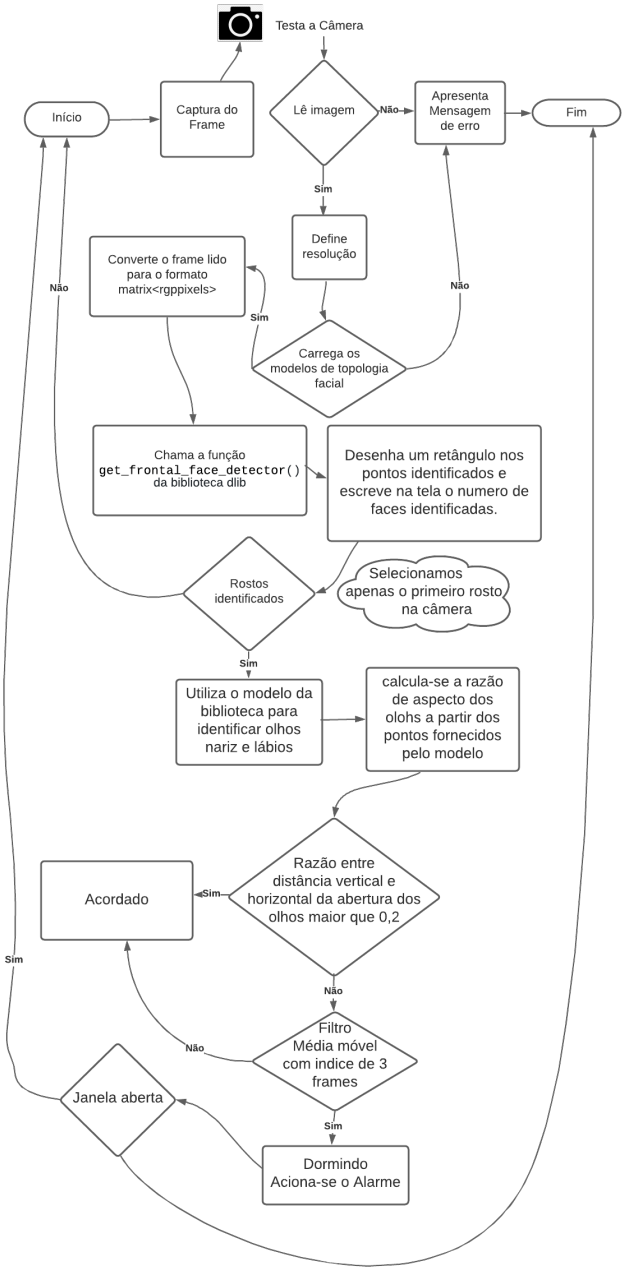


Figura 2. Estrutura lógica do código criado.

### *B. Descrição de Hardware*

Lista de componentes:

- Raspberry Pi 4B;
- Webcam Lehmox modelo LEY-52;
- Buzzer 5V.
- Fonte para Raspberry pi 4 USB 5V 4,2A.
- Cabo USB tipo C ,4A.

Ao invés de utilizar um módulo de câmera próprio para Raspberry foi utilizado uma Webcam HD 1080 conectada a entrada usb, não houve necessidade de instalação de nenhum

software para o reconhecimento da câmera, após conectá-la, a Raspberry reconheceu automaticamente a câmera.

A figura 3 descreve o esquemático do circuito, que conta com a Raspberry Pi 4B com um processador Quad-core Cortex-A72 64 bit SoC de 1,5 GHz, 8 Gb de memória Ram , interface para display e para câmera,Wifi de 2,4 GHz, Bluetooth 5.0 , 2 portas USB 2.0, 2 portas USB 3.0,GPIO com 40 pinos, conectada via USB a Webcam Lehmox LEY-52 HD 1080P para capturar a imagem do condutor e conectada a um Buzzer de 5 Volts pelo GPIO , que é acionado se no código do projeto é detectado pelo processamento da imagem que o motorista está dormindo. Para alimentação do circuito foi utilizada uma fonte para celular compatível com a Raspberry de 5 Volts e 4,2 Ampères e um cabo USB tipo C que suporta 4 Ampères.

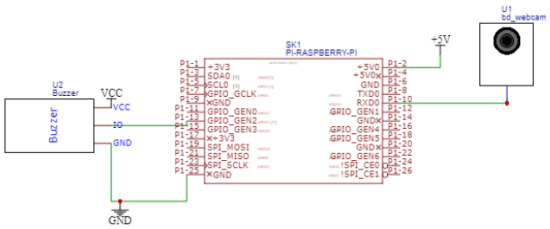


Figura 3. Desenho eletrônico do circuito.

## III. Resultados

Os resultados obtidos nesta seção foram desenvolvidos a partir do comportameto do sistema em três situações diferentes: ambiente com luminosidade controlada; sistema montado dentro de um veículo e testado durante os periodos noturno e diurno. Considerou-se também, em ambos os casos, a situação em que o motorista esteja usando máscara facial. Cada teste foi feito com 5400 frames, a uma taxa de 30 frames por segundo, e foram simulados 15 vezes o estado de sonolência por um homem e uma mulher.

A figura 4 abaixo descreve o posicionamento do sistema no veículo para realização dos testes. A câmera, destacada no lado 'a' da imagem, foi posicionada para focar no rosto do motorista e pode ser ajustada de acordo com o porte físico do usuário.

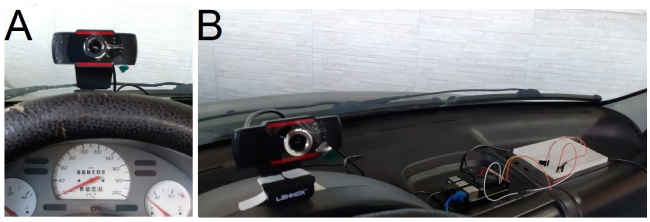


Figura 4. **A.** Posicionamento da câmera nos testes; **B.** disposição do sistema completo no veículo (https://www.youtube.com/watch?v=NNRwEN6Vmmc)

A tabela I a seguir descreve os resultados obtidos nos experimentos dentro do veículo durante o dia. Comparando-se os resultados obtidos sem máscara e aqueles com o uso da máscara observa-se uma robustez do sistema quanto a identificação do rosto nessas condições.

Tabela I
TESTES DENTRO DO VEÍCULO DURANTE O DIA. FONTE: AUTORES.

| | **Detectou sonolência** | **$n^o$ de testes** |
|---|---|---|
| Com máscara | 73,3% | 15 |
| Sem máscara | 80% | 15 |

A figura 5 abaixo mostra o reconhecimento facial em cada um dos casos descritos.


Figura 5. Reconhecimento facial **a.** Estado acordado com máscara; **b.** Estado dormindo com máscara; **c.** Estado acordado sem máscara; **d.** Estado dormindo sem máscara. Fonte: Autores (https://www.youtube.com/watch?v=dZpvxjU484k)

O mesmo procedimento foi seguido para fazer testes noturnos. Nesse ponto, foi utilizada apenas a iluminação pública e dos faróis. Observou-se uma diminuição expressiva do número de testes bem-sucedidos na detecção de sonolência. A tabela II resume os resultados obtidos nessa situação.

Tabela II
TESTES DENTRO DO VEÍCULO DURANTE A NOITE. FONTE: AUTORES.

| | **Detectou sonolência** | **$n^o$ de testes** |
|---|---|---|
| Com máscara | 60% | 15 |
| Sem máscara | 66,66% | 15 |

A fim de determinar a influência da luminosidade do ambiente, os testes foram reproduzidos em uma sala com luminosidade controlada. Com isso, observou-se um aumento de 30%, em média, no número de detecções corretas do sistema. A tabela III apresenta os resultados obtidos.

## IV. Conclusão

## Referências


[1] FAPESP. (2007) A vigília necessária. [Online]. Available: https://revistapesquisa.fapesp.br/medicina-vigilia-necessaria/

[2] T. Horberry, L. Hartley, G. Krueger, and N. Mabbott, "Review of fatigue detection and prediction," *National Road Transport Commission*.

Tabela III
TESTES COM CONTROLE DE LUMINOSIDADE SOBRE O ROSTO DO USÁRIO. FONTE: AUTORES.

| | **Detectou sonolência** | $n^o$ **de testes** |
|---|---|---|
| Com máscara | 93,33% | 15 |
| Sem máscara | 100% | 15 |
| Com óculos | 85,71% | 35 |
| Sem óculos | 94,28% | 35 |


[3] T. C. Chieh, M. M. Mustafa, A. Hussain, S. F. Hendi, and B. Y. Majlis, "Development ofvehicle driver drowsiness detection system using electrooculogram (eog)," *1st International Conference on Computers, Communications, Signal Processing with Special Track on Biomedical Engineering*.

[4] a. o. A. Chellappa., "Fatigue detection using raspberry pi 3," *International Journal of Engineering Technology*, no. 10.14419/ijet.v7i2.24.11993, 2018.

[5] M. Hashmi *et al.*, "Drowsiness detection system using raspberry pi and opencv," *Raj J.S. (eds) International Conference on Mobile Computing and Sustainable Informatics. ICMCSI 2020. EAI/Springer Innovations in Communication and Computing*, 2021.

[6] T. MUSALE, "Driver drowsiness detection technique using raspberry pi," *International Journal Of Development Research*, pp. 11 499–11 503, Feb. 2017.


# V. APÊNDICES

## *A. Código para a captura de imagens e detecção de sonolência*

```
#include <dlib/opencv.h>
#include <opencv2/highgui/highgui.hpp>
#include <dlib/image_processing/frontal_face_detector.h>
#include <dlib/image_processing/render_face_detections.h>
#include <dlib/image_processing.h>
#include <dlib/gui_widgets.h>
#include <dlib/image_io.h>


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


image_window win;
shape_predictor sp;
std::vector<cv::Point> righteye;
std::vector<cv::Point> lefteye;
char c;
cv::Point p;
int const fps = 30;

double compute_EAR(std::vector<cv::Point> vec)
{

    double a = cv::norm(cv::Mat(vec[1]),
            cv::Mat(vec[5]));
    double b = cv::norm(cv::Mat(vec[2]),
            cv::Mat(vec[4]));
    double c = cv::norm(cv::Mat(vec[0]),
            cv::Mat(vec[3]));
    //compute EAR
    double ear = (a + b) / (2.0 * c);
    return ear;
}

int main()
{
    try {
        cv::VideoCapture cap(0);

        if (!cap.isOpened()) {
            cerr << "Não foi possível
                encontrar a câmera" << endl;
            return 1;
        }
        //Definindo a Resolução
        cap.set(CV_CAP_PROP_FRAME_WIDTH, 640);
        cap.set(CV_CAP_PROP_FRAME_HEIGHT, 480);

        //Definindo saída
        system("echo 4 > /sys/class/gpio/export");
        system("echo out > /sys/class/
                gpio/gpio4/direction");
        system("echo 0 > /sys/class/gpio/gpio4/value");
        frontal_face_detector detector =
        get_frontal_face_detector();

        deserialize("../../landmarks/shape_predi
            ctor_68_face_landmarks.dat") >> sp;

        while (!win.is_closed()) {
            cv::Mat temp;
            if (!cap.read(temp)) {
                break;
            }

            cv_image<bgr_pixel> cimg(temp);
            full_object_detection shape;

            // Detecta faces
            std::vector<rectangle> faces
                = detector(cimg);
            cout << "Quantidade de
                    rostos na imagem: "
            <<faces.size() << endl;

            win.clear_overlay();
            win.set_image(cimg);
```

```
            // Encontra posição do rosto
                if (faces.size() > 0) {

                shape = sp(cimg, faces[0]);
                for (int b = 36; b < 42; ++b) {
                    p.x = shape.part(b).x();
                    p.y = shape.part(b).y();
                    lefteye.push_back(p);
                }
                for (int b = 42; b < 48; ++b) {
                    p.x = shape.part(b).x();
                    p.y = shape.part(b).y();
                    righteye.push_back(p);
                }
     //Calcula a razão de aspecto dos olhos
                double right_ear
                    = compute_EAR(righteye);
                double left_ear
                    = compute_EAR(lefteye);


                if ((right_ear + left_ear) / 2 < 0.2){
                           win.add_overlay(
                               dlib::image_window
                               ::overlay_rect(
                                   faces[0], rgb_pixel(
                                   255, 255, 255),
                                   "Dormindo"));
                          system("echo 1 >/sys/class/
                                  gpio/gpio4/value");
                }else{
                           win.add_overlay(
                           dlib::image_window::
                           overlay_rect(
                           faces[0], rgb_pixel(
                           255, 255, 255), "Acordado"));
                        system("echo 0> /sys/class/
                                gpio/gpio4/value");
                }
                righteye.clear();
                lefteye.clear();
                win.add_overlay(
                render_face_detections(shape));
                c = (char)waitKey(1000/fps);
                if (c == 27)
                    break;
            }
        }
        system("echo 4 > /sys/class/gpio/unexport");
    }
    catch (serialization_error& e) {
        cout << "Houve um erro na compilação
            do modelo de características
            faciais." << endl;
        cout << "Você consegue o arquivo
            completo através da  URL: " << endl;
        cout << "
        http://dlib.net/files/
        shape_predictor_68_face_landmarks.dat.bz2"
        << endl;
        cout << endl
         << e.what() << endl;
    }
    catch (exception& e) {
        cout << e.what() << endl;
    }
}
```

### B. Código CMake utilizado

```
cmake_minimum_required(VERSION 2.8)
project( CriaImagem )
include(FetchContent)
      FetchContent_Declare(dlib
      GIT_REPOSITORY https://github.com/davisking/dlib.git
      GIT_TAG        v19.18)
FetchContent_MakeAvailable(dlib)
find_package( OpenCV REQUIRED )
include_directories( ${OpenCV_INCLUDE_DIRS} )
add_executable( CriaImagem CriaImagem.cpp )
target_link_libraries(CriaImagem dlib::dlib ${OpenCV_LIBS} )
```