UNIVERSIDADE FEDERAL DO PARANÁ

FERNANDO ECKHARDT VALLE

DROWSILY LITE: DETECTOR DE SONOLÊNCIA

CURITIBA

2021

FERNANDO ECKHARDT VALLE

DROWSILY LITE: DETECTOR DE SONOLÊNCIA

TCC apresentado ao curso de Pós-Graduação em Inteligência Artificial Aplicada, Setor de Educação Profissional e Tecnológica, Universidade Federal do Paraná, como requisito parcial à obtenção do título de Especialista em Inteligência Artificial Aplicada.

Orientador: Prof. Dr. Alexander Kutzke

CURITIBA

2021

TERMO DE APROVAÇÃO

Os membros da Banca Examinadora designada pelo Colegiado do Programa de Pós-Graduação INTELIGÊNCIA ARTIFICIAL APLICADA da Universidade Federal do Paraná foram convocados para realizar a arguição da Monografia de Especialização de **FERNANDO ECKHARDT VALLE** intitulada: **DROWSILY LITE: DETECTOR DE SONOLENCIA**, que após terem inquirido o aluno e realizada a avaliação do trabalho, são de parecer pela sua __Aprovação_____ no rito de defesa.

A outorga do título de especialista está sujeita à homologação pelo colegiado, ao atendimento de todas as indicações e correções solicitadas pela banca e ao pleno atendimento das demandas regimentais do Programa de Pós-Graduação.

Curitiba, 28 de Outubro de 2021.

ALEXANDER ROBERT KUTZKE
Presidente da Banca Examinadora

RAZER ANTHOM NIZER ROJAS MONTAÑO
Avaliador Interno (UNIVERSIDADE FEDERAL DO PARANÁ)

AGRADECIMENTOS

Agradeço especialmente a meu orientador Alexander Kutzke, por dividir por longo período, nesses tempos de trevas, as agruras para se produzir esse trabalho.

“It’s been a long time
since I rock and rolled“
(Page, Plant, Jones, Bonham; 1972)

RESUMO

Um dos fatores que podem levar a acidentes de trânsito é o cansaço do motorista atrás do volante. Independente dos fatores que levam a esse estado, uma das caraterísticas que demonstram o cansaço de uma pessoa é a sonolência e uma característica marcante da sonolência são os olhos se fechando. Este trabalho apresenta uma forma de monitorar os olhos do motorista utilizando técnicas de visão computacional e inteligência artificial. Através desse monitoramento é possível tomar ações que vão desde tentar acordar o motorista até enviar a localização de onde o incidente ocorreu.

Palavras-chaves: Monitoramento de motorista. Sonolência. Inteligência artificial. Visão computacional

ABSTRACT

One of the factors that can lead to traffic accidents is the tiredness of the driver behind the wheel. Regardless of the factors that lead to this state, one of the characteristics that demonstrate the tiredness of a person is drowsiness and a hallmark of drowsiness is the eyes closing. This work presents a way to monitor the driver's eyes using computer vision and artificial intelligence techniques. Through this monitoring, it is possible to take actions that range from trying to wake up the driver to sending the location where the incident occurred.

Key-words: Driver monitoring. Drowsiness. Artificial intelligence. Computer vision

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

EAR	Eye Aspect Ratio
FPS	Frames per second
ARM	Advanced RISC Machine

SUMÁRIO

1 INTRODUÇÃO

O aumento do número de acidentes de trânsito que ocorrem devido a diminuição do nível de atenção do motorista é um dos grandes problemas enfrentados no trânsito atual. De acordo com a ABRAMET (Associação Brasileira de Medicina de Tráfego) as principais causas de acidentes em rodovias no Brasil é a fadiga (18%) e o sono (42%), totalizando mais da metade dos acidentes (JUNIOR, 2021).

Muitas pesquisas já foram realizadas numa tentativa de desenvolver sistemas de monitoramento para motoristas usando diferentes técnicas. Os melhores métodos de detecção do estado de vigilância do motorista são baseados em detecção de fenômenos fisiológicos como monitoramento de ondas cerebrais, frequência cardíaca, taxa de pulsação e de respiração (SARADADEVI; BAJAJ, 2008). Porém esses métodos são intrusivos, necessitando colar eletrodos no corpo do motorista para realizar o monitoramento, o que obviamente pode causar um desconforto desnecessário durante a condução do veículo.

Pessoas cansadas demostram alguns comportamentos visuais característicos. O estado de sonolência tem como uma das características mais marcante os olhos sendo gradativamente fechados. A visão computacional, aliada a técnicas de *machine learning*, pode entrar então como uma maneira de detectar o estado de vigilância de um motorista, pensando agora especificamente em um estado de sonolência, de modo não intrusivo.

O presente trabalho tem como objetivo conseguir identificar se uma pessoa encontra-se em estado de sonolência utilizando inteligência artificial e visão computacional.

1.1 JUSTIFICATIVA

A indústria automobilística sempre prezou pela segurança de motoristas e passageiros. Mesmo construindo veículos cada vez mais seguros, ainda não é possível garantir 100 por cento de segurança ao andar em um veículo automotivo. A computação embarcada consegue hoje auxiliar o motorista em situações de risco como alertar uma possível colisão, reduzir automaticamente a velocidade em situações de perigo, indicar com precisão a autonomia do veículo e muitas outras ajudas, sempre prezando pelo bem estar das pessoas que estão em seu interior.

A inteligência artificial também já entrou nesse mercado, tanto que hoje, mesmo que muitas vezes em fases experimentais, já existem veículos controlados totalmente pelo computador com auxilio de uma inteligência artificial, como modelos experimentais

da Tesla (SIDDIQUI, 2019).

Além disso a inteligência artificial também pode contribuir para o ambiente automotivo não apenas pensando no veículo em si, mas também no provável ator mais importante desse cenário: o motorista.

Pensando nisso, no motorista, e como a inteligência artificial poderia agir diretamente sobre ele, desenvolveu-se a proposta desse trabalho: criar uma aplicação que através de inteligência artificial consiga dizer se o motorista está entrando em um estado de sonolência que possa causar algum acidente.

1.2 OBJETIVO GERAL

Desenvolver uma aplicação que seja capaz de analisar se o usuário está em um estado de sonolência (começando a dormir) através da sua captura em vídeo.

1.3 OBJETIVOS ESPECÍFICOS

- Analisar através do uso de inteligência artificial o comportamento dos olhos do usuários e com isso conseguir dizer se o usuário pode estar em estado de sonolência;
- Se a aplicação detectar que o usuário está em estado de sonolência executar as seguintes ações: Disparar um alarme na tentativa de acordar o usuário e enviar uma mensagem, via email, para algum responsável, dizendo o que aconteceu e a localização do evento.

1.4 ORGANIZAÇÃO DO TRABALHO

Além do presente capítulo, este documento apresenta os seguintes conteúdos: o Capítulo 2 embasa teoricamente o trabalho, analisando tecnologias utilizadas, algumas aplicações relacionadas e também explica o funcionamento da aplicação.

O Capítulo 3 detalha os materiais e métodos utilizados, além de expor a cronograma do projeto.

O Capítulo 4 apresenta os resultados obtidos através de testes feitos com a aplicação.

O Capítulo 5 faz a conclusão do trabalho, além de algumas recomendações para trabalhos futuros.

Os apêndices contêm os diagramas de classe, casos de uso e de atividade, além do quadro com as especificações de casos de uso.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulos são abordadas algumas tecnologias relacionadas ao escopo deste trabalho, desde conceitos mais genéricos como machine learning, processamento de imagem e visão computacional, até bibliotecas que auxiliam, e muito, nesses conceitos, como o OpenCV. Além disso também é detalhado o funcionamento da aplicação.

2.1 PROCESSAMENTO DE IMAGEM

Processamento de imagens é um processo onde a entrada do sistema é uma imagem e a saída é um conjunto de valores numéricos, que podem ou não compor uma outra imagem (MARENGONI; STRINGHINI, 2010).

Uma das maneiras mais tradicionais de utilização de processamento de imagem na visão computacional é no tratamento de imagens a fim de retirar principalmente ruídos que venham a aparecer provenientes do processo de aquisição dessa imagem. Por exemplo, na FIGURA 1 é possível verificar à esquerda a imagem original com ruído, e à direita, após passar por um processo de filtragem, é possível verificar a exclusão do ruído mais grosseiro e dessa forma ter uma melhor definição da composição da imagem.

FIGURA 1 – Imagem com ruído e imagem sem ruído

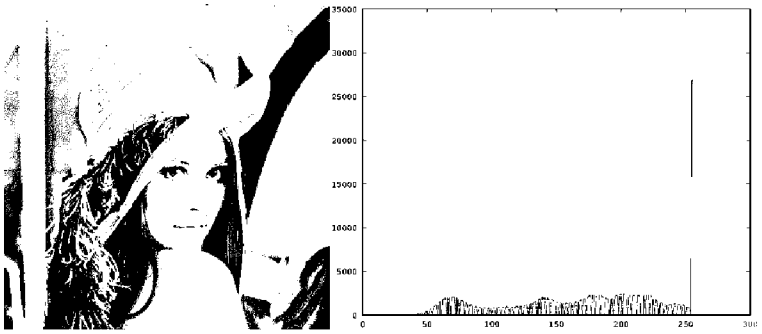
Fonte: (MARENGONI; STRINGHINI, 2010)

Além da remoção de ruído para melhorar a nitidez de uma imagem, filtros também podem ser usados para outros tipos de otimização, tal como detecção de bordas, aumento de contraste para melhorar a distinção de objetos na imagem e identificação de contornos.

Outra técnica interessante de processamento de imagem é o negativo da imagem, muito útil em situações onde a imagem original é escura. Com esse tipo de processamento, os objetos brilhantes, mas fracos, passarão a aparecer como objetos escuros contra um fundo claro, tornando a visualização mais intuitiva.

Por fim, vale citar também o histograma. Histograma é uma representação gráfica de um conjunto de dados. Por exemplo, na FIGURA 2 temos uma imagem e ao lado o histograma que representa o nível de brilho da imagem. É possível observar uma grande quantidade de pixels com valores de brilho muito altas (255), o que acarreta em uma imagem muito estourada, ou seja, sem o contraste ideal, puxando tudo para uma tonalidade mais pálida.

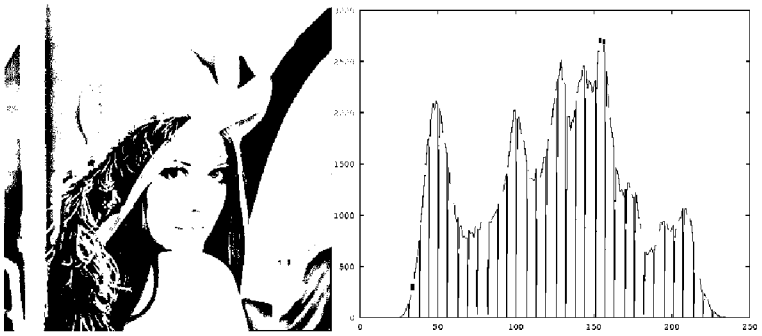
FIGURA 2 – Imagem com brilho alto



Fonte: (JO, 2015)

Já na FIGURA 3, após a imagem passar por uma equalização do histograma, é possível perceber uma distribuição muito melhor dos *pixels* em relação ao brilho, gerando assim uma imagem muito melhor definida e fácil de se observar.

FIGURA 3 – Imagem com histograma equilibrado



Fonte: (JO, 2015)

2.2 VISÃO COMPUTACIONAL

A visão computacional tenta replicar a visão humana se utilizando de recursos de software e de hardware. Essa tentativa de replicação da visão humana pode ser explicado pelo seu principio básico de funcionamento: uma imagem como parâmetro de entrada e um conjunto de valores numéricos como saída, porém a saída é uma interpretação da imagem como um todo, ou parcialmente (MARENGONI; STRINGHINI, 2010). Esses valores numéricos, que podem ser, por exemplo, vetores e matrizes, são os dados de onde é possível, computacionalmente, sua manipulação para resolver algum problema, como por exemplo o reconhecimento de um objeto.

Numa tentativa de simulação da visão natural, a visão computacional tem um embasamento em estudos de algoritmos que buscam compreender e analisar conceitos da visão humana nas áreas de biologia, óptica, álgebra, geometria, estatística, etc.

Na FIGURA 4 existem dois exemplos clássicos de visão computacional. Na imagem a esquerda pode-se verificar que é feito o reconhecimento de bicicletas, e a direita é feito o reconhecimento de pessoas. Essas duas situações ilustram bem o conceito de visão computacional, através de um recurso de hardware (a câmera que capturou essas imagens), um software faz a análise e através de seu processamento consegue reconhecer objetos (tal qual a visão humana, que de uma maneira bem sintética, temos os olhos mandando as informações do que é visto para o cérebro, que reconhece tudo o que ocorre na observação), bastando então apenas tomar alguma ação com esse reconhecimento, que poderia por exemplo simplesmente fazer uma contagem dos objetos reconhecidos ou verificar modelo e marca (para o caso das bicicletas) ou ainda tipo de roupa ou tamanho do cabelo (para o caso do reconhecimento de pessoas).

FIGURA 4 – Dois exemplos de visão computacional usada para reconhecer objetos.

Fonte: (FARIA, 2014)

Atualmente existem algumas ferramentas que auxiliam no desenvolvimento de aplicação que se utilizem de visão computacional. Essas ferramentas conseguem fazer grande parte do trabalho, como processar a imagem, reconhecer padrões e etc.

O OpenCV é uma delas, na próxima seção uma visão geral dessa ferramenta.

2.2.1 OPENCV

O OpenCV (*Open Source Computer Vision Library*), é uma biblioteca multiplataforma, totalmente livre ao uso acadêmico e comercial, inicialmente desenvolvida pela Intel, que tem como objetivo tornar a visão computacional menos complexa e mais acessível para desenvolvedores. O OpenCV possui módulos de processamento de imagens e vídeos, estrutura de dados, álgebra linear, interface gráfica básica de usuário, controle de dispositivos de entrada como mouse e teclado além de centenas de algoritmos de visão computacional como filtros de imagem, calibração de câmera e reconhecimento de objetos (SOBRAL, 2013).

OpenCV foi desenvolvida utilizando as linguagens de programação C e C++, porém tem suporte para diversas outras linguagens como Java, Python e Ruby. Deste modo então possível desenvolver aplicações utilizando a biblioteca em diversas plataformas diferentes, como linux, windows, android e ios.

2.3 MACHINE LEARNING

Machine Learning ou aprendizado de máquina é a tecnologia onde os computadores tem a capacidade de aprender de acordo com respostas esperadas por meio de associação de diferentes dados, que podem ser imagens, números e tudo o mais que esta tecnologia possa identificar.

De acordo com Arthur Samuel (1959), o aprendizado de máquina é o "campo de estudo que dá aos computadores a habilidade de aprender sem serem explicitamente programados" (RUSSELL; NORVIG, 2011). Além disso, o aprendizado de máquina explora a construção de algoritmos que podem aprender com seus próprios erros e fazer previsões sobre os dados através das seguintes abordagens de aprendizado:

- Aprendizado supervisionado: Ao computador são apresentados exemplos de entrada e saída de dados, fornecidas por uma espécia de 'professor' tendo como objetivo aprender uma regra regal que ira mapear a essas entradas e essas saídas;
- Aprendizado não supervisionado: Nenhum tipo de classificação é dado ao algoritmo de aprendizado, ele mesmo irá encontrar alguma estrutura nas entrada fornecidas. Este aprendizado pode ser um objetivo em si mesmo (que seria descobrir novos padrões nos dados) ou um meio para atingir um fim;
- Aprendizado por reforço: neste método, o computador é estimulado a aprender com base em tentativas e erros. O processo é otimizado por meio da prática direta.

Pode se ainda fornecer ao programa *feedback* quanto premiações e punições. Um clássico exemplo são algoritmos para aprender a jogar um jogo, como o xadrez;

2.4 DESENVOLVIMENTO ÁGIL

Metodologias para desenvolvimento ágil de software começaram a surgir na década de 1990 como resposta às necessidades de agilizar a entrega de softwares, quebrando paradigma da modelagem estruturada de sistemas e a produção antecipada de todos os artefatos e modelagens, que tornavam o processo mais demorado. O documento que inaugurou essa tendência foi o Manifesto Ágil, assinado em 2001 e que continha as seguintes premissas (BECK et al., 2014):

- Valorizar indivíduos e suas interação mais do que processos e ferramentas;
- Valorizar software em funcionamento mais do que documentação abrangente;
- Colaboração com o cliente mais do que negociação do contrato;
- Responder às mudanças mais do que seguir um plano.

Essas premissas são opostas ao desenvolvimento tradicional de software, que tenta seguir um plano traçado antes do inicio do desenvolvimento do software, que produz uma série de artefatos e documentos sobre o sistema, que está atrelado à ferramentas e processos engessados e que não tem uma relação de proximidade com o cliente.

Um dos principais expoentes da metodologia de desenvolvimento ágil é o *framework SCRUM*, que tem como principais características a adoção de valores, princípios e práticas que podem ser adaptados às necessidades da organização (VIREIRA, 2014). A base do *SCRUM* está fundamentada em papéis, atividades e artefatos. Os papéis fundamentais são:

- *Product Owner*: é o responsável pelo produto, quais funcionalidades serão desenvolvidas e em qual ordem. Deve estar em constante contato com o time de desenvolvimento e com o *Scrum master*, sendo ponto central do processo de comunicação. Para Vieira (2014), é o papel responsável pelo "sucesso global da solução";
- *Scrum Master*: é o responsável por manter os valores, princípios e práticas do *SCRUM* entre os membros envolvidos. Tem também um papel de facilitador na resolução de problemas, previnindo interferências externas e removendo barreiras que podem comprometer a produtividade;

- Time *scrum*: o time de desenvolvimento *SCRUM* é auto-organizado e multidisciplinar. As pessoas da equipe de desenvolvimento são responsáveis pela concepção, construção e testes do produto. São equipes pequenas eque devem possuir todas as habilidades necessárias para produzir, com qualidade, o software desenvolvido (VIREIRA, 2014).

Algumas das atividades e artefatos do *framework* são:

- *Product Backlog*: é o documento que contém o que será desenvolvido e qual a prioridade dos itens previstos. É um documento que está em constante evolução e pode ser alterado à medida em que o software é construído;
- *Sprint*: são ciclos de trabalhos de duração fixa em que as funcionalidades definidas no *product backlog*. Geralmente tem duração fixa, não ultrapassando um mês;
- *Sprint Planning*: antes da execução do *Sprint* propriamente dito, o *Product Owner*, o time de desenvolviment e o *Scrum Master* definem os objetivos do próximo ciclo ou iteração e quais itens do *Product Backlog* serão implementados;
- *Daily Scrum*: também chamada de *Stend-up meeting*, é uma reunião de curta duração (até 15 minutos) feita diariamente entre os membros da equipe de desenvolvimento para alinhar as atividades desenvolvidas ao objetivo da iteração e compartilhar dificuldades e impedimentos para sua realização;
- *Sprint Review, Sprint Retrospective*: realizadas após a execução de um *Sprint*, tem como objetivo apresentar as funcionalidades desenvolvidas e verificar a necessidade de adaptações no produto ou no processo de trabalho;

Cohn elenca seis motivos para que equipes de desenvolvimento passem a usar a metodologia SCRUM (COHN, 2011, p. 33):

- Maior produtividade e menores custos;
- Maior engajamento e satisfação no trabalho por parte dos funcionários;
- Time-to-market mais rápido;
- Maior qualidade;
- Maior satisfação dos *stakeholders*;
- O que estamos fazendo não funciona mais.

Apesar de reconhecer a dificuldade de mensurar a produtividade de programadores, o autor traz dados de pesquisas que mostram aumento de produtividade na adoção de métodos ágeis, tendo como consequência uma redução dos custos: "Segundo a lógica, se as pessoas forem produtivas, os custos serão menores"(COHN, 2011, p. 34). Um dos fatores para o aumento de produtividade, segundo o autor, está diretamente ligado à maior satisfação dos funcionários com o trabalho que estão realizando, aliado ao fato de que com a adoção da metodologia ágil eles passaram a fazer menos horas-extras (COHN, 2011, p. 36).

Outra vantagem é que diferente dos métodos tradicionais, onde todo o planejamento é realizado antes do início da construção do software, métodos ágeis "tendem a lançar produtos mais rapidamente"(COHN, 2011, p. 36). Isso se deve a dois fatores: a maior produtividade da equipe leva a um desenvolvimento de funcionalidades mais rápido e as equipes ágeis costumam entregar versões incrementais do software.

Para Cohn, a maior qualidade da metodologia ágil está relacionada ao ritmo de trabalho das equipes ágeis, que costumam produzir consistentemente trabalho de maior qualidade, evitando que erros produzidos em algum momento do desenvolvimento voltem a ser tornar um problema em momentos posteriores. Além disso, metodologias ágeis empregam recursos como refatoração, programação em pares e testes automatizados.

Com os elementos citados acima, Cohn acha natural que os demais envolvidos no processo também apresentem maior satisfação, alertando para o fato da metodologia ágil ainda possuir um outro diferencial: a metodologia ágil permite que sejam feitas alterações de requisitos e prioridades (COHN, 2011, p. 38).

Por fim, o autor reconhece a inadequação de metodologias de desenvolvimento tradicionais, alertando para o fato de que "quando um processo que funcionava no passado deixa de funcionar, uma tendência comum é que ele continue sendo executado"(COHN, 2011, p. 39).

2.5 SOFTWARE DE APOIO

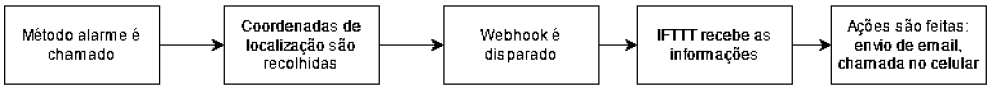
Como forma de otimizar o tempo de desenvolvimento e também de testar uma tecnologia diferente, foi utilizado um software de apoio quando o alarme da aplicação era chamado. Esse software é o IFTTT.

2.5.1 IFTTT

O IFTTT (*If This Then That*) é uma aplicação onde é possível preparar vários gatilhos para inicializar determinadas ações (IFTTT Developers, 2021). Para o Drowsily Lite, ele foi configurado para que quando o método contendo o alarme (que indica que foi

capturado um estado de sonolência) fosse chamado, seria enviado, via *webhook* (uma forma de trocar informações entre dois sistemas), informações com as coordenadas de localização do usuário para um *e-mail* previamente cadastrado. Além disso também foi criada uma ação que dispara um alarme em um celular especificado. A FIGURA 5 ilustra esse fluxo.

FIGURA 5 – Envio de notificação



Fonte: O autor (2021)

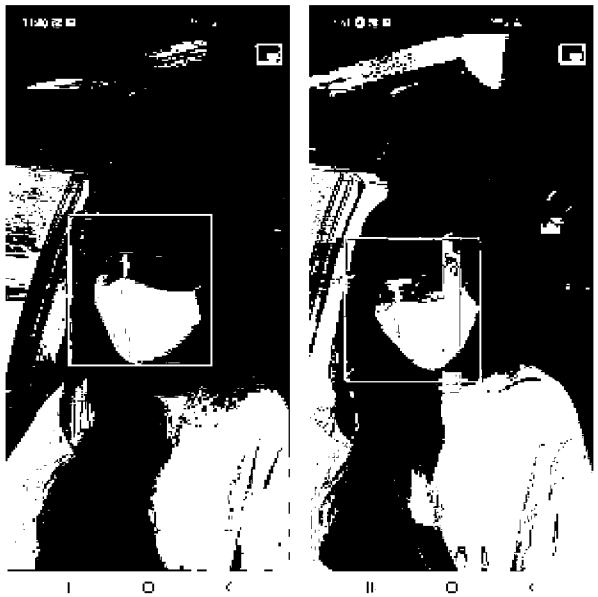
2.6 APLICAÇÕES RELACIONADAS

Existem ainda algumas aplicações interessantes de serem citadas pois tratam também do reconhecimento de sonolência tal qual o escopo deste trabalho. A exposição a seguir trata de 3 aplicações que estão no mercado.

2.6.1 Drowsy Driving Alert: Sleepy Driver Warning

Essa aplicação pode ser encontrada na Google Play e instalada em sistemas Android. A funcionalidade dela, de acordo com a descrição, é de analisar os olhos do usuário para verificar o momento que se fecham (Galaxy Lab, 2020), como mostrado na FIGURA 6.

FIGURA 6 – Imagem demonstrativa do app Drowsy Driving Alert



Fonte: (Galaxy Lab, 2020)

2.6.2 Awake: Drowsy Driving

Outro aplicativo que faz detecção de fadiga analisando os olhos do usuário, dessa vez para o ecossistema da Apple. Seu funcionamento segue a mesma linha de analisar os olhos do usuário e indicar fadiga caso se fechem, uma tela do *app* na FIGURA 7 faz a demonstração da detecção em tempo real. Um diferencial é a utilização de mais itens da família de hardware da Apple para fazer a detecção da fadiga. O *app* pode ser integrado com o *smartwatch* da Apple e verificar o batimento cardíaco do usuário para um diagnóstico mais preciso (SIMMONS, 2020).

FIGURA 7 – Imagem demonstrativa do app Awake: Drowsy Driving

Fonte: (SIMMONS, 2020)

2.6.3 Argus DS

O Argus DS é um software da empresa Argus Solutions, no site da aplicação existe uma descrição do software:

> Por meio de sensores e algoritmos avançados, o sistema da Argus Solutions acompanha em tempo real a operação dos motoristas, monitorando seus olhos, boca, expressões faciais e suas ações, permitindo identificar os primeiros sinais de fadiga e também de distração na operação. Quando esses eventos são identificados, o motorista recebe imediatamente um alerta sonoro para alertá-lo e evitar assim um possível acidente. (Argus, 2021)

Como pode-se perceber esse software faz uma análise em outros elementos para fazer o diagnóstico da sonolência, além dos olhos, a boca e expressões faciais entram na equação para determinar a fadiga do motorista.

2.6.4 Comparação com o Drowsily Lite

As três aplicações relatadas nesse capítulo têm uma proposta bem clara, detectar sonolência do usuário e disparar um alarme quanto essa sonolência for detectada. Em relação ao alarme, a aplicação *Drowsy Driving Alert* tem um diferencial, além de uma alerta sonoro do celular, é possível ainda notificar um *smartwatch* da Apple através de alerta sonoro ou vibração. As outras aplicações apenas disparam alerta sonoro no dispositivo que está sendo executado. Neste ponto o Drowsily Lite tem também um diferencial, pois trabalhando em conjunto com o IFTTT (explicado melhor na seção 2.5.1), além de disparar um alerta sonoro para um celular previamente especificado, ainda pode enviar para um *e-mail*, também previamente especificado, as coordenadas geográficas que foi localizada o sonolência do usuário.

Quando ao método de reconhecimento da fadiga, o Drowsily Lite apenas faz a análise dos olhos do usuário, essa é uma desvantagem em relação ao Argus Ds, que além de análise dos olhos também analisa a boca e expressões faciais do motorista, sendo possível então um diagnóstico mais preciso devido a possibilidade de verificação de mais variáveis.

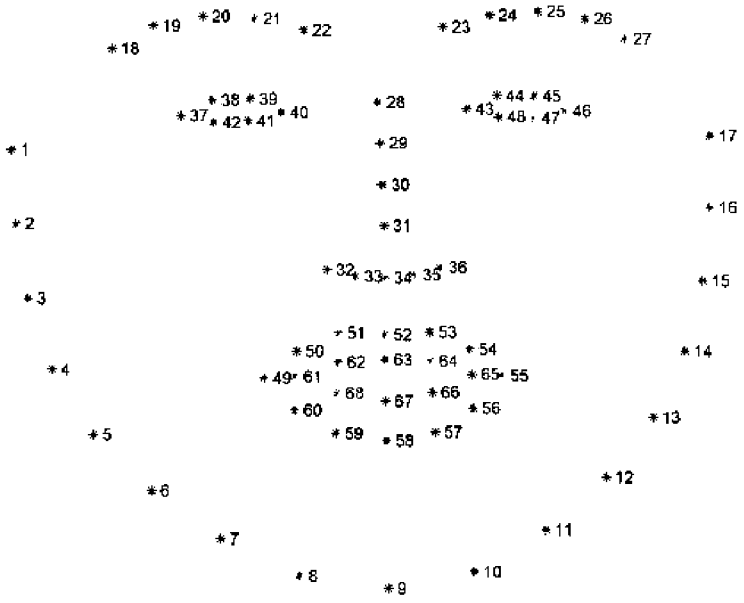
2.7 FUNCIONAMENTO DA APLICAÇÃO

O diagrama de atividade, que encontra-se no apêndice, mostra o fluxo de funcionamento do software.

Ao iniciar a aplicação a primeira ação a ser feita é a inicialização da câmera para dar inicio ao processo de *streaming* de vídeo através dela. Com essa etapa iniciada, a próxima ação é a chamada do método *ml_face_detector()* que é o responsável pela aplicação de técnicas de *machine learning* na aplicação.

O método *ml_face_detector()* utiliza a biblioteca *dlib*, com ela é possível detectar um rosto em uma imagem e também fazer predição de áreas comuns no rosto humano, como sobrancelhas, olhos, nariz, lados do rosto e boca. Para reconhecer essas áreas comuns a *dlib* utiliza um conjunto de algoritmos de *machine learning* de *regression trees* onde é possível identificar 68 marcas comuns no rosto humano (chamado de 68 *facial landmarks*), a FIGURA 8 ilustra essas *facial landmarks*.

FIGURA 8 – 68 facial landmarks



Fonte: (ROSEBROCK, 2017)

Com o rosto sendo detectado e sendo possível identificar os seus locais comuns, então é possível identificar os pontos dos olhos com exatidão, sendo os pontos 37 a 42 representando o olho direito e os pontos 43 a 48 representando o olho esquerdo. Sabendo que 6 pontos representam cada olho, é possível com os 2 pontos superiores e os dois pontos inferiores, saber se um olho está 'mais aberto' ou 'mais fechado', calculando a distância entre esses pontos. Essa distância é chamada de *Eye Aspect Ratio (EAR)*.

Quanto maior o *EAR*, mais amplamente o olho está aberto. Logo, quanto menor o *EAR*, consequentemente teremos a representação de um olho mais fechado.

Relacionando os pontos que representam o *facial landmark* da seguinte maneira (considerando apenas o olho esquerdo), exposto no QUADRO 1:

QUADRO 1 – Relação pontos facial landmark com representação na equação

Facial Landmarks	Representação
37	p1
38	p2
39	p3
40	p4
41	p5
42	p6

Fonte: O autor (2021)

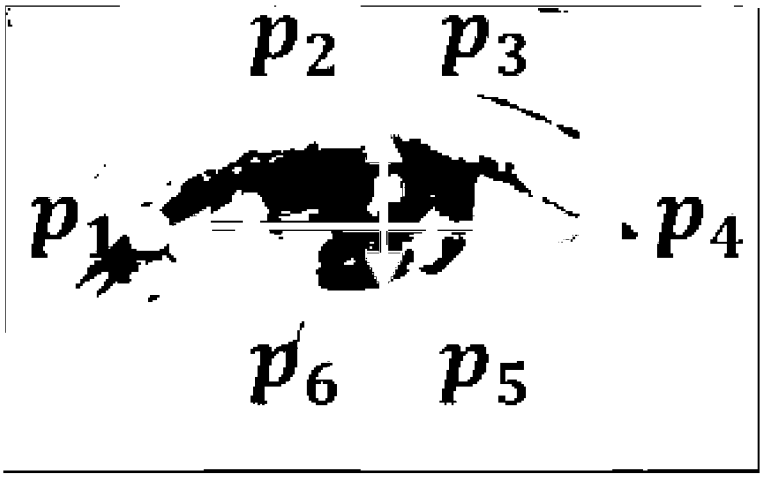
É possível descobrir o valor do EAR utilizando a seguinte fórmula:

$$EAR = \frac{||p2 - p6|| + ||p3 - p5||}{2||p1 - p4||}$$

Sendo que $||p2-p6||$, $||p3-p5||$ e $||p1-p4||$ é o cálculo da distância euclidiana.

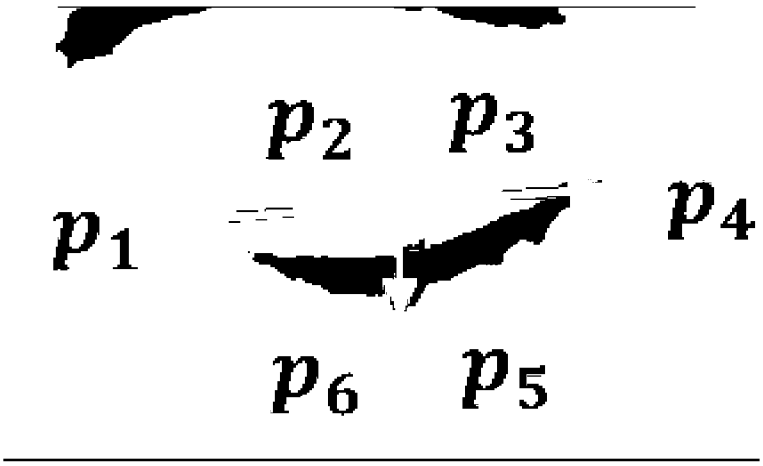
Esta relação pode ser visualizada graficamente como mostrada nas FIGURAS 9 e 10:

FIGURA 9 – EAR alto, distância entre p2 e p6 e entre p3 e p5 altas



Fonte: (PANDEY, 2021)

FIGURA 10 – EAR baixa, distância entre p2 e p6 e entre p3 e p5 baixas



Fonte: (PANDEY, 2021)

Um modelo treinado (explicado na seção 3.3) é utilizado com a ferramenta *shape_predictor* da biblioteca *dlib*, que fará a predição dos *68 facial landmarks*.

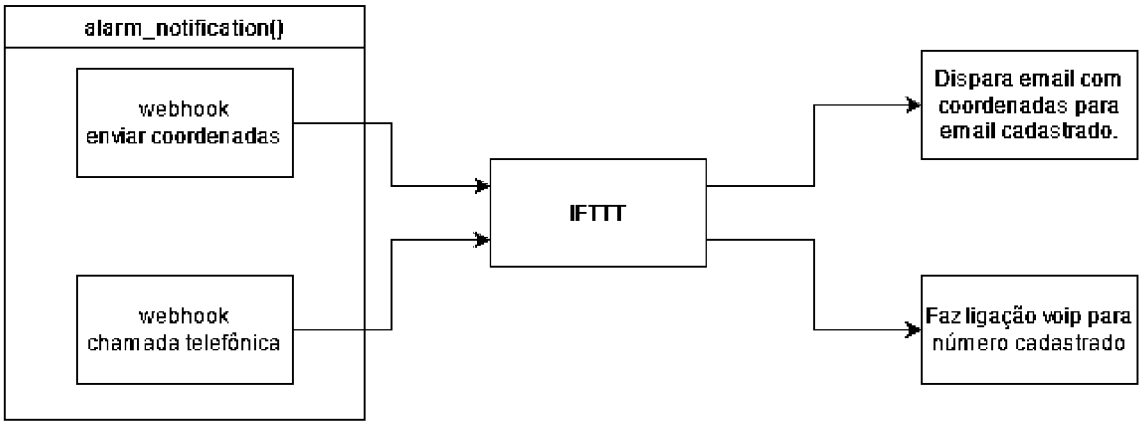
O programa entra em um *looping* analisando pelo preditor os *frames* do *streaming* feito pela câmera. A cada *frame* do *streaming* é realizado um cálculo pelo

método *eye_aspect_ratio()* que retorna o valor do *EAR*. Se o valor do *EAR* estiver muito baixo então isso significa que os olhos não estão muito abertos, o que significa que o motorista pode estar em um estado de sonolência.

Porém, essa verificação é feita em todo *frame*. Se simplesmente fosse detectado um *EAR* baixo em apenas um *frame*, significaria que uma simples piscada poderia corresponder a um estado de sonolência. Para evitar isso existe uma variável de controle chamada *EYE_AR_CONSEC_FRAMES*. Essa variável determina a quantidade de *frames* consecutivos que devem ter o valor *EAR* baixo para que seja considerado realmente um estado de sonolência. Então, se por uma quantidade X de *frames* o *EAR* apresentar o valor baixo, o estado de sonolência será detectado e uma ação será tomada.

A ação tomada está configurada no método *alarm_notification()*, quando esse método é chamado, é enviado para um *e-mail* previamente cadastrado as informações das coordenadas de onde foi detectado o estado de sonolência do motorista além de disparar um alarme (no caso disparado pelo próprio celular do motorista) numa tentativa de alerta sonoro. Essas duas ações são feitas disparando *webhooks*, que serão capturados pela aplicação IFTTT, que então fará efetivamente as ações. A FIGURA 11 ilustra este fluxo.

FIGURA 11 – Uso do IFTTT



Fonte: O autor (2021)

Após tomada de ação, o contador que era comparado com o valor da variável *EYE_AR_CONSEC_FRAMES* é restaurado para voltar novamente à monitoração.

3 MATERIAIS E MÉTODOS

Este capítulo descreve todo o processo de desenvolvimento da aplicação. Aborda-se a utilização da metodologia ágil, sua adaptação às necessidades do trabalho, o cronograma de atividades e os artefatos produzidos ao longo do desenvolvimento.

3.1 ADAPTAÇÃO DA METODOLOGIA ÁGIL

Para a adoção de uma metodologia ágil para o desenvolvimento do *software* proposto nesse trabalho foram necessárias algumas adaptações. Primeiramente por se tratar de um trabalho que foi desenvolvido por apenas uma pessoa os papéis de *Product Owner, Scrum master* e time de desenvolvimento não puderam ser adequadamente distribuídos, sendo assim, o *Product backlog* e demais atividades ficaram a cargo de uma pessoa só, o elaborador do trabalho.

Outra adaptação foi a não adoção da reunião diária. Ela foi substituída por um dia de reflexões feitas inicialmente de modo quinzenal, na qual eram pensados os objetivos de uma nova *sprint* de acordo com o *Product backlog*, posteriormente isso foi alterado para intervalos de tempo não fixos. Ainda, de modo mensal foram feitas reuniões com o professor orientador onde eram discutidas as principais dificuldades enfrentadas, quais itens se encontravam atrasados e quais ações seriam executadas para adequar os prazos.

3.2 HARDWARE UTILIZADO

O hardware utilizado para o desenvolvimento da aplicação foi:

- Notebook Dell com processador i5 de 5ª geração, 12GB de memória RAM, 240 GB de armazenamento do tipo SSD, com câmera VGA com resolução de 0,3 *megapixel*: utilizado para efetivamente rodar a aplicação.
- Notebook Dell com processador i7 de 8ª geração, 16GB de memória RAM, 256 GB de armazenamento NVME: utilizado para treinar o modelo.
- Celular Samsung A30s com câmera frontal de 5 *megapixel*: utilizado para realizar filmagens para testes.

A princípio a aplicação seria embarcada em um Raspberry Pi (modelo B+, com processador Broadcom BCM2835 de 700MHz e 512MB de RAM), inclusive utilizando uma câmera própria para o dispositivo (câmera genérica Raspberry Pi Camera Rev 1.3),

porém, após inúmeras tentativas, não foi possível nem mesmo inicializar o Raspberry Pi. Ao que tudo indica o dispositivo estragou, não sendo possível embarcar o *Drowsily Lite* nele.

3.3 BASE DE DADOS

Para construir o modelo de *machine learning* utilizado na aplicação foi utilizado o *dataset* público iBUG 300-W (SAGONAS et al., 2016). Este *dataset* contém as coordenadas de mais de 7 mil imagens de pessoas.

3.3.1 Facial Landmarks

As *facial landmarks* são as coordenadas de 68 pontos no rosto onde é possível definir as seguintes áreas:

- Boca ;
- Sobrancelha esquerda;
- Sobrancelha direita;
- Olho esquerdo;
- Olho direito;
- Nariz;
- Mandíbula;

A FIGURA 8 mostra a distribuição desses pontos no rosto.

O *dataset* iBUG 300-W contém as coordenadas de todos esses pontos em suas mais de 7 mil imagens. A estrutura principal do dataset contém 3 arquivos:

- labels_ibug_300W.xml: *dataset* completo;
- labels_ibug_300W_test.xml: *dataset* dividido para utilização em testes de modelos;
- labels_ibug_300W_train.xml: *dataset* dividido para utilização em treinamento de modelos.

Uma imagem nesses XMLs tem um estrutura onde é identificado o endereço da imagem (*image file*), localização do rosto (*box*) e as coordenadas de todos os 68 pontos da *facial landmark*. Na FIGURA 12 um recorte dessa estrutura:

FIGURA 12 – labels_ibug_300W_train.xml - recorte

```
<images>
  <image file='lfpw/trainset/image_0457.png' width='350' height='464'>
    <box top='78' left='74' width='138' height='140'>
      <part nome='00' x='55' y='141'/>
      <part nome='01' x='59' y='161'/>
      <part nome='02' x='66' y='182'/>
      <part nome='03' x='75' y='197'/>
      <part nome='04' x='90' y='209'/>
      <part nome='05' x='108' y='220'/>
      <part nome='06' x='131' y='226'/>
      <part nome='07' x='149' y='232'/>
      <part nome='08' x='167' y='230'/>
      <part nome='09' x='181' y='225'/>
      <part nome='10' x='184' y='208'/>
      <part nome='11' x='186' y='193'/>
```

Fonte: O autor (2021)

3.3.1.1 Treinamento do modelo

Para o treinamento do modelo a primeira ação a ser feita foi reduzir o tamanho do *dataset*. Como a intenção é trabalhar apenas com os olhos então inicialmente foi feito um tratamento no *dataset* retirando todos os pontos que não fossem dos olhos, logo apenas as coordenadas 37, 38, 39, 40, 41, e 42 (olho direito) e 43, 44, 45, 46, 47 e 48 (olho esquerdo) foram mantidos nos arquivos. Tanto o arquivo de treinamento quanto de teste passaram por esse tratamento.

O treinamento do modelo foi feito utilizando a biblioteca dlib, mais especificamente utilizando a função *train_shape_predictor()*. O tempo total do treinamento feito no notebook com processador Intel i7 foi de aproximadamente 15 minutos. Como resultado foi gerado um arquivo .dat, que é o modelo que será utilizado na aplicação (Drowsily Lite).

Para avaliar o modelo, a biblioteca dlib fornece uma função para calcular o erro médio (*Mean Average Error*) chamada *test_shape_predictor()*, fazendo a avaliação na base de treinamento e na base de testes foi possível obter os seguinte resultados:

- erro médio com base de treinamento: 3.659473224958382
- erro médio com base de teste: 7.402431623435015

Os resultados dos erros são condizentes (o erro na base que foi utilizada na geração do modelo foi menor que o erro da base de testes).

Para finalizar a avaliação do modelo, foi feito uma avaliação visual, ou seja, verificar se o modelo está realmente reconhecendo os olhos. Na FIGURA 13 pode-se verificar que o modelo funcionou corretamente conseguindo estabelecer as *landmarks* dos olhos.

FIGURA 13 – Validação visual do modelo

Fonte: O autor(2021)

A partir dessa validação, foi estabelecido que esse modelo seria usado na aplicação.

3.4 CRONOGRAMA DE ATIVIDADES

A seguir, no QUADRO 2, o cronograma de atividades executado.

QUADRO 2 – Cronograma de atividades

Iteração	Data final	Atividades
Sprint 0	18/06/20	Discussão e definição sobre o tema do trabalho
Sprint 1	14/12/20	Definição do novo tema do trabalho
Sprint 2	04/01/21	Definição das tecnologias utilizadas
Sprint 3	18/01/21	Criação do ambiente de desenvolvimento. Criação do repositório GIT
Sprint 4	25/01/21	Teste de conceito

QUADRO 2 – Cronograma de atividades

Iteração	Data final	Atividades
Sprint 5	08/02/21	Elaboração dos diagramas de caso de uso e de atividades.
Sprint 6	15/02/21	Implementação completa da aplicação
Sprint 7	17/05/21	Recursos extras de alerta
Sprint 8	07/06/21	Inicio da elaboração do texto do trabalho.
Sprint 9	29/08/21	Testes da aplicação

Fonte: O autor (2021)

3.4.1 Sprint 0: Discussão e definição sobre o tema do trabalho

Na *Sprint* inicial, juntamente com o professor orientador, foi feita uma discussão e reflexão sobre o tema do trabalho. A ideia inicial definida então foi fazer uma análise de textos de *twitter* para conseguir, através do uso de inteligência artificial, classificar se determinado *twitt* teria um cunho racista ou preconceituoso. Ainda nesse *Sprin* foram feitas algumas pesquisas sobre base de dados com *fake news* para um eventual treinamento do algoritmo e também foram pesquisados alguns trabalhos e aplicativos que detectam *fake news*.

3.4.2 Sprint 1: Definição do novo tema do trabalho

Nesta *Sprint*, depois de um grande hiato, foi definida uma nova ideia para o trabalho, dessa vez uma aplicação que consegue analisar os olhos de uma pessoa, através de técnicas de *machine learning* e visão computacional, e consegue dizer se a pessoa está com os olhos fechados ou abertos. Esse foi o escopo definido para o trabalho.

3.4.3 Sprint 2: Definição das tecnologias utilizadas

A escolha das tecnologias utilizadas é um ponto crucial para o desenvolvimento, entendendo que essas escolham podem afetar de forma positiva ou negativa o que se pretende desenvolver. O QUADRO 3 apresenta as tecnologias utilizadas para o inicio do desenvolvimento do projeto, a versão utilizada e uma breve descrição de sua finalidade.

QUADRO 3 – Tecnologias Utilizadas

Tecnologia	Versão	Descrição
Python	3.7.3	Linguagem de Programação
OpenCV	4.3.0.36	Biblioteca para desenvolvimento de aplicações de visão computacional

QUADRO 3 – Tecnologias Utilizadas

Tecnologia	Versão	Descrição
Dlib	19.20.0	Toolkit contendo algoritmos de aprendizado de máquina
Git	2.28.0	Controle de versão

Fonte: O autor (2021)

3.4.4 Sprint 3: Criação do ambiente de desenvolvimento e do repositório Git

Nesta *Sprint* é criado o ambiente de desenvolvimento utilizando o próprio *python* com o uso de *virtenv* (ambiente virtual), dessa forma foi possível isolar o projeto do restante do sistema evitando o conflito de eventuais pacotes utilizados no desenvolvimento. Também é nessa *Sprint* criado o repositório *Git* para o controle de versão, hospedados no serviço *Github*.

3.4.5 Sprint 4: Teste de conceito

Esta *Sprint* foi fundamental para a confirmação da ideia, o conceito básico de reconhecimento do rosto utilizando um preditor de 68 pontos do rosto humano usando aprendizado de máquina com modelo pré treinado foi realizado com sucesso. A partir disso foi pesquisado como isolar apenas os pontos dos olhos e analisar um comportamento que permitisse dizer com clareza quando eles poderiam estar abertos ou fechados.

3.4.6 Sprint 5: Elaboração dos diagramas de caso de uso e de atividades

Nesta *Sprint* os diagramas de classe, de atividades, de caso de uso e as especificações de caso de uso foram elaboradas. Todos os diagramas e a descrição das especificações de caso de uso estão no apêndice.

3.4.7 Sprint 6: Implementação completa da aplicação

Com o teste de conceito bem sucedido, a próxima etapa seria desenvolver todo o conceito da aplicação. As bibliotecas *dlib* e *Opencv* foram facilitadoras fazendo todo o trabalho pesado da aplicação, onde o crucial foi fazer a leitura da documentação das bibliotecas para conseguir utilizá-las de maneira adequado. Essa *Sprint* foi dividida em 3 etapas bem definidas:

- Aplicação do conhecimento adquirido para desenvolver a ideia
- Polimento do código
- Estilização final do código

3.4.8 Sprint 7: Recursos extra de alertas

Nessa *Sprint* foram criados os recursos de alerta. Se inicialmente apenas era exibida uma notificação na tela que mostra a captura do rosto do usuário de que ele estava com os olhos fechados, agora um alerta sonoro era disparado, numa tentativa de chamar a atenção do usuário, e também é enfiado uma notificação utilizando a aplicação *IFTTT (If This Than That)*.

3.4.9 Sprint 8: Início da elaboração do texto do trabalho

Nesta *Sprint* é iniciado efetivamente o processo de escrita do trabalho, com os capítulos de Introdução, Revisão de Literatura e Metologia sendo os primeiros a serem abordados.

3.4.10 Sprint 9: Testes da aplicação

Nesta *Sprint* foram realizados testes da aplicação em variados cenários.

4 RESULTADOS

Alguns cenários foram testados para uma melhor avaliação da eficácia da aplicação. Necessário dizer que não foi encontrado um estudo que mostre por quanto tempo os olhos fechados de uma pessoa podem ser considerados para algum tipo de estado de sonolência, então, de modo empírico, chegou-se a conclusão de que os olhos fechados por mais de 1 segundo e meio pode ser um indício de sonolência. Todos os testes foram feitos no *FPS* padrão dos dispositivos de captura, que é 30 FPS. Isso é importante pois a cada *frame* é feita uma verificação do *EAR* dos olhos. Então, se por 45 *frames* seguidos for verificado um valor de *EAR* abaixo do esperado, significa que o estado de sonolência ocorre e o alerta deve ser ativado. Porém é necessário saber o valor correto do *EAR*. Feito alguns testes em um sujeito (o autor), chegou-se aos valores médios de *EAR* de acordo com o QUADRO 4.

QUADRO 4 – EAR médio

Estado dos olhos	EAR
Abertos	entre 0,32 até 0,38
Semicerrados	entre 0,29 até 0,31
Fechados	entre 0,12 até 0,17

Fonte: O autor (2021)

Como um *EAR* dentro da faixa dos olhos fechados é bastante arriscado pois é possível que o usuário já tenha passado de um estado de sonolência para um estado de sono, para os testes foi considerado o valor de alerta do *EAR* como 0,3. Ou seja, se por 45 *frames* consecutivos o *EAR* estiver abaixo desse valor, significa que o usuário entrou no estado de sonolência. Em resumo, os testes foram realizados seguindo os valores do QUADRO 5.

QUADRO 5 – Valores considerados para os testes

Dimensão	Valor
EAR limite	0,30
Frames avaliados em sequência	45

Fonte: O autor (2021)

Foram testados 9 cenários, com dois sujeitos diferentes, sendo um deles, o sujeito 1, o autor. No QUADRO 6 são mostradas as características de cada cenário.

QUADRO 6 – Cenários de testes

Cenário	Sujeito	Aquisição da imagem	Condição
1	1	Camêra VGA notebook	Dentro de casa
2	1	Câmera Android	Dentro de casa
3	1	Câmera Android	Carro em movimento
4	2	Câmera Android	Carro em movimento
5	2	Câmera Android	Carro em movimento Câmera no painel
6	1	Câmera Android	Carro em movimento Sujeito de máscara
7	2	Câmera Android	Dentro de casa Sujeito óculos de grau
8	2	Câmera Android	Dentro de casa Sujeito óculos escuro
9	1	Câmera Android	Dentro de casa Baixa iluminação

Fonte: O autor (2021)

O alerta configurado para uso em todos os cenários foi o mesmo: disparar email com coordenadas e também fazer uma ligação para o celular do usuário. Em todos os cenários, quando foi exigido, as duas ações do alerta funcionaram em todos os testes.

4.1 CENÁRIO 1

Neste teste foi usada a câmera de um *notebook* dell (i5 - 12GB de ram), uma câmera VGA com resolução de 640x480 (0,3 *megapixel*). Foi o único teste feito usando *streaming* do dispositivo de captura para a aplicação.

Mesmo com uma resolução muito inferior ao que existe hoje no mercado (e é possível perceber a baixa resolução da aplicação rodando nesses condições como mostra a FIGURA 14), o resultado foi muito satisfatório, a aplicação, durantes os testes, dentro de casa em ambiente bem iluminado pela luz do sol, conseguiu em 100% do tempo capturar o rosto e analisar os olhos do sujeito.

FIGURA 14 – Cenário 1 - Câmera VGA

Fonte: O autor (2021)

4.2 CENÁRIO 2

A partir desse cenário foi utilizado a câmera de um celular Android Samsung A30s. Para o cenário 2 o teste foi feito em um ambiente interno (dentro de casa), com boa iluminação natural, utilizando a câmera frontal do celular (5 *megapixel*). Se com uma resolução baixa a aplicação conseguiu funcionar satisfatoriamente, com uma resolução melhor o funcionamento também se deu de modo correto, na FIGURA 15 é possível perceber a aplicação rodando com a câmera de melhor qualidade. Na FIGURA 16 o alerta em execução com a detecção de sonolência. Lembrando que sempre que o celular Android é utilizado nos testes, é gravado o vídeo com o celular e então utilizado esse vídeo na aplicação.

FIGURA 15 – Cenário 2 - Aplicação em funcionamento

Fonte: O autor (2021)

FIGURA 16 – Cenário 2 - Alerta disparado

Fonte: O autor (2021)

4.3 CENÁRIO 3

No cenário 3 entra em cena um ambiente considerado externo, isso porque foi feita uma tentativa de se chegar o mais perto possível do ambiente ideal de uso, dentro de um carro. O interessante desse cenário é que não temos um sujeito parado olhando fixamente para frente, dessa vez temos todas as trepidações e sacolejos que acontecem dentro de um carro.

Neste cenário, o sujeito não estava dirigindo, mas sentado no banco de trás de um carro, como novamente foi usada a câmera do celular Android então foi obtida uma imagem com uma boa resolução, com isso a aplicação conseguiu fazer o reconhecimento dos olhos e seu funcionamento foi dentro do esperado. Na FIGURA 17 é mostrado a aplicação rodando a captura feita neste cenário.

FIGURA 17 – Cenário 3 - Carro em movimento

Fonte: O autor (2021)

Foi interessante notar que em alguns momentos desse teste, com o sujeito olhando para baixo, a aplicação percebia isso como olhos fechando e disparava o alerta. É uma observação interessante para eventuais ajustes. Nos demais casos, os olhos foram capturados em todos os momentos do teste e o alerta foi disparado corretamente.

4.4 CENÁRIO 4

O cenário 4 tem as mesma características do cenário 3, com a diferença que o teste foi feito pelo sujeito 2. Dessa vez, o sujeito estava realmente dirigindo, em alguns momentos ele desviou os olhos para olhar no retrovisor, mas não de modo demorado (o teste foi feito em um trecho de linha reta) e com isso a aplicação conseguiu fazer a captura dos olhos 100% do tempo. A FIGURA 18 mostra a aplicação rodando dessa vez com o sujeito 2.

FIGURA 18 – Cenário 4 - Carro em movimento - Sujeito 2

Fonte: O autor (2021)

4.5 CENÁRIO 5

Este cenário também foi feito pelo sujeito 2, mas teve um diferencial interessante, a câmera foi colocada no painel do carro (com um suporte colocado junto as saídas centrais do ar-condicionado), essa posição poderia ser um lugar interessante para esse tipo de aplicação, pensando em algum tipo de dispositivo embarcado ou mesmo uma aplicação para *smartphone*.

Nessa posição, mesmo não sendo de frente para o sujeito, a aplicação se comportou bem, conseguindo captura e analisar todo o tempo os olhos do sujeito. Na FIGURA 19 é possível perceber a aplicação em funcionamento com esse teste.

FIGURA 19 – Cenário 5 - Carro em movimento - Ângulo diferente

Fonte: O autor (2021)

4.6 CENÁRIO 6

Este cenário continua com os testes em um carro em movimento, porém dessa vez o sujeito 1 utilizou uma máscara (PFF2). Este teste apresentou uma falha. Com a máscara a aplicação não conseguiu reconhecer o rosto corretamente, e com isso não fez o monitoramento do rosto durante todo o tempo, apenas em alguns raros momentos com um segundo ou menos de tempo. Temos então o primeiro caso de um cenário com falha da aplicação. Para ilustrar, a FIGURA 20 é a aplicação rodando este cenário, é possível perceber que não é feito o monitoramento dos olhos pois o rosto não foi reconhecido.

FIGURA 20 – Cenário 6 - Carro em movimento - Uso máscara

Fonte: O autor (2021)

Como a técnica dos *68 facial landmarks* reconhece 68 pontos no rosto, ao que parece não foi possível reconhecer alguns ponto laterais do rosto (área das bochechas) e com isso a aplicação falhou.

4.7 CENÁRIO 7

Neste cenário temos o sujeito 2, em um ambiente interno, com boa iluminação, e desta vez usando um óculos de grau. A aplicação conseguir fazer o monitoramento dos olhos durante todo o teste. Foi feito teste para disparar o alerta, que também funcionou corretamente. Óculos de grau, pelo menos com modelos similares ao do teste, e mostrado na FIGURA 21, não parece causar problemas para a aplicação.

FIGURA 21 – Cenário 7 - Óculos de grau

Fonte: O autor (2021)

4.8 CENÁRIO 8

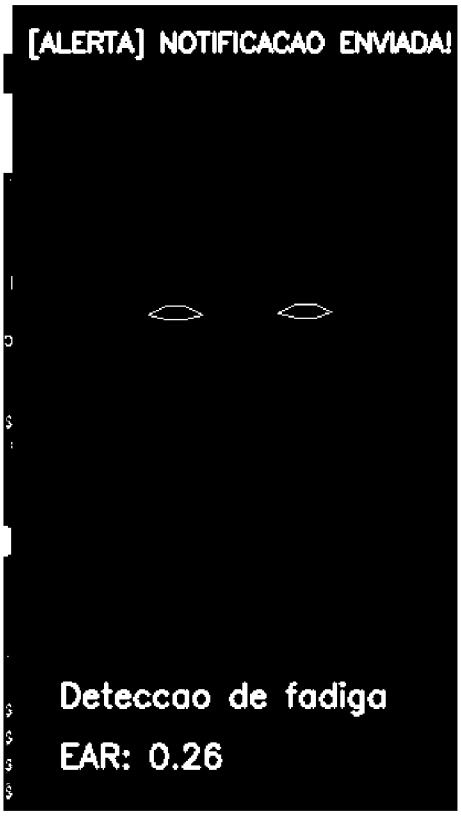
Neste cenário, o sujeito 2 utiliza um óculos escuro, como era de se esperar não é possível ver os olhos devido ao óculos escuro e o reconhecimento dos olhos simplesmente não é executado pela aplicação. Novamente um cenário inviável de utilizar a aplicação.

4.9 CENÁRIO 9

Neste último cenário o sujeito 1 faz o teste em um ambiente com pouco iluminação. É possível perceber o rosto do sujeito pela filmagem, o rosto é reconhecido pela aplicação, porém os olhos não são reconhecidos corretamente.

Durante toda a execução do teste a aplicação indica um *EAR* muito baixo, sempre na casa de 0,20, mesmo o usuário estando com os olhos bem aberto e variando a abertura dos olhos (bem abertos para fechados). Esse valor de aproximadamente 0,20 se manteve durante todas as alterações do sujeito na abertura dos olhos. Na FIGURA 22 é possível perceber essa situação.

FIGURA 22 – Cenário 9 - Pouca iluminação



Fonte: O autor (2021)

4.10 RESUMO DOS CENÁRIOS

Em resumo é possível perceber que em cenários de boas condições de iluminação e com o rosto livre, a aplicação consegue desempenhar bem a sua funcionalidade, mesmo utilizando dispositivos de captura de imagem de baixa resolução. As falhas da aplicação ocorreram em cenários onde não foi possível identificar corretamente o rosto do usuário e onde a iluminação foi insuficiente.

No QUADRO 7 temos os resultados dos cenários analisados.

QUADRO 7 – Resultados dos testes

Cenário	Observação	Resultado
1	Câmera VGA	Sucesso
2	Ambiente interno	Sucesso
3	Carro em movimento	Sucesso
4	Carro em movimento	Sucesso
5	Carro em movimento	Sucesso
6	Sujeito de máscara	Falha
7	Sujeito óculos de grau	Sucesso

QUADRO 7 – Resultados dos testes

Cenário	Observação	Resultado
8	Sujeito óculos escuro	Falha
9	Pouca iluminação	Falha

Fonte: O autor (2021)

5 CONSIDERAÇÕES FINAIS

Foram propostos dois objetivos iniciais neste trabalho: conseguir analisar se o usuário estava em estado de sonolência e também conseguir disparar um alarme quando essa sonolência fosse detectada. De acordo com os resultados obtidos a aplicação mostrou uma funcionalidade muito satisfatória para alcançar esses dois objetivos. Claro, como foi possível perceber na analise dos resultados dos cenários, algumas condições devem ser satisfeitas para o pleno funcionamento da aplicação.

Uma ideia interessante de se propor seria a utilização da funcionalidade da aplicação em conjunto com uma outra funcionalidade, como por exemplo a detecção de bocejo, que também é possível fazer utilizando os pontos da boca dos 68 pontos de *facial landmarks*. Com isso seria possível ter 2 tipos de análise, incrementando e fortalecendo a condição para disparar o alerta.

5.1 RECOMENDAÇÕES PARA TRABALHOS FUTUROS

Como ideia futura a aplicação caberia muito bem em um ambiente embarcado utilizando tecnologia ARM, isso facilitaria o desenvolvimento pois é possível instalar distribuições Linux nessa arquitetura. O *Raspberry Pi* seria o modelo ideal para implementar essa ideia pois é um sistema completo onde seria possível instalar as dependências da aplicação e executá-la tal qual em um ambiente x86. Ainda seria possível transformar a aplicação em um serviço do sistema bastando apenas ligar o *Raspberry* para que a aplicação iniciasse. Com módulo adicional no embarcado ainda seria possível pegar pontos de GPS em tempo real e se conectar a rede móvel (3G ou 4G) para envio de informações.

Também seria interessante como recomendação futura sanar as dificuldades da aplicação em condições desfavoráveis, como por exemplo ambiente com pouca iluminação, usuário usando máscara ou mesmo o usuário utilizando óculos escuro.

REFERÊNCIAS

Argus. **Solução Argus DS**. 2021. Disponível em: <https://argusolutions.com.br/#argus-ds>. Acesso em: 25 de outubro de 2021. Citado na página 22.

BECK, K. et al. **O Manifest Ágil**. [S.l.], 2014. Disponível em: <http://www.manifestoagil.com.br/>. Citado na página 18.

COHN, M. **Desenvolvimento de Software com Scrum: aplicando métodos ágeis com sucesso**. Porto Alegre: Bookman, 2011. ISBN 978-85-7780-807-6. Citado 2 vezes nas páginas 19 e 20.

FARIA, A. de O. **Conceito: Evitando acidentes com visão computacional**. 2014. Disponível em: <https://www.vivaolinux.com.br/artigo/Conceito-Evitando-acidentes-com-visao-computacional>. Acesso em: 25 de outubro de 2021. Citado na página 16.

Galaxy Lab. **Drowsy Driving Alert: Sleepy driver warning**. 2020. Disponível em: <https://play.google.com/store/apps/details?id=com.galaxylab.drowsydriver&hl=pt_BR&gl=US>. Acesso em: 25 de outubro de 2021. Citado na página 21.

IFTTT Developers. **IFTTT - Do more with the things you love**. 2021. Disponível em: <https://ifttt.com/>. Acesso em: 26 de outubro de 2021. Citado na página 20.

JO, M. **Histogramas**. 2015. Disponível em: <https://www.embarcados.com.br/histograma/>. Acesso em: 25 de outubro de 2021. Citado na página 15.

JUNIOR, D. A. **Pobre Profissional do volante!** 2021. Disponível em: <http://abramet-rs.com.br/artigo/pobre-profissional-do-volante/10>. Acesso em: 26 de outubro de 2021. Citado na página 12.

MARENGONI, M.; STRINGHINI, D. **Tutorial: Introdução à Visão Computacional usando OpenCV. Revista de Informática Teórica e Aplicada**. [S.l.], 2010. Disponível em: <https://www.seer.ufrgs.br/rita/article/view/rita_v16_n1_p125/7289>. Citado 2 vezes nas páginas 14 e 16.

PANDEY, D. **Eye Aspect Ratio(EAR) and Drowsiness detector using dlib**. 2021. Disponível em: <https://medium.com/analytics-vidhya/eye-aspect-ratio-ear-and-drowsiness-detector-using-dlib-a0b2c292d706>. Acesso em: 25 de outubro de 2021. Citado na página 25.

ROSEBROCK, A. **Facial landmarks with dlib, OpenCV, and Python**. 2017. Disponível em: <https://www.pyimagesearch.com/2017/04/03/facial-landmarks-dlib-opencv-python/>. Acesso em: 25 de outubro de 2021. Citado na página 24.

RUSSELL, S. J.; NORVIG, P. **Artificial Intelligence: A Modern Approach**. [S.l.]: Prentice Hall, 2011. ISBN 0-13-461099-7. Citado na página 17.

SAGONAS, C. et al. **300 faces In-the-wild challenge: Database and results. Image and Vision Computing (IMAVIS), Special Issue on Facial Landmark Localisation "In-The-Wild"**. 2016. Disponível em: <https://ibug.doc.ic.ac.uk/media/uploads/documents/sagonas_2016_imavis.pdf>. Acesso em: 25 de outubro de 2021. Citado na página 28.

SARADADEVI, M.; BAJAJ, P. **Driver fatigue detection using mouth and yawning analysis. International journal of Computer science and network security**. [S.l.], 2008. Disponível em: <https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.542.1708&rep=rep1&type=pdf>. Citado na página 12.

SIDDIQUI, F. **Tesla floats fully self-driving cars as soon as this year. Many are worried about what that will unleash.** 2019. Disponível em: <https://www.washingtonpost.com/technology/2019/07/17/tesla-floats-fully-self-driving-cars-soon-this-year-many-are-worried-about-what-that-will-unleash/>. Acesso em: 25 de outubro de 2021. Citado na página 13.

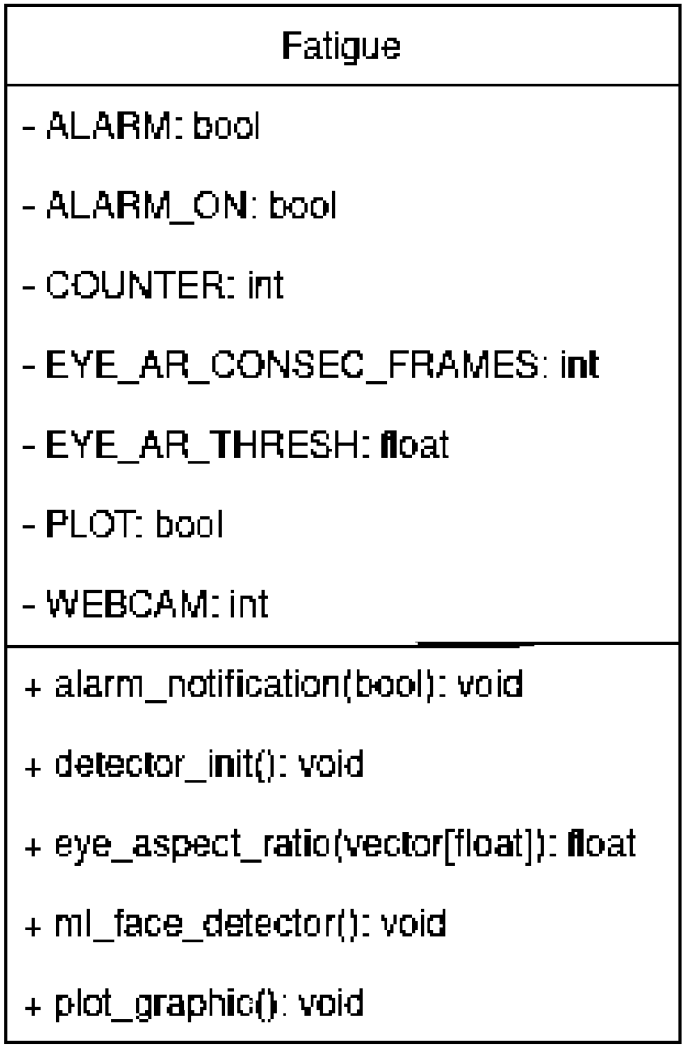
SIMMONS, B. **Awake: Drowsy Driving**. 2020. Disponível em: <https://apps.apple.com/us/app/awake-drowsy-driving/id1493097609>. Acesso em: 25 de outubro de 2021. Citado na página 22.

SOBRAL, A. **BGSLibrary: An OpenCV C++ Background Subtraction Library**. [S.l.], 2013. Disponível em: <https://www.academia.edu/download/54817476/BGSLibrary_An_OpenCV_C_Background_Subtra.pdf>. Citado na página 17.

VIREIRA, D. **Scrum: a metodologia ágil explicada de forma definitiva**. [S.l.], 2014. Disponível em: <http://www.mindmaster.com.br/scrum/>. Citado 2 vezes nas páginas 18 e 19.

APÊNDICE A – DIAGRAMA DE CLASSES

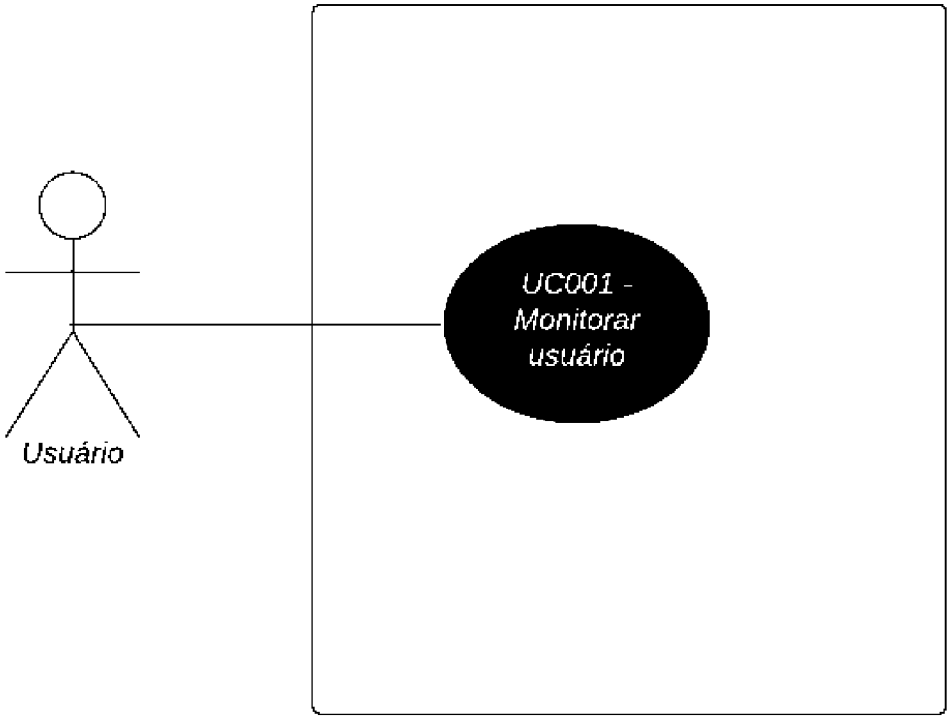
FIGURA 23 – Diagrama de Classes



Fonte: O autor (2021)

APÊNDICE B – DIAGRAMA DE CASOS DE USO

FIGURA 24 – Diagrama de Casos de Uso - Simplificado



Fonte: O autor (2021)

APÊNDICE C – ESPECIFICAÇÕES DE CASO DE USO

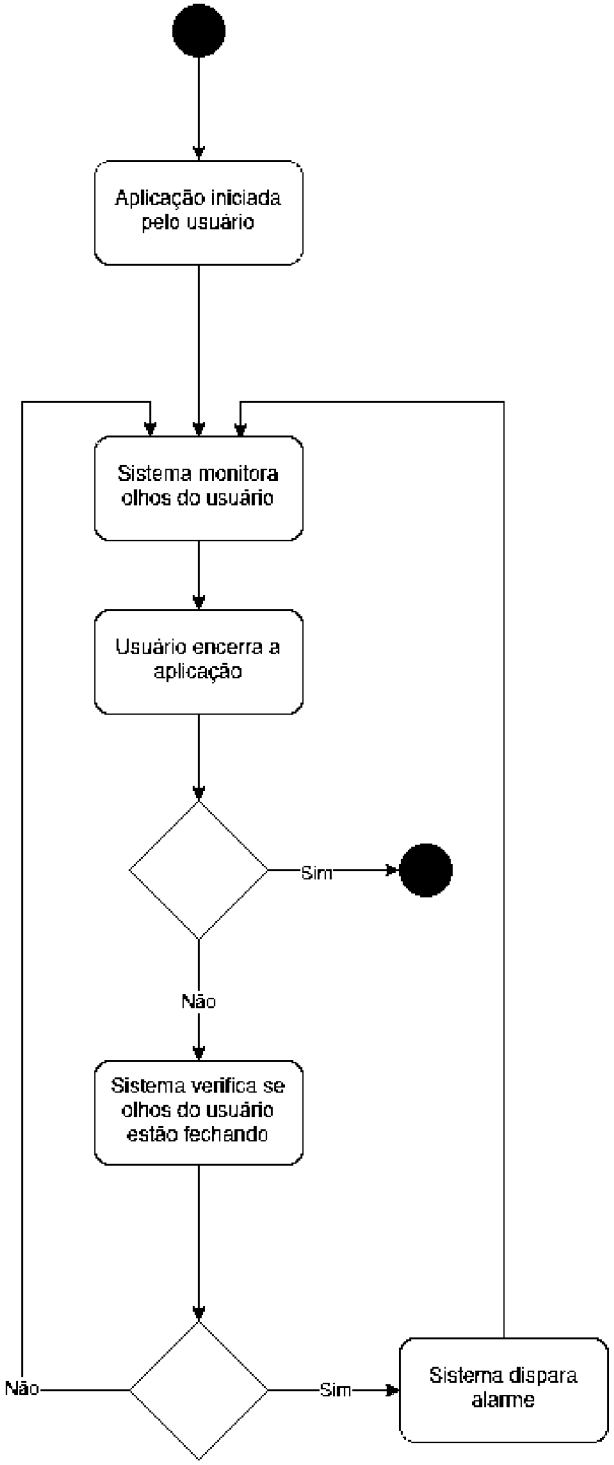
QUADRO 8 – Especificação de caso de uso: monitorar usuário

Nome do caso de uso	Monitorar Usuário
Ator principal	Sistema
Resumo	Esse caso de uso descreve as etapas para sistema inicializar e monitorar o usuário
Pré-condições	Usuário precisa estar disponível para o sistema fazer sua captura em vídeo
Fluxo principal	
1. O usuário inicia a aplicação 2. O sistema inicia o *streaming* de vídeo pela câmera configurada 3. O sistema faz a capturada de um *frame* do vídeo 4. O sistema faz o reconhecimento do rosto do *frame* capturado 5. O sistema reconhece 68 pontos faciais no *frame* com o rosto reconhecido 6. O sistema extrai apenas os pontos faciais referentes ao olhos esquerdo e direito 7. O sistema calcula se os olhos estão abertos ou fechados 8. O sistema avalia se os olhos estão fechados por 45 *frames* consecutivos (E1) 9. O usuário desliga a aplicação 10. O caso de uso é encerrado	
Fluxo de exceção 1 - Olhos fechados por 45 *frames* consecutivos	
1. O sistema exibe a mensagem de alerta "[ALERTA] NOTIFICACAO ENVIADA!" 2. O sistema envia uma notificação por *email* 3. O sistema faz uma chamada de celular	

Fonte: O autor (2021)

APÊNDICE D – DIAGRAMA DE ATIVIDADE

FIGURA 25 – Diagrama de Atividade



Fonte: O autor (2021)