UNIVERSIDADE FEDERAL DE CAMPINA GRANDE
CENTRO DE ENGENHARIA ELÉTRICA E INFORMÁTICA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

LYANG LEME DE MEDEIROS

SISTEMA DE CÂMERA AUTÔNOMA PARA SALA DE AULA

CAMPINA GRANDE
2022

LYANG LEME DE MEDEIROS

SISTEMA DE CÂMERA AUTÔNOMA PARA SALA DE AULA

Trabalho de Conclusão de Curso submetido à Coordenação de Graduação em Engenharia Elétrica da Universidade Federal de Campina Grande, Campus Campina Grande, como parte dos requisitos necessários para obtenção do título de Graduado em Engenharia Elétrica.

Orientador: Gutemberg Gonçalves dos Santos Júnior, D.Sc.

CAMPINA GRANDE
2022

Lyang Leme de Medeiros

Sistema de Câmera Autônoma para Sala de Aula

Trabalho de Conclusão de Curso submetido à Coordenação de Graduação em Engenharia Elétrica da Universidade Federal de Campina Grande, Campus Campina Grande, como parte dos requisitos necessários para obtenção do título de Graduado em Engenharia Elétrica.

Trabalho aprovado em: / /

Gutemberg Gonçalves dos Santos Júnior, D.Sc.
Orientador

Danilo Freire de Souza Santos, D.Sc.
Avaliador

Campina Grande
2022

Dedico este trabalho à minha família, em especial, aos meus pais.

AGRADECIMENTOS

Aos meus pais, Maria das Graças Medeiros e Luiz Leme de Medeiros, por todo amor, carinho, apoio e esforços dedicados a mim e por me servirem de inspiração durante minha trajetória.

A toda a minha família, por torcerem por mim, por servirem de exemplo e apoio ontem, hoje e sempre.

Aos meus amigos, que estiveram sempre ao meu lado, pela paciência, pela compreensão, pela cumplicidade e por amplificarem as alegrias nos momentos felizes e amortecerem as tristezas nos momentos de dificuldade.

A todos os professores do Departamento de Engenharia Elétrica que compartilharam seus conhecimentos em sala de aula e enriqueceram a minha jornada enquanto universitário. Sou grato especialmente aos professores Adolfo Fernandes Herbster e Gutemberg Gonçalves dos Santos Júnior por todos os ensinamentos e oportunidades.

The future cannot be predicted, but futures can be invented. (GABOR, Denis, 1993).

RESUMO

Neste trabalho são apresentadas as bases teóricas, bem como as etapas de implementação, necessárias para o desenvolvimento de um sistema de câmera que automaticamente enquadra o rosto de um apresentador e consegue receber comandos através de gestos manuais. Além disso, foi desenvolvida uma interface gráfica para permitir a visualização da entrada e da saída do sistema, além de permitir a interação por parte do usuário. Por fim, foi desenvolvido um *driver* para Windows 10 que cria uma câmera virtual no sistema operacional, o que possibilita usar o sistema de câmera desenvolvido em aplicações de videochamada.

Palavras-chave: Visão computacional; Rastreamento de faces; Reconhecimento de gestos.

ABSTRACT

In this work, the theoretical bases are presented, as well as the implementation steps necessary for developing a camera system that automatically tracks a presenter's face and can receive commands through manual gestures. In addition, a graphical interface was developed to allow the visualization of the input and output of the system and allow user interaction. Finally, a Windows 10 device driver was developed to create a virtual camera that makes it possible to use the developed camera system in video call applications.

Keywords: Computer vision; Face tracking; Gesture recognition.

LISTA DE FIGURAS

LISTA DE ABREVIATURAS E SIGLAS

2D	Bidimensional
3D	Tridimensional
ABS	*Acrylonitrile Butadiene Styrene*
API	*Application Programming Interface*
DLL	*Dinamic Link Libary*
FPS	*Frames per Second*
GB	*Gigabyte*
HD	*High Definition*
ICH	Interação Humano-Computador
ML	*Machine Learning*
PnP	*Perspective-n-Point*
PTZ	*Pan-Tilt-Zoom*
RAM	*Random Access Memorie*
RGB	*Red, Green and Blue*
SCASA	Sistema de Câmera Autônoma para Sala de Aula
SSD	*Secure Socket Shell*
SSH	*Solid State Drive*
USB	*Universal Serial Bus*

SUMÁRIO

1 INTRODUÇÃO

Neste capítulo serão apresentadas a justificativa, os objetivos gerais e específicos e, por fim, uma breve explicação sobre a organização do trabalho.

1.1 JUSTIFICATIVA

A tecnologia sempre desempenhou um papel importante para a educação. É sabido que o uso de equipamentos modernos e tecnológicos nesta área aumenta o aprendizado e a interação entre alunos e professores (RAJA; NAGASUBRAMANI, 2018). Desta forma, ao longo dos últimos anos, muitas universidades começaram a disponibilizar cursos em modalidade online como forma de facilitar a disseminação de conhecimento.

Além disso, os efeitos da pandemia de COVID-19 causaram um enorme crescimento na demanda por formas de educar através de meios virtuais. Apesar de existirem diversas formas de disponibilizar materiais educativos na internet, ao longos dos anos, aulas em vídeo vem sendo um dos formatos mais usados (SANTOS-ESPINO; AFONSO-SUÁREZ; GUERRA-ARTAL, 2016).

Em geral, a gravação de aulas é feita com o uso de uma câmera comum gravando uma região estática de um ambiente de uma sala de aula. Dado que a maioria das câmeras comuns não têm resolução suficiente para capturar toda a região da lousa de uma só vez e ainda permitir a leitura de seu conteúdo, geralmente as câmeras capturam apenas um pequena parte da lousa. Por isto, o professor fica restrito a movimentar-se dentro da região de captura da câmera. Isto acaba por limitar as possibilidades de interação do professor e causa, porventura, a perda de interesse por parte dos alunos no conteúdo das aulas.

Para solucionar este problema, se faz necessário algum ator que direcione a câmera para a região de interesse da aula a cada momento, buscando sempre manter o melhor enquadramento para mostrar o professor e o conteúdo da lousa que está sendo escrito ou explicado. O que este trabalho propõe é uma forma de automatizar este processo, usando um sistema eletromecânico capaz de rotacionar a câmera em torno de seus eixos horizontal e vertical.

Para isto, foi desenvolvido um sistema de visão computacional que detecta o professor na imagem da câmera e envia as informações de sua posição para o sistema eletromecânico. Com isso, o professor permanece sempre enquadrado na imagem enquanto o mesmo se movimenta pela sala. Por fim, o uso de algoritmos de visão computacional e aprendizado de máquina, possibilita que o professor controle a câmera com gestos manuais, e com isso, aplique *zoom* na imagem e dê destaque para regiões específicas da lousa.

1.2 OBJETIVOS GERAIS

Este trabalho tem como objetivo geral o desenvolvimento de um sistema que permita que um professor ministre aulas com alunos presenciais, ao mesmo tempo em que as transmite em uma videoconferência, de forma que possa mover-se dentro da sala de aula ou auditório sem precisar preocupar-se com o seu enquadramento na câmera.

1.3 OBJETIVOS ESPECÍFICOS

Como forma de mapear e organizar as etapas etapas necessárias para atingir o objetivo geral proposto, foram definidos os seguintes objetivos específicos:

- Estudar e avaliar bibliotecas de código livre baseadas em aprendizado de máquina e/ou inteligência artificial voltadas para o processamento de vídeo;
- Desenvolver um algoritmo capaz de detectar e rastrear pessoas, reconhecer faces e também gestos manuais;
- Desenvolver um sistema mecânico controlado por software que permita a movimentação de uma câmera.

1.4 ORGANIZAÇÃO DO TRABALHO

O presente trabalho encontra-se dividido em 5 capítulos, incluindo este introdutório. A seguir temos uma breve descrição do conteúdo de cada capítulo.

No capítulo 2, é apresentada a fundamentação teórica do projeto, expondo os conceitos de visão computacional, processamento de imagens, estimação de pose, reconhecimento de gestos manuais e uma revisão bibliográfica de trabalhos conexos ao tema.

No capítulo 3, estão descritas as etapas e os componentes de *hardware* necessários para a montagem do sistema. Além disso são apresentadas as bibliotecas de *software* e *frameworks* utilizados para o desenvolvimento da aplicação, assim como os módulos que a compõem.

No capítulo 4, estão expostos os resultados positivos e negativos que foram obtidos nos testes realizados no protótipo construído.

Por fim, no capítulo 5, são apresentadas as conclusões que podem ser retiradas deste trabalho, assim como sua contribuições e sugestões para futuros trabalhos.

2 REFERENCIAL TEÓRICO

Neste capítulo serão apresentadas as teorias e conceitos que embasaram o desenvolvimento deste trabalho, assim como as publicações em periódicos científicos onde são expostas soluções similares à proposta deste trabalho e que serviram como referência para o mesmo.

2.1 VISÃO COMPUTACIONAL

Nos últimos anos, a visão computacional vem se tornando uma tecnologia cada vez mais presente nas mais diversas áreas e aplicações. Segundo Barrow e Tenenbaum (1981), a intensão do uso da visão computacional, de um modo geral, é analisar e entender as cenas do mundo real através de imagens. Apesar de ser uma tecnologia relativamente recente na história da humanidade, desde o seus primórdios, nos anos 60, até os dias atuais, a visão computacional apresentou uma evolução bastante considerável.

Inicialmente, as aplicações de visão computacional eram desenvolvidas para ambientes restritos e controlados. Um dos primeiros exemplos deste tipo de aplicação ficou conhecida como *blocks world*, onde os objetos eram caracterizados por regiões poligonais de brilho aproximadamente uniforme, que eram mapeadas como superfícies de blocos. Já as descontinuidades do brilho em seus limites eram mapeados como as arestas. Porém, este sistema era bastante sensível e tinha problemas para identificar corretamente objetos com texturas, em ambientes com muitas sombras ou sob luzes difusas (BARROW; TENENBAUM, 1981).

Hoje em dia, a visão computacional já está presente em aplicações onde se requer bastante robustez nos mais diversos ambientes e tipos iluminação. Alguns exemplos destes tipos de aplicações são os usos de visão computacional em carros autônomos, drones de resgate, logística em estoques, auxílio em cirurgias médicas e dentre outros. A evolução que permitiu esta gama de aplicações atuais da visão computacional se deu, primordialmente, graças à evolução e à redução do custo do *hardware* dos computadores e câmeras, permitindo o aumento do número de pesquisas nesta área (BUCH; VELASTIN; ORWELL, 2011).

Graças a isto, atualmente temos disponível um grande número de bibliotecas de código aberto voltadas para o processamento digital de imagens, tais como OpenCV, DLib, SimpleCV, TensorFlow, MediaPipe, OpenPose, Pytorch, dentre muitas outras que permitem a criação de aplicações de visão computacional.

2.2 PROCESSAMENTO DIGITAL DE IMAGENS

O termo processamento digital de imagem geralmente se refere ao processamento de uma matriz bidimensional por um computador digital (CHITRADEVI; SRIMATHI, 2014), pois, uma imagem digital é representada como uma matriz de *pixels*, a menor unidade de uma imagem digital. Cada *pixel* tem um valor que representa a intensidade de sua cor. Em imagens em preto e

branco, cada *pixel* apresenta um valor discreto entre 0 e 255, onde o valor mais baixo é branco e o mais alto é o preto, ou vice-versa a depender do tipo de sistema de cores da imagem.

Já em imagens digitais coloridas, um dos sistemas de cores mais comum é composto pela adição das cores Vermelho, Verde e Azul, chamado RGB (*Red*, *Green*, *Blue*). Neste sistema de cores, cada *pixel* é representado por três valores discretos, um para cada cor que compõe o *pixel*, de 0 à 255.

Segundo Gonzalez e Woods (2018), o processamento deste tipo de imagem pode ser dividido em três níveis: nível baixo, médio e alto. No processamento de baixo nível, tanto a saída quanto a entrada do processo são imagens. Este nível inicia-se com a aquisição, que é feita por sensores de luminosidade e em seguida é composto por operações primitivas, como escalonamento e pequenas correções realizadas por computadores, que podem ser necessárias para o próximo nível de processamento, tais como correção de níveis de brilho, correção de contraste, equalização de histograma e filtragem de ruído.

Já no processamento de nível médio, como entrada tem-se uma imagem, mas como saída tem-se as informações que são extraídas da imagem. Estas informações são os resultados de processos como segmentação, reconhecimento e detecção de objetos e extração de características da imagem. Neste nível de processamento, o número de operações necessárias, assim como seus tipos, são definidos pela informação que se deseja extrair da imagem.

Por fim, no processamento de alto nível é dado sentido ao conjunto de objetos reconhecidos, desempenhando as funções normalmente associadas à visão humana, tais como a atribuição de sentido à um determinado gesto manual reconhecido ou a estimativa dos ângulos de rotação que formam a pose na qual se encontra a cabeça de uma pessoa em uma imagem.

2.3 ESTIMATIVA DE POSE DA CABEÇA

Em um grande número das aplicações de visão computacional, tentamos reproduzir atividades simples realizadas no dia-a-dia por humanos. Estimar a orientação da cabeça de uma pessoa no nosso campo de visão é uma destas tarefas. A comunicação humana se dá de modo verbal e não-verbal e a comunicação não-verbal é feita e compreendida até mesmo por crianças que não sabem falar (BUCK, 1975). Desde a mais tenra idade, somos capazes de interpretar a orientação e o movimento da cabeça das pessoas e com isso obter informação importantes para a comunicação, como por exemplo pra onde estão olhando enquanto falam.

Em um contexto de visão computacional, a estimativa da pose da cabeça é o processo de inferir a orientação de uma cabeça humana a partir de imagens digitais (MURPHY-CHUTORIAN; TRIVEDI, 2008). Em aplicações da visão computacional, o objetivo da estimativa da pose da cabeça é inferir a orientação da cabeça de uma pessoa em relação à posição de uma câmera.

Para transformar uma representação bidimensional, baseada em *pixels*, de uma cabeça em um conceito de direção de alto nível, são necessários vários passos de processamento. Assim como, outras etapas de processamento de visão computacional para extração de informações,

tais como coordenadas da posição de uma face em uma imagem.

Um algoritmo ideal para estimar a pose de cabeça deve demonstrar invariância e robustez sob os mais diversos fatores de disformidades na imagem. Esses fatores incluem fenômenos físicos como distorção da câmera, geometria projetiva, bem como aparência biológica, expressão facial e a presença de acessórios como óculos, chapéus e máscaras hospitalares.

Algoritmos robustos de estimativa de pose de cabeça em tempo real têm o potencial de avançar muito nos campos de interação humano-computador (IHC). Segundo Morency, Whitehill e Movellan (2008), as aplicações possíveis incluem novos dispositivos de entrada de computador, reconhecimento de gestos de cabeça, sistemas de reconhecimento de fadiga do motorista, percepção de falta de atenção para sistemas tutores inteligentes e análise de interação social.

2.4 RECONHECIMENTO DE GESTOS MANUAIS

É bem comum pessoas usarem gestos com as mãos para expressar seus sentimentos e comunicar, por vezes até de forma involuntária, seus pensamentos. Os gestos das mãos são uma forma muito útil de IHC, pois são a forma mais primária e expressiva de comunicação humana (WACHS et al., 2011). Desta forma, a interpretação visual dos gestos das mãos pode ajudar a alcançar a facilidade e naturalidade desejadas para uma IHC. Por isto, este tipo de interface já é aplicado em várias áreas como sistemas médicos, tecnologias assistivas, controle de robôs, automação residencial, jogos e entretenimento (OUDAH; AL-NAJI; CHAHL, 2020).

Segundo Murthy e Jadon (2009), o principal objetivo da pesquisa sobre reconhecimento de gestos, é criar um sistema que possa identificar gestos humanos específicos e usá-los para transmitir informações ou para controlar dispositivos. Pode-se definir um gesto como sendo um movimento físico das mãos, braços, rosto e corpo com a intenção de transmitir informação ou significado. Por isto, o reconhecimento de gestos consiste não apenas no rastreamento do movimento humano, mas também na interpretação desse movimento como comandos semanticamente significativos.

Existem duas principais abordagens usadas para o reconhecimento de gestos manuais: o reconhecimento auxiliado por hardware e o reconhecimento auxiliado por vídeo. No reconhecimento auxiliado por hardware é construído um dispositivo vestível, em geral no formato de luva, equipado com sensores que permitam obter as informações de como está a pose da mão e, preferencialmente, de cada dedo individualmente. Segundo Oudah, Al-Naji e Chahl (2020), os sensores comumente usados neste tipo de abordagem são: sensores de curvatura, sensores de deslocamento angular, transdutores de fibra óptica, sensores flexíveis e acelerômetros.

Já o reconhecimento de gestos manuais auxiliado por vídeo, baseia-se no uso de câmeras e algoritmos que processam as imagens capturadas pelas câmeras e identificam a pose das mãos e dedos. Para isso, os algoritmos tentam segmentar e detectar na imagem as características da mão, como cor da pele, aparência, movimento, esqueleto, modelo 3D, detecção por aprendizado profundo dentre outras características.

2.5 TRABALHOS RELACIONADOS

No trabalho de Sutanto et al. (2021), os autores propõem um sistema composto por um computador — que pode ser o computador pessoal do apresentador ou professor, uma Raspberry Pi e um suporte *Pan/Tilt* acionado por servomotores. A câmera é conectada ao computador, via *Universal Serial Bus* (USB), e acoplada mecanicamente ao suporte *Pan/Tilt*.

Com isso, o computador captura as imagens da câmera, procura por um rosto no *frame* e mapeia sua posição. Caso o rosto encontrado esteja dentro de uma região de interesse dentro do *frame*, a câmera não será movida. Por outro lado, caso o rosto esteja fora da região de interesse, o computador calcula qual deve ser a posição da câmera para que o rosto seja enquadrado. Logo em seguida, envia um comando, via *Secure Socket Shell* (SSH), para a Raspberry Pi, que por sua vez controla os servomotores para movimentar a câmera e colocá-la na posição desejada.

Além disso, o sistema de processamento implementado pelos autores, também realiza o reconhecimento de gestos manuais. A procura por gestos manuais acontece uma vez a cada dois *frames*, o que melhora o desempenho do sistema. Os autores fizeram a classificação de dois gestos manuais que são usados para controlar o *zoom* da câmera.

Já o trabalho de Haghighi et al. (2020), os autores propõem um sistema de câmeras para gravação automática de aulas composto por duas câmeras e um dispositivo vestível. Uma das câmeras é uma câmera fixa e a outra é uma câmera *Pan-Tilt-Zoom* (PTZ).

A câmera fixa apresenta um grande ângulo de captura e seus *frames* são tratados com a biblioteca OpenPose CNN que identifica a pose do professor. Os autores definiram um conjunto de poses as quais o professor comumente pode apresentar.

A partir disso, implementaram uma Cadeia de Markov para estimar as possibilidades de transição entre cada uma das poses definidas. Por fim, a posição na qual o professor se encontra na sala é enviada para a câmera PTZ, que tem um ângulo de captura menor e, por sua vez, realiza o enquadramento do professor em sua imagem.

O dispositivo vestível, por sua vez, tem várias funcionalidades. A principal delas é a captura em áudio, mas também serve como uma forma de o sistema de câmeras detectar a presença do professor no ambiente da sala de aula e, com isso, a câmera pode começar a gravar automaticamente a aula sempre que o professor está presente.

3 MATERIAIS E MÉTODOS

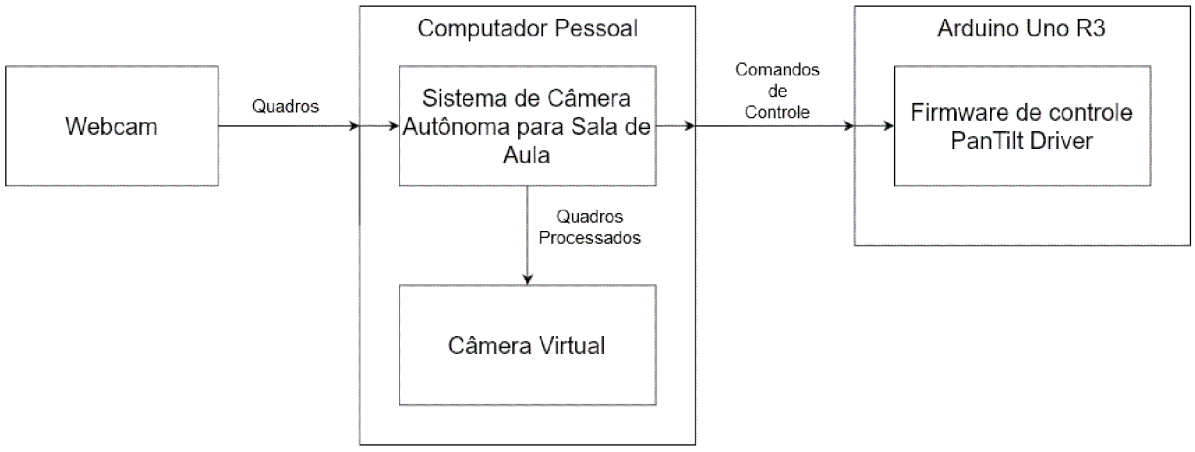
Neste capítulo, são descritos os componentes, tecnologias e procedimentos implementados para o desenvolvimento do sistema que foi projetado ao decorrer deste trabalho, detalhando cada módulo que foi desenvolvido para se obter uma câmera capaz de automaticamente manter um apresentador ou professor enquadrado enquanto o mesmo se movimenta dentro de um ambiente de sala de aula.

3.1 VISÃO GERAL DO SISTEMA

Em alto nível, o sistema é composto por três componentes. O componente principal é o *software* desenvolvido em Python 3.8 para o sistema operacional Windows 10 nomeado *Classroom Autonomous Camera System* ou Sistema de Câmera Autônoma para Sala de Aula (SCASA), em Português. Este *software* deve ser executado em um computador pessoal ao qual estejam conetados uma câmera USB e um Arduino Uno R3 programado com o *firmware* de controle nomeado *PanTilt Driver*.

Nessas condições, o SCASA realiza a aquisição de quadros por meio da câmera e realiza uma série passos, os quais serão descritos nas seções posteriores deste capítulo, para o processamento das imagens. Como resultado deste processamento, são enviados comandos para o *PanTilt Driver* que, com isso, movimentará o suporte *Pan/Tilt* com a câmera de forma a enquadrar corretamente uma região de interesse da sala de aula. Além disso, o SCASA também manda os quadros processados para a câmera virtual, o que permite que estes quadros sejam transmitidos em uma videochamada. A Figura 1 contém um diagrama com uma visão geral das relações dos componentes descritos nesta seção.

Figura 1 – Visão geral do sistema projetado

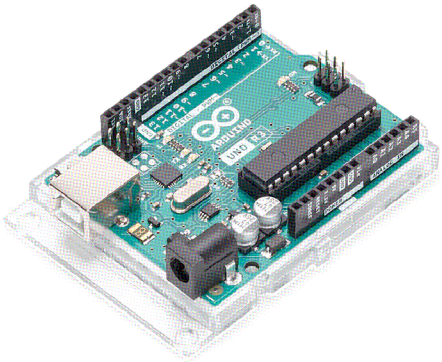


Fonte: Autoria própria.

3.2 COMPONENTES DE *HARDWARE*

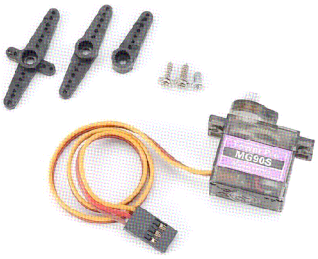
Os componentes de *hardware* necessários para a construção do sistema aqui descrito são: um Arduino Uno R3 (Figura 2a), um suporte *Pan/Tilt* para câmera (Figura 2c), dois micro servos MG90S TowerPro (Figura 2b) e uma Webcam Logitech HD C270 (Figura 2d).

Figura 2 – Componentes de hardware usados na montagem do sistema



(a) Arduino UNO R3

Fonte: Arduino (2010)



(b) Micro Servo MG90S TowerPro

Fonte: Bauermeister e Thomsen (2022)

(c) Suporte Pan/Tilt para Câmera

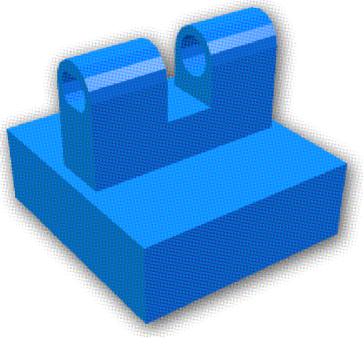
Fonte: RoboCore (2022)

(d) Webcam Logitech HD C270

Fonte: Logitech (2010)

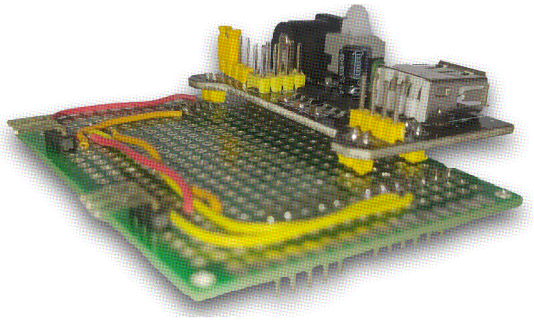
Além destes componentes que foram comprados, foi necessária a confecção de dois componentes para a montagem do sistema final. O primeiro foi um suporte impresso em 3D (Figura 3a) com o material ABS que permite acoplar a câmera C270 no suporte Pan/Tilt. O outro componente produzido foi uma placa de circuito (Figura 3b) que conecta os dois micro servos motores usados ao Arduino Uno R3 e permite que os mesmos sejam alimentados por uma fonte de tensão externa. O *hardware* completo montado para o uso do sistema pode ser visto na Figura 4.

Figura 3 – Componentes de hardware confeccionados para a montagem do sistema



(a) Vista do suporte impresso em 3D

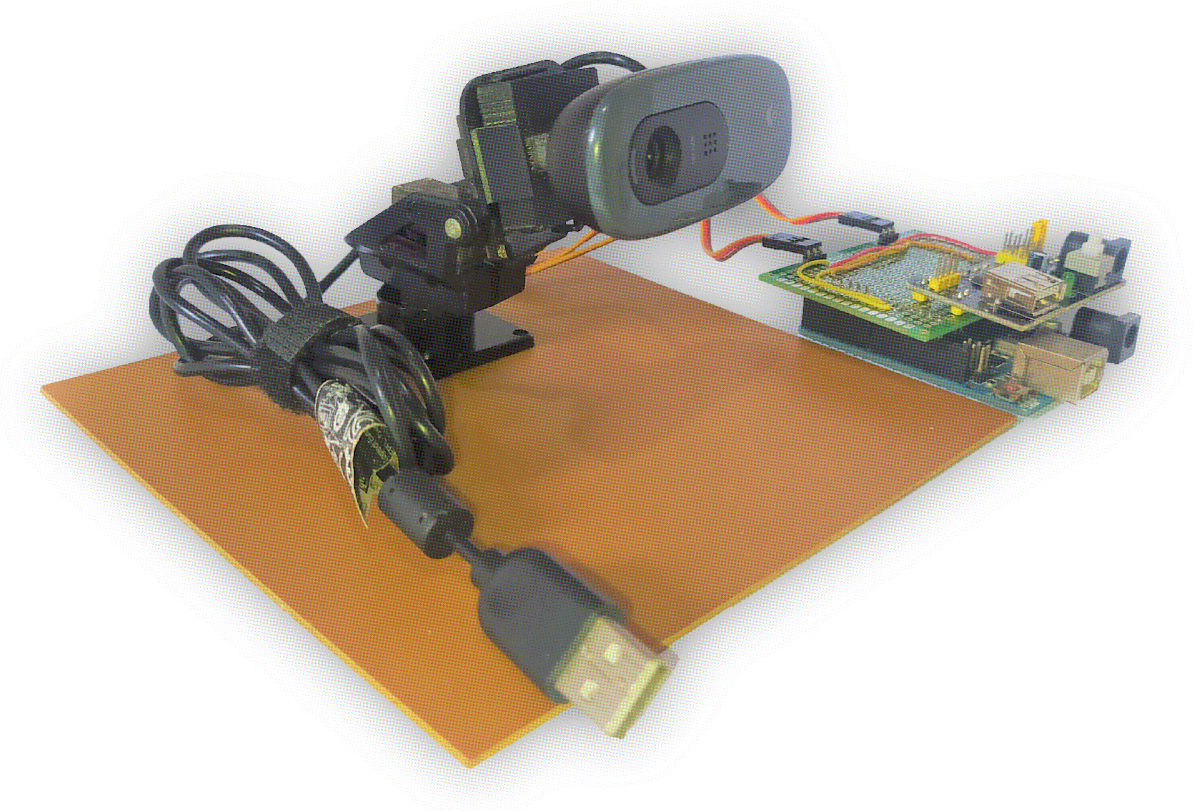
Fonte: Autoria própria



(b) Placa para conexão dos servo motores

Fonte: Autoria própria

Figura 4 – *Hardware* do sistema



Fonte: Autoria própria.

3.3 COMPONENTES DE *SOFTWARE*

Nesta seção, serão apresentadas as tecnologias e bibliotecas utilizadas no desenvolvimento do sistema resultante deste trabalho, assim como os módulos e classes que foram desenvolvidos.

3.3.1 **OpenCV**

OpenCV é uma biblioteca de software de visão computacional e aprendizado de máquina fruto de um projeto de pesquisa da Intel iniciado em 1998 que se tornou púbico no ano 2000 (PULLI et al., 2012). Como o próprio nome sugere, trata-se de uma biblioteca de código aberto, distribuído sob a licença BSD[1], permitindo seu uso em projetos com finalidades acadêmicas ou comerciais. Seu principal objetivo é fornecer uma infraestrutura comum para aplicativos de visão computacional e acelerar o uso da percepção da máquina (OPENCV, 2020).

Para isso, o OpenCV possui interfaces em C++, Python, Java e MATLAB. Além disso, é multiplataforma oferecendo compatibilidade com os sistemas operacionais Windows, Linux, Android e Mac OS, disponibilizando mais de 2500 algoritmos otimizados (OPENCV, 2020).

Dentre estes, estão algoritmos para detecção e reconhecimento de rostos, identificação de objetos, classificação de ações humanas em vídeos, rastreamento de movimento de câmera, rastreamento de objetos em movimento, extração modelos 3D de objetos, produção de nuvens de pontos 3D, mesclagem de imagens para produzir uma imagem de alta resolução, comparação e seleção de imagens semelhantes em um banco de dados dentre muitas outras aplicações.

O OpenCV possui uma comunidade de usuários com mais de 47 mil pessoas e mais de 18 milhões de downloads. A biblioteca é amplamente utilizada em empresas como Google, Microsoft, Intel, IBM, Sony, Honda e Toyota, grupos de pesquisa e órgãos governamentais (OPENCV, 2020).

3.3.2 **Tkinter**

O módulo Tkinter é uma biblioteca padrão da linguagem Python que permite a criação de interfaces gráficas com aparência nativa em sistemas Unix, Windows e Macintosh.

Tkinter é uma interface para o kit de ferramentas GUI para Tcl/Tk. Tcl/Tk é o recurso de scripts e gráficos inicialmente desenvolvido por John Ousterhout e atualmente é mantido pela Scriptics Corporation, fundada por Ousterhout (GRAYSON, 2000). A Figura 5 mostra uma interface gráfica de exemplo que foi contruída usando Tkinter na linguagem Python 3.8.

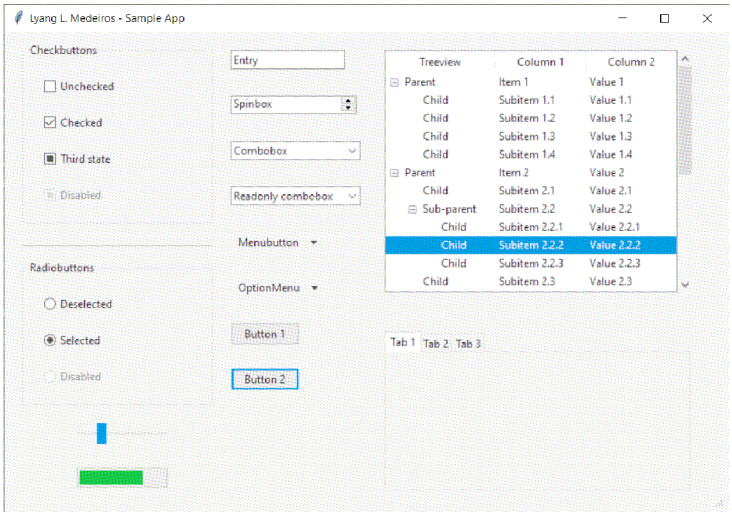
3.3.3 **MediaPipe**

O MediaPipe é uma biblioteca de *software*, com interfaces em C++, Python e JavaScript, desenvolvida e mantida pelo Google, que foi projetada para profissionais de aprendizado de máquina, do inglês *Machine Learning* (ML), incluindo pesquisadores, estudantes e desenvolvedores de *software*, para facilitar o desenvolvimento de aplicações de ML prontas para produção, desenvolvimento de trabalhos de pesquisa acadêmica e criação protótipos de tecnologia.

O principal caso de uso do MediaPipe é a prototipação rápida de pipelines de percepção com modelos de inferência e outros componentes reutilizáveis. O MediaPipe também facilita a implantação da tecnologia de percepção em demonstrações e aplicativos em uma ampla

[1]The FreeBSD Copyright <https://www.freebsd.org/copyright/freebsd-license/>

Figura 5 – Exemplo de interface gráfica criada com o Tkinter



Fonte: Autoria própria.

variedade de plataformas de hardware diferentes. O MediaPipe permite melhorias incrementais nos pipelines de percepção por meio de sua rica linguagem de configuração e ferramentas de avaliação (LUGARESI et al., 2019).

3.3.4 **Sistema de Câmera Autônoma para Sala de Aula (SCASA)**

Este sistema consiste em um *software* com interface gráfica (Figura 6) desenvolvida com a biblioteca Tkinter em Python 3.8 que permite um usuário conecte e controle o *hardware* desenvolvido neste projeto. O sistema é constituído por oito módulo separados, cada um com uma responsabilidade específica que trocam mensagens entre si como mostrado no diagrama de comunicação da Figura 7. Nas seções a seguir, serão tratados em mais detalhes cada um dos módulos desenvolvidos para esta aplicação.

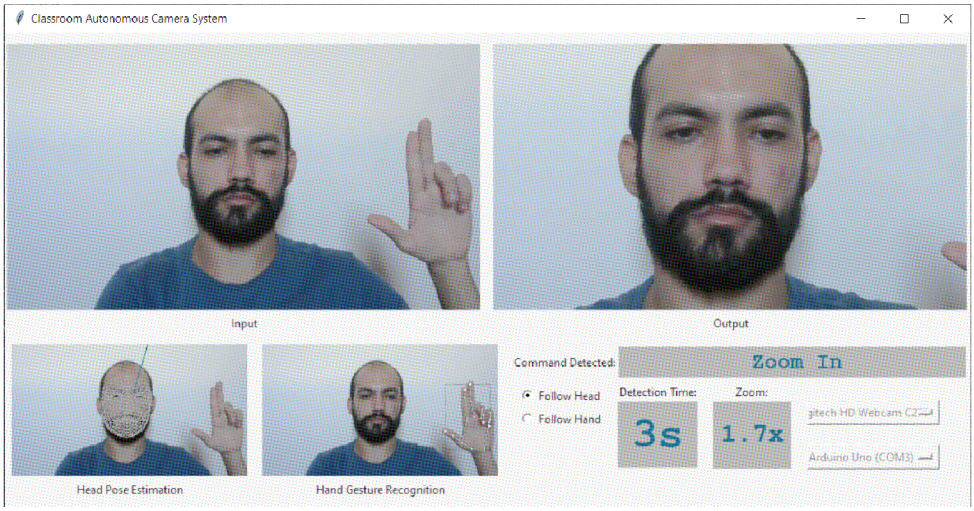
3.3.4.1 ***Módulo de gerenciamento de processos***

Uma vez que o processamento de imagens demanda um grande poder computacional e o sistema aqui descrito se propõe a ser um sistema de tempo real, surgiu a necessidade de paralelizar as tarefas realizadas pelo sistema para permitir que os quadros processado chegassem o mais rápido possível na saída do sistema.

Para implementar este paralelismo, optou-se por usar a técnica de multiprocessamento. Com isso, cada módulo que desempenha uma tarefa isolada e específica deveria ser executado em um processo separado. Uma vez que os processos não compartilham a mesma região de memória do computador onde estão sendo executado, surgiu a necessidade de implementar também meios de comunicação entre processos.

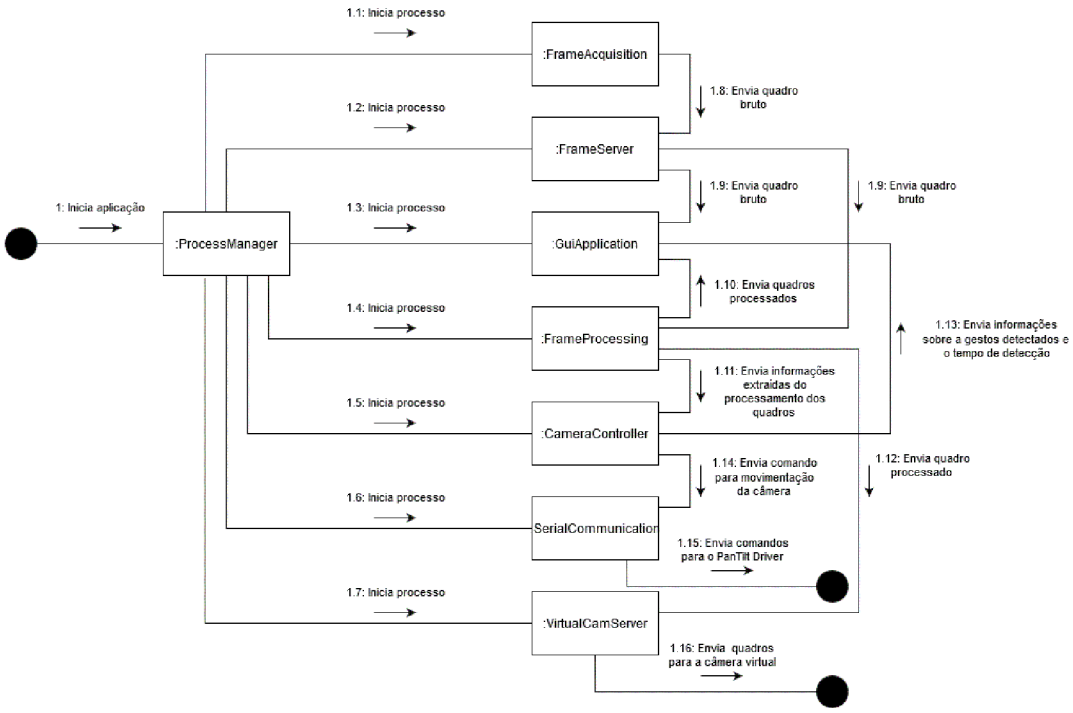
Para resolver esta questão, foi desenvolvido um módulo específico para gerenciar tanto a criação e encerramento dos processos que compõem o SCASA, assim como gerenciar as filas e

Figura 6 – Interface do Sistema de Câmera Autônoma para Sala de Aula



Fonte: Autoria própria.

Figura 7 – Diagrama de comunicação entre as classes que compõem o SCASA



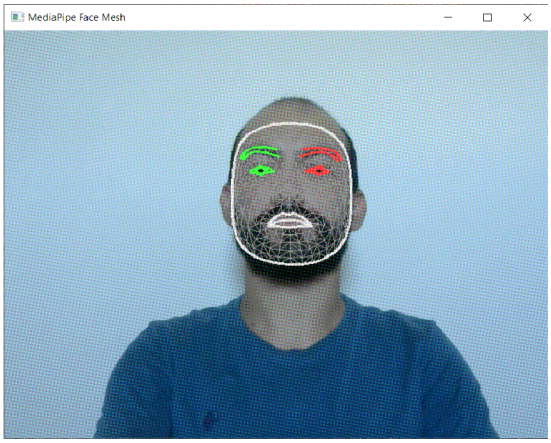
Fonte: Autoria própria.

pipes que são usados na comunicação entre os processos. Este módulo foi implementado em uma classe nomeada `ProcessManager`.

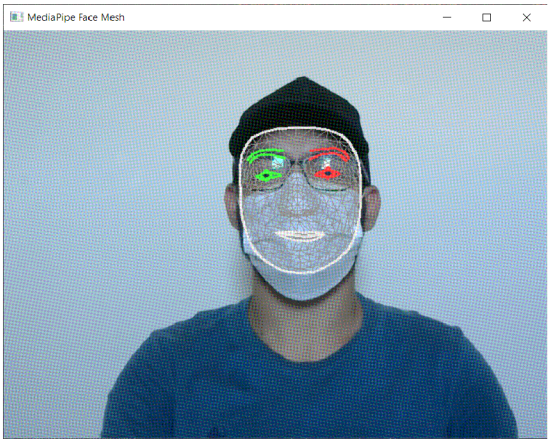
3.3.4.2 *Módulo de aquisição*

Este módulo foi implementado na classe `FrameAcquisition`. Sua finalidade é inicializar a câmera *webcam*, usando a biblioteca OpenCV e, após isso, entra em um laço onde

Figura 8 – Exemplos de uso do MediaPipe Face Mesh



(a) Pontos de referência encontrados em uma face sem acessórios usando o MediaPipe Face Mesh.



(b) Pontos de referência encontrados em uma face com acessórios usando o MediaPipe Face Mesh.

Fonte: Autoria própria

fica constantemente obtendo quadros da câmera e enviado estes para o módulo de distribuição de quadros através de uma fila.

3.3.4.3 *Módulo de distribuição de quadros*

Da necessidade de mais de um processo receber o mesmo quadro ao mesmo tempo, surgiu o módulo de distribuição de quadros. Implementado na classe `FrameServer`, este módulo recebe os quadro recém obtidos pelo módulo de aquisição, e os distribui simultaneamente, através de filas, para os módulos de interface gráfica e de processamento de quadros.

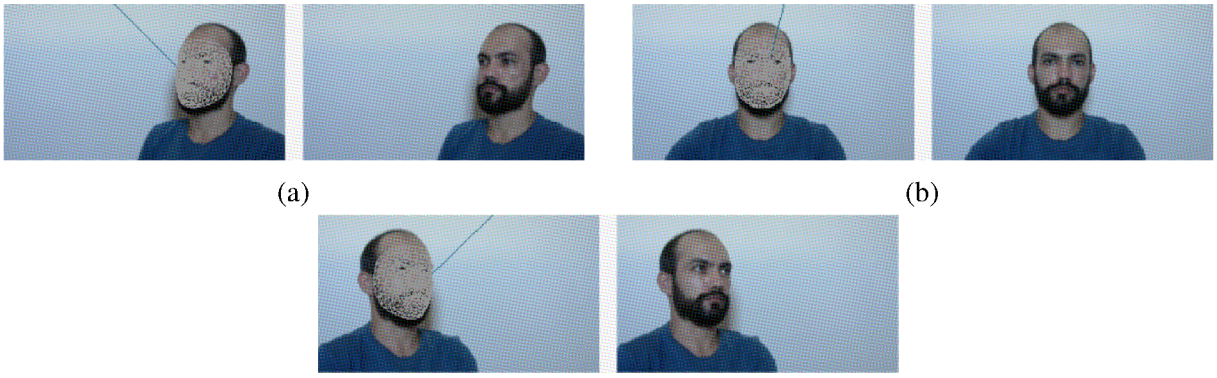
3.3.4.4 *Módulo de processamento de quadros*

A partir deste módulo, que é implementado na classe `FrameProcessing`, são criados dois processos diferentes, um para cada tipo dos processamentos que são realizados: o processamento para estimativa da posição da cabeça, e o processamento para reconhecimento de gestos manuais.

A estimativa da posição da cabeça foi implementado usando o módulo Face Mesh da biblioteca MediaPipe. Este módulo emprega aprendizado de máquina (ML) para inferir a superfície facial 3D e conseguir demarcar 468 pontos de referência em uma face (Figura 8a), exigindo apenas uma única entrada de câmera sem a necessidade de um sensor de profundidade dedicado, funcionando até mesmo em pessoas usando acessórios como óculos, chapéus e máscaras, como mostrado na Figura 8b.

Existem diversas abordagens na literatura para realizar a estimativa de pose de uma cabeça. Neste trabalho, foi usada uma abordagem que consiste na resolução de um problema conhecido na computação como *Perspective-n-Point* (PnP). Este tipo de solução permite corre-

Figura 9 – Estimativas de pose da cabeça realizada pelo SCASA



(a) (b)

(c)

Fonte: Autoria própria

lacionar a posição de pontos em um sistema de coordenadas tridimensional com a posição de pontos conhecidos em um sistema de coordenadas bidimensional.

Para isso, é necessário conhecer um certo um número de pontos de um objeto em um sistema de coordenadas 3D, assim como a projeção desses pontos em um plano em outro sistema de coordenadas, nesse caso, uma imagem 2D (ROCCA; MANCAS; GOSSELIN, 2014). Com isso, é possível encontrar a transformação entre os dois sistemas de coordenadas a partir da seguinte relação:

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \cdot \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{1}$$

- u e v são as coordenadas do ponto de projeção em *pixels*;
- A é uma matriz de câmera ou uma matriz de parâmetros intrínsecos;
- (c_x, c_y) é um ponto principal que geralmente está no centro da imagem;
- f_x e f_y são as distâncias focais expressas em unidades de *pixel*;
- (X, Y, Z) são as coordenadas de um ponto 3D no espaço;
- E $[R|t]$ é a matriz de rotação e transformação

Com um conjunto de N coordenadas 2D e as correspondências 3D, e conhecendo a matriz da câmera A, é possível encontrar a matriz rotação-translação. A própria biblioteca OpenCV contém a função `solvePnP`[2] que fora usada neste trabalho para resolver o problema de PnP em cada quadro capturado. A Figura 9 mostra a estimativa de pose da cabeça de um usuário sendo realizada pelo SCASA.

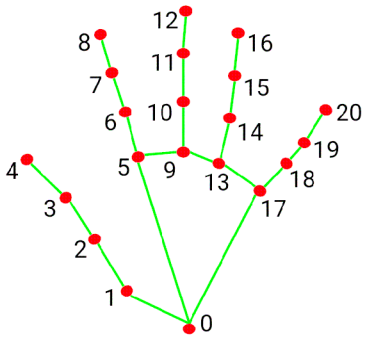
Já o reconhecimento de gestos manuais é implementado usando outro módulo da biblioteca MediaPipe nomeado Hands. Este módulo é uma solução de rastreamento de mãos

[2]Perspective-n-Point (PnP) pose computation <https://docs.opencv.org/4.x/d5/d1f/calib3d_solvePnP.html>

e dedos de alta fidelidade, que emprega aprendizado de máquina para inferir 21 pontos de referência de uma mão (Figura 10) a partir de um único quadro, (ZHANG et al., 2020).

O módulo Hands utiliza um pipeline de ML que consiste em vários modelos trabalhando juntos, sendo: um modelo de detecção de palma, que opera na imagem completa e retorna uma caixa delimitadora de mão orientada; E um modelo de referência de mão que opera na região da imagem recortada definida pelo detector de palma, retornando ao final pontos de referência com alta fidelidade (ZHANG et al., 2020).

Figura 10 – Pontos de referência marcados pelo módulo Hands do MediaPipe



0. WRIST
1. THUMB_CMC
2. THUMB_MCP
3. THUMB_IP
4. THUMB_TIP
5. INDEX_FINGER_MCP
6. INDEX_FINGER_PIP
7. INDEX_FINGER_DIP
8. INDEX_FINGER_TIP
9. MIDDLE_FINGER_MCP
10. MIDDLE_FINGER_PIP
11. MIDDLE_FINGER_DIP
12. MIDDLE_FINGER_TIP
13. RING_FINGER_MCP
14. RING_FINGER_PIP
15. RING_FINGER_DIP
16. RING_FINGER_TIP
17. PINKY_MCP
18. PINKY_PIP
19. PINKY_DIP
20. PINKY_TIP

Fonte: MediaPipe (2020)

Estes pontos de referência foram usados para criar um modelo de ML com a biblioteca TensorFlow capaz de detectar quatro gestos manuais a partir dos pontos de referência detectados. Para isso foi criado um *dataset* de treinamento contendo 3.408 conjuntos de pontos de referências relativos aos quatro gestos que desejava-se identificar.

Após o treinamento, o modelo foi adicionado ao SCASA para realizar inferência em tempo real dos gestos performados por um usuário do sistema, como mostrado nas imagens da Figura 11.

Por fim, terminados todos os passos de processamento mencionados nesta seção, o módulo de processamento de quadros envia a posição do rosto do usuário no quadro, o gesto manual identificado (caso algum esteja sendo realizado) e a posição da mão no quadro para o módulo de controle.

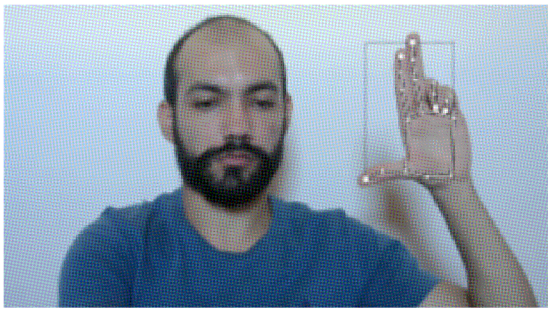
3.3.4.5 *Módulo de controle*

Implementado na classe `CameraController`, o módulo de controle recebe as informações do módulo de processamento e verifica se algum gesto manual está sendo detectado. Caso esteja, é iniciado um contador de cinco segundos. Se durante este tempo o mesmo gesto permanecer sendo detectado nos novos quadros que estão sendo processados, então o módulo de controle aplica o comando relativo ao gesto.

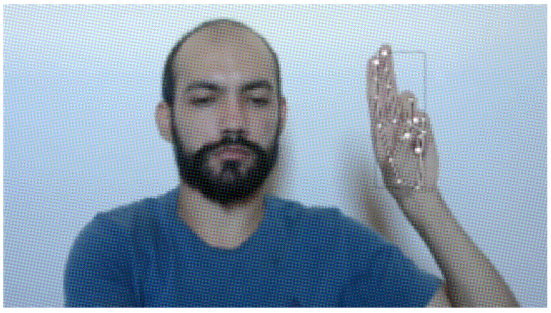
Além disso, o SCASA possui dois modos de movimentação:

- Modo I: a câmera é movimentada de forma a manter o rosto do usuário centralizado no quadro;

Figura 11 – Reconhecimento de gestos manuais realizado pelo SCASA

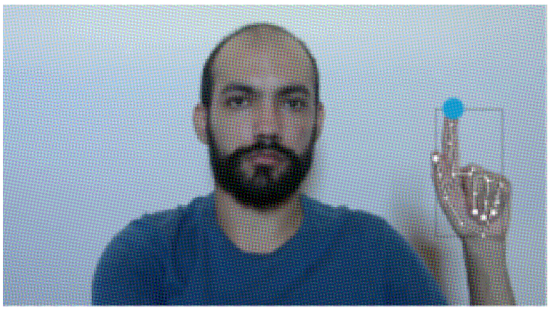


(a) Gesto para aumentar o *zoom*



(b) Gesto para diminuir o *zoom*

(c) Gesto para parar ou voltar a seguir o rosto



(d) Gesto para seguir a posição da mão

Fonte: Autoria própria

- Modo II: a câmera é movimentada de forma a manter a ponta do dedo indicador no centro da câmera caso o usuário esteja realizando o gesto mostrado na Figura 11d.

O módulo de controle calcula os comandos que devem ser enviados ao PanTilt Driver atavés do módulo de comunicação serial. Caso o sistema esteja no Modo I, estes comandos são calculados a partir da posição do rosto que aparece no quadro. Por outro lado, caso o sistema esteja no Modo II, os comandos são calculados a partir da posição da ponta do indicador quando o usuário estiver fazendo o gesto para que a posição da mão seja seguida.

3.3.4.6 *Módulo de comunicação serial*

O módulo de comunicação serial é implementado na classe `SerialMessenger`. Neste módulo, são recebidos os comandos calculados no módulo de controle para serem enviados ao Arduino Uno R3, onde está implementado o PanTilt Driver.

3.3.4.7 *Módulo de câmera virtual*

Este módulo é dividido em duas partes, um desenvolvido em Python, implementado na classe `VirtualCamServer`, e outro desenvolvido em C++ 14. Na parte desenvolvida em Python, tem-se um processo que recebe os quadros que foram processados e distribui estes através de um servidor *web socket*.

Já na parte implementada em C++, temos um *driver* implementado usando o *framework*

Directshow[3]. Este driver consiste em uma biblioteca de *link* dinâmico (DLL), que ao ser registrada no sistema operacional Windows 10 passa a a ser interpretada como uma *webcam* virtual, ou seja, que não representa um dispositivo físico real conectado ao computador.

Com isso, foi implementado dentro da DLL um cliente *web socket*, que conecta-se com o servidor escrito em Python e passa a receber os quadros enviados. Desta forma, a *webcam* virtual pode ser usada por navegadores, em aplicações de vídeo chamadas, como o Google Meet para transmitir os quadros processados pelo SCASA.

3.3.4.8 *Módulo de interface gráfica*

Este módulo está implementado na classe `GuiApplication` e é responsável por possibilitar a interação do usuário com o sistema. Uma visão geral da interface é mostrada na Figura 6. Nela estão presentes três indicadores: um indicador de gesto detectado, um indicador de tempo de detecção e um indicador de nível de *zoom*.

O indicador de gesto detectado mostra o nome do gesto manual que está sendo detectado em tempo real. Já o indicador de tempo de detecção serve para que o usuário possa acompanhar a contagem de cinco segundos até que o gestos manuais sejam considerados válidos, como explicado na seção 3.3.4.5. Por fim, o indicador de *zoom* mostra o nível de *zoom* que está sendo aplicado atualmente.

Além disso, existem alguns controles implementados na interface que podem ser acionados pelo usuário, mostrados nas imagens da Figura 12. Os botões radiais mostrados na Figura 12a são equivalentes ao gesto mostrado na Figura 11c, ou seja, ambos comandos servem para alternar entre os modos de movimentação da câmera.

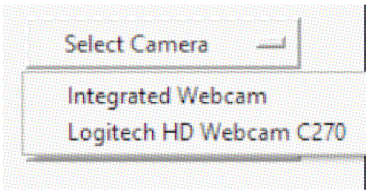
Já no menu mostrado na Figura 12b são exibidos todas as câmeras que estão conectadas ao computador onde o SCASA está sendo executado, permitindo ao usuário escolher qual a câmera que será usada pelo sistema.

E por fim, tem-se o controle mostrado na Figura 12c. Este trata-se de um menu onde são listados todos os dispositivos conetados às portas seriais do computador onde o SCASA está sendo executado. Com isso, o usuário pode selecionar o dispositivo onde está implementado o PanTilt Diver.

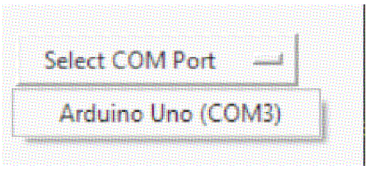
[3]Documentação Directshow: <https://docs.microsoft.com/pt-br/windows/win32/directshow/directshow>

Figura 12 – Controles da interface gráfica

(a) Botões radiais para seleção entre os modos de movimentação



(b) Menu para a seleção de câmeras



(c) Menu para seleção de portas seriais.

Fonte: Autoria própria

4 ANÁLISE E DISCUSSÃO DOS RESULTADOS

Neste capítulo são tratados os resultados dos testes da implementação realizada para o sistema proposto.

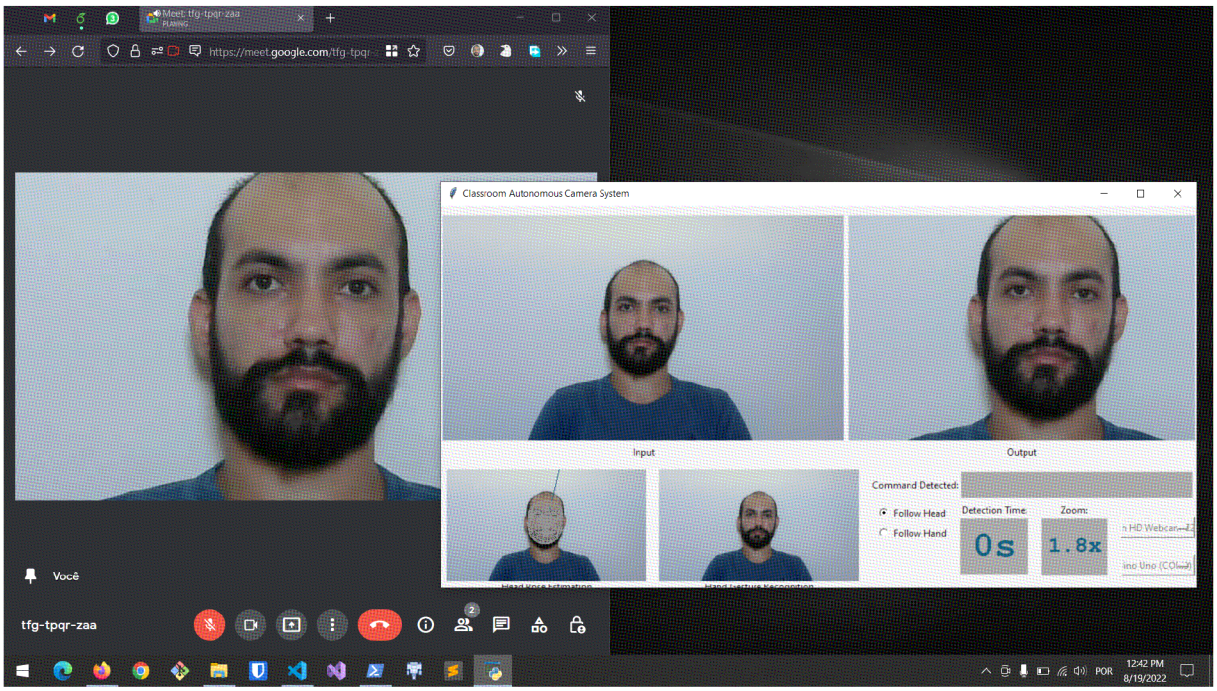
4.1 PLATAFORMA DE TESTES

O SCASA foi testado sendo executado em um computador com Windows 10 Pro, versão 10.0.19044, cujo o processador era um Intel Core i7 da 8ª geração, com 16 GB de memória RAM e SSD de 480 GB.

4.2 RESULTADOS

Os resultados obtidos foram satisfatórios, uma vez que o sistema mostrou-se razoavelmente estável sendo capaz de permanecer mais de uma hora sendo usado para transmissão em uma vídeo chamada no Google Meet (Figura 13).

Figura 13 – Captura de tela durante uma chamada no Google Meet usando a o SCASA

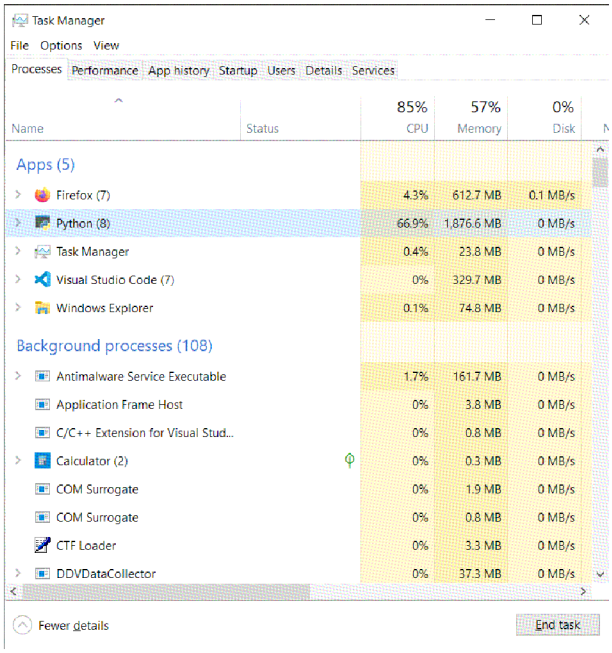


Fonte: Autoria própria.

Porém, durante estes testes verificou-se que o SCASA demanda uma grande quantidade de processamento durante sua execução, como pode ser visto na Figura 14, ficando sempre em torno dos 65%, com picos de até 70%.

Além disso, a taxa de quadro pro segundo (FPS) da câmera virtual durante uma chamada no Google Meet mostrou-se baixa, ficando em torno dos 15 FPS. Isso é devido ao alto consumo de

Figura 14 – Captura do gerenciador de tarefas durante a execução do SCASA



Fonte: Autoria própria.

processamento demandado pelo SCASA, pois, foram realizados testes usando outros programas, escritos tanto em C++ quanto em Python, onde estes enviaram quadros via *web socket* para a câmera virtual e nesses casos observou-se uma taxa de quadros superiores. Sendo assim, uma possível solução para este problema seria a otimização dos algoritmos do SCASA para que o mesmo se torne mais rápido e demande menos processamento durante sua execução, ou utilizar uma GPU para realizar o processamento das imagens e, com isso, diminuir o uso do processador.

Outro problema encontrado foi a falta de compatibilidade do driver de câmera virtual com alguns navegadores e aplicações para área de trabalho. Em navegadores como Google Chrome e Microsoft Edge não foi possível usar a câmera virtual. Apesar de a mesma aparecer disponível para uso em aplicações como o Google Meet quando aberto nestes navegadores, ao se tentar usar a câmera virtual em uma vídeo chamada, a mesma ficou constantemente exibindo quadros totalmente pretos, mesmo a aplicação SCASA sendo executada normalmente.

Durante as investigações realizadas acerca deste problema, verificou-se que este problema é causado devido uma incompatibilidade entre o *framework* Directshow e estes navegadores, o que não ocorre na navegador Mozilla Firefox. O Directshow já possui mais de dez anos de desenvolvimento e recentemente foi descontinuado e vem sendo substituído pelo *framework* Media Foundation[1]. Por isso, muitas aplicações recentes e até alguns sistemas operacionais, como o Windows 11, não oferecem mais compatibilidade com esta API. Por isso, uma possível solução para esta incompatibilidade com os navegadores seria desenvolver um novo *driver* usando o *framework* Media Foundation.

[1] Documentação Media Foundation <https://docs.microsoft.com/pt-br/windows/win32/medfound/microsoft-media-foundation-sdk>

Já em aplicações de vídeo chamada de área de trabalho, como Microsoft Teams e Zoom, a câmera virtual não aparece listada como um dispositivo para ser usado. Isso é devido à forma como estas aplicações procuram por dispositivos de vídeo no sistema. Para que a câmera virtual possa ser listada e usada em aplicações de área de trabalho, seria preciso fazer com que o *driver* desenvolvido seja listado na pilha de dispositivos Plug-and-Play do Windows, isto é possível de ser feito pois é usado em outras aplicações que geram câmeras virtuais como o OBS Studio[2] e o NVIDIA Broadcast[3]

[2]OBS Studio <https://obsproject.com/pt-br>
[3]NVIDIA Broadcast https://www.nvidia.com/pt-br/geforce/broadcasting/broadcast-app/

5 CONCLUSÃO

Com o desenvolvimento deste trabalho, foi possível obter um bom conhecimento prático sobre o processamento de imagens digitais, desenvolvimento de aplicações com multiprocessamento e desenvolvimento de *drivers* para plataformas Windows. Com isso, foi desenvolvido um protótipo de um sistema capaz de permitir que um professor movimente-se dentro de sala de aula enquanto realiza transmissões por videochamada.

Em relação a outros trabalhos semelhantes já publicados nesta área, o sistema aqui documentado apresenta contribuições por incluir uma interface gráfica, onde usuários podem controlar e interagir com a aplicação de forma mais facilitada. Além disso, foi também desenvolvido um *driver* que cria uma câmera virtual que permite o uso do sistema em videochamadas.

5.1 TRABALHOS FUTUROS

Como ficou claro no capítulo de análise e discussão dos resultados, alguns pontos do sistema requerem aprimoramentos e melhorias que podem ser exploradas em trabalhos futuros, dentre elas estão:

1. Otimizar os algoritmos da aplicação com interface gráfica de forma a reduzir a sua alta demanda de processamento;
2. Utilizar aceleração por GPU para acelerar o processamento de imagens;
3. Utilizar a estimativa da pose da cabeça do usuário para direcionar o enquadramento da câmera. De forma que caso o usuário esteja olhando para a esquerda, por exemplo, a câmera tenta enquadrar o usuário e também parte do que estiver à sua esquerda;
4. Desenvolver um novo *driver* de câmera virtual, baseado no *framework* Media Foundation, para que o mesmo possa ser usado em aplicações e sistemas operacionais mais recentes;
5. Desenvolver um novo *driver* que seja listado como dispositivo *Plug-and-Play*, de forma que a câmera virtual possa ser usada em aplicações de área de trabalho.

Referências

ARDUINO. **Arduino Uno REV3**. Arduino, 2010. Acessado em: 18 de Agosto de 2022. Disponível em: <https://store-usa.arduino.cc/products/arduino-uno-rev3?selectedStore=us>. Citado na página 8.

BARROW, H. G.; TENENBAUM, J. M. Computational vision. **Proceedings of the IEEE**, IEEE, v. 69, n. 5, p. 572–595, 1981. Citado na página 3.

BAUERMEISTER, G.; THOMSEN, A. **Micro Servo MG90S towerpro**. FilipeFlop, 2022. Acessado em: 18 de Agosto de 2022. Disponível em: <https://www.filipeflop.com/produto/micro-servo-mg90s-towerpro/>. Citado na página 8.

BUCH, N.; VELASTIN, S. A.; ORWELL, J. A review of computer vision techniques for the analysis of urban traffic. **IEEE Transactions on intelligent transportation systems**, IEEE, v. 12, n. 3, p. 920–939, 2011. Citado na página 3.

BUCK, R. Nonverbal communication of affect in children. **Journal of Personality and Social Psychology**, American Psychological Association, v. 31, n. 4, p. 644, 1975. Citado na página 4.

CHITRADEVI, B.; SRIMATHI, P. An overview on image processing techniques. **International Journal of Innovative Research in Computer and Communication Engineering**, Citeseer, v. 2, n. 11, p. 6466–6472, 2014. Citado na página 3.

GONZALEZ, R.; WOODS, R. **Digital Image Processing**. Pearson India, 2018. ISBN 9789353062989. Disponível em: <https://books.google.com.br/books?id=ODG7zQEACAAJ>. Citado na página 4.

GRAYSON, J. E. **Python and Tkinter programming**. [S.l.]: Manning Publications Co. Greenwich, 2000. Citado na página 10.

HAGHIGHI, M. S. et al. Automation of recording in smart classrooms via deep learning and bayesian maximum a posteriori estimation of instructor's pose. **IEEE Transactions on Industrial Informatics**, IEEE, v. 17, n. 4, p. 2813–2820, 2020. Citado na página 6.

LOGITECH. Logitech, 2010. Acessado em: 18 de Agosto de 2022. Disponível em: <https://www.logitech.com/pt-br/products/webcams/c270-hd-webcam.960-000694.html>. Citado na página 8.

LUGARESI, C. et al. Mediapipe: A framework for building perception pipelines. **arXiv preprint arXiv:1906.08172**, 2019. Citado na página 11.

MEDIAPIPE, G. **MediaPipe Hands**. Google, 2020. Acessado em: 16 de Agosto de 2022. Disponível em: <https://google.github.io/mediapipe/solutions/hands>. Citado na página 15.

MORENCY, L.-P.; WHITEHILL, J.; MOVELLAN, J. Generalized adaptive view-based appearance model: Integrated framework for monocular head pose estimation. In: IEEE. **2008 8th IEEE International Conference on Automatic Face & Gesture Recognition**. [S.l.], 2008. p. 1–8. Citado na página 5.

MURPHY-CHUTORIAN, E.; TRIVEDI, M. M. Head pose estimation in computer vision: A survey. **IEEE transactions on pattern analysis and machine intelligence**, IEEE, v. 31, n. 4, p. 607–626, 2008. Citado na página 4.

MURTHY, G.; JADON, R. A review of vision based hand gestures recognition. **International Journal of Information Technology and Knowledge Management**, v. 2, n. 2, p. 405–410, 2009. Citado na página 5.

OPENCV, T. **About**. 2020. Acessado em: 15 de Agosto de 2022. Disponível em: <https://opencv.org/about/>. Citado na página 10.

OUDAH, M.; AL-NAJI, A.; CHAHL, J. Hand gesture recognition based on computer vision: a review of techniques. **journal of Imaging**, MDPI, v. 6, n. 8, p. 73, 2020. Citado na página 5.

PULLI, K. et al. Real-time computer vision with opencv. **Communications of the ACM**, ACM New York, NY, USA, v. 55, n. 6, p. 61–69, 2012. Citado na página 10.

RAJA, R.; NAGASUBRAMANI, P. Impact of modern technology in education. **Journal of Applied and Advanced Research**, v. 3, n. 1, p. 33–35, 2018. Citado na página 1.

ROBOCORE. **Suporte pan/tilt para Câmera**. 2022. Acessado em: 18 de Agosto de 2022. Disponível em: <https://www.robocore.net/item-mecanico/suporte-pan-tilt-para-camera-raspberry-pi>. Citado na página 8.

ROCCA, F.; MANCAS, M.; GOSSELIN, B. Head pose estimation by perspective-n-point solution based on 2d markerless face tracking. In: SPRINGER. **International Conference on Intelligent Technologies for Interactive Entertainment**. [S.l.], 2014. p. 67–76. Citado na página 14.

SANTOS-ESPINO, J. M.; AFONSO-SUÁREZ, M. D.; GUERRA-ARTAL, C. Speakers and boards: A survey of instructional video styles in moocs. **Technical Communication**, Society for Technical Communication, v. 63, n. 2, p. 101–115, 2016. Citado na página 1.

SUTANTO, D. I. et al. Auto-tracking camera system for remote learning using face detection and hand gesture recognition based on convolutional neural network. In: IEEE. **2021 1st International Conference on Computer Science and Artificial Intelligence (ICCSAI)**. [S.l.], 2021. v. 1, p. 451–457. Citado na página 6.

WACHS, J. P. et al. Vision-based hand-gesture applications. **Communications of the ACM**, ACM New York, NY, USA, v. 54, n. 2, p. 60–71, 2011. Citado na página 5.

ZHANG, F. et al. Mediapipe hands: On-device real-time hand tracking. **arXiv preprint arXiv:2006.10214**, 2020. Citado na página 15.