VisionHead: Alternativa de uso do mouse para pessoas tetraplégicas utilizando visão computacional

José David Melo dos Santos[1], Petrucio Neves Filho[1], M. Simone Mendes Nunes[2], Marcelo Martins da Silva[1]

[1]Universidade Federal do Ceará - Brasil

[2]Universidade Federal do Pará - UFPA

{dmelo8185,petruciofilho}@alu.ufc.br

{simone.mnunes, martins2016eng}@gmail.com

Abstract. *This work develops a computer vision-based mouse control system using MediaPipe for real-time facial detection and PyAutoGUI to convert movements into cursor commands. The evaluation, based on Fitts' Law, assessed efficiency and accuracy in target selection. Results showed a sensitivity of 75%, specificity of 67.78%, and precision of 60.81%, indicating improvements in click accuracy and error reduction, though further refinements are needed to minimize false negatives.*

Resumo. *Este trabalho desenvolve um sistema de controle do mouse por visão computacional, utilizando MediaPipe para detecção facial em tempo real e PyAutoGUI para converter movimentos em comandos do cursor. A avaliação, baseada na Lei de Fitts, analisou eficiência e precisão na seleção de alvos. Os resultados mostraram sensibilidade de 75%, especificidade de 67,78% e precisão de 60,81%, indicando avanços na acurácia dos cliques e redução de erros, embora melhorias sejam necessárias para minimizar falsos negativos.*

1. Introdução

A OMS estimou que, em 2021, cerca de 15,4 milhões de pessoas viviam com lesão medular, condição geralmente causada por traumas e que resulta na perda de funções motoras [Organization 2024]. No Brasil, a PNS de 2019 indicou que 17,3 milhões de pessoas (8,4% da população) possuem deficiência, sendo quase metade idosos [Brasil 2024]. A tetraplegia impõe desafios à interação com dispositivos computacionais, exigindo ferramentas auxiliares, como bastonetes bucais, que podem dificultar a EaD e a aprendizagem [Neves Filho et al. 2023]. A baixa qualificação e o despreparo organizacional também ampliam a exclusão desse público do mercado de trabalho [Lorenzo and Silva 2017].

O avanço tecnológico tem impulsionado as Tecnologias Assistivas (TA), promovendo a inclusão e autonomia de Pessoas com Deficiência (PcDs) [Bersch 2008]. Na Interação Humano-Computador (IHC), a acessibilidade requer interfaces adaptadas [Barbosa and Silva 2010], sendo o design e a avaliação de tecnologias acessíveis essenciais para garantir usabilidade e inclusão [Melo 2014]. A visão computacional se destaca por interpretar informações visuais em tempo real, viabilizando soluções automatizadas [Klann et al. 2024]. Segundo [Karn et al. 2024], essa área combina inteligência artificial, aprendizado de máquina e engenharia. O *eye tracking* também se mostra relevante ao monitorar movimentos oculares para interação, beneficiando pessoas com

mobilidade reduzida [Fernandes 2020]. Conforme [Bergstrom and Schall 2014], essa técnica mapeia a atenção visual do usuário, promovendo inclusão digital e autonomia.

Diversas pesquisas exploram tecnologias para auxiliar tetraplégicos na interação com computadores. Em [Neves Filho et al. 2023], um dispositivo vestível detecta movimentos sutis da face para navegação sem periféricos. Outras abordagens empregam visão computacional para substituir mouse e teclado com expressões faciais, como em [Oliveira et al. 2022] e [Matoba et al. 2022], que utilizam o MediaPipe e a biblioteca DLIB. Em [Sampaio 2018], o rastreamento de cabeça, olhos e boca permite interação sem contato físico. No entanto, desafios como precisão, adaptabilidade e conforto ainda limitam a adoção dessas soluções.

Este trabalho propõe um controle de mouse para tetraplégicos baseado em visão computacional, sem dispositivos vestíveis. A solução utiliza o MediaPipe para detectar movimentos faciais e o PyAutoGUI para funções como clique por piscar de olhos, tornando a interação mais acessível. Além disso, foi comparada a uma abordagem anterior com DLIB e a API do Windows, considerando sensibilidade, especificidade e precisão.

2. Referencial Teórico

2.1. Acessibilidade e Tecnologias Assistivas

A acessibilidade refere-se à garantia de que pessoas com deficiência possam acessar lugares e recursos de forma segura e autônoma [Melo 2014] [Silva and Freire 2018]. Nos sistemas computacionais, isso envolve o desenvolvimento de interfaces que auxiliem usuários na realização de suas tarefas [Melo 2014]. As interfaces digitais frequentemente impõem barreiras à acessibilidade, como botões pequenos, menus complexos e baixo contraste visual, dificultando a interação para pessoas com deficiência motora [de Sales 2002]. Além disso, links próximos e ações que exigem múltiplos cliques aumentam os desafios na navegação [Larson et al. 2003]. A ausência de suporte a necessidades específicas, aliada a dificuldades cognitivas, visuais, auditivas e motoras, compromete a autonomia no uso de sistemas computacionais [Kotzé et al. 2004] [Microsoft 2025].

A falta de controles alternativos e personalizações, como suporte para joysticks ou comandos de voz, restringe ainda mais a experiência do usuário, tornando-a limitada e frustrante [Dusik 2013]. A Tecnologia Assistiva (TA) aplica avanços tecnológicos para restaurar e potencializar funções humanas, promovendo autonomia e qualidade de vida. De natureza interdisciplinar, envolve áreas como engenharia, saúde e educação no desenvolvimento de equipamentos e estratégias para pessoas com deficiência [Bersch 2008].

2.2. Visão Computacional e Acessibilidade

A visão computacional é um campo de pesquisa que capacita os computadores a ”ver”e a extrair informações a partir de dados de imagens. Essa área é considerada multidisciplinar, reunindo conceitos de inteligência artificial, aprendizado de máquina e diversas técnicas de engenharia e ciência da computação [Karn 2021]. O interesse nessa área tem crescido, impulsionado pelas melhorias em hardware, que proporcionam maior capacidade de processamento e memória.

A visão computacional tem desempenhado um papel importante na acessibilidade, possibilitando interações mais intuitivas para pessoas com limitações motoras[Islam et al. 2021]. Entre suas aplicações, o eyetracking se destaca ao permitir o controle de interfaces digitais por meio do movimento ocular, eliminando a necessidade de dispositivos físicos como teclados e mouses [Karn et al. 2024]. Essa tecnologia utiliza algoritmos avançados para detectar e rastrear a posição

da pupila, traduzindo os movimentos dos olhos em comandos para navegação, seleção e até mesmo digitação [Karn et al. 2024]. Combinado a inteligência artificial e técnicas de aprendizado de máquina, o eyetracking pode se adaptar às necessidades individuais dos usuários, aumentando a precisão e a usabilidade. Além disso, sua integração com assistentes virtuais e interfaces personalizáveis amplia a inclusão digital, tornando a tecnologia acessível a pessoas com tetraplegia, esclerose lateral amiotrófica (ELA) e outras condições que afetam a mobilidade [Klann et al. 2024]

3. Trabalhos Relacionados

A pesquisa de [Gomez 2007] propõe um controle por imagem baseado em lógica *fuzzy* para auxiliar tetraplégicos na interação com computadores. Usando LEDs infravermelhos, o sistema captura e interpreta movimentos oculares para gerar comandos, como cliques e seleções. A otimização dos filtros e algoritmos reduziu significativamente o tempo de processamento, embora o impacto do controlador *fuzzy* tenha sido considerado insignificante.

Em [Oliveira et al. 2022], um sistema baseado em visão computacional substitui o mouse tradicional, permitindo controle por gestos manuais e cliques pelo fechamento dos olhos, utilizando apenas uma webcam. Implementado em Python com bibliotecas como Mediapipe, Dlib e OpenCV, permite personalizar comandos. Os testes indicaram boa estabilidade, mas o rastreamento da pupila apresentou falhas devido à iluminação e posição da cabeça, afetando a segmentação da íris.

Já [Neves Filho et al. 2023] apresenta o AuriCheeks, um dispositivo vestível para navegação em desktops por pessoas com deficiência motora. Sensores inerciais no ouvido e um sensor de pressão na bochecha permitem movimentar o cursor e clicar. Baseado em Arduino Pro Micro, o sistema mostrou boa adaptação, mas exigiu ajustes na sensibilidade e no peso. Diferente das soluções por câmera, prioriza portabilidade, mas enfrenta desafios de custo e conforto.

Este trabalho se destaca por utilizar exclusivamente software livre e uma webcam comum, tornando-o acessível e replicável. Ao priorizar a movimentação facial em vez do rastreamento ocular, como o de [Oliveira et al. 2022] reduz-se a complexidade e a dependência de iluminação ideal. A aplicação de tarefas práticas baseadas na Lei de Fitts também evidencia o potencial de usabilidade, mesmo com limitações no perfil dos voluntários, Assim, enquanto os trabalhos anteriores contribuíram com inovações relevantes, este estudo busca um equilíbrio entre viabilidade técnica, acessibilidade econômica e aplicabilidade real, abrindo espaço para estudos futuros com usuários-alvo reais.

4. Metodologia

4.1. Visão Geral

O dispositivo proposto é um sistema de controle de cursor para computadores, desenvolvido especificamente para pessoas tetraplégicas. Ele utiliza uma webcam para detectar os movimentos da cabeça do usuário, permitindo a navegação pelo ambiente digital de forma intuitiva. Além disso, os cliques do mouse são acionados por meio de piscadas oculares, com o olho esquerdo correspondendo ao clique esquerdo e o direito ao clique direito. A Figura1 dispõe da visão geral de utilização do VisionHead

A identificação da posição da cabeça do usuário é um aspecto fundamental do sistema. Para isso, são empregadas técnicas de rastreamento facial baseadas em visão computacional, que capturam e analisam a direção do movimento da cabeça. Esse mecanismo garante um controle

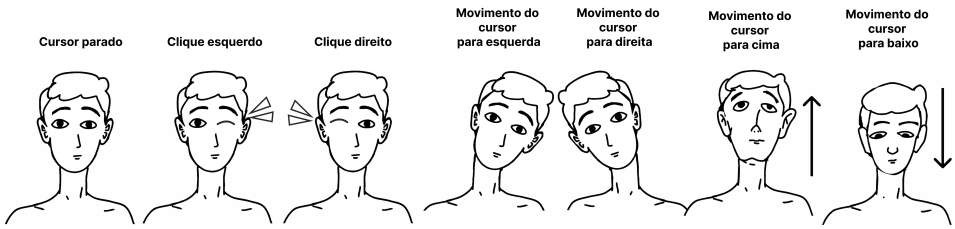


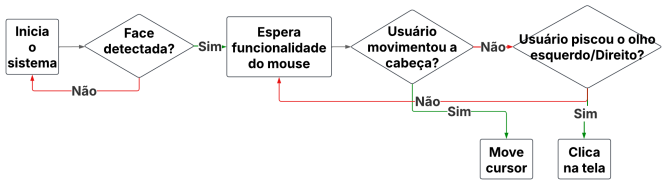
Figura 1. Visão Geral de utilização da Tecnologia

preciso do cursor, reduzindo movimentos involuntários e proporcionando uma experiência de uso mais confortável e eficiente. A movimentação do cursor é determinada pela análise da posição da face do usuário. O sistema utiliza um modelo de rastreamento facial que identifica pontos-chave do rosto e calcula a inclinação da cabeça. Dessa forma, os deslocamentos laterais e verticais da cabeça são mapeados para movimentos correspondentes do cursor na tela, assegurando uma interação fluida e natural. Os cliques são acionados por meio da detecção de piscadas, empregando algoritmos que distinguem fechamentos voluntários dos olhos das piscadas reflexas. Isso garante que apenas comandos intencionais sejam registrados, evitando cliques acidentais e melhorando a precisão do sistema.

4.2. Implementação

Para a implementação do sistema VisionHead, optou-se pela linguagem Python, amplamente utilizada em projetos de Visão Computacional devido à sua versatilidade e vasto ecossistema de bibliotecas. Desenvolvemos um algoritmo capaz de replicar as funcionalidades do mouse por meio do rastreamento dos movimentos faciais e da detecção de piscadas oculares, seguindo o fluxo apresentado na Figura 2.

Figura 2. Fluxograma do Sistema



O algoritmo inicia com a configuração do sistema, carregamento das bibliotecas necessárias e definição das variáveis essenciais. Após a inicialização, o sistema tenta detectar uma face por meio da webcam do usuário. Caso uma face seja detectada, o sistema aguarda uma ação relacionada ao mouse. A partir desse momento, duas situações podem ocorrer: **1)** Caso o usuário mova a cabeça, a função de movimentação do cursor é ativada. **2)** Caso o usuário pisque o olho esquerdo ou direito, a função de clique na tela é acionada.

4.3. Desenvolvimento

O desenvolvimento deste trabalho foi estruturado em duas etapas principais. A primeira foi dedicada à detecção da face do usuário, onde foi realizada uma comparação entre uma solução anterior, que utilizava ferramentas menos robustas, e uma abordagem mais moderna, que empregou tecnologias atualizadas. A segunda etapa metodológica consistiu na execução de testes do sistema

e na análise dos resultados obtidos. Para avaliar a usabilidade e a eficiência da solução, os testes foram baseados no Teste de Fitts (*Fitts' Law Test*), um modelo amplamente utilizado para medir o desempenho de interfaces de entrada, que analisa a relação entre a distância do movimento e a precisão na seleção de alvos.

4.4. Detecção da Face do Usuário

Para a detecção da face do usuário, neste sistema, foi utilizado o framework MediaPipe, em conjunto com a biblioteca OpenCV, com o objetivo de capturar os frames faciais por meio da webcam do dispositivo (desktop ou notebook) e identificar contornos oculares para a detecção de piscadas.

O MediaPipe foi escolhido por sua eficiência e precisão na detecção de pontos faciais em tempo real, sendo uma solução otimizada para diferentes dispositivos, incluindo aqueles com recursos computacionais limitados. Sua integração simplificada com Python também facilita o desenvolvimento de aplicações interativas. A biblioteca OpenCV, amplamente utilizada na comunidade de visão computacional, oferece ferramentas robustas para manipulação e processamento de imagens, permitindo a aplicação de filtros e a extração de características relevantes. A combinação dessas tecnologias proporciona um sistema leve, rápido e acessível para a implementação da solução proposta. Para fins de comparação e análise de desempenho, foram implementadas duas versões distintas do sistema. A primeira versão foi baseada na biblioteca DLIB para detecção facial e utilizou a função SetCursorPos da API do Windows para realizar o movimento do cursor, com a ativação do clique sendo feita por meio da detecção do piscar do olho. Esta versão inicial utilizou pontos faciais 2D, e o controle do cursor foi diretamente vinculado aos movimentos da face, com o clique sendo acionado pelo movimento das pálpebras durante o piscar do olho.

A segunda versão, desenvolvida como uma abordagem alternativa, utilizou o framework MediaPipe em conjunto com a biblioteca PyAutoGUI. Nessa versão, a detecção facial foi realizada com o modelo Face Mesh do MediaPipe, que fornece um conjunto de 468 pontos (landmarks) que detalham a face de forma mais precisa. O movimento do cursor foi gerenciado pela função *moveTo* da biblioteca PyAutoGUI, enquanto o clique foi acionado pela detecção do piscar do olho.

4.4.1. Versão do Sistema com DLIB e API do Windows

Na primeira versão, utilizou-se a biblioteca DLIB para detectar os pontos faciais e a função *SetCursorPos* da API do Windows para controlar o cursor. A função *SetCursorPos* move o cursor para as coordenadas especificadas na tela, e o controle do movimento do cursor era feito com base nos deslocamentos da face. O clique era realizado pela detecção do piscar do olho esquerdo ou direito, identificado pela aproximação das pálpebras. A detecção da face foi realizada por meio de pontos faciais 2D, utilizando o algoritmo Landmark para fornecer uma posição aproximada dos principais pontos da face. Na figura 3, é possível observar os 68 pontos na face do usuário, utilizando a técnica de Landmark 2D.

Figura 3. Marcações dos 68 pontos na face pela biblioteca DLIB.

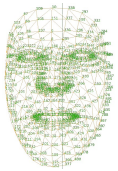
Fonte:[Practical CV 2024]

A função disponível `get_frontal_face_detector` carrega um detector de faces baseado no algoritmo *Histogram of Oriented Gradients (HOG)* combinado com uma *Support Vector Machine (SVM)*. O detector identifica a posição aproximada do rosto do usuário na imagem e, após detectar o rosto, o preditor utiliza um modelo pré-treinado para localizar os 68 pontos faciais de referência. Já a função `SetCursorPos` da API do Windows ajusta a posição do ponteiro de acordo com a proporção da tela, permitindo que o usuário controle o cursor apenas movimentando a cabeça. Esse método proporciona acessibilidade para pessoas com dificuldades motoras severas, permitindo a navegação no ambiente digital sem uso das mãos.

4.4.2. Versão do Sistema com MediaPipe e PyAutoGUI

Na segunda versão, foi utilizado o MediaPipe, que possibilita uma detecção mais detalhada da face através de 468 pontos faciais, proporcionando um controle mais refinado do cursor. A movimentação do cursor foi feita pela função *moveTo* da biblioteca PyAutoGUI, que oferece mais flexibilidade e precisão no controle do cursor. A detecção do piscar do olho foi mantida, mas com a vantagem de uma detecção mais precisa da região dos olhos, proporcionada pela técnica de Face Mesh. Na figura 4, é possível observar os 468 pontos utilizando a técnica de Face Mesh.

Figura 4. 468 pontos na face humana com Face Mesh



Fonte: [AI 2025]

No código desenvolvido para o sistema, a função `process_face_landmarks` processa pontos faciais detectados, calculando interações e movendo o cursor do mouse com base na posição do nariz. A função desenha um contorno ao redor do olho esquerdo usando os pontos faciais definidos e realiza um clique do mouse quando a distância entre as pálpebras superior e inferior é pequena, simulando um piscar. A posição do nariz é mapeada para coordenadas da tela, movendo o cursor para essa posição. Essa abordagem permite controlar o cursor com movimentos faciais, oferecendo uma solução de acessibilidade.

Para detectar o clique simples do mouse por meio do piscar do olho esquerdo, foi necessário extrair os pontos centrais das pálpebras superior e inferior. A partir desses pontos, calculou-se a distância euclidiana entre as extremidades das pálpebras, permitindo quantificar a aproximação entre elas durante o ato de piscar. Essa distância foi então comparada a um limite previamente definido (*threshold*). Quando o valor calculado fica abaixo desse limite, o sistema interpreta como um piscar intencional e aciona o comando de clique simples. Esse método proporciona uma resposta precisa e eficiente aos movimentos faciais do usuário. A fórmula utilizada para o cálculo da distância euclidiana está apresentada na equação 1. Detalhes adicionais sobre o desenvolvimento podem ser consultados no repositório do projeto VisionHead, disponível em [Melo 2025].

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \tag{1}$$

Na aplicação, X1 e X2 representam as coordenadas x, e Y1 e Y2 as coordenadas y dos pontos centrais das pálpebras superior e inferior. Duas versões tecnológicas foram comparadas quanto à precisão do movimento do cursor e à taxa de cliques, utilizando a mesma metodologia.

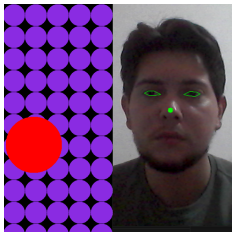
4.5. Experimentos

A segunda etapa do desenvolvimento consistiu na realização de testes do sistema e na análise dos resultados obtidos. Para avaliar desempenho da solução, os testes foram baseados no Teste de Fitts (*Fitts' Law Test*, um modelo amplamente utilizado para medir o desempenho de interfaces de entrada, analisando a relação entre a distância do movimento e a precisão da seleção de alvos. Os testes foram realizados em laboratório, com ambiente controlado para garantir reprodutibilidade. Participaram quatro voluntários sem deficiência (dois homens e duas mulheres), entre 20 e 45 anos, com alturas de 1,60m a 1,87m. Optou-se por recrutar participantes sem deficiência devido à dificuldade inicial de acesso ao público-alvo. Nesta fase preliminar, o foco foi validar o funcionamento do sistema e coletar dados em condições controladas.

Essa abordagem é comum em pesquisas similares, onde os testes com usuários com deficiência ocorrem em etapas posteriores, após a verificação da viabilidade técnica e da usabilidade. Estudos futuros contemplarão testes com o público-alvo para validar a eficácia do sistema em contextos reais. A seleção dos quatro voluntários, sem deficiência, visou avaliar o desempenho dos cliques no sistema e identificar possíveis ajustes na calibragem antes de sua aplicação em usuários com deficiência. Os experimentos seguiram as resoluções 466/2012 e 510/2016 do CNS (Conselho Nacional de Saúde), com consentimento assinado por todos. Os testes avaliaram o clique simples (via piscar de olho) e a movimentação do cursor conforme os deslocamentos faciais. Foram observados a precisão do rastreamento, tempo de resposta e adaptação do usuário, gerando insights para ajustes de sensibilidade e calibração dos comandos.

Na implementação com MediaPipe Face Mesh, foi necessário escolher um ponto central na face do usuário para garantir um controle eficiente do cursor ao movimentar a cabeça. Optou-se pelo ponto 4, localizado no nariz, por ser o ponto mais central da face humana. Com base nisso, utilizou-se a biblioteca PyAutoGUI, especificamente o método *moveTo*, para converter os movimentos da cabeça do usuário em deslocamentos do cursor, permitindo que o cursor acompanhe a posição do nariz durante os movimentos. A figura 5 monstra como se observa estas configurações no rosto do usuário.

Figura 5. Simulação de Movimentação de Mouse por Software



É possível identificar a marcação do landmark 4, facilitando a identificação precisa de sua localização no rosto. Além disso, a figura ilustra o teste do sistema, onde o usuário tenta mover o cursor até o alvo e realizar o clique. Para a análise dos resultados, foi utilizado um software capaz de capturar os dados das interações dos usuários no formato JSON. A análise dos registros foi realizada com a linguagem Python, empregando ferramentas específicas para cada

etapa do processo. A biblioteca Pandas foi utilizada para o processamento e manipulação eficiente dos dados, enquanto as bibliotecas Matplotlib e Seaborn foram empregadas para a criação de visualizações gráficas detalhadas, facilitando a interpretação dos resultados.

5. Resultados e Discussões

Extraindo índices de desempenho do sistema nas duas situações abordadas, a primeira situação com a utilização da biblioteca DLIB para detecção facial e a função *SetCursorPos* da API do Windows para converter piscadas oculares em cliques do mouse e uma segunda, onde foi utilizado o framework MediaPipe em conjunto com a biblioteca PyAutoGUI para conversão das piscadas oculares em cliques é possível compararmos e encontrarmos diferenças entre as abordagens para melhorias. Para isso, utilizamos da técnica de matriz de confusão que resume os resultados dos experimentos realizados, como mostram as tabelas 1 e 2.

Utilizamos um software de Teste de Fitts disposto no GitHub[1] para gerar os alvos: os participantes deviam acertar o alvo, e cada alvo mudava de tamanho e posição de acordo com o Clique. A primeira situação envolvendo o uso da DLIB e API do Windows teve os seguintes resultados: Amostragem: 1 alvo que muda de tamanho e posição, 150 cliques - **Sensibilidade**: $TP/(TP + FN) \rightarrow 27/(27 + 33) = 0,45 \rightarrow 45\%$ - **Especificidade**: $TN/(TN + FP) \rightarrow 52/(52 + 38) = 0,5778 \rightarrow 57,78\%$ - **Precisão**: $TP/(TP + FP) \rightarrow 27/(27 + 38) = 0,4154 \rightarrow 41,54\%$

Onde: **TP (Verdadeiro Positivo)**: O participante clicou e o alvo realmente mudou; **FN (Falso Negativo)**: O participante não clicou, mas o alvo mudou; **FP (Falso Positivo)**: O participante clicou, mas o alvo não mudou; **TN (Verdadeiro Negativo)**: O participante não clicou e o alvo também não mudou.

Tabela 1. Matriz de Confusão Utilizando DLIB e API do Windows

Situação Real	Alvo	Não Alvo	Total
Alvo	VP: 27 (True Positive)	FN: 33 (False Negative)	60
Não Alvo	FP: 38 (False Positive)	VN: 52 (True Negative)	90
Total	65	85	150

Durante os 150 testes realizados, os cliques foram detectados em algumas situações, mas um número significativo de falsos negativos (FN) foi observado, o que indica que, em diversas ocasiões, o clique não foi registrado corretamente. Esse problema pode estar relacionado a uma lentidão na detecção do clique, talvez por parte da biblioteca DLIB ou API do Windows, dificuldades na detecção do clique, como uma falha na identificação precisa da posição dos olhos ou pela qualidade de iluminação do ambiente, que pode prejudicar a captura das interações. Além disso, a ocorrência de falsos positivos (FP) sugere que o sistema registrou cliques em momentos em que não deveria registrar, indicando uma possível sensibilidade excessiva do sistema, capturando movimentos que não correspondem a um clique real. Esses fatores juntos contribuem para um desempenho inferior, refletido na sensibilidade de 45%, especificidade de 57.78% e precisão de 41.54%.

Já no segundo momento dos testes, os resultados mostraram uma melhoria significativa na detecção de cliques: Amostragem: 1 alvo que muda de tamanho e posição, 150 cliques: - **Sensibilidade**: $TP/(TP+FN) \rightarrow 45/(45+15) = 0,75 \rightarrow 75\%$ - **Especificidade**: $TN/(TN+$

[1]https://github.com/topics/fitts-law

$FP) \rightarrow 61/(61+29) = 0,6778 \rightarrow 67,78\%$ - **Precisão**: $TP/(TP+FP) \rightarrow 45/(45+29) = 0,6081 \rightarrow 60,81\%$

Tabela 2. Matriz de Confusão Utilizando MediaPipe e PyAutoGUI

Situação Real	Alvo	Não Alvo	Total
Alvo	VP: 45 (True Positive)	FN: 15 (False Negative)	60
Não Alvo	FP: 29 (False Positive)	VN: 61 (True Negative)	90
Total	74	76	150

A sensibilidade aumentou para 75%, indicando que uma maior proporção de cliques foi registrada corretamente, minimizando os falsos negativos (FN). Essa melhoria pode ser atribuída ao aprimoramento do desempenho do sistema, tornando a detecção do piscar dos olhos mais eficiente e precisa. A especificidade também apresentou um avanço, alcançando 67,78%, o que sugere uma redução na quantidade de falsos positivos (FP), ou seja, menos cliques foram erroneamente registrados quando não houve alteração do alvo. A precisão, que atingiu 60,81%, demonstrou uma boa taxa de acerto entre os cliques detectados e as interações reais, refletindo um desempenho geral mais eficaz do sistema. Esses resultados evidenciam um avanço significativo na acurácia da detecção, embora ainda haja espaço para melhorias, especialmente na redução dos falsos negativos. Apesar dos bons resultados técnicos, métricas de usabilidade, como tempo de tarefa, carga cognitiva e satisfação, não foram avaliadas. Futuramente, pretende-se realizar testes qualitativos com usuários para aprimorar a experiência de uso.

5.1. Análise dos Dados Utilizando DLIB e API do Windows

Realizamos uma primeira experiência focada na posição do cursor, com o objetivo de identificar movimentos nas seguintes direções: frente, cima, baixo, direita e esquerda, além de analisar os cliques em cada posição. Para capturar os dados dos usuários, utilizamos a interface do Visual Studio Code, armazenando as informações no formato JSON. A análise dos logs de interação foi feita com a linguagem Python, empregando as seguintes bibliotecas: Pandas para processamento e manipulação dos dados, Matplotlib e Seaborn para visualização gráfica.

Durante essa análise, observamos que, em todos os casos, houve ruídos e uma trajetória instável do cursor antes de alcançar o alvo de clique. Esse comportamento pode ser explicado pela natureza da biblioteca DLIB e das funcionalidade da API do Windows, que exigem um alto poder de processamento e um ambiente bem iluminado para realizar detecções precisas. A ausência de uma ferramenta mais robusta nesse aspecto pode resultar em informações inconsistentes, causando tremores no cursor ou até mesmo dificultando seu controle. Para avaliar os resultados, utilizamos a funcionalidade de gráficos de dispersão da Seaborn, o que nos permitiu visualizar os dados capturados durante o teste inicial. Nesse teste, a pessoa utilizando o protótipo olhava para frente e deveria realizar um clique no alvo. A Figura 6 ilustra o gráfico gerado, evidenciando as variações nos movimentos do cursor.

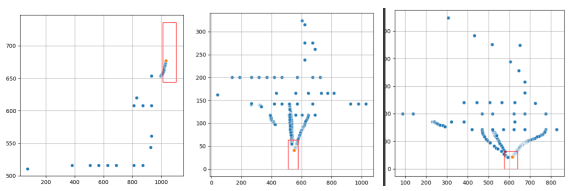


Figura 6. Saída do sistema utilizando a DLIB, na situação em que a pessoa está sentada olhando para a tela

É importante ressaltar que a presença desses ruídos e a inconsistência na captura dos movimentos podem impactar negativamente a experiência do usuário. O tremor excessivo no cursor compromete a precisão do controle, dificultando a interação com o sistema. Portanto, o uso de bibliotecas de detecção facial mais avançadas é essencial para garantir uma captura mais estável e confiável dos movimentos da cabeça, melhorando assim a usabilidade do VisionHead.

5.2. Análise dos Dados Utilizando MediaPipe e PyAutoGUI

Realizamos uma segunda experiência, desta vez utilizando o framework MediaPipe em conjunto com a biblioteca PyAutoGUI para a detecção dos cliques. O ambiente e a metodologia de avaliação permaneceram os mesmos adotados na análise do modelo anterior. Durante a análise, observamos que o novo sistema apresentou um desempenho mais estável, com menos ruídos e uma trajetória do cursor mais fluida e precisa. Essa melhoria pode ser atribuída ao uso do MediaPipe, um framework mais eficiente e leve em comparação à DLIB para detecção facial. Além disso, a substituição das funções da API do Windows pela biblioteca PyAutoGUI contribuiu para um sistema mais enxuto e responsivo, como mostrado nos resultados das três situações na figura 7.

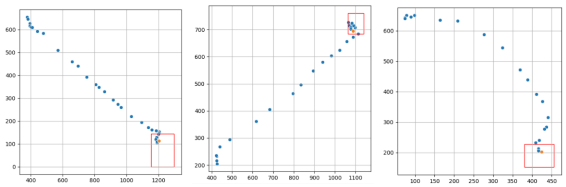


Figura 7. Saída do sistema utilizando a MediaPipe

6. Considerações Finais

Diante dos desafios enfrentados por pessoas com tetraplegia no acesso ao meio digital, observa-se que muitas soluções existentes apresentam baixa usabilidade, métodos de interação desconfortáveis e custos elevados, o que limita seu alcance. A revisão bibliográfica evidenciou avanços significativos no desenvolvimento de Tecnologias Assistivas, especialmente quando associadas à Visão Computacional, promovendo inovações em dispositivos de uso cotidiano, conforme demonstrado por estudos como [Gomez 2007, Oliveira et al. 2022, Sampaio 2018]. Nesse contexto, este trabalho apresenta o VisionHead, uma solução voltada à melhoria da navegação em desktops e notebooks por pessoas tetraplégicas, utilizando algoritmos robustos de Visão Computacional por meio do framework MediaPipe, além de bibliotecas como OpenCV e PyAutoGUI.

Durante a pesquisa, testes foram realizados com voluntários sem deficiência, que controlaram o mouse sem o uso das mãos. Quatro participantes sem familiaridade prévia com o sistema permitiram avaliar a curva de aprendizado e melhorias contínuas no uso. Embora pessoas com tetraplegia não tenham sido incluídas, a análise de desempenho ajudou a refinar o sistema. Entre as principais contribuições, destaca-se o desenvolvimento de uma solução acessível para navegação digital de usuários tetraplégicos, estendendo-se a pessoas sem mobilidade nos membros superiores ou com dificuldades motoras. A tecnologia permite acesso a plataformas como web e redes sociais sem dispositivos invasivos, tornando a acessibilidade mais eficiente.

As principais contribuições do trabalho incluem: (1) o desenvolvimento e avaliação de um sistema de controle de cursor por movimentos oculares, com foco em acessibilidade para pessoas com tetraplegia; (2) a comparação de duas abordagens técnicas — DLIB com API do Windows

e MediaPipe com PyAutoGUI — evidenciando ganhos em sensibilidade, precisão e estabilidade com a segunda; (3) a aplicação de métodos de avaliação quantitativa (como matrizes de confusão) e análise de logs para medir desempenho; e (4) a proposta de uma solução acessível, de baixo custo e não invasiva, que alia Visão Computacional à IHC de forma interdisciplinar, com potencial de ampliar a inclusão digital. Destaca-se a integração entre Visão Computacional e IHC como um motor de avanço para tecnologias assistivas, viabilizando interfaces que interpretam gestos e movimentos oculares como formas de controle. Essa colaboração interdisciplinar potencializa a inclusão de pessoas com limitações motoras, ao aliar processamento inteligente de movimentos a princípios de usabilidade centrada no usuário.

Ao analisar os resultados dos testes, considerando limitações identificadas no estudo destacamos que posição do usuário dificultou a detecção do clique, assim como a iluminação inadequada, especialmente em ambientes escuros, onde o fechamento das pálpebras pode não ser identificado. Além disso, a distância excessiva da tela gerou inconsistências na detecção.

Este trabalho, ainda em desenvolvimento, prevê aprimoramentos como duplo clique, arrastar, rolagem, zoom e inicialização autônoma. A avaliação considerará variáveis como iluminação, presença de terceiros e acessórios, buscando movimentos mais confortáveis aos usuários. Futuras versões poderão incorporar Inteligência Artificial e diretrizes WAI-ARIA, um novo protocolo de avaliação com usuários finais focados em desempenho, confiabilidade e acessibilidade.

Referências

AI, G. (2025). Mediapipe face mesh solution map. *ResearchGate*. Accessed: 2025-03-07.

Barbosa, S. and Silva, B. (2010). *Interação humano-computador*. Elsevier Brasil.

Bergstrom, J. R. and Schall, A. (2014). *Eye Tracking in User Experience Design*. Elsevier.

Bersch, R. (2008). Introdução à tecnologia assistiva. *Porto Alegre: CEDI*, 21.

Brasil, C. (2024). Brasil tem mais de 17 milhões de pessoas com deficiência, segundo ibge. `https://www.cnnbrasil.com.br/nacional/brasil-tem-mais-de-17-milhoes-de-pessoas-com-deficiencia-segundo-ibge/#:~:text=Perfil%20de%20quem%20tem%20defici%C3%AAncia,%25%20pardas%20e%208%25%20brancas`. Acesso em: 16 dez. 2024.

de Sales, M. B. (2002). *Desenvolvimento de um checklist para a avaliação de acessibilidade da web para usuários idosos*. PhD thesis, Universidade Federal de Santa Catarina, Centro Tecnológico. Programa de Pós ....

Dusik, C. L. (2013). Teclado virtual silábico-alfabético: tecnologia assistiva para pessoas com deficiência física.

Fernandes, K. C. d. S. (2020). Tecnologia de rastreamento ocular como ferramenta auxiliar para avaliação audiológica básica de pessoas com múltiplas deficiências no sus.

Gomez, B. A. (2007). *Sistema de controle por imagem para indivíduos tetraplégicos ou com lesões cerebrais*. PhD thesis, Universidade de São Paulo.

Islam, R., Rahman, S., and Sarkar, A. (2021). Computer vision based eye gaze controlled virtual keyboard for people with quadriplegia. In *2021 International Conference on Automation, Control and Mechatronics for Industry 4.0 (ACMI)*, pages 1–6. IEEE.

Karn, A. (2021). Artificial intelligence in computer vision. *International Journal of Engineering Applied Sciences and Technology*, 6(1):249–254.

Karn, A., Mehta, R., Hiriyanna, G., Sayyed Johar, K., Chhabra, A., Ty, C., and Rajahrajasingh, H. (2024). Artificial intelligence in computer vision. *SUSTAINABLE DEVELOPMENT THROUGH MACHINE LEARNING, AI AND IOT: Second*, page 102.

Klann, D., da Rocha Fernandes, A. M., da Silva, E. A., and Parreira, W. D. (2024). Explorando algoritmos de visão computacional em tecnologias assistivas: uma revisão sistemática da literatura. *Journal of Health Informatics*, 16(Especial).

Kotzé, P., Eloff, M., Adesina-Ojo, A., and Eloff, J. (2004). Accessible computer interaction for people with disabilities the case of quadriplegics. In *ICEIS 2004-Proceedings of the Sixth International Conference on Enterprise Information Systems*.

Larson, H., Gips, J., Hall, F., and Stephanidis, C. (2003). A web browser for people with quadrip legia. In *HCI (4)*, pages 226–230.

Lorenzo, S. M. and Silva, N. R. (2017). Contratação de pessoas com deficiência nas empresas na perspectiva dos profissionais de recursos humanos. *Revista Brasileira de Educação Especial*, 23:345–360.

Matoba, F., Pedrini, H., Técnico-IC-PFG, R., and de Graduação, P. F. (2022). Controle de mouse baseado em rastreamento ocular.

Melo, A. M. (2014). Acessibilidade e inclusão digital. *Livro dos Tutoriais do XIII Simpósio Brasileiro sobre Fatores Humanos em Sistemas Computacionais*, pages 29–54.

Melo, D. (2025). Visionhead - controle de mouse com movimentos da cabeça e piscadas. `https://github.com/Dvt55/VisionHead`. Acesso em: 24 maio 2025.

Microsoft (2025). Accessibility - microsoft. `https://www.microsoft.com/en-us/accessibility/`. Accessed: 2025-02-03.

Neves Filho, P. D. C., Feitosa, C. E. A., Barreto, J. S., and Silva, M. M. d. (2023). Auricheeks: Assistive technology to assist in desktop´ s navigation. In *Proceedings of the XXII Brazilian Symposium on Human Factors in Computing Systems*, pages 1–11.

Oliveira, F. V. et al. (2022). Avaliação de alternativas ao mouse utilizando técnicas da visão computacional.

Organization, W. H. (2024). Spinal cord injury - fact sheet. `https://www.who.int/news-room/fact-sheets/detail/spinal-cord-injury`. Acesso em: 16 dez. 2024.

Practical CV (2024). Facial landmarks detection with dlib. `https://github.com/Practical-CV/Facial-Landmarks-Detection-with-DLIB`. Acessado em: 04 nov. 2024.

Sampaio, G. S. (2018). Desenvolvimento de uma interface computacional natural para pessoas com deficiência motora baseada em visão computacional.

Silva, C. A. and Freire, A. P. (2018). Inspeção da acessibilidade de aplicativos móveis utilizando software leitor de telas. In *Anais Estendidos do XVII Simpósio Brasileiro sobre Fatores Humanos em Sistemas Computacionais*. SBC.