UNEB - Universidade do Estado da Bahia

DCET - Campus I - Salvador

Sistemas de Informação

Desenvolvimento de um Sistema de Reconhecimento de Gestos para Subistituição do Mouse e Mousepad em Laptops

Alunos: Aurelicio dos Santos Pereira e Hélio José Ferreira Soares Filho
Professor: Alexandre Rafael Lenz

Outubro
2023

Universidade do Estado da Bahia
Departamento de Ciências Exatas e da Terra
Sistemas de Informação

Relatório

Desenvolvimento de um Sistema de Reconhecimento de Gestos para Subistituição do Mouse e Mousepad em Laptop do Curso Sistemas de Informação da Universidade do Estado da Bahia, como requisito parcial para conclusão da disciplina.

Alunos: Aurelicio dos Santos Pereira, Hélio José Ferreira Soares Filho

Professor: Alexandre Rafael Lenz

Outubro
2023

Contents

1 Problema

A utilização de periféricos, como mouses e mousepads, muitas vezes se revela um desafio para diversos indivíduos, seja devido a limitações físicas pessoais ou à inconveniência de transportar e configurar esses dispositivos, que frequentemente requerem espaço e esforço adicionais. Enquanto outras áreas da tecnologia têm visto avanços significativos, especialmente no que diz respeito aos drivers, o cenário tem sido diferente quando se trata da interação com computadores, tanto laptops quanto desktops. A introdução de dispositivos com tela sensível ao toque revolucionou a forma como interagimos com smartphones e tablets, mas, curiosamente, a interação com computadores pessoais ainda exige o uso de "intermediários" como o mouse e o teclado. Imagine se fosse possível utilizar seu computador sem depender desses periféricos convencionais. E se você pudesse simplesmente usar gestos naturais das mãos para controlar o cursor ou rolar a tela, tudo sem sequer tocar no monitor? A solução que desenvolvemos foi criada com o objetivo de abordar essa questão, aproveitando a câmera já incorporada à maioria dos notebooks modernos. Com essa abordagem inovadora, você pode usar gestos das mãos para simular as ações realizadas com os periféricos tradicionais, eliminando a necessidade de carregar dispositivos físicos adicionais e simplificando consideravelmente a forma como você interage com o seu sistema. Em vez de depender de objetos externos, agora você pode simplesmente estender a mão e controlar intuitivamente seu computador, tornando a experiência de computação mais acessível, conveniente e natural.

2 Objetivo geral

O propósito fundamental deste projeto é criar um protótipo inovador que ofereça uma alternativa ao uso exclusivo do mouse ou do mousepad como meio de interação entre o usuário e o sistema operacional. Para isso, empregamos câmeras webcam incorporadas em laptops, permitindo o reconhecimento de gestos que mimetizam os movimentos do cursor do mouse.

3 Objetivos específicos

Implementar um sistema de captura de vídeo para adquirir imagens da câmera. Aplicar uma técnica de mudança da crominância imagem, para escala de cinza para melhor identificação do contorno das mãos Detectar e rastrear gestos de mão na imagem em tempo real. Atribuir a esses gestos operações equivalentes às do mouse, a navegação do ponteiro como cliques esquerdo e direito, arrastar e soltar. Avaliar a precisão e o desempenho do sistema de reconhecimento de gestos em diferentes cenários e com diferentes pessoas. Desenvolver uma interface de usuário para configurar e controlar o sistema.

4 Metodologia

Utilização da biblioteca OpenCV para capturar e processar vídeo em tempo real. Implementação da técnica de subtração de fundo para destacar gestos. Detecção de contornos e cálculo de áreas para identificar gestos de interesse. Simulação de operações do mouse com base na posição e gestos detectados. Avaliação do sistema por meio de testes com usuários para verificar a precisão e a usabilidade do sistema.

5 Relevância

A importância deste projeto reside na crescente demanda por interações computacionais mais intuitivas e acessíveis. A utilização de periféricos tradicionais, como mouse e mousepad, pode ser limitante para muitos usuários, especialmente aqueles com mobilidade reduzida ou que necessitam de acessibilidade. O reconhecimento de gestos oferece uma alternativa promissora que simplifica a interação com o computador, tornando-a mais eficiente e acessível para um público mais amplo.

6 Justificativa

A justificativa para este pré-projeto é baseada na necessidade de superar as limitações dos dispositivos de entrada convencionais. A utilização de câmeras já presentes em notebooks para o reconhecimento de gestos representa uma solução eficiente e econômica. Além disso, o projeto visa preencher uma lacuna na pesquisa de interações naturais com computadores, explorando o potencial do reconhecimento de gestos para simular operações do mouse. Uma vez implementado com sucesso, o sistema proposto pode ter aplicações em diversos campos, incluindo acessibilidade, entretenimento e design de interfaces de usuário mais intuitivas, contribuindo assim para o avanço da pesquisa em interações computacionais inovadoras.

7 Validação

A validação do sistema será realizada por meio de testes com usuários. Os usuários serão convidados a interagir com o sistema de reconhecimento de gestos em cenários controlados e tarefas específicas que envolvem operações do mouse. Serão coletados dados sobre a precisão do reconhecimento de gestos, a eficácia na simulação das operações do mouse e a usabilidade geral do sistema. Os resultados dos testes serão analisados e comparados com métricas pré-definidas para avaliar o desempenho do sistema e identificar áreas de melhoria.

8 Relatório final

A realização desse projeto se deu repleta de desafios técnicos, dos quais alguns foram solucionados e outros, ficaram como pendentes de solução para trabalhos futuros. Entretanto, de certa forma, foram esse desafios que ditaram o ritmo e o roteiro desse projeto, pois a medida que avançávamos em suas resoluções eles nos mostravam novos rumos e novas demandas necessárias para implementação, além daquelas previstas no nosso plano de ação, por se tratar de uma área completamente nova para nós.
Resumimos esses desafios em duas fases: Captura e tratamento da imagem; Reconhecimento de gestos e associação destes às funcionalidades do mouse.
Após o reconhecimento da linguagem *Python* como a melhor opção para a construção do sistema por sua diversidade de bliotecas para o uso da visão computacional, utilizamos a biblioteca *OpenCV 2* para inicializarmos a captura da imagem pela webcam e identificação de formas.
Entretanto, com a realização da captura de vídeo decidimos por transformar a imagem em escala de cinza, uma vez que trabalhar a imagem em cores não estava sendo muito eficiente. A imagem na escala de cinza, se mostrou muito melhor para o reconhecimento do contorno das mãos, facilitando a identificação de formas por parte do algorítmo, enquanto que em escala de cores o estava se confundindo as mãos com a cor de fundo.
Outra situação que identificamos foi que a era necessário estabelecer as configurações da resolução da webcam como parâmetros para o algorítmo de reconhecimento de gestos, o que nos leva a conclusão que cada câmera exigirá uma configuração específica.

Captura e tratamento da imagem

Inicialização da Câmera e Parâmetros: A câmera é inicializada usando *cv2.VideoCapture(0)* para capturar o vídeo da câmera padrão (índice 0). As dimensões da imagem capturada (largura e altura) são obtidas para futura referência.

Configuração da Região de Interesse (ROI): Uma área de detecção de gestos é definida na imagem capturada usando coordenadas específicas para x, y, *width*, e *height*. Essa região é denominada ROI (*Region of Interest*).

Conversão para Escala de Cinza e Suavização: A ROI é convertida para escala de cinza para simplificar a detecção de bordas. Uma operação de suavização (*Gaussian Blur*) é aplicada à imagem em escala de cinza para reduzir o ruído e suavizar a transição entre as intensidades de pixel.

Detecção de Bordas usando *Canny*: O detector de bordas *Canny* é aplicado à imagem em escala de cinza para identificar transições de intensidade que representam as bordas na imagem.

Identificação de Contornos: A função *cv2.findContours* é utilizada para encontrar contornos na imagem resultante da detecção de bordas. Os contornos são filtrados com base em uma área mínima (*detect area*), e retângulos são desenhados ao redor dos contornos detectados na ROI para visualização.

Identificação de Ponto Central do Contorno: Para cada contorno válido, o centro do retângulo envolvente é calculado como (cx, cy). Essas coordenadas são usadas para determinar a posição do gesto na ROI.

Simulação de Cliques do Mouse com *PyAutoGUI*: Com base na posição do centro do retângulo, são simulados cliques do mouse usando a biblioteca *PyAutoGUI*. Se o centro do retângulo estiver à esquerda, um clique direito é simulado; se estiver à direita, um clique esquerdo é simulado.

Essas etapas combinadas permitem a detecção de gestos na região especificada da imagem capturada pela câmera, com a simulação de cliques do mouse e arraste conforme necessário. O código utiliza técnicas de processamento de imagem para filtrar informações relevantes e interagir com o sistema com base nos gestos identificados.

Reconhecimento de gestos

Para que fosse possível a associação de gestos à ações do mouse utilizamos da estratégia de associar primeiramente movimentos capturados na porção esquerda da tela de captura ao clique esquerdo do mouse, e a porção direita da tela ao clique direito do mouse.

A detecção de gestos neste código é implementada usando a biblioteca *OpenCV* para processamento de imagem. Aqui está uma descrição passo a passo da lógica de detecção de gestos:

Inicialização da Câmera e Parâmetros: A câmera é inicializada usando a biblioteca *OpenCV* (*cv2.VideoCapture*). Os parâmetros da largura (*width*) e altura (*height*) da imagem capturada pela câmera são obtidos para posterior referência.

Detecção de Gestos no *Loop* Principal: O código entra em um *loop* principal onde, em cada iteração, captura um frame da câmera e realiza as operações de detecção. A região de interesse (ROI) é definida para focar na área onde os gestos serão detectados.

Pré-processamento da Imagem: A ROI é convertida para escala de cinza (*gray roi*) para simplificar o processamento. A imagem em escala de cinza é suavizada usando um filtro Gaussiano para reduzir o ruído. As bordas da imagem suavizada são detectadas usando o detector de bordas *Canny*.

Identificação de Contornos: Os contornos são identificados na imagem resultante da detecção de bordas. Os contornos são filtrados com base em uma área mínima (*detect area*). Para cada contorno identificado, um retângulo é desenhado na ROI para visualização.

Detecção de Gestos Específicos: O centro de cada retângulo é calculado (cx e cy). Com base na posição do centro do retângulo, são simulados cliques do mouse correspondentes (esquerdo ou direito) usando a biblioteca *PyAutoGUI*.

A detecção de gestos é realizada iterativamente no *loop* principal, e a temporização é controlada pelo tempo mínimo de espera entre os gestos (*gestures cooldown*). O código reage aos gestos detectados, simulando cliques do mouse e permitindo o arraste em resposta a condições específicas.

Definição de aspectos visíveis ao usuário

Exibição da Resolução da Imagem: A resolução da imagem capturada pela câmera é exibida na janela de visualização do vídeo. Isso fornece uma informação útil ao usuário sobre a qualidade da imagem e as características da câmera. Neste trecho, o texto contendo a resolução da câmera ("Resolucao da camera: *"width x height"*") é exibido na posição (10, 30) da janela com uma fonte, tamanho e cor específicos.

Exibição de Mensagens de Texto para Gestos Detectados: Mensagens de texto são exibidas na janela de visualização para informar ao usuário quais gestos foram detectados. Por exemplo, se o centro de um gesto estiver à esquerda na ROI, é exibida a mensagem "Clique Esquerdo"; se estiver à direita, é exibida a mensagem "Clique Direito". Aqui, as mensagens "Clique Esquerdo" e "Clique Direito" são exibidas nas posições (50, 50) e (100, 100),

respectivamente. Além disso, as mensagens são estilizadas com uma fonte vermelha (BGR: (0, 0, 255)).

Exibição de Áreas de Detecção e Contornos: A ROI é visualizada na janela de visualização, permitindo ao usuário ver a área específica da imagem onde os gestos estão sendo detectados. Além disso, retângulos verdes são desenhados ao redor dos contornos identificados, fornecendo uma indicação visual dos gestos em tempo real.

Esses aspectos visíveis ao usuário contribuem para uma experiência interativa e informativa. Eles permitem que o usuário compreenda o que está acontecendo durante a detecção de gestos, fornecendo feedback visual sobre os eventos identificados e as ações que estão sendo simuladas com base nesses gestos.

A classificação dos eventos implementados no projeto está definida em: clique esquerdo ou clique primário, clique direito ou secundário e ação de arrastar. Sendo utilizados os respectivos métodos para cada evento:

click() - Método que, ao ser invocado, solicita ao sistema operacional uma realização do evento de clique esquerdo ou primário do mouse na posição corrente do cursor.

rightClick() - Método que, ao ser invocado, solicita ao sistema operacional uma realização do evento de clique direito ou secundário do mouse na posição corrente do cursor.

drag() - Método que, ao ser invocado, solicita ao sistema operacional o movimento do mouse enquanto o botão do mouse está pressionado, para realizar a operação de "arrastar" a partir da posição corrente do cursor.

Controle e temporização de eventos

A temporização dos eventos de clique esquerdo, clique direito e arrastar (drag) no código é controlada principalmente pelos elementos *gestures cooldown* e *last gesture time*:
- *gestures cooldown* - é uma variável que define o tempo mínimo de espera entre os gestos (em segundos). Neste código, está configurado como 1.7 segundos.

- *last gesture time* - é uma variável que armazena o tempo do último gesto

detectado. Se o tempo atual menos *last gesture time* for maior ou igual a *gestures cooldown*, o código considera que um novo gesto pode ser detectado.

Lógica de detecção de gestos:
A detecção de gestos ocorre dentro do bloco condicional *"if current time - last gesture time ¿= gestures cooldown:"*. Se este bloco for verdadeiro, o código verifica a presença de contornos na imagem, indicando a detecção de um gesto.

Temporização de arraste (*dragging*):
A temporização para ativar o arraste é controlada pela variável *arrast persist*. Se o último gesto detectado foi um clique direito e o tempo desde o último gesto é maior que 1.0 segundo, então *arrast persist* é atualizado e *arraste ativo* é marcado como verdadeiro. O arraste é ativado quando arraste ativo é verdadeiro e o cursor está à direita.

Temporização do arraste (dentro do *loop*):
Dentro do *loop* principal, se o arraste está ativo (*dragging* é verdadeiro), as coordenadas do ponto final (*end point*) são atualizadas, e a função *pyautogui.drag* é chamada para simular o movimento do mouse. O parâmetro *duration* é utilizado para controlar a velocidade do arraste, neste caso, é configurado como 0.2 segundos.

Essa lógica de temporização garante que o código espere um certo período de tempo entre os gestos para evitar respostas indesejadas e controla o arraste (*drag*) com base nas condições especificadas. Isso é crucial para o funcionamento adequado do sistema de controle de gestos implementado no código.

Bibliotecas e métodos utilizados

OpenCV (cv2):
OpenCV é uma biblioteca amplamente utilizada para processamento de imagem e visão computacional. Métodos principais utilizados:
cv2.VideoCapture: Inicializa a captura de vídeo da câmera. *cv2.flip*: Espelha horizontalmente o frame capturado. *cv2.cvtColor*: Converte uma imagem de um espaço de cor para outro. *cv2.GaussianBlur*: Aplica um filtro Gaussiano para suavizar a imagem. *cv2.Canny*: Aplica o detector de bordas *Canny* para identificar bordas na imagem. *cv2.findContours*: Encontra contornos em uma imagem. *cv2.boundingRect*: Encontra o retângulo envolvente de um

contorno.

PyAutoGUI:
PyAutoGUI é uma biblioteca para automação de tarefas do mouse e teclado. Métodos principais utilizados: *pyautogui.click*: Simula um clique do mouse. *pyautogui.rightClick*: Simula um clique do botão direito do mouse. *pyautogui.drag*: Simula o movimento do mouse enquanto um botão está pressionado. *pyautogui.position*: Retorna as coordenadas atuais do cursor do mouse. *pyautogui.FAILSAFE*: Variável global que, quando configurada como True, lança uma exceção se o mouse for movido para o canto superior esquerdo da tela.

Time:
A biblioteca time é utilizada para lidar com operações relacionadas ao tempo. Método principal utilizado: time.time: Retorna o tempo atual em segundos.

Pontos não trabalhados

Embora o código forneça uma implementação funcional para a detecção de gestos e interações com o mouse, há alguns pontos que poderiam ser aprimorados ou considerados dependendo dos requisitos específicos do projeto. Aqui estão alguns desses pontos: Calibração Dinâmica: O código atualmente assume uma posição fixa para a região de interesse (ROI). Em um ambiente dinâmico, seria benéfico incorporar um processo de calibração que permitisse ao usuário ajustar ou recalibrar a posição da ROI conforme necessário.

Melhorias na Detecção de Gestos: A detecção de gestos pode ser aprimorada usando técnicas mais avançadas, como o uso de redes neurais para reconhecimento de gestos ou a aplicação de algoritmos de aprendizado de máquina. Isso poderia aumentar a precisão e robustez da detecção.

Tratamento de Condições Limites: O código atual desativa o recurso de segurança pyautogui.FAILSAFE para evitar exceções ao mover o mouse para o canto superior esquerdo da tela. No entanto, isso pode resultar em problemas se o usuário tentar sair da aplicação durante a execução. Uma abordagem mais robusta para tratar essa condição limite pode ser implementada.

Interface Gráfica de Usuário (GUI): Adicionar uma interface gráfica de usuário (GUI) poderia melhorar a experiência do usuário, permitindo configurações interativas, feedback visual e controles adicionais.

Gestão de Threads: Em aplicações mais complexas, a gestão de threads pode ser considerada para separar tarefas intensivas em CPU (como o processamento de imagem) das tarefas relacionadas à interface do usuário.

Aprimoramentos na Simulação de Arraste (Drag): Atualmente, o código simula um arraste simples ao mover o cursor para a direita. Dependendo dos requisitos, podem ser adicionadas melhorias, como detecção da direção do movimento da mão para um arraste mais intuitivo.

Manuseio de Erros: A implementação atual assume condições ideais. Adicionar tratamento de erros e exceções poderia tornar o código mais robusto em situações não previstas.

Personalização de Gestos: Permitir aos usuários personalizar ou definir gestos específicos para ações específicas pode adicionar flexibilidade ao sistema.

Impedimentos de implementação

Dependência de Condições de Iluminação: O desempenho da detecção de gestos pode ser sensível às condições de iluminação. Mudanças significativas na iluminação ambiente podem afetar a precisão da detecção.

Limitações na Precisão da Detecção: Métodos de detecção de gestos baseados em processamento de imagem podem ter limitações na precisão, especialmente em ambientes com fundos complexos ou gestos sobrepostos.

Restrições na Variedade de Gestos: O código atual está otimizado para a detecção de gestos específicos (clique esquerdo, clique direito e arraste). Implementar a detecção de uma variedade maior de gestos pode ser desafiador e requer métodos mais avançados, como redes neurais.

Desempenho em Tempo Real: Dependendo da complexidade dos gestos e do poder de processamento do hardware, pode haver limitações no desempenho em tempo real, especialmente ao lidar com vídeo de alta resolução.

Adaptação a Diferentes Ambientes: O código pode não ser facilmente adaptado a diferentes ambientes ou cenários de uso sem alguma forma de calibração ou configuração dinâmica.

Requerimentos de Hardware e Câmera: O código assume o acesso a uma

câmera de vídeo. Dependendo do ambiente de implementação, a disponibilidade de hardware compatível pode ser um impedimento.

Sensibilidade a Movimentos Não Desejados: O código pode ser sensível a movimentos não intencionais, resultando em cliques ou arrastes indesejados. Isso pode ser um desafio, especialmente em ambientes onde os gestos são comuns.

Segurança e Privacidade: O uso de uma câmera pode levantar preocupações de segurança e privacidade, especialmente se o código estiver sendo executado em um ambiente compartilhado ou público.

Dependência de Bibliotecas Externas: O código depende de bibliotecas externas, como OpenCV e PyAutoGUI. Garantir que essas bibliotecas estejam instaladas e atualizadas pode ser um ponto de atenção.

Limitações na Simulação de Arraste: A simulação de arraste é realizada de forma simples, movendo o cursor para a direita. Isso pode ser limitado em termos de intuitividade e personalização.

Não Detecção de Orientação ou Rotação: O código atual não leva em consideração a orientação ou rotação da mão do usuário, o que pode ser relevante para certos tipos de gestos.

Sensibilidade a Obstruções: Obstruções na visão da câmera, como objetos entre a câmera e a mão do usuário, podem afetar a detecção de gestos.

Referências

RIBEIRO, L. H. Reconhecimento de gestos usando segmentação de imagens dinâmicas de mãos baseada no modelo de mistura de gaussianas e cor de pele. São Carlos, Brasil, USP. 2006.

HAMESTER, A. M. Método de reconhecimento de gestos aplicado em smartphones. Porto Alegre, Brasil, UFRGS. 2013.

RAMOS, C. L. Reconhecimento de Gestos usando Redes Neurais Artificiais. Brasília, Brasil, UnB. 2011.