Desenvolvimento de um Servidor Remoto para Controle de Jogos Eletrônicos Através do Movimento Ocular

Silvano Malfatti
Universidade Federal de Viçosa
Silvano.malfatti@ufv.br

Resumo: **Este trabalho apresenta o desenvolvimento de um servidor TCP projetado para permitir que jogos eletrônicos possam ser controlados a partir do movimento ocular do jogador. O sistema utiliza a câmera do computador para capturar imagens em tempo real, aplica técnicas de visão computacional para detectar e classificar a direção do olhar, e transmite essa informação por meio de uma conexão de rede a qualquer jogo ou aplicação cliente que esteja conectada ao servidor. Dessa forma, a solução proposta estabelece uma interface de controle baseada no olhar que promove a inclusão e oferece uma alternativa adaptável a qualquer jogo independente do motor gráfico ou linguagem utilizados em sua implementação.**

I. Introdução

Uma das principais características dos jogos eletrônicos é a interatividade, que consiste na troca contínua de informações entre o jogador e o sistema. Essa interação ocorre, tradicionalmente, por meio de dispositivos de entrada como teclados, mouses, joysticks e, mais recentemente, por sensores de movimento corporal, como os utilizados pelo Wiimote, Kinect e o Meta Quest — este último, um sistema de realidade virtual autônomo desenvolvido pela empresa Meta.

Entretanto, esses dispositivos apresentam limitações quando se trata de jogadores com deficiências motoras, como em casos de paralisia parcial ou total. A ausência de acessibilidade pode restringir a participação desses usuários em ambientes digitais interativos. Visando promover a inclusão e ampliar as possibilidades de controle, este trabalho propõe uma solução baseada na utilização dos movimentos oculares como meio de entrada para o controle de aplicações interativas.

A proposta consiste na implementação de um servidor TCP desenvolvido em Python, capaz de capturar imagens em tempo real da câmera do computador, identificar a direção do olhar do usuário e disponibilizar essa informação a aplicações externas, como jogos eletrônicos. Para isso, foi utilizado um modelo de rede neural convolucional treinado previamente a partir de um dataset rotulado com fotos de olhos e suas respectivas direções.

O desenvolvimento da solução foi dividido em etapas: (i) identificação e validação de um dataset apropriado; (ii) carregamento e pré-processamento das imagens; (iii) treinamento e persistência do modelo de predição; e (iv) implementação do servidor remoto responsável por realizar a captura da imagem do usuário e, com base no modelo treinado, inferir a direção do olhar a partir do movimento da íris. Essas etapas serão detalhadas nas seções seguintes.

II. Identificação e validação do dataset

Após uma busca por datasets com as características necessárias foi escolhido o MOBIUS (Mobile Ocular Biometrics In Unconstrained Settings) dataset fornecido pelo Laboratório de Visão Computacional da Faculdade de Ciências da Computação e Informação da Universidade de Liubliana [1].

Este dataset [2] oferece um conjunto completo de imagens de olhos classificados de acordo com o formato do olho esquerdo ou direito e para cada olho a direção do olhar sendo possível quatro direções: esquerda, direita, para cima e para frente como demonstra a Figura 1.

Figura 1 - imagens do MOBIUS dataset.

As imagens contidas no dataset possuem uma nomenclatura que permite identificar a qual dos olhos corresponde a foto e qual direção do olhar. É importante salientar que apesar de ser um dataset aberto, para obtê-lo é preciso solicitar permissão para o uso através da página do projeto.

III. LEITURA E ARMAZENAMENTO DO DATASET

Portanto, a primeira etapa foi realizar a codificação em Python para a leitura e armazenamento dos dados do modelo conforme o código apresentado pela Figura 2.

```python
# Diretório base contendo subpastas com imagens
base_path = './images'

# Listas para imagens e rótulos
X = []
y = []

# Classes válidas (left, right, straight, up)
valid_gaze = {'left': 0, 'right': 1, 's': 2, 'u': 3}

# Tamanho fixo para redimensionar as imagens
target_size = (128, 128)

# Função para extrair o gaze a partir do nome do arquivo
def extract_gaze_label(filename):
    try:
        parts = filename.split('_')
        gaze = parts[2][1]  # Terceiro segmento, segundo caractere (ex: 'Rs' -> 's')
        return valid_gaze[gaze] if gaze in valid_gaze else None
    except:
        return None

# Percorrer todos os arquivos nas subpastas
for root, dirs, files in os.walk(base_path):
    for file in files:
        if file.lower().endswith(('.jpg', '.jpeg', '.png')):
            label = extract_gaze_label(file)
            if label is not None:
                image_path = os.path.join(root, file)
                try:
                    img = Image.open(image_path).convert('RGB')
                    img = img.resize(target_size)
                    img_np = np.array(img) / 255.0  # Normaliza
                    X.append(img_np)
                    y.append(label)
                except Exception as e:
                    print(f"Erro ao processar {file}: {e}")

# Converter para arrays NumPy
X = np.array(X)
y = np.array(y)
```

Figura 2 - Código para o carregamento completo do dataset.

Como resultado da etapa de carregamento obteve-se o dataset carregado e rotulado pronto para o treinamento do modelo como mostra a Figura 3.

Figura 3 - imagens carregadas e rotuladas.

IV. TREINAMENTO DO MODELO

Nesta etapa, foi utilizado um modelo sequencial da biblioteca Keras/TersorFlow para o treinamento de uma rede neural convolucional composta por oito camadas. A arquitetura intercala camadas convolucionais responsáveis pela extração de características com camadas de pooling, que realizam a redução espacial das representações. Ao final da rede, são adicionadas camadas densas que realizam a classificação probabilística das imagens com base nas características extraídas. O treinamento foi configurado para ser executado por dez épocas, ao término das quais o modelo treinado foi salvo para ser utilizado na etapa de predição como demonstra o código da Figura 4.

```python
# Dividir em treino e validação
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2, stratify=y, random_state=42)

# Encoding para a saída
num_classes = 4
y_train_cat = to_categorical(y_train, num_classes)
y_val_cat = to_categorical(y_val, num_classes)

# Arquitetura de camadas da CNN
model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=X.shape[1:]),
    MaxPooling2D((2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Flatten(),
    Dense(128, activation='relu'),
    Dropout(0.5),
    Dense(num_classes, activation='softmax')
])

model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

# Treinamento
model.fit(X_train, y_train_cat, epochs=10, batch_size=32, validation_data=(X_val, y_val_cat))

# Salvar o modelo
model.save('modelo_gaze.h5')
```

Figura 4 - Etapa de treinamento do modelo.

Utilizando 80% do dataset para treinamento e 20% para teste, ao final de 10 épocas foi possível obter uma taxa de acerto de 96.04%.

V. IMPLEMENTAÇÃO DO SERVIDOR REMOTO

Nesta etapa o código desenvolvido tem como finalidade capturar em tempo real a imagem do rosto do usuário utilizando a câmera do computador e, a partir disso, detectar os olhos, prever a direção do olhar (esquerda, direita, reto ou cima) com base no modelo previamente treinado e transmitir essa informação a um cliente remoto por meio de um socket TCP.

Inicialmente, o sistema realiza o carregamento de um modelo previamente treinado com arquitetura de rede neural convolucional (CNN), armazenado no arquivo modelo_gaze.h5. Este modelo foi treinado para reconhecer quatro possíveis direções de olhar: esquerda (0), direita (1), reto (2) e cima (3), com base em imagens segmentadas dos olhos. Cada imagem é redimensionada para 128x128 pixels, convertida para RGB e normalizada antes da inferência.

Na sequência, o código estabelece uma conexão TCP por meio do módulo socket, configurando o servidor para escutar na porta 5001 do endereço local 127.0.0.1. Após aceitar a conexão de um cliente, o sistema se prepara para iniciar o processamento contínuo dos frames capturados pela câmera.

A biblioteca MediaPipe é utilizada para identificar os landmarks da face em tempo real. Especificamente, os índices de pontos faciais relacionados aos olhos são usados para

isolar e extrair as regiões dos olhos esquerdo e direito. Um cálculo conhecido como EAR (Eye Aspect Ratio ou razão entre distâncias verticais e horizontais da pálpebra) é utilizado para verificar se os olhos estão abertos, evitando que o sistema processe olhos fechados ou durante piscadas, o que melhoraria a precisão da predição. Durante o loop principal, a cada 300 milissegundos, o sistema captura um novo frame da câmera, detecta os pontos faciais com o FaceMesh e extrai as regiões correspondentes aos olhos. Cada imagem de olho válida é submetida ao modelo de predição. A saída do modelo é um vetor de probabilidades que indica a direção estimada do olhar. A classe com maior valor de probabilidade é mapeada para uma das quatro direções possíveis.

Sempre que o sistema identifica uma mudança na direção do olhar em relação à predição anterior, um código representando a nova direção é enviado ao cliente TCP conectado. Especificamente, o código 0\n indica olhar para a esquerda, 1\n para a direita, e 2\n é utilizado tanto para o olhar reto quanto para cima. Esse envio é realizado apenas quando há mudança de direção, minimizando o tráfego de dados e evitando redundâncias.

Além disso, o sistema pode apresentar visualmente as imagens recortadas dos olhos em janelas separadas, com atualização contínua, auxiliando em testes e avaliações em tempo real. O código desta etapa é apresentado através da Figura 5.

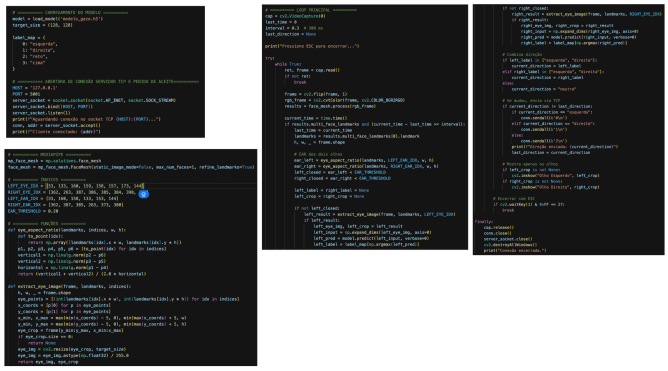

Figura 5 - Implementação do servidor remoto.

Como resultado da execução deste código, temos um servidor que captura a imagem de seleção dos olhos esquerdo e direito do usuário, realiza a predição e envia para o canal de saída a direção um código (0 esquerda, 1 direita, 2 neutra) que representa a direção do olhar. A Figura 6 demonstra o servidor em execução.

Figura 6 - servidor em execução.

VI. Estudo de Caso

Como estudo de caso, foi desenvolvido um game 2D estilo scroll vertical desenvolvido em Java no qual o jogador controla o movimento do avião para a esquerda ou para a direita. Por limitação das direções oferecidas pelo dataset, não foi possível realizar movimentações verticais nem identificar o evento de tiro com base no piscar dos olhos. A Figura 7 demonstra o protótipo em execução

Figura 7 - captura de tela do jogo 2D.

VII. Resultados e Conclusão

A execução do sistema desenvolvido demonstrou resultados promissores no que diz respeito à viabilidade do uso do movimento ocular como forma de controle para jogos eletrônicos. Durante o treinamento do modelo, utilizando 80% das imagens para treino e 20% para validação, foi obtida uma acurácia de 96,04% no conjunto de treinamento e 91,84% no conjunto de validação, com uma perda de validação (val_loss) de 0,3466. Esses resultados indicam que o modelo foi capaz de generalizar bem para dados não vistos, classificando corretamente a direção do olhar a partir das imagens segmentadas dos olhos.

Na etapa de inferência em tempo real, o servidor demonstrou estabilidade e desempenho adequados ao capturar os olhos do usuário, aplicar o modelo de predição e transmitir a direção do olhar por meio de uma conexão TCP. A estratégia de atualização apenas em casos de mudança de direção contribuiu para a redução do tráfego de dados e aumento da eficiência da comunicação com o cliente remoto.

O estudo de caso aplicado ao game 2D comprovou a aplicabilidade prática da solução. Apesar das limitações impostas pelo dataset – como a ausência de suporte para movimentos verticais contínuos e a impossibilidade de mapear eventos como disparo por piscar de olhos – a integração com o jogo foi realizada com sucesso. O jogador foi capaz de controlar o movimento horizontal da aeronave com base exclusivamente na direção do olhar, sem a necessidade de qualquer outro dispositivo de entrada físico.

Conclui-se, portanto, que a proposta de um servidor remoto baseado em visão computacional e redes neurais convolucionais para controle de aplicações interativas a partir do movimento ocular é tecnicamente viável e eficaz. A arquitetura modular do sistema permite sua adaptação a diferentes tipos de jogos ou aplicações, independentemente

do motor gráfico ou linguagem utilizados, promovendo acessibilidade e inclusão digital.

Como trabalhos futuros, sugere-se a ampliação do modelo para incluir mais direções, como diagonais e para baixo, bem como o mapeamento de piscadas como eventos adicionais de controle. Também seria pertinente explorar técnicas de calibração individual para melhorar ainda mais a acurácia da predição, considerando as variações fisiológicas entre os usuários.

VIII. Referências

[1] GOLOB, Ožbej; PEER, Peter; VITEK, Matej. Semi-automated correction... In: IEEE... 2020. p. 344-347.

[2] MOBIUS dataset. Disponível em https://sclera.fri.uni-lj.si/datasets.html. Acessado em 01/08/2025