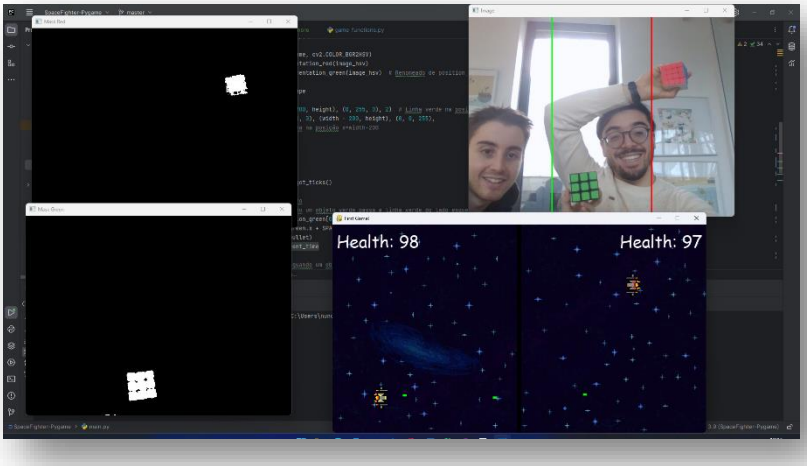
2023

Space Fighters

Relatório para Introdução à Visão por Computador



Engenharia em Desenvolvimento de Jogos Digitais
INTITUTO POLITÉCNICO AVE E CÁVADO

Trabalho Prático 1

Relatório

Grupo :

20115 - André Cerqueira
20116 - Nuno Fernandes

Orientador(a): José Brito

Engenharia em Desenvolvimento de Jogos Digitais

Barcelos | novembro, 2023

Índice

1. Introdução

1.1. Contextualização

No âmbito da disciplina "Introdução à Visão por Computador" do curso de Licenciatura em Engenharia em Desenvolvimento de Jogos Digitais no Instituto Politécnico do Cávado e do Ave, foi proposto o desenvolvimento de um jogo utilizando técnicas de Visão por Computador. Esta iniciativa visa proporcionar uma experiência prática e aprofundada na aplicação de algoritmos de visão computacional em contextos de jogos digitais.

1.2. Motivação e Objetivos

A motivação principal deste trabalho reside na exploração e aplicação de técnicas de segmentação em visão por computador para criar uma interface de utilizador inovadora e interativa. O objetivo da primeira fase é implementar um sistema de controlo para um jogo, onde o jogador pode interagir através da câmara, utilizando algoritmos de segmentação para detetar e rastrear objetos de cor predefinida.

1.3. Estrutura do Documento

Este relatório está organizado em seções que detalham a abordagem adotada, a metodologia de implementação, os resultados obtidos e uma análise crítica dos mesmos. A seção seguinte aborda a metodologia, seguida pelos resultados e, finalmente, as conclusões.

2. Metodologia e Desenvolvimento

Para esta fase, optou-se por desenvolver um jogo inspirado no clássico "SpaceFighter". A principal interação dos jogadores é controlar as naves espaciais, movendo-as verticalmente. A segmentação baseia-se na deteção de objetos de cores específicas, neste caso, vermelho e verde.

2.1. Escolha do Jogo: SpaceFighter

Para a implementação deste projeto, optou-se pelo jogo "SpaceFighter". Este jogo foi escolhido devido à sua mecânica simples, mas envolvente, onde duas naves espaciais duelam, disparando balas uma contra a outra. A simplicidade do jogo permite que a ênfase seja colocada nas técnicas de Visão por Computador, conforme estipulado pelo enunciado.

2.2. Segmentação de Cores

A segmentação de cores é uma técnica fundamental em visão computacional, especialmente quando se deseja isolar objetos específicos numa imagem. No projeto "SpaceFighter", essa técnica foi empregue para detetar e rastrear objetos coloridos que servem como controlos para as naves no jogo.

Utilizando a biblioteca OpenCV, o espaço de cores HSV (Hue, Saturation, Value) foi empregue devido à sua capacidade de separar a componente de luminosidade (Value) das componentes de cor (Hue e Saturation), tornando a segmentação de cores mais robusta sob diferentes condições de iluminação.

2.2.1. Definição de cores

No script color_segmentation.py, são definidos intervalos específicos para as cores vermelha e verde no espaço de cores HSV (Hue, Saturation, Value). O espaço de cores HSV é frequentemente usado em visão computacional devido à sua capacidade de separar a informação de luminância da informação de cor, tornando-o menos sensível às variações de iluminação.

As cores foram retiradas de cubos mágicos que é o objeto utilizado para fazer a movimentação da nave.

2.2.2. Segmentação de Objetos Vermelhos:

A segmentação de objetos vermelhos é um pouco mais complexa devido à natureza do espaço de cores HSV, onde o vermelho pode aparecer em ambos os extremos do espectro e matiz. Assim, foram definidos dois intervalos para capturar objetos vermelhos:

- Função update_segmentation_red

Aqui, dois intervalos de matiz são definidos para capturar o vermelho. As duas máscaras resultantes são então combinadas usando uma operação OR bitwise para criar uma máscara final que segmenta objetos vermelhos.

2.2.3. Segmentação de Objetos Verdes

Para segmentar objetos verdes, foi definido um intervalo específico no espaço HSV:

- Função update_segmentation_red;

Neste trecho, a imagem capturada pela câmara é primeiro convertida para o espaço de cores HSV. Em seguida, é definido um intervalo de cores para o verde e uma máscara binária é criada pelo cv2.inRange(), onde os pixels dentro do intervalo são definidos como brancos (valor 255) e os pixels fora do intervalo são definidos como pretos (valor 0).

Estas segmentações permitem que o sistema detete e rastreie objetos coloridos em tempo real, proporcionando uma interface de utilizador interativa e envolvente para o jogo.

2.3. Deteção e Rastreamento

Após a segmentação, uma etapa crucial é a deteção e o rastreamento dos objetos segmentados. Esta etapa permite que o sistema identifique a posição e o movimento dos objetos, traduzindo-os em comandos para o jogo.

Identificação de Contornos:

Com a biblioteca OpenCV, os contornos dos objetos segmentados foram identificados. A função findContours foi empregue para este propósito:

- findContours;

Neste exemplo, mask é a máscara binária resultante da segmentação de objetos vermelhos. A função retorna uma lista de contornos detetados na máscara.

Seleção do Contorno Principal:

Dado que podem existir múltiplos contornos detetados, é essencial identificar qual deles representa o objeto de controlo principal. Para isso, o contorno de maior área foi selecionado:

- largest_contour

A função contourArea calcula a área de um contorno, e a função max é usada para selecionar o contorno com a maior área.

Determinação da Posição:

Com o contorno principal identificado, a posição central do objeto foi calculada para determinar a posição da nave no jogo:

- Calcular a posição

2.4. Disparo de Tiros

O disparo de tiros é uma mecânica central no jogo "SpaceFighter", permitindo que os jogadores ataquem o adversário. Esta funcionalidade é implementada no script main.py, onde a lógica para disparar tiros é cuidadosamente integrada com a deteção de movimento através da visão computacional.

2.4.1. Mecânica de Disparo:

A mecânica de disparo é controlada por duas variáveis principais: last_green_shot e last_red_shot, que registam o tempo desde o último tiro disparado por cada nave. Isso é usado para implementar um atraso entre os tiros, evitando que as naves disparem continuamente sem qualquer restrição.

2.4.2. Deteção de Posição para Disparo:

A posição dos objetos coloridos é continuamente atualizada através das funções update_segmentation_red e update_segmentation_green. Quando um objeto é detetado

cruzando uma linha virtual definida na imagem da câmara, um tiro é disparado.

2.4.3. Condições para Disparo:

O jogo verifica se a posição detetada do objeto vermelho está à esquerda de uma linha vermelha virtual (red_line_x) e se a posição detetada do objeto verde está à direita de uma linha verde virtual (green_line_x). Se essas condições forem atendidas e o tempo desde o último tiro for maior que o atraso definido (BULLET_DELAY), um novo tiro é disparado.

- Excerto de código com a condição de disparo

3. Conclusões

3.1. Avaliação do Sistema de Controlo

O sistema de controlo implementado demonstrou ser eficaz na deteção e rastreamento de objetos coloridos, permitindo uma interação intuitiva e responsiva no contexto do jogo "SpaceFighter". A utilização do espaço de cores HSV provou ser uma escolha acertada, dada a sua robustez em diferentes condições de iluminação, o que é crucial para a consistência da jogabilidade em ambientes variados.

3.2. Integração da Visão por Computador no Jogo

A integração das técnicas de visão por computador no jogo foi bem-sucedida, proporcionando uma nova forma de interação que vai além dos métodos tradicionais de input, como teclado e rato. Esta abordagem não só enriqueceu a experiência do utilizador, mas também demonstrou o potencial da visão por computador aplicada ao desenvolvimento de jogos digitais.

3.3. Desafios e Limitações

Durante o desenvolvimento, enfrentaram-se desafios relacionados com a precisão da segmentação de cores em diferentes ambientes de iluminação. Além disso, a deteção de múltiplos objetos e a seleção do objeto de controlo principal foram aspetos que exigiram atenção especial para garantir a precisão do sistema.

3.4. Impacto da Interface de Utilizador Inovadora

A interface de utilizador baseada em movimento proporcionou uma dimensão adicional de imersão e interatividade. Este aspeto do projeto não só cumpriu os objetivos pedagógicos da disciplina, mas também ofereceu insights sobre como interfaces de utilizador inovadoras podem ser desenvolvidas e integradas em jogos futuros.

4. Conclusões Finais e Perspetivas Futuras

Em suma, o projeto alcançou seus objetivos de implementar um sistema de controlo interativo utilizando visão por computador e de explorar a aplicabilidade desta tecnologia no desenvolvimento de jogos. Para trabalhos futuros, considera-se a expansão do sistema para incluir mais cores e objetos, bem como a otimização da segmentação para ambientes com iluminação ainda mais variável. Adicionalmente, a implementação de algoritmos de *machine learning* para melhorar o rastreamento e a deteção de objetos poderia ser explorada, aumentando a precisão e a robustez do sistema de controlo.

Este projeto não só confirmou a viabilidade da visão por computador em jogos digitais, mas também abriu caminho para investigações futuras que podem levar a avanços significativos na forma como interagimos com os jogos.

5. Anexos

```python
def update_segmentation_red(image_hsv):
    mask_h1 = cv2.inRange(image_hsv[:, :, 0], hmin_red1, hmax_red1)
    mask_h2 = cv2.inRange(image_hsv[:, :, 0], hmin_red2, hmax_red2)
    mask_h = cv2.bitwise_or(mask_h1, mask_h2)
    mask_s = cv2.inRange(image_hsv[:, :, 1], smin_red, smax_red)
    mask_v = cv2.inRange(image_hsv[:, :, 2], vmin_red, vmax_red)

    mask = cv2.bitwise_and(mask_h, cv2.bitwise_and(mask_s, mask_v))
    cv2.imshow("Mask Red", mask)

    contours, _ = cv2.findContours(mask, cv2.RETR_TREE, cv2.CHAIN_APPROX_NONE)
    if contours:
        largest_contour = max(contours, key=cv2.contourArea)
        x, y, w, h = cv2.boundingRect(largest_contour)
        return x + w // 2, y + h // 2, w, h
    else:
        return None
```

Figura 1 - Excerto de código

```python
def update_segmentation_green(image_hsv):
    mask_h = cv2.inRange(image_hsv[:, :, 0], hmin_green, hmax_green)
    mask_s = cv2.inRange(image_hsv[:, :, 1], smin_green, smax_green)
    mask_v = cv2.inRange(image_hsv[:, :, 2], vmin_green, vmax_green)

    mask = cv2.bitwise_and(mask_h, cv2.bitwise_and(mask_s, mask_v))
    cv2.imshow("Mask Green", mask)

    contours, _ = cv2.findContours(mask, cv2.RETR_TREE, cv2.CHAIN_APPROX_NONE)
    if contours:
        largest_contour = max(contours, key=cv2.contourArea)
        x, y, w, h = cv2.boundingRect(largest_contour)
        return x + w // 2, y + h // 2, w, h
    else:
        return None
```

Figura 2 - Excerto de Código

```
contours, _ = cv2.findContours(mask, cv2.RETR_TREE, cv2.CHAIN_APPROX_NONE)
```

Figura 3 - Excerto de Código

```
largest_contour = max(contours, key=cv2.contourArea)
```

Figura 4 - Excerto de Código

```
x, y, w, h = cv2.boundingRect(largest_contour)
return x + w // 2, y + h // 2, w, h
```

Figura 5 - Excerto de código

```
# Shooting logic for green and red spaceships
if position_green and position_green[0] > green_line_x and current_time - last_green_shot > BULLET_DELAY:
    bullet = pygame.Rect(green.x + SPACESHIP_WIDTH, green.y + SPACESHIP_HEIGHT // 2 - 2, 10, 5)
    green_bullets.append(bullet)
    last_green_shot = current_time

if position_red and position_red[0] < red_line_x and current_time - last_red_shot > BULLET_DELAY:
    bullet = pygame.Rect(red.x - 10, red.y + SPACESHIP_HEIGHT // 2 - 2, 10, 5)
    red_bullets.append(bullet)
    last_red_shot = current_time
```

Figura 6 - Excerto de código