IPCA ESCOLA SUPERIOR DE TECNOLOGIA

IPCA INSTITUTO POLITÉCNICO DO CÁVADO E DO AVE

# Relatório Projeto IVC – Fase 2

## EDJD 2022/2023

*Este projeto serviu de prepósito a um projeto para a cadeira deIntrodução a visão por computador, em python dividido em três fases.*

# Índice

# Índice de imagens

# Introdução

Este trabalho foi proposto pelo professor José Brito, no contexto da unidade curricular Introdução à Visão por Computador, integrada no primeiro semestre do 2º ano de licenciatura de Desenvolvimento de Jogos Digitais, que visa o reforço e a aplicação dos conhecimentos adquiridos ao logo do semestre.

# Propósitos e Objetivos

No jogo, o jogador controla um paddle de forma que a bola destrua os tijolos da parede. O objetivo do trabalho é que o jogador controle o paddle através da câmara. O trabalho tem 3 fases. Em cada fase o controlo do paddle deve basear-se em diferentes técnicas de Visão por Computador. Em todas as fases deve ser apresentada a janela com o jogo e pelo menos outra janela com a imagem captada pela câmara e a visualização do resultado dos algoritmos de Visão por Computador. Poderão ser apresentadas várias janelas com resultados intermédios dos algoritmos de Visão por Computador. Em todas as fases, poderá ser usado o rato para inicializar os algoritmos usados. Na fase 1 o controlo do paddle deve ser baseado em algoritmos de segmentação. Poderá ser aplicado qualquer algoritmo de segmentação de entre os abordados na aula. O objeto a ser segmentado na imagem poderá ser um ou dois objetos de cor predefinida. Se o(s) objeto(s) a segmentar forem as mãos ou a cara do jogador, a segmentação poderá basear-se na cor da pele O controlo do paddle deve basear-se na posição do(s) objeto(s) segmentados na imagem.

# Propósitos e Objetivos

## Fase 1

Na fase 1 o controlo do paddle deve ser baseado em algoritmos de segmentação. Poderá ser aplicado qualquer algoritmo de segmentação de entre os abordados na aula. O objeto a ser segmentado na imagem poderá ser um ou dois objetos de cor predefinida. Se o(s) objeto(s) a segmentar forem as mãos ou a cara do jogador, a segmentação poderá basear-se na cor da pele O controlo do paddle deve basear-se na posição do(s) objeto(s) segmentados na imagem.

## Fase 2

Na fase 2 o controlo do paddle deve ser baseado em algoritmos de deteção de movimento. Poderá ser aplicado qualquer algoritmo de deteção de movimento de entre os abordados na aula. O controlo do paddle deverá basear-se no movimento dos pixels da imagem.

## Fase 3

Na fase 3 o controlo do paddle deve ser baseado em algoritmos de deteção de objetos. Poderá ser aplicado qualquer algoritmo de deteção de objetos de entre os abordados na aula. O controlo do paddle deve basear-se na posição do(s) objeto(s) detetados na imagem.

# Desenvolvimento da Fase 2

Nesta segunda fase decidi complementar a primeira, reutilizei o código que já segmentava uma cor selecionada pelo utilizador e com recurso a funções de deteção de movimento isolei o objeto em movimento dos restantes da mesma cor que poderia existir no background, daí resolvendo o principal problema com o movimento baseado apenas em segmentação.

Acrescento também que não fiquei por aqui nesta fase do projeto, realizei protótipos dos diversos métodos dispostos pelo professor, desde optical flow com lk ou dense, absdiff e combinações entre os mesmos, tentei diversas versões até optar pelo kalman filter complementado por background subtraction e segmentação, que me porpocionou a melhor jogabilidade. A segmentação permitiu me isolar a deteção de movimento a uma amostra mais pequena, evitando movimentos não pretendidos por parte de outros objetos ou até mesmo pelo corpo do jogador. O kalman filter permitiu que interligasse uma previsão do percurso do objeto diretamente ao paddle, tornando a jogabilidade mais fluida, dado que esta previsão simula a aceleração do objeto, tornando o jogo mais responsivo às intenções do jogador. Já o background subtractor apesar de parecer contraditório visto que já utilizo o filtro kalman, este resolve o problema de existir objetos parados de maior dimensão no background, isto pois faz com que os únicos objetos passados para o filtro kalman já se encontram em movimento.

Dado isto, para evitar a repetição das explicações do relatório da primeira fase, apenas focarei no desenvolvimento e integração da deteção de movimento no código já existente.

# Conclusão

Começamos por inicializar como globais os métodos createBackgroundSubtractor versão MOG2 e kalmanFilter.

OpenCV (Workspace) - Cam.py

```
8  # initialize background subtractor
9  background_subtraction = cv.createBackgroundSubtractorMOG2(1000, 50)
10
11 # initialize kalman filter
12 kalman_filter = cv.KalmanFilter(4, 2)
```

*Figura 1 - Inicialização das funções de deteção de movimento*

Após lermos o frame atual do vídeo_vapture verificamos se o roi (region of interest) é nulo, que só acontece no primeiro frame, nesse caso o utilizador pode selecionar uma região no ecrã ou cancelar a operação selecionando assim o frame na sua totalidade como roi.

OpenCV (Workspace) - Cam.py

```
30 # If region of interest isn't defined prompt user to define one
31 if roi is None:
32     roi = cv.selectROI('Select Roi', frame, False)
33     cv.destroyWindow('Select Roi')
34     if roi == ((0,0,0,0)): # if selection is canceled use whole frame
35         height, width, _ = frame.shape
36         roi = ((0,0,width,height))
```

*Figura 2 - ROI selection*

Com o roi agora definido cortamos o frame com as suas coordenadas iniciais e dimensões obtendo assim o frame_roi. Este permitira apenas realizar as deteções na região pretendida.

OpenCV (Workspace) - Cam.py

```
40 # Crop image with roi's dimensions
41 frame_roi = frame[roi[1] : roi[1] + roi[3], roi[0] : roi[0] + roi[2]]
```

*Figura 3 - Frame cropping*

# Conclusão

Aplicamos de seguida a função background_subtractor ao frame resultante da segmentação de cor do frame_roi. Com isto filtramos os objetos segmentados obtendo apenas aqueles que demonstram movimento.

OpenCV (Workspace) - Cam.py

```
50  # Remove background, isolating moving objects
51  frame_movement = background_subtraction.apply(frame_segmented)
```

*Figura 4 - Background subtraction*

Filtramos também as áreas em movimento de dimensão reduzida, para evitar movimentos acidentais devido a pequenos movimentos involuntários por parte do utilizador.

OpenCV (Workspace) - Cam.py

```
53 # Filter smaller areas out
54     frame_thresholded = cv.threshold(src=frame_movement, thresh=20, maxval=255, type=cv.THRESH_BINARY)[1]
```

*Figura 5 - Filtragem das áreas de dimensão reduzida*

Desta vez começamos a função draw_contours por desenhar um retângulo azul com as dimensões do roi no frame, representando assim a área de interesse que o user selecionou.

OpenCV (Workspace) - Cam.py

```
85  def draw_contours(frame_draw, frame_contours):
86      # draw a blue rectangle where roi is located at
87      cv.rectangle(frame_draw, (0, 0), (roi[2]-1, roi[3]-1), (255, 0, 0), 2)
```

*Figura 6 - Retângulo para representar o roi*

# Conclusão

Antes de usarmos a função findContours para obter os contornos das áreas em movimento da cor segmentada, filtramos mais uma vez o frame processado deixando apenas as áreas que mostram uma clara diferença de 254 em 255.

OpenCV (Workspace) - Cam.py

```
89  # Filter only clear differences
90  _, frame_contours = cv.threshold(frame_contours, 254, 255, cv.THRESH_BINARY)
```

*Figura 7 - Filter only clear differences*

Depois de termos os contours e de verificarmos que existe pelo menos uma área em movimento desenhamos novamente um retângulo que representa a região de interesse mas verde, esta mudança de cor dá um feedback ao jogador de que o programa está a detetar movimento.

OpenCV (Workspace) - Cam.py

```
99  # check if there's at least 1 object with the segmented color
100     if len(contours) > 0:
101         # draw a green rectangle on top of the blue rectangle to indicate that movement was detected
102         cv.rectangle(frame_draw, (0, 0), (roi[2]-1, roi[3]-1), (0, 255, 0), 2)
```

*Figura 8 - Retângulo para representar o roi quando movimento é detetado*

Através do centro de massa calculado colocamos um texto da mesma cor do retângulo que contorna o objeto identificando assim o mesmo como a representação da posição atual do movimento.

OpenCV (Workspace) - Cam.py

```
119 # draw text to identify the current position
120 cv.putText(frame_draw, "Current", (int(center_mass_current[0]), int(center_mass_current[1] + 50)), FONT ,0.6, [0,0,255])
```

*Figura 9 - Texto para identificar a posição atual*

Com recurso à função defenida predict_movement calculamos uma previsão através do filtro de kalman que nos retorna o centro de massa da posição para onde prevê que o objeto se deslocará.

OpenCV (Workspace) - Cam.py

```
122 center_mass_predicted = predict_movement(center_mass_current[0],center_mass_current[1])
```

*Figura 10 - Cálculo da previsão através da função definida*

# Conclusão

A função predict_movement mencionada começa por definir as matrizes de medida e transição necessárias para a previsão, de seguida armazena as coordenadas num tuple para ser usado na função correct que atualiza o estado da previsão da medida. Após atualizadas as medidas usamos a função predict para computar o estado da próxima posição com base nas mesmas. Por fim retornamos o tuple com as coordenadas do centro da previsão.

OpenCV (Workspace) - Cam.py

```
76 # correct current position and predict the next, returns the prediction
77 def predict_movement(coord_x, coord_y):
78     kalman_filter.measurementMatrix = np.array([[1,0,0,0],[0,1,0,0]],np.float32)
79     kalman_filter.transitionMatrix = np.array([[1,0,1,0],[0,1,0,1],[0,0,1,0],[0,0,0,1]],np.float32)
80     curent = np.array([[np.float32(coord_x)], [np.float32(coord_y)]])
81     kalman_filter.correct(curent)
82     predicted = kalman_filter.predict()
83     return predicted
```

*Figura 11 - Função para calcular uma previsão usando o Kalman Filter*

Para dar uma visualização desta previsão ao utilizador, adicionamos um offset igual à diferença entre a posição do centro atual e do previsto aos pontos utilizados para o retângulo de contorno do objeto, e da mesma forma desenhamos este retângulo, mas de cor azul para facilitar a destinação entre ambos.

OpenCV (Workspace) - Cam.py

```
123 #  draw the same rotated bounding rectangle but offset to the predicted position
124 for point in box:
125     point[0] += int(center_mass_predicted[2])
126     point[1] += int(center_mass_predicted[3])
127 cv.drawContours(frame_draw, [box], 0, (255, 0, 0), 2)
```

*Figura 12 - Retângulo para representar a posição prevista*

Voltamos também a colocar o texto da mesma cor identificando assim o mesmo como a representação da posição prevista para o movimento.

OpenCV (Workspace) - Cam.py

```
129 # draw text to identify the predicted position
130 cv.putText(frame_draw, "Predicted", (int(center_mass_predicted[0]), int(center_mass_predicted[1] - 50)), FONT , 0.6, [255, 0, 0])
```

*Figura 13 - Texto para identificar a posição prevista*

# Conclusão

Pelo contrário da fase 1, desta vez retornamos o centro de massa previsto. Assim o movimento do paddle será derivado da previsão do filtro kalman, tornando a gampelay mais fluida do que préviamente.

OpenCV (Workspace) - Cam.py

```
133 return center_mass_predicted
```

*Figura 14 - Retorno do centro de massa previsto*

Devido à utilização do roi, encontrámos um obstáculo ao antigo método de movimentação do paddle, como as coordenadas do roi estavam limitadas pela sua dimensão o paddle não se movimentava para além das mesmas, fazendo com que não podesse se movimentar no mapa todo, para resolver este problema devolvemos o resultado da multiplicação da coordenada x do centro de massa previsto pela razão entre a largura da câmara e da largura do roi, resultando assim em coordenadas válidas para a totalidade do ecrã do breakout.

OpenCV (Workspace) - Cam.py

```
71 if center_mass is not None:
72     # compensate for roi's different resolution
73     paddle_x = int(center_mass[0]) * 610 / roi[2]
74     return paddle_x
```

*Figura 15 - Cálculo da coordenada a devolver*

# Conclusão

Com esta segunda fase foi possível aprofundar mais conhecimentos acerca da deteção e rastreamento de movimentos numa imagem através de diversos métodos, e a utilização da mesma para múltiplas aplicações. Aprendi também a importância de analisar as diversas opções de soluções que estavam à minha disposição antes de resolver o problema em questão.