IPCA ESCOLA SUPERIOR DE TECNOLOGIA

IPCA INSTITUTO POLITÉCNICO DO CÁVADO E DO AVE

Relatório Projeto IVC – Fase 1

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 1

Primeiramente para a resolução desta fase precisamos de uma library que nos permita utilizar e manipular a câmara, para isto usamos a library de openCV chamada cv2. Importámos também a library numpy para o uso de matrizes.

OpenCV (Workspace) - Cam.py

```
1  import cv2
2  import numpy as np
```

Figura 1 - Importação das libraries utilizadas para a segmentação

Começamos por definir thresholds globais e iniciá-los com um valor predefinido para a segmentação de verde.

Iniciamos também a captura de vídeo associando a mesma a uma variável global para permitir a suma manipulação.

OpenCV (Workspace) - Cam.py

```
4  # initialize HSV color range for green colored objects
5  thresholdLower = (50, 100, 100)
6  thresholdUpper = (70, 255, 255)
7
8  # Start capturing the video from webcam
9  video_capture = cv2.VideoCapture(0)
```

Figura 2 - Declaração do threshold inicial e início da captura de vídeo

Conclusão

Definimos uma função chamada cam(), esta começa por verificar se a vídeo_capture está aberta, caso não abre a mesma. Gurdamos o frame atual na variável frame e usamos a função flip da library cv2 para evitar o efeito da imagem estar espelhada.

OpenCV (Workspace) - Cam.py

```
11 def cam():
12     global frame # global frame so it can be used in mouse_get_threshold()
13     if not video_capture.isOpened():
14         video_capture.open(0)
15     # Store the current frame of the video in the variable frame
16     ret, frame = video_capture.read()
17     # Flip the image to make it right
18     frame = cv2.flip(frame,1)
```

Figura 3 – Função cam() parte 1

Damos uso à função GaussianBlur da library cv2 com o kernel de tamanho 5 para remover ruído excessivo no frame. Convertemos também o frame para o formato HSV com recurso ao método cvtColor da library cv2 permitindo assim uma segmentação mais eficaz.

OpenCV (Workspace) - Cam.py

```
20 # Blur the frame using a Gaussian Filter of kernel size 5, to remove excessive noise
21 frame_blurred = cv2.GaussianBlur(frame, (5,5), 0)
22 # Convert the frame to HSV as it allows better segmentation.
23 frame_hsv = cv2.cvtColor(frame_blurred, cv2.COLOR_BGR2HSV)
```

Figura 4 - Função cam() parte 2

Conclusão

Segmentamos o frame já convertido para HSV com a função inRange da library cv2 baseada nos valores atuais dos thresholds. Obtendo assim uma imagem onde a cor segmentada está representada a branco e as restantes a preto.

Após segmentado, erodimos o frame_segmented para eliminar o ruído restante, obtendo assim o frame_eroded. Este é então dilatado dando origem ao frame_masked.

Por fim, após esta filtração, mostramos o frame_masked na janela “Masked Output” com a intenção de dar uma visualização da segmentação ao utilizador.

OpenCV (Workspace) - Cam.py

```
25 # Create a mask for the frame, showing threshold values
26 frame_segmented = cv2.inRange(frame_hsv, thresholdLower, thresholdUpper)
27 # Erode the masked output to delete small white dots present in the masked image
28 frame_eroded  = cv2.erode(frame_segmented, None, 10)
29 # Dilate the resultant image to restore our target
30 frame_masked = cv2.dilate(frame_eroded, None, 10)
31
32 # Display the masked output in a different window
33 cv2.imshow('Masked Output', frame_masked)
```

Figura 5 - Função cam() parte 3

Com a função findContours da library cv2 obtemos todas as áreas da cor segmentada e guardamos o output na variável contours.

Declaramos também o centro do objeto como noneType, para caso não haver um objeto da cor segmentada a função devolver none.

OpenCV (Workspace) - Cam.py

```
35 # Find all contours in the masked image
36 contours,_ = cv2.findContours(frame_masked.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
37
38 # Define center of the object to be detected as None
39 center = None
```

Figura 6 - Função cam() parte 4

Verificamos se o comprimento da variável contours é superior a 0, ou seja, que existe um ou mais de um objeto de cor igual à da segmentação.

OpenCV (Workspace) - Cam.py

```
41  # check if there's at least 1 object with the segmented color
42  if len(contours) > 0:
```

Figura 7 - Função cam() parte 5

Caso não exista mostramos o frame na janela “Frame”, passamos também o output da função setMouseCallback da library cv2 para a função mouse_get_threshold e devolvemos o center com o valor de none.

OpenCV (Workspace) - Cam.py

```
57  # Show the output frame
58  cv2.imshow('Camera Output', frame)
59  cv2.setMouseCallback('Camera Output',mouse_get_threshold)
60  return center
```

Figura 8 - Função cam() parte 6

Caso exista calculamos o objeto de maior área recorrendo à função max na variável contours, com o método contourArea da library cv2 como chave de busca para analisar apenas as áreas dos contornos.

OpenCV (Workspace) - Cam.py

```
43  # Find the contour with maximum area
44  contours_max = max(contours, key = cv2.contourArea)
```

Figura 9 - Função cam() parte 7

Determinamos o retângulo de menor área que contorna a nossa área máxima, calculamos os vértices desse retângulo com recurso ao método boxPoints de modo a ter os pontos necessários para desenhar o retângulo no frame utilizando a função drawContours. Isto para ter uma referência visual que indique o objeto ou área que está a ser segmentada.

OpenCV (Workspace) - Cam.py

```
46  #  rotated bounding rectangle (https://docs.opencv.org/3.4/dd/d49/tutorial_py_contour_features.html)
47  rect = cv2.minAreaRect(contours_max)
48  box = cv2.boxPoints(rect)
49  box = np.int0(box)
50  cv2.drawContours(frame,[box],0,(0,0,255),2)
```

Após usarmos o método moments da library cv2, utilizamos o output na fórmula para encontrar o centro da área segmentada.

$$\mathbf{c} = \frac{1}{n}\sum_{i=1}^{n}\mathbf{x}_i$$

Figura 10 - formula do centroide (https://learnopencv.com/find-center-of-blob-centroid-using-opencv-cpp-python/)

$$C_x = \frac{M_{10}}{M_{00}}$$

Figura 11 - formula do x do centroide (https://learnopencv.com/find-center-of-blob-centroid-using-opencv-cpp-python/)

$$C_y = \frac{M_{01}}{M_{00}}$$

Figura 12 - formula do y do centroide (https://learnopencv.com/find-center-of-blob-centroid-using-opencv-cpp-python/)

OpenCV (Workspace) - Cam.py

```
52  # Calculate the centroid of the object
53  # "formula from (https://learnopencv.com/find-center-of-blob-centroid-using-opencv-cpp-python/)"
54  M = cv2.moments(contours_max)
55  center = (int(M['m10'] / M['m00']), int(M['m01'] / M['m00']))
```

Figura 13 - Função cam() parte 9

Por fim, mostramos o frame na janela “Frame” e devolvemos o center com o valor da posição do centro do objeto.

OpenCV (Workspace) - Cam.py

```
57  # Show the output frame
58  cv2.imshow('Camera Output', frame)
59  cv2.setMouseCallback('Camera Output',mouse_get_threshold)
60  return center
```

Figura 14 - Função cam() parte 11

Conclusão

Na função mouse_get_threshold verificamos se o evento registado foi um clique no botão esquerdo do rato, se sim utilizamos as coordenadas do rato na janela frame para obter os valores da cor nesse pixel e convertemos simultaneamente essa cor para o formato HSV com a função cvtColor da library cv2 e o parâmetro COLOR_BGR2HSV.

OpenCV (Workspace) - Cam.py

```
62 def mouse_get_threshold(event,x,y,flags,param):
63     if event == cv2.EVENT_LBUTTONDOWN: # checks mouse left button down condition
64         # convert rgb to hsv format
65         colorsHSV = cv2.cvtColor(np.uint8([[[frame[y,x,0] ,frame[y,x,1],frame[y,x,2] ]]]),cv2.COLOR_BGR2HSV)
```

Figura 15 - Função mouse_get_threshold() parte 1

Com esta cor geramos novos intervalos de cor e alteramos os valores das variáveis threshold para os mesmos. Segmentando assim a partir da próxima iteração do game_loop a nova cor selecionada.

OpenCV (Workspace) - Cam.py

```
67 # create a threshold based on the color values
68 tempLower = colorsHSV[0][0][0]  - 10, 100, 100
69 tempUpper = colorsHSV[0][0][0] + 10, 255, 255
70
71 global thresholdLower
72 global thresholdUpper
73 # set the threshold
74 thresholdLower = np.array(tempLower)
75 thresholdUpper = np.array(tempUpper)
```

Figura 16 - Função mouse_get_threshold() parte 2

Conclusão

Já no código do breakout, mais especificamente na função game_loop() da classe Game, utilizamos a função cam() do ficheiro Cam.py para obter as coordenadas do centro do objeto, após verificarmos que o centro não é nonetype usamos a função get_position() da classe GameObject, para obter as coordenadas do objeto paddle. Tendo ambas as coordenadas declaramos que o offset é a diferença entre a coordenada X das mesmas, obtendo-se assim o valor necessário para através da função move() da classe Paddle alterar a posição do objeto paddle para coincidir com o objeto de cor igual à da segmentação.

OpenCV (Workspace) - Breakout.py

```
184 def game_loop(self):
185     object_coords = Cam.cam()
186     if object_coords != None:
187         paddle_coords = self.paddle.get_position()
188         offset = object_coords[0] - paddle_coords[0]
189         self.paddle.move(offset)
```

Figura 17 - Função game_loop()

Conclusão

Com esta primeira fase foi possível aprofundar mais conhecimentos acerca da segmentação de uma imagem através de thresholds, e a utilização da mesma para certas aplicações. Aprendi também que fundamentalmente que ter um bom código e bem documentado é melhor que ter um código que funciona.