IPCA ESCOLA SUPERIOR DE TECNOLOGIA

IPCA INSTITUTO POLITÉCNICO DO CÁVADO E DO AVE

Relatório Projeto IVC – Fase 3

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 3

Nesta terceira fase decidi complementar a primeira parte do projeto, complementando o código que já controlava o paddle com base na posição absoluta do objeto detetado na câmara.

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.

Começamos por importar a library do yolov5, que é um algoritmo de deteção de objetos que divide imagens num sistema de grelha. Cada célula na grelha é responsável por detetar objetos nela mesma.

OpenCV (Workspace) - Cam.py

```
3  import yolov5
4
5  # load yolov5model
6  models = yolov5.load('../yolov5n.pt')
7  models.conf = 0.33
8
9  # Start capturing the video from webcam
10 video_capture = cv.VideoCapture(0)
11 roi = None
```

Figura 1 – Importação da library yolov5

Conclusão

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

Aplicamos de seguida a função cvtColor ao frame_roi para alterar assim o seu formato de cor do BGR para o RGB.

OpenCV (Workspace) - Cam.py

```
38 # Convert the frame to HSV as it allows better segmentation.
39 frame_rgb = cv.cvtColor(frame_roi, cv.COLOR_BGR2RGB)
```

Figura 4 – Conversão do frame para o formato rgb

Conclusão

Analisámos o frame já em rgb com base nos modelos de yolov5 carregados, para detetar assim os objetos nestes especificados.

OpenCV (Workspace) - Cam.py

```
41  # Analyse the frame with the loaded yolov5 models
42  results = models(frame_rgb)
```

Figura 5 – Análise do frame pela library yolov5

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

```
56  def draw_contours(frame_draw, results):
57      # draw a blue rectangle where roi is located at
58      cv.rectangle(frame_draw, (0, 0), (roi[2]-1, roi[3]-1), (255, 0, 0), 2)
```

Figura 6 - Retângulo para representar o roi

Inicializamos o centro como None para o caso não detetarmos.

OpenCV (Workspace) - Cam.py

```
60  # Define center of the object to be detected and it's predicted center as None
61  center = None
```

Figura 7 - Inicializar o centro de massa com valor None

Conclusão

Iteramos pelos objetos resultantes do modelo do yolov5 com o objetivo de encontrar o objeto pretendido através da condição que compara o nome do objeto detetado com o nome do selecionado.

OpenCV (Workspace) - Cam.py

```
63 # check every object to see if it is the selected object
64 for pred in enumerate(results.pred):
65     im_boxes = pred[1]
66     for *box, conf, cls in im_boxes:
67         box_class = int(cls)
68         if results.names[box_class] == "person":
```

Figura 8 – Verificação com iteração por todos os objetos

Caso detetarmos o objeto selecionado nos resultados, 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 o objeto.

OpenCV (Workspace) - Cam.py

```
69 # draw a green rectangle on top of the blue rectangle to indicate that selected object as detected
70 cv.rectangle(frame_draw, (0, 0), (roi[2]-1, roi[3]-1), (0, 255, 0), 2)
```

Figura 9 - Retângulo para representar o roi quando o objeto é detetado

De seguida guardamos os parâmetros do objeto selecionado em variáveis para facilitar a leitura da restante função.

OpenCV (Workspace) - Cam.py

```
72 conf = float(conf)
73 pt1 = np.array(np.round((float(box[0]), float(box[1]))), dtype=int)
74 pt2 = np.array(np.round((float(box[2]), float(box[3]))), dtype=int)
75 box_color = (0, 0, 255)
```

Figura 10 – Parâmetros do objeto selecionado

Com estes parâmetros começamos por desenhar um retângulo que delimita o objeto detetado.

OpenCV (Workspace) - Cam.py

```
77 # draw rectangle around the detected object using it's parameters
78 cv.rectangle(img = frame_draw,
79                 pt1 = pt1,
80                 pt2 = pt2,
81                 color = box_color,
82                 thickness = 1)
```

Figura 11 - Retângulo para representar a posição do objeto

E com os mesmos colocamos um texto acima da posição do retângulo que identifica o nome do modelo de objeto detetado.

OpenCV (Workspace) - Cam.py

```
84 # write the text indenting the object
85 cv.putText(img = frame_draw,
86             text = "{}:{:.2f}".format(results.names[box_class], conf),
87             org = np.array(np.round((float(box[0]), float(box[1]-1))), dtype = int),
88             fontFace = FONT,
89             fontScale = 0.5,
90             color = box_color,
91             thickness = 1)
```

Figura 12 Texto para identificar o objeto

Guardamos o valor aproximado do centro do objeto selecionado que foi detetado para o retornar no fim da função.

OpenCV (Workspace) - Cam.py

```
93 # store the aproximate center of the selected object to return
94 center = (int(box[0] + box[2]/2), int(box[1] + box[3]/2))
```

Figura 13 – centro do objeto selecionado e detetado

Conclusão

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 pudesse se movimentar no mapa todo, para resolver este problema devolvemos o resultado da multiplicação da coordenada x do centro 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

```
50  if center is not None:
51      # compensate for roi's different resolution
52      paddle_x = int(center[0]) * 610 / roi[2]
53      return paddle_x
```

Figura 14 - 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.