



UNIVERSIDADE FEDERAL DE PERNAMBUCO

Grupo de Pesquisa em Engenharia Biomédica

Computação Visual

Aquisição de Vídeo por Programação em Python

Arthur Gois

Carlos Gabriel

Gabriel Florentino

Jairo Magno

Recife, 2023

Sumário

1	Objetivos	1
2	Descrição	2
2.1	OpenCV	2
2.1.1	Aquisição de Vídeo RGB	2
2.1.2	Aquisição Utilizando Raspberry PI	4
2.2	MediaPipe	5
2.2.1	Reconhecimento de Gestões da Mão em Vídeo RGB	5
2.2.2	Detecção de Pontos de Referência do Corpo Humano em Vídeo RGB	8
2.2.3	Segmentação de Imagem em Vídeo RGB	10
2.2.3.1	Separar Primeiro Plano e Segundo Plano	10
2.2.3.2	Borrar Plano de Fundo	13
2.2.4	Detecção de Objetos em Vídeo RGB	14
3	Referências Bibliográficas	17

1 Objetivos

Este relatório é referente ao trabalho de computação visual da Sprint 02 : 30/06/2023 - 14/07/2023 do grupo de IC's do GPEB. O principal objetivo desse trabalho é a introdução do uso das bibliotecas *OpenCV* e *MediaPipe* da linguagem de programação Python. Nele será documentado a construção e funcionamento dos projetos propostos.

2 Descrição

2.1 OpenCV

Este relatório aborda o uso do *OpenCV* para aquisição em vídeo, uma biblioteca de visão computacional amplamente utilizada. O *OpenCV* fornece uma variedade de recursos e ferramentas essenciais para capturar e processar vídeos de diferentes fontes, como câmeras e arquivos de vídeo. Será utilizado algumas funcionalidades do *OpenCV* relacionadas à aquisição de vídeo. A partir disso, será criado um código capaz de fazer aquisição de vídeo RGB em tempo real. Após construído o código e testado pelo computador, será feito outro teste em uma *Raspberry Pi*.

2.1.1 Aquisição de Vídeo RGB

Nesta seção, será abordado a implementação do *OpenCV* para aquisição e gravação de vídeo. Será explicado com detalhes o código da Figura 2.1.

```

save_video.py > ...
1   import cv2 as cv
2
3   cap = cv.VideoCapture(0)
4   # Define o codec e cria um VideoWriter object
5   fourcc = cv.VideoWriter_fourcc(*'XVID')
6   out = cv.VideoWriter('Teste.avi', fourcc, 20.0, (640, 480))
7
8   while cap.isOpened():
9       ret, frame_rgb = cap.read()
10      if not ret:
11          print("Não foi possível receber do frame_rgb (fim da stream?). Saindo ...")
12          break
13      # Inverte a Imagem para uma visualização de selfie
14      frame_rgb = cv.cvtColor(frame_rgb, 1), cv.COLOR_BGR2RGB
15      frame_rgb = cv.cvtColor(frame_rgb, cv.COLOR_RGB2BGR) #Garantia da conversão para RGB
16      # Escreve no frame_rgb
17      out.write(frame_rgb)
18      cv.imshow('Rec...', frame_rgb)
19      if cv.waitKey(1) == ord('s'): break #Clica no botão 's' para parar a gravação
20  # Libera tudo se o vídeo for terminado
21
22  cap.release()
23  out.release()
24  cv.destroyAllWindows()

```

Figura 2.1: Código da Aquisição em RGB usando OpenCV

Na linha 3, será inicializado o objeto VideoCapture para capturar o vídeo. O parâmetro 0 indica que está sendo capturando o vídeo da primeira câmera disponível. Dependendo da configuração do usuário, será necessário alterar esse parâmetro.

As linhas 4 e 5 definem o codec de vídeo usado para gravar o vídeo e criar um objeto VideoWriter para escrever o vídeo em um arquivo. Neste exemplo, está sendo usando o codec "XVID" e definindo o nome do arquivo de saída como "Teste.avi". O parâmetro 20.0 indica o *FPS* (20 frames per second), e (640, 480) especifica a largura e altura do vídeo em pixels. É possível ajustar esses parâmetros para outro desejado.

Após isso será inicializado um loop que continuará enquanto a captura de vídeo estiver aberta. A condição na linha 10 garante que se houver algum problema na leitura do vídeo, a stream será fechada e irá mostar um print no logo informando que houve algum erro.

Nas linhas 14 e 15 é usado o `cv.flip(frame_rgb, 1)` para inverter horizontalmente a imagem, para uma visualização de "selfie". Em seguida, é usado `cv.cvtColor()` para converter o quadro de BGR para RGB. Essa conversão é necessária para garantir que a exibição em tempo real do quadro seja correta. Em seguida, o quadro é convertido novamente de RGB para BGR, para que seja gravado corretamente no arquivo de vídeo.

As linhas 17 até 19 gravam o quadro no arquivo de vídeo usando `out.write(frame_rgb)`. Também exibem o quadro usando `cv.imshow()` com o título "Rec..." na janela de exibição. A função `cv.waitKey(1)` aguarda por 1 milissegundo e verifica se alguma tecla foi pressionada. Se a tecla 's' for pressionada, o loop é interrompido e a gravação do vídeo é encerrada.

Por fim, ao se encerrar o loop, as linhas que faltam são responsáveis por liberar os recursos após a gravação do vídeo ser concluída.

2.1.2 Aquisição Utilizando Raspberry PI

Para realizar a aquisição na *Raspberry Pi*, é necessário instalar a biblioteca *OpenCV 4*. Como a linguagem *python* será utilizada, será necessário instalar um wrapper do *OpenCV*, chamado "opencv-python". Este wrapper possibilita a utilização das api's do *OpenCV* através da linguagem *python*. É possível instalá-lo através do pip (Gerenciador de pacotes da linguagem *python*), com o seguinte comando: "pip install opencv-python", porém, como o programa será executado em uma plataforma *Arm*, é necessário complementar o comando de execução do programa *python* com a seguinte variável de ambiente: "LD_PRELOAD=/usr/lib/arm-linux-gnueabihf/libatomic.so.1". No final, para executar o programa, é necessário fazer: "LD_PRELOAD=/usr/lib/arm-linux-gnueabihf/libatomic.so.1 python nome_do_script". Para que as bibliotecas utilizadas no projeto fiquem encapsuladas, sem afetar o ambiente principal da linguagem, o módulo *venv* foi utilizado. Na Figura ?? é possível verificar a aquisição de vídeo dentro de uma *Raspberry Pi*.

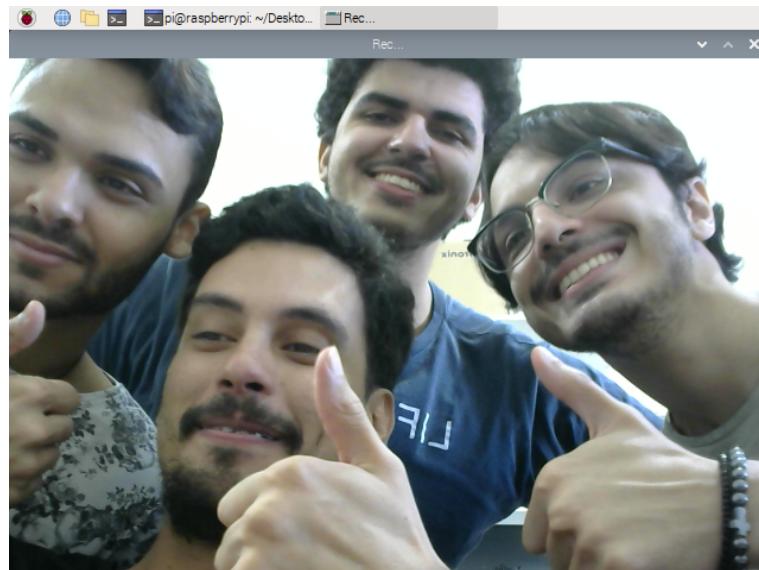


Figura 2.2: Aquisição de Vídeo em RGB usando OpenCV na Raspberry

2.2 MediaPipe

O *MediaPipe* é uma estrutura de código aberto desenvolvida pelo Google, que oferece uma variedade de recursos poderosos para o processamento de vídeo em tempo real. O *MediaPipe* tem sido amplamente utilizado para diversas finalidades, como reconhecimento de gestos da mão em vídeo RGB, detecção de pontos de referência do corpo humano em vídeo RGB, segmentação de imagem em vídeo RGB e detecção de objetos em vídeo RGB.

Ao empregar técnicas avançadas de visão computacional e aprendizado de máquina, o *MediaPipe* permite a análise e interpretação precisa de vídeos, possibilitando a detecção de gestos específicos da mão, como movimentos de *pinoching*, deslizamento e rotação. Além disso, a estrutura também é capaz de identificar pontos de referência-chave do corpo humano, como articulações e extremidades, o que permite a análise de movimentos e posturas corporais em tempo real.

Outra funcionalidade importante do *MediaPipe* é a segmentação de imagem, que possibilita a separação de objetos em primeiro plano e plano de fundo em um vídeo RGB. Essa técnica é especialmente útil para aplicações de realidade aumentada, remoção de fundo e outras tarefas de edição de vídeo.

Por fim, o *MediaPipe* oferece recursos de detecção de objetos em vídeo RGB, permitindo identificar e rastrear objetos específicos em tempo real. Essa funcionalidade é amplamente utilizada em aplicações de vigilância, reconhecimento facial e sistemas de controle de tráfego.

Todos esses recursos são utilizados em conjunto com o *OpenCV*.

2.2.1 Reconhecimento de Gestões da Mão em Vídeo RGB

Nesta seção, será abordado a implementação da biblioteca *MidiaPipe* e *OpenCV* para o reconhecimento de gesto de uma mão. O código criado que torna possível essa implementação pode ser visto na Figura 2.3.

```

hand_tracker > handtracker.py > ...
1 """
2 1. Necessário ter o openCV baixado
3 2. Necessário ter o MediaPipe baixado
4 """
5 import cv2 as cv
6 import mediapipe as mp
7
8 mp_drawing = mp.solutions.drawing_utils
9 mp_drawing_styles = mp.solutions.drawing_styles
10 mphands = mp.solutions.hands
11
12 capture = cv.VideoCapture(0)
13 fourcc = cv.VideoWriter_fourcc(*'XVID')
14 out = cv.VideoWriter('Teste_Mao.avi', fourcc, 20.0, (640, 480)) #Salvando o vídeo
15 hands = mphands.Hands()
16
17 while True:
18     data, image = capture.read()
19     #Inverter a imagem
20     image = cv.cvtColor(cv.flip(image, 1), cv.COLOR_BGR2RGB)
21     #Armazenando os resultados
22     result = hands.process(image)
23     image = cv.cvtColor(image, cv.COLOR_RGB2BGR)
24     out.write(image)
25     if result.multi_hand_landmarks:
26         for hand_landmarks in result.multi_hand_landmarks:
27             mp_drawing.draw_landmarks(
28                 image,
29                 hand_landmarks,
30                 mphands.HAND_CONNECTIONS)
31     cv.imshow('Reconhecimento Gestos Mao', image)
32     if cv.waitKey(1) == ord('s'): break
33                                         #apertar a tecla 's' para sair
33
34 capture.release()
35 out.release()

```

Figura 2.3: Código de Reconhecimento de Gesto da Mão

O trecho de código das linhas 8 até 10 é onde é definido o módulo 'drawing_utils' do *MediaPipe* como `mp_drawing`. Esse módulo será usado para desenhar os landmarks (pontos de referência) da mão no vídeo. Além disso foi utilizado o módulo `drawing_styles` do *MediaPipe* como `mp_drawing_styles`. Essa biblioteca fornece estilos visuais que podem ser aplicados aos landmarks desenhados na imagem. E por fim, foi importado o módulo `hands` do *MediaPipe* como `mphands`, que é responsável pela detecção e rastreamento da mão. As linhas 12 até 14 já foram bem explicadas na seção referente a aquisição de vídeo em RGB. Na linha 15 foi criado um objeto `Hands` do *MediaPipe* para realizar o processamento de reconhecimento de gestos da mão. Na Figura 2.4 é possível ver os pontos de reconhecimento em uma mão que a biblioteca é capaz de reconhecer.

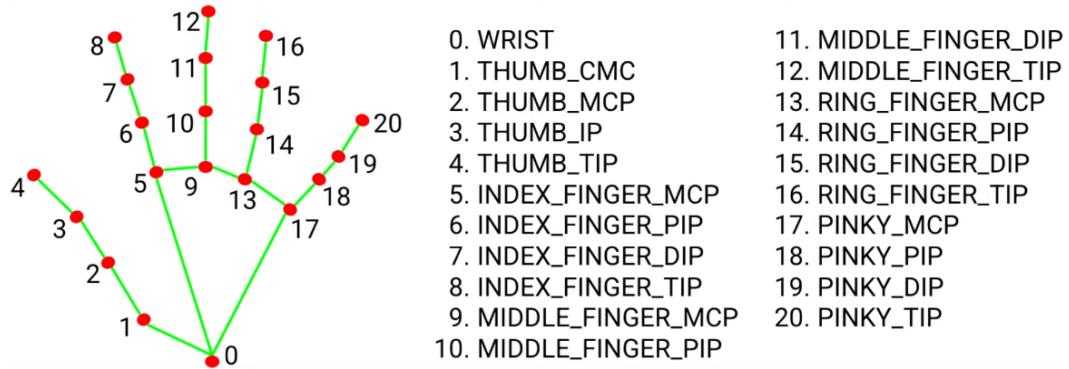


Figura 2.4: Pontos de Reconhecimento da Mão

A partir disso, será iniciado um loop que executa continuamente a captura de quadros de vídeo. A função `capture.read()` lê um quadro do vídeo capturado. O quadro é invertido horizontalmente e convertido de BGR para RGB para garantir uma visualização correta. O objeto `Hands` processa o quadro para detectar e rastrear a mão. A imagem é convertida novamente de RGB para BGR e gravada no arquivo de vídeo. Caso sejam detectadas múltiplas mãos, os landmarks das mãos são desenhados na imagem usando a função `draw_landmarks`. A imagem resultante é exibida em uma janela com o título "Reconhecimento Gestos Mao". O loop é interrompido se a tecla 's' for pressionada. Na Figura 2.3 é possível ver como ficariam os pontos de reconhecimento de uma mão quando a stream começa.

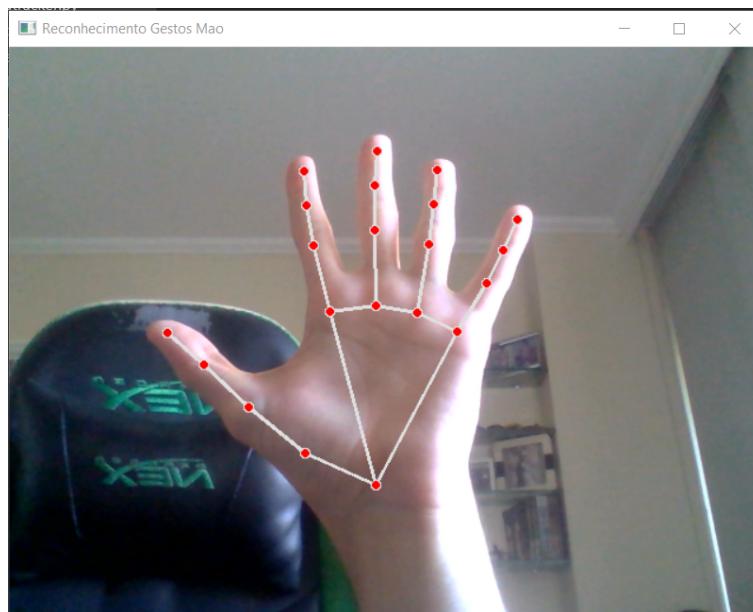


Figura 2.5: Vídeo de Reconhecimento da Mão

2.2.2 Detecção de Pontos de Referência do Corpo Humano em Vídeo RGB

Nesta seção, será apresentado a implementação da detecção de pontos de referência do corpo humano em vídeo RGB usando as bibliotecas MediaPipe e OpenCV. O código criado pode ser visto na Figura 2.6. Alguns trechos de código são idênticos a códigos posteriores, dessa forma eles serão menos focados.

```
pose_landmarker > 📁 landmark_pose.py > ...
1   import cv2 as cv
2   import mediapipe as mp
3
4   mp_drawing = mp.solutions.drawing_utils
5   mp_drawing_styles = mp.solutions.drawing_styles
6   mp_pose = mp.solutions.pose
7
8   capture = cv.VideoCapture(0)
9
10  with mp_pose.Pose(min_detection_confidence = 0.5, min_tracking_confidence = 0.5) as pose:
11      while capture.isOpened():
12          sucess, image = capture.read()
13          image.flags.writeable = False
14          image = cv.cvtColor(image, cv.COLOR_BGR2RGB)
15          results = pose.process(image)
16          #Escreve a anotação da pose na imagem
17          image.flags.writeable = True
18          image = cv.cvtColor(image, cv.COLOR_RGB2BGR)
19          mp_drawing.draw_landmarks(
20              image,
21              results.pose_landmarks,
22              mp_pose.POSE_CONNECTIONS,
23              landmark_drawing_spec = mp_drawing_styles.get_default_pose_landmarks_style()
24          )
25          #Inverte a imagem horizontalmente para uma visualização estilo selfie
26          cv.imshow('LandMark Pose', cv.flip(image, 1))
27          if cv.waitKey(1) == ord('s'): #apertar o botão s para sair
28              break
29
30  capture.release()
```

Figura 2.6: Código Referente a Detecção de Pontos no Corpo

Na linha 6 é importado o módulo pose do *MediaPipe* como `mp_pose`, que é responsável pela detecção e rastreamento dos pontos de referência do corpo humano. Na figura 2.7 é possível ver os pontos de reconhecimento chave que a biblioteca é capaz de visualizar.

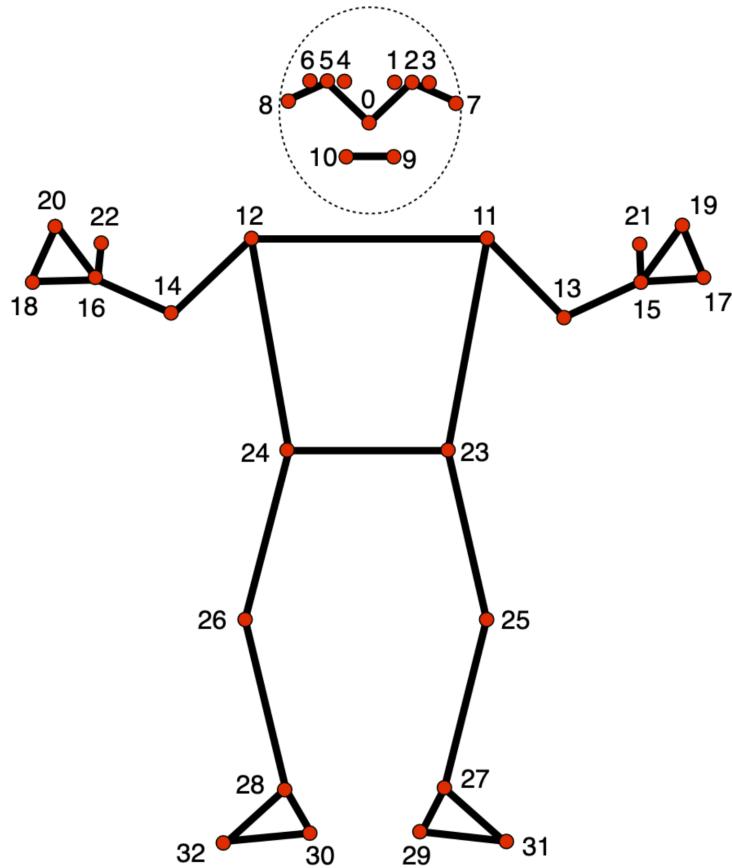


Figura 2.7: Pontos de Reconhecimento Chaves do Corpo Humano

Após isso será iniciado um loop para capturar e processar os quadros do vídeo em tempo real. Foi feito a utilização do `with` statement para criar um objeto `Pose` do *MediaPipe*, que permite a detecção e o rastreamento dos pontos de referência do corpo humano. Dentro do loop, é capturado os quadros de vídeo e convertido de BGR para RGB para garantir uma visualização em forma de 'Selfie'. Em seguida, o objeto `Pose` processa os quadros e obtém os resultados dos landmarks de pose. Os landmarks são desenhados na imagem usando a função `draw_landmarks` do *MediaPipe*. Por fim, é exibido a imagem resultante com os landmarks desenhados em uma janela com o título "LandMark Pose". O loop é interrompido se a tecla 's' for pressionada. Na Figura 2.8 é possível ver o código em ação capturando os pontos de referência de um corpo humano.

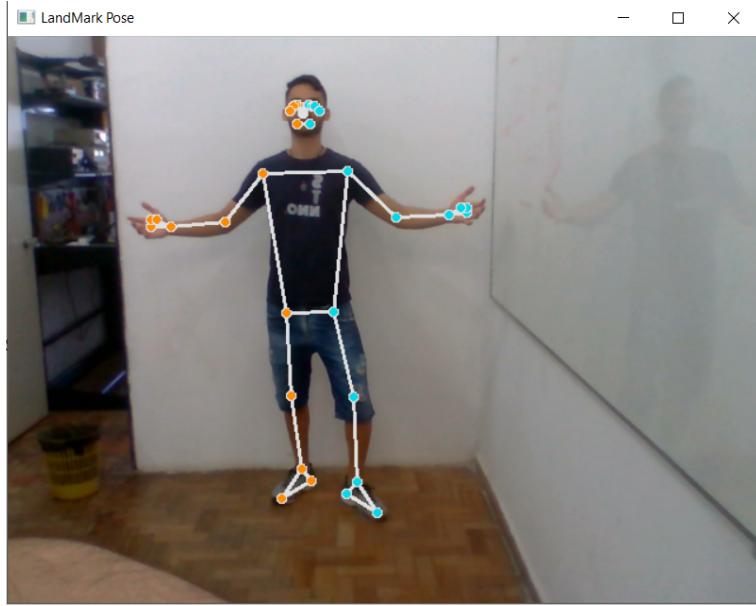


Figura 2.8: Pontos de Reconhecimento Chaves do Corpo Humano em Vídeo

2.2.3 Segmentação de Imagem em Vídeo RGB

A parte de segmentação de imagem foi dividido em dois projetos: Separar o primeiro plano e segundo plano; borrar o plano de fundo. A primeira coisa a se fazer é realizar o download do modelo de segmentação de imagem que será utilizado para esses exemplos. Dentro do terminal utilize o comando `'wget -O deeplabv3.tflite -q https://storage.googleapis.com/mediapipe-models/image_segmenter/deeplab_v3/float32/1/deeplab_v3.t'`. Certifique-se de utilizar esse comando dentro do diretório onde os scripts em python serão armazenados.

2.2.3.1 Separar Primeiro Plano e Segundo Plano

Com o modelo baixado, o código responsável por separar o primeiro e segundo plano da imagem pode ser visto na Figura 2.9

```

1  import math
2  import cv2 as cv
3  import numpy as np
4  import mediapipe as mp
5
6  from mediapipe.tasks import python
7  from mediapipe.tasks.python import vision
8
9  # Performa o redimensionamento e mostra a imagem
10 def resize_and_show(image):
11     h, w = image.shape[:2]
12     if h < w:
13         img = cv.resize(image, (DESIRED_WIDTH, math.floor(h/(w/DESIRED_WIDTH))))
14     else:
15         img = cv.resize(image, (math.floor(w/(h/DESIRED_HEIGHT)), DESIRED_HEIGHT))
16     cv.imshow('Image', img)
17     cv.waitKey(0)
18     cv.destroyAllWindows()
19
20 BG_COLOR = (192, 192, 192) # cinza
21 MASK_COLOR = (255, 255, 255) # branco
22 # Altura e largura que será usada pelo modelo
23 DESIRED_HEIGHT = 480
24 DESIRED_WIDTH = 480
25
26 image_filenames = ['teste_1.jpg', 'teste_2.jpg']
27
28 # Cria as opções que serão utilizadas para o ImageSegmenter
29 base_options = python.BaseOptions(model_asset_path='deeplabv3.tflite')
30 options = vision.ImageSegmenterOptions(base_options=base_options, output_category_mask=True)
31
32 # Cria o segmentador de imagem
33 with vision.ImageSegmenter.create_from_options(options) as segmenter:
34
35     # Loop dentro das imagens desejadas
36     for image_file_name in image_filenames:
37
38         # Criando o arquivo da imagem em MediaPipe que será segmentado
39         image = mp.Image.create_from_file(image_file_name)
40
41         # Recupera as máscaras para a imagem segmentada
42         segmentation_result = segmenter.segment(image)
43         category_mask = segmentation_result.category_mask
44
45         # Gera imagens de cores sólidas para mostrar a máscara de segmentação de saída.
46         image_data = image.numpy_view()
47         fg_image = np.zeros(image_data.shape, dtype=np.uint8)
48         fg_image[:] = MASK_COLOR
49         bg_image = np.zeros(image_data.shape, dtype=np.uint8)
50         bg_image[:] = BG_COLOR
51
52         condition = np.stack((category_mask.numpy_view(),) * 3, axis=-1) > 0.2
53         output_image = np.where(condition, fg_image, bg_image)
54
55         print(f'Máscara de segmentação de: {image_file_name}:')
56         resize_and_show(output_image)

```

Figura 2.9: Código Primeiro e Segundo Plano pt1

As linhas 6 e 7 são responsáveis por importar os módulos específicos do *MediaPipe* para a tarefa de segmentação de imagem.

Logo após, foi-se criado uma função chamada `resize_and_show` que redimensiona a imagem para uma altura e largura desejadas e a exibe em uma janela utilizando o *OpenCV*. Essa função será utilizada para visualizar a imagem resultante da segmentação.

No trecho da linha 20 até a 26 fora definidos algumas variáveis para configuração do processo de segmentação. `BG_COLOR` e `MASK_COLOR` representam as cores de fundo e da máscara, respectivamente. `DESIRED_HEIGHT` e `DESIRED_WIDTH` são as dimensões desejadas para redimensionamento da imagem. `image_filenames` é uma lista com os nomes dos arquivos de imagem que serão segmentados.

No restante do código é criado o segmentador de imagem utilizando o *MediaPipe*. Define-se as opções do segmentador, incluindo o modelo de segmentação baixado através do comando no terminal, o "deeplabv3.tflite". Em seguida, dentro de um loop, carregam-se as imagens a serem segmentadas e passa cada uma delas para o segmentador. O resultado da segmentação é obtido e a máscara de categorias é extraída. Com base nessa máscara, será gerado uma imagem com as cores de fundo e da máscara. Por fim, é feito a chamada da função `resize_and_show` para exibir a máscara de segmentação resultante com o tamanho pré estabelecido no início do código.

Na Figura 2.10 é possível ver a transformação de uma das imagens testes após passar pelo código, demonstrando assim o funcionamento a segmentação.



Figura 2.10: Resultados Segmentação Primeiro e Segundo Plano

2.2.3.2 Borrar Plano de Fundo

O próximo código é responsável por borrar o plano fundo, estilo o efeito disponibilizado pelo Google Hangouts. Boa parte desse código é igual ao código de separar o primeiro e segundo plano, dessa forma será explicado apenas as partes que mostram novas funcionalidades.

```

1  import math
2  import cv2 as cv
3  import numpy as np
4  import mediapipe as mp
5
6  from mediapipe.tasks import python
7  from mediapipe.tasks.python import vision
8
9  # Performa o redimensionamento e mostra a imagem
10 def resize_and_show(image):
11     h, w = image.shape[:2]
12     if h < w:
13         img = cv.resize(image, (DESIRED_WIDTH, math.floor(h/(w/DESIRED_WIDTH))))
14     else:
15         img = cv.resize(image, (math.floor(w/(h/DESIRED_HEIGHT)), DESIRED_HEIGHT))
16     cv.imshow('Image', img)
17     cv.waitKey(0)
18     cv.destroyAllWindows()
19
20 # Altura e largura que será usada pelo modelo
21 DESIRED_HEIGHT = 480
22 DESIRED_WIDTH = 480
23
24 image_filenames = ['teste_1.jpg', 'teste_2.jpg']
25
26 # Cria as opções que serão utilizadas para o ImageSegmenter
27 base_options = python.BaseOptions(model_asset_path="deeplabv3.tflite")
28 options = vision.ImageSegmenterOptions(base_options=base_options, output_category_mask=True)
29
30 # Desfoca o fundo da imagem com base na máscara de segmentação.
31
32 # Cria o segmentador
33 with python.vision.ImageSegmenter.create_from_options(options) as segmenter:
34
35     # Loop dentro das imagens desejadas
36     for image_file_name in image_filenames:
37
38         # Criando o arquivo da imagem em MediaPipe que será segmentado
39         image = mp.Image.create_from_file(image_file_name)
40
41         # Recupera as máscaras para a imagem segmentada
42         segmentation_result = segmenter.segment(image)
43         category_mask = segmentation_result.category_mask
44
45         # Converte de BGR para RGB
46         image_data = cv.cvtColor(image.numpy_view(), cv.COLOR_BGR2RGB)
47
48         # Aplica Efeitos
49         blurred_image = cv.GaussianBlur(image_data, (55,55), 0)
50         condition = np.stack((category_mask.numpy_view() * 3, axis=-1) > 0.1
51         output_image = np.where(condition, image_data, blurred_image)
52
53         print(f'Plano de fundo borrado de: {image_file_name}')
54         resize_and_show(output_image)
```

Figura 2.11: Código Borrar Plano de Fundo

A única parte realmente nova nesse código é o trecho das linhas 45 até 51. Nela, a imagem original é convertida de BGR para RGB. É utilizado a função GaussianBlur do *OpenCV* para aplicar o desfoque no plano de fundo. Após isso é criado uma condição com base na máscara de segmentação, onde os pixels com valores maiores que 0.1 são considerados como parte do objeto principal. Depois é usado a função np.where do NumPy para escolher entre a imagem original e a imagem desfocada com base na condição. A imagem resultante é então exibida utilizando a função resize_and_show.

Na Figura 2.12 é possível ver a transformação de uma das imagens testes após passar pelo código, demonstrando assim o funcionamento a segmentação



Figura 2.12: Resultados Segmentação Primeiro e Segundo Plano

2.2.4 Detecção de Objetos em Vídeo RGB

Também é possível detectar objetos com o *MediaPipe* com soluções já existentes, ou seja, com um banco de dados que já retorna alguns objetos e sua probabilidade de acordo com o algoritmo que foi implementado para gerar o mesmo. Na Figura 2.13, encontra-se o código implementado.

Nota-se que o código é dividido em duas partes, uma função definida que é responsável por desenhar na imagem as boundboxs, os nomes dos objetos e as porcentagens de compatibilidade e a parte principal.

```

1 ˜ import cv2
2  import numpy as np
3  import mediapipe as mp
4  from mediapipe.tasks import python
5  from mediapipe.tasks.python import vision
6
7  MARGIN = 10
8  ROW_SIZE = 10
9  FONT_SIZE = 1
10 FONT_THICKNESS = 1
11 TEXT_COLOR = (255, 0, 0)
12
13 ˜ def visualize(image, detection_result) -> np.ndarray:
14 ˜˜ for detection in detection_result.detections:
15 ˜˜˜ # Draw bounding_box
16 ˜˜˜ bbox = detection.bounding_box
17 ˜˜˜ start_point = bbox.origin_x, bbox.origin_y
18 ˜˜˜ end_point = bbox.origin_x + bbox.width, bbox.origin_y + bbox.height
19 ˜˜˜ cv2.rectangle(image, start_point, end_point, TEXT_COLOR, 3)
20
21 ˜˜ # Draw label and score
22 ˜˜ category = detection.categories[0]
23 ˜˜ category_name = category.category_name
24 ˜˜ probability = round(category.score, 2)
25 ˜˜ result_text = category_name + ' (' + str(probability) + ')'
26 ˜˜ text_location = (MARGIN + bbox.origin_x, MARGIN + ROW_SIZE + bbox.origin_y)
27 ˜˜ cv2.putText(image, result_text, text_location, cv2.FONT_HERSHEY_PLAIN, FONT_SIZE, TEXT_COLOR, FONT_THICKNESS)
28 ˜˜ return image
29
30 # STEP 2: Create an ObjectDetector object.
31 base_options = python.BaseOptions(model_asset_path='efficientdet.tflite')
32 options = vision.ObjectDetectorOptions(base_options=base_options, score_threshold=0.5)
33 detector = vision.ObjectDetector.create_from_options(options)
34 webcam = cv2.VideoCapture(0)
35
36 while True:
37
38     sucess, frame = webcam.read()
39
40     frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
41     frame = cv2.flip(frame, 1)
42
43     # STEP 3: Load the input image.
44     image = mp.Image(image_format=mp.ImageFormat.SRGB, data=frame)
45
46     # STEP 4: Detect objects in the input image.
47     detection_result = detector.detect(image)
48
49     # STEP 5: Process the detection result. In this case, visualize it.
50     image_copy = np.copy(image.numpy_view())
51     annotated_image = visualize(image_copy, detection_result)
52     rgb_annotated_image = cv2.cvtColor(annotated_image, cv2.COLOR_BGR2RGB)
53     cv2.imshow("Out", rgb_annotated_image)

```

Figura 2.13: Código Detectar Objetos

Da linha 17 até a 19 são soluções do *MediaPipe* para os pontos do boundboxs, enquanto na linha 20 é usado o *opencv* para desenhar o retângulo.

As linhas 23 e 24, tem-se qual foi o objeto detectado enquanto das linhas 25 até a 28 tem-se o texto a ser inserido com a *boundbox* (nome e probabilidade), o *opencv* faz a escrita do texto.

No código principal (linhas 32 até 35), primeiro se carrega a base de informações dos objetos a serem detectados (soluções) e declara o objeto da webcam.

Em "*while True*", basicamente se aplica a detecção na linha 47 e passa como parâmetro na função citada anteriormente. No final, mostra-se a imagem com os objetos detectados. Na Figura 2.14 é possível ver um teste feito em uma webcam de notebook.

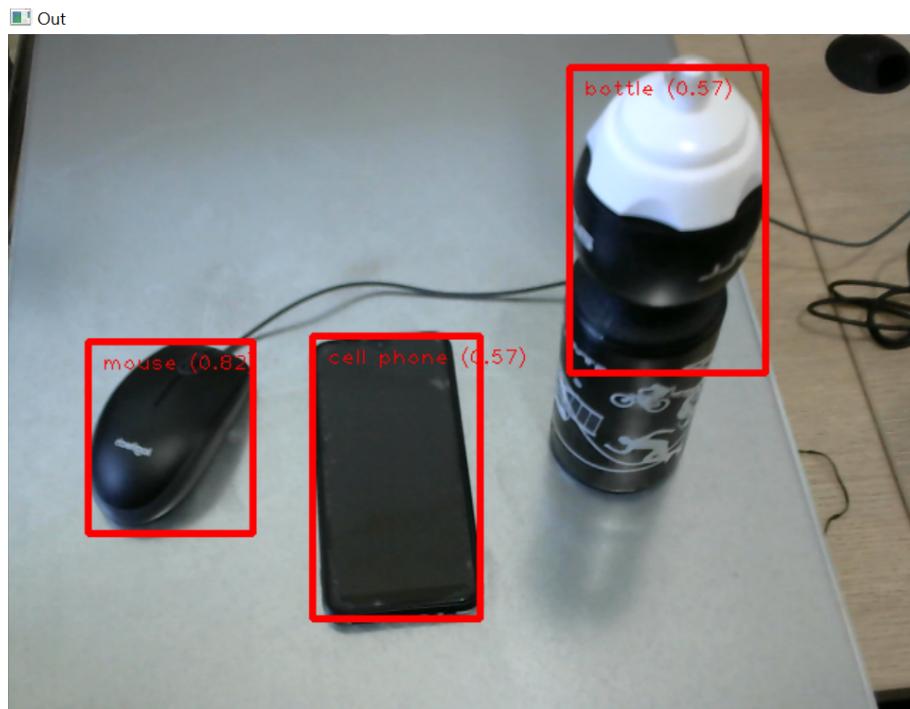


Figura 2.14: Teste Detectar Objetos

3 Referências Bibliográficas

- [1] Instalação da biblioteca OpenCV na Raspberry PI. Disponível em:
<https://www.youtube.com/watch?v=QzVYnG-WaM4&t=1s>.
- [2] Detecção de Objetos em Vídeo. Disponível em :
https://developers.google.com/mediapipe/solutions/vision/object_detector#get_started.
- [3] Detecção Pontos Chaves do Corpo Humano. Disponível em :
https://developers.google.com/mediapipe/solutions/vision/pose_landmarker#get_started
- [4] Detecção de Gestos de uma Mão:
https://developers.google.com/mediapipe/solutions/vision/gesture_recognizer#get_started
- [5] Segmentação de Imagens:
https://developers.google.com/mediapipe/solutions/vision/image_segmenter
- [6] Aquisição Vídeo RGB OpenCV:
https://docs.opencv.org/3.4/dd/d43/tutorial_py_video_display.html
- [7] Funcionamento do OpenCV na Raspberry:
<https://stackoverflow.com/questions/60676022/how-to-make-opencv-work-fully-on-raspberry-pi>