# 1. 导入必要的库

|  |
| --- |
| from picamera2 import Picamera2  import cv2  import numpy as np  import face\_recognition  import os  import time  import sys  import webbrowser |

* from picamera2 import Picamera2：从 picamera2 库中导入 Picamera2 类，用于控制树莓派相机。
* import cv2：导入 OpenCV 库，用于图像处理，如读取、写入、显示图像，以及颜色空间转换等操作。
* import numpy as np：导入 NumPy 库，用于数值计算，例如处理数组和矩阵运算。
* import face\_recognition：导入 face\_recognition 库，用于人脸识别，包括检测人脸位置和获取人脸编码等功能。
* import os：导入 os 库，用于与操作系统进行交互，如文件和目录的操作。
* import time：导入 time 库，用于处理时间相关的操作，如设置等待时间。
* import sys：导入 sys 库，用于访问命令行参数和与 Python 解释器进行交互。
* import webbrowser：导入 webbrowser 库，用于在默认浏览器中打开指定的 URL。

# 2. 加载参考图像并编码人脸的函数

|  |
| --- |
| def load\_reference\_image(path):  image = cv2.imread(path)  if image is None:  raise FileNotFoundError(f"Failed to load image: {path}")  rgb = cv2.cvtColor(image, cv2.COLOR\_BGR2RGB)  face\_locations = face\_recognition.face\_locations(rgb)  if not face\_locations:  raise ValueError("No face detected in reference image")  return face\_recognition.face\_encodings(rgb, face\_locations)[0] |

* image = cv2.imread(path)：使用 OpenCV 的 imread 函数从指定路径 path 读取图像，并将其存储在 image 变量中。
* if image is None:：检查图像是否成功加载，如果 image 为 None，表示图像加载失败，抛出 FileNotFoundError 异常，并显示相应的错误信息。
* rgb = cv2.cvtColor(image, cv2.COLOR\_BGR2RGB)：将图像的颜色空间从 BGR（OpenCV 默认的颜色空间）转换为 RGB，因为 face\_recognition 库需要 RGB 格式的图像。
* face\_locations = face\_recognition.face\_locations(rgb)：使用 face\_recognition 库的 face\_locations 函数检测图像中人脸的位置，返回一个包含人脸位置坐标的列表。
* if not face\_locations:：检查是否检测到人脸，如果列表为空，表示没有检测到人脸，抛出 ValueError 异常，并显示相应的错误信息。
* return face\_recognition.face\_encodings(rgb, face\_locations)[0]：使用 face\_recognition 库的 face\_encodings 函数获取检测到的人脸的编码，返回第一个人脸的编码（假设图像中只有一个人脸）。

# 3. 初始化相机的函数

|  |
| --- |
| def init\_camera():  picam2 = Picamera2()  config = picam2.create\_preview\_configuration(  main={  "size": (640, 480), # Camera resolution  "format": "BGR888"  },  controls={  "FrameDurationLimits": (33333, 66666), # 30fps  "AwbMode": 0, # Auto white balance  "ExposureTime": 20000 # 20ms exposure  }  )  picam2.configure(config)  picam2.start()  return picam2 |

* picam2 = Picamera2()：创建一个 Picamera2 相机对象。
* config = picam2.create\_preview\_configuration(...)：创建相机的预览配置，设置相机的分辨率为 (640, 480)，格式为 BGR888，帧率限制为 30fps（通过 FrameDurationLimits 设置），白平衡模式为自动（AwbMode 设置为 0），曝光时间为 20ms（ExposureTime 设置为 20000）。
* picam2.configure(config)：将配置应用到相机上。
* picam2.start()：启动相机。
* return picam2：返回初始化并启动后的相机对象。

# 4. 保存新人脸图像并更新已知编码的函数

|  |
| --- |
| def save\_new\_face(frame, face\_location, name):  top, right, bottom, left = face\_location  face\_image = frame[top:bottom, left:right]  # Convert BGR to RGB  face\_image\_rgb = cv2.cvtColor(face\_image, cv2.COLOR\_BGR2RGB)  # Create a directory to save face images if it doesn't exist  if not os.path.exists('faces'):  os.makedirs('faces')  image\_path = os.path.join('faces', f'{name}.jpg')  cv2.imwrite(image\_path, face\_image\_rgb)  new\_encoding = load\_reference\_image(image\_path)  return new\_encoding |

* top, right, bottom, left = face\_location：从 face\_location 元组中解包出人脸在帧中的上、右、下、左坐标。
* face\_image = frame[top:bottom, left:right]：从原始帧 frame 中提取出人脸图像。
* face\_image\_rgb = cv2.cvtColor(face\_image, cv2.COLOR\_BGR2RGB)：将提取的人脸图像的颜色空间从 BGR 转换为 RGB。
* if not os.path.exists('faces'):：检查是否存在名为 faces 的目录，如果不存在，使用 os.makedirs 创建该目录。
* image\_path = os.path.join('faces', f'{name}.jpg')：构建保存人脸图像的路径，使用 os.path.join 函数将目录名 faces 和文件名 name.jpg 组合起来。
* cv2.imwrite(image\_path, face\_image\_rgb)：将转换后的人脸图像保存到指定路径。
* new\_encoding = load\_reference\_image(image\_path)：加载保存的人脸图像，并获取其编码。
* return new\_encoding：返回新保存人脸的编码。

# 5. 加载所有已知人脸的函数

|  |
| --- |
| def load\_all\_known\_faces():  known\_encodings = []  known\_names = []  if os.path.exists('faces'):  for filename in os.listdir('faces'):  if filename.endswith('.jpg'):  name = os.path.splitext(filename)[0]  image\_path = os.path.join('faces', filename)  try:  encoding = load\_reference\_image(image\_path)  known\_encodings.append(encoding)  known\_names.append(name)  except Exception as e:  print(f"Error loading {image\_path}: {str(e)}")  return known\_encodings, known\_names |

* known\_encodings = [] 和 known\_names = []：初始化两个空列表，分别用于存储已知人脸的编码和名称。
* if os.path.exists('faces'):：检查 faces 目录是否存在。
* for filename in os.listdir('faces'):：遍历 faces 目录中的所有文件。
* if filename.endswith('.jpg'):：检查文件是否以 .jpg 结尾，如果是，则处理该文件。
* name = os.path.splitext(filename)[0]：获取文件名（不包含后缀）。
* image\_path = os.path.join('faces', filename)：构建图像的完整路径。
* try: 块：尝试加载图像并获取其编码，将编码添加到 known\_encodings 列表中，将名称添加到 known\_names 列表中。如果加载过程中出现异常，打印错误信息。
* return known\_encodings, known\_names：返回包含所有已知人脸编码和名称的两个列表。

# 6. 主函数

|  |
| --- |
| def main():  if len(sys.argv) != 2:  print("Please provide a mode (1 or 2) as a command-line argument.")  return  choice = sys.argv[1]  # Load known faces  try:  known\_encodings, known\_names = load\_all\_known\_faces()  except Exception as e:  print(f"Initialization failed: {str(e)}")  return  # Initialize the camera  camera = init\_camera()  # Display parameters  SCALE\_FACTOR = 0.5  FONT = cv2.FONT\_HERSHEY\_SIMPLEX  THRESHOLD = 0.5  # Timer settings  TIMEOUT = 3 # Waiting time to ask for saving (seconds)  unknown\_start\_time = None  try:  while True:  try:  # Capture a frame  frame = camera.capture\_array()  except Exception as e:  print(f"Error capturing frame: {e}")  continue  frame\_rgb = cv2.cvtColor(frame, cv2.COLOR\_BGR2RGB)  # Preprocess  small\_frame = cv2.resize(  frame\_rgb,  (0, 0),  fx=SCALE\_FACTOR,  fy=SCALE\_FACTOR,  interpolation=cv2.INTER\_AREA  )  # Face detection  face\_locations = face\_recognition.face\_locations(small\_frame)  try:  face\_encodings = face\_recognition.face\_encodings(small\_frame, face\_locations)  except Exception as e:  print(f"Error encoding faces: {e}")  continue  # Recognition processing  all\_unknown = True  for (top, right, bottom, left), face\_encoding in zip(face\_locations, face\_encodings):  # Restore original coordinates  top = int(top / SCALE\_FACTOR)  right = int(right / SCALE\_FACTOR)  bottom = int(bottom / SCALE\_FACTOR)  left = int(left / SCALE\_FACTOR)  # Calculate matching distances  distances = face\_recognition.face\_distance(known\_encodings, face\_encoding)  if len(distances) > 0:  min\_distance = np.min(distances)  match\_index = np.argmin(distances)  # Determine identity  name = "Unknown"  color = (0, 0, 255) # Red  if min\_distance <= THRESHOLD:  name = known\_names[match\_index]  color = (0, 255, 0) # Green  all\_unknown = False  else:  name = "Unknown"  color = (0, 0, 255)  min\_distance = float('nan')  # Draw bounding box and label  cv2.rectangle(frame, (left, top), (right, bottom), color, 2)  text = f"{name} ({min\_distance:.2f})"  cv2.putText(frame, text, (left + 6, bottom - 6),  FONT, 0.5, color, 1)  # Handle unknown faces  if choice == '2' and all\_unknown:  if unknown\_start\_time is None:  unknown\_start\_time = time.time()  elif time.time() - unknown\_start\_time >= TIMEOUT:  # 打开保存人脸的网页，并传递当前帧和人脸位置信息  # 这里简单假设第一个检测到的人脸为要保存的人脸  if face\_locations:  face\_location = (  int(face\_locations[0][0] / SCALE\_FACTOR),  int(face\_locations[0][1] / SCALE\_FACTOR),  int(face\_locations[0][2] / SCALE\_FACTOR),  int(face\_locations[0][3] / SCALE\_FACTOR)  )  # 保存当前帧  cv2.imwrite('temp\_frame.jpg', frame)  # 保存人脸位置  with open('temp\_face\_location.txt', 'w') as f:  f.write(f"{face\_location[0]},{face\_location[1]},{face\_location[2]},{face\_location[3]}")  # 打开网页  webbrowser.open('http://127.0.0.1:5000/save\_face')  unknown\_start\_time = None  else:  unknown\_start\_time = None  # Display output  cv2.imshow('Face Recognition', cv2.cvtColor(frame, cv2.COLOR\_RGB2BGR))  # Exit condition  if cv2.waitKey(1) & 0xFF == ord('q'):  print("User terminated the program")  break  finally:  # Clean up resources  camera.stop()  camera.close()  cv2.destroyAllWindows()  print("System resources have been released") |

* if len(sys.argv) != 2:：检查命令行参数的数量是否为 2，如果不是，打印提示信息并返回。
* choice = sys.argv[1]：获取命令行参数中的模式选择（1 或 2）。
* try: 块：尝试加载所有已知人脸的编码和名称，如果加载过程中出现异常，打印错误信息并返回。
* camera = init\_camera()：初始化相机并获取相机对象。
* 定义显示参数和定时器设置：SCALE\_FACTOR 用于缩放图像，FONT 用于绘制文本的字体，THRESHOLD 用于判断人脸匹配的阈值，TIMEOUT 是等待保存未知人脸的时间，unknown\_start\_time 用于记录检测到未知人脸的起始时间。
* 进入 while True 循环：
  + try: 块：尝试捕获一帧图像，如果捕获过程中出现异常，打印错误信息并继续循环。
  + frame\_rgb = cv2.cvtColor(frame, cv2.COLOR\_BGR2RGB)：将捕获的图像颜色空间从 BGR 转换为 RGB。
  + small\_frame = cv2.resize(...)：对图像进行缩放处理，减小计算量。
  + 进行人脸检测和编码：使用 face\_recognition 库检测人脸位置和获取人脸编码，如果编码过程中出现异常，打印错误信息并继续循环。
  + 遍历检测到的人脸，进行识别处理：
    - 恢复人脸在原始图像中的坐标。
    - 计算人脸编码与已知人脸编码的距离，找到最小距离和匹配索引。
    - 根据最小距离和阈值判断人脸的身份，设置相应的名称和颜色，并在图像上绘制边界框和标签。
  + 处理未知人脸：如果选择模式 2 且所有检测到的人脸都是未知的，记录开始时间，当等待时间达到 TIMEOUT 时，保存当前帧和人脸位置信息，并打开网页。
  + cv2.imshow('Face Recognition', cv2.cvtColor(frame, cv2.COLOR\_RGB2BGR))：显示处理后的图像。
  + if cv2.waitKey(1) & 0xFF == ord('q'):：检查用户是否按下 q 键，如果是，打印提示信息并退出循环。
* finally: 块：在程序结束时，停止相机，关闭相机资源，销毁所有 OpenCV 窗口，并打印提示信息。

# 7. 程序入口

|  |
| --- |
| if \_\_name\_\_ == "\_\_main\_\_":  main() |

* 当脚本直接运行时，调用 main 函数启动程序。如果脚本被导入作为模块，则不会执行 main 函数。