Optical Character Reader

Code

```
import pytesseract
import PIL.Image
import cv2

# Specify the path to Tesseract executable if not in PATH
pytesseract.pytesseract.tesseract_cmd = r'C:\Program Files\Tesseract-OCR\tesseract.exe'

# Configuration for Tesseract OCR
myconfig = r"--psm 6 --oem 3"

# Path to the image (adjust this path to your image file)
image_path = r"C:\Users\shekh\Documents\Projects\OCR\dsp.jpg"

try:
    # Extract text from image using Tesseract OCR
    text = pytesseract.image_to_string(PIL.Image.open(image_path), config=myconfig)
    print("Extracted Text:\n", text)

    # Read image using OpenCV
    img = cv2.imread(image_path)
    height, width, _ = img.shape

    # Perform OCR on the image and get bounding boxes
    boxes = pytesseract.image_to_boxes(img, config=myconfig)

    # Draw bounding boxes on the image
    for box in boxes.splitlines():
        box = box.split(" ")
        img = cv2.rectangle(img, (int(box[1]), height - int(box[2])), (int(box[3]), height -
int(box[4])), (0, 255, 0), 2)

    # Display the image with bounding boxes
    cv2.imshow("img", img)
    cv2.waitKey(0)
    cv2.destroyAllWindows()

    # Extract text from the image again (optional, as it has already been done)
    text = pytesseract.image_to_string(PIL.Image.open(image_path), config=myconfig)
    print("Extracted Text Again:\n", text)

except PermissionError:
    print("Permission denied: Check the file path and ensure you have read permissions.")
except FileNotFoundError:
    print("File not found: Ensure the file exists at the specified path.")
```

```
except Exception as e:
    print(f"An error occurred: {e}")
```

How To Run

1. Install Required Libraries: Make sure you have Python installed on your machine. Then install the required libraries using pip.
   >>pip install pytesseract pillow opencv-python-headless

2. Install Tesseract-OCR: You need to have Tesseract installed on your system. Download from here: https://github.com/UB-Mannheim/tesseract/wiki

3. Download or Prepare Your Image: Make sure you have an image named dsp.jpg and adjust the image_path variable to point to your image file.

Explanation

### Code Explanation with Relevant Theory

```python
import pytesseract
import PIL.Image
import cv2
```
- **Imports**:
  - `pytesseract`: Python wrapper for Tesseract OCR engine.
  - `PIL.Image`: Part of the Pillow library, used for handling images.
  - `cv2`: OpenCV library for computer vision tasks.

#### Setting Tesseract Path and Configuration

```python
# Specify the path to Tesseract executable if not in PATH
pytesseract.pytesseract.tesseract_cmd = r'C:\Program Files\Tesseract-OCR\tesseract.exe'

# Configuration for Tesseract OCR
myconfig = r"--psm 6 --oem 3"
```
- **Tesseract Path**:
  - `pytesseract.pytesseract.tesseract_cmd`: Specifies the path to the Tesseract executable. This is necessary if Tesseract is not in the system's PATH.

- **Tesseract Configuration**:
  - `myconfig`: Defines configuration options for Tesseract OCR.
    - `--psm 6`: Page Segmentation Mode 6, which assumes a single uniform block of text.
    - `--oem 3`: OCR Engine Mode 3, which uses the default OCR engine.

#### Image Path and Text Extraction

```python
# Path to the image (adjust this path to your image file)
image_path = r"C:\Users\shekh\Documents\Projects\OCR\dsp.jpg"

try:
    # Extract text from image using Tesseract OCR
    text = pytesseract.image_to_string(PIL.Image.open(image_path), config=myconfig)
    print("Extracted Text:\n", text)
```
- **Image Path**:
  - `image_path`: Specifies the path to the image file (`dsp.jpg`) that will be processed.

- **Text Extraction**:
  - `PIL.Image.open(image_path)`: Opens the image using Pillow's Image module.
  - `pytesseract.image_to_string()`: Invokes Tesseract OCR to extract text from the opened image.
  - The extracted text is printed to the console.

#### Reading Image and Bounding Box Visualization

```python
    # Read image using OpenCV
    img = cv2.imread(image_path)
    height, width, _ = img.shape

    # Perform OCR on the image and get bounding boxes
    boxes = pytesseract.image_to_boxes(img, config=myconfig)
```
- **Image Reading**:
  - `cv2.imread(image_path)`: Reads the image using OpenCV (`img`).
  - `height, width, _ = img.shape`: Retrieves the dimensions (`height` and `width`) of the image.

- **Bounding Box Extraction**:
  - `pytesseract.image_to_boxes()`: Applies Tesseract OCR on the image to obtain bounding boxes (`boxes`) around detected characters.

#### Drawing Bounding Boxes

```python
    # Draw bounding boxes on the image
    for box in boxes.splitlines():
        box = box.split(" ")
        img = cv2.rectangle(img, (int(box[1]), height - int(box[2])), (int(box[3]), height - int(box[4])), (0, 255, 0), 2)
```
- **Drawing Bounding Boxes**:

  - Iterates over each line (`box`) in the `boxes` string (each line represents a bounding box).
  - Splits each line into components (`box.split(" ")`), where `box[1]` to `box[4]` represent coordinates of the bounding box.
  - `cv2.rectangle()`: Draws rectangles on the image (`img`) around the detected text regions using coordinates extracted from `boxes`.

#### Displaying and Saving Image

```python
    # Display the image with bounding boxes
    cv2.imshow("img", img)
    cv2.waitKey(0)
    cv2.destroyAllWindows()

    # Extract text from the image again (optional, as it has already been done)
    text = pytesseract.image_to_string(PIL.Image.open(image_path), config=myconfig)
    print("Extracted Text Again:\n", text)

except PermissionError:
    print("Permission denied: Check the file path and ensure you have read permissions.")
except FileNotFoundError:
    print("File not found: Ensure the file exists at the specified path.")
except Exception as e:
    print(f"An error occurred: {e}")
```
- **Displaying Image**:
  - `cv2.imshow("img", img)`: Displays the image (`img`) with drawn bounding boxes using OpenCV.
  - `cv2.waitKey(0)`: Waits for a key press to close the image window.
  - `cv2.destroyAllWindows()`: Closes all OpenCV windows.

- **Optional Text Extraction**:
  - Performs text extraction again (optional) after displaying the image.

#### Error Handling

```python
except PermissionError:
    print("Permission denied: Check the file path and ensure you have read permissions.")
except FileNotFoundError:
    print("File not found: Ensure the file exists at the specified path.")
except Exception as e:
    print(f"An error occurred: {e}")
```
- **Error Handling**:
  - Catches and handles specific exceptions (`PermissionError` and `FileNotFoundError`) related to file access and existence.

  - Catches any other exceptions with a generic `Exception` handler, printing the error message.

### Relevant Theory

- **Optical Character Recognition (OCR)**:
  - **OCR** is a technology that converts different types of documents containing printed or handwritten text into editable and searchable data.
  - **Applications**: OCR finds applications in document scanning, digital archiving, automated data entry, and more.

- **Tesseract OCR**:
  - **Tesseract** is one of the most popular open-source OCR engines developed by Google. It supports multiple languages and can process various image formats to extract text.

- **Page Segmentation Modes (PSM)**:
  - **PSM 6**: Assumes a single uniform block of text. This mode is appropriate for detecting text in typical documents where the text is presented as a single block.

- **OCR Engine Modes (OEM)**:
  - **OEM 3**: Uses the default OCR engine provided by Tesseract.

- **OpenCV (cv2)**:
  - **OpenCV** is a library of programming functions mainly aimed at real-time computer vision tasks. It provides tools for image processing, including reading, writing, and manipulating images.

- **Bounding Boxes**:
  - **Bounding boxes** are rectangular regions around detected objects or text in an image.
  - **Drawing Bounding Boxes**: Visualizes the positions and extents of recognized text regions.

### Summary

This Python script demonstrates how to leverage Tesseract OCR and OpenCV to perform optical character recognition on an image (`dsp.jpg`), visualize bounding boxes around detected text regions, display the annotated image, and handle common file access errors. Adjustments to Tesseract configuration (`myconfig`) and image path (`image_path`) can be made to suit specific OCR requirements and input images. This approach is foundational in automated text extraction from images for applications such as document digitization, text analysis, and data extraction.