UCAS
الكلية الجامعية
للعلوم التطبيقية

Student Name	Student ID
SHAHD EYAD ETHALATHINI	2202100527

Bar-Code Detection and Decoding

2D BARCODE
1D BARCODE

Introduction:

What is a Barcode?

A barcode is a machine-readable representation of information, typically in the form of a visual pattern of parallel lines (1D barcodes) or squares, dots, or other geometric patterns (2D barcodes, like QR codes). Barcodes are widely used for data storage, identification, and inventory management in industries such as retail, logistics, and healthcare.

Project Steps :

Image Acquisition:
The process begins with capturing an image using a camera or scanner. The image might include a barcode alongside other content.

Preprocessing:
Techniques such as grayscale conversion, contrast enhancement, and noise removal are applied to improve the quality of the image.

Region of Interest (ROI) Identification:
Algorithms like thresholding are used to identify areas of the image likely to contain a barcode.

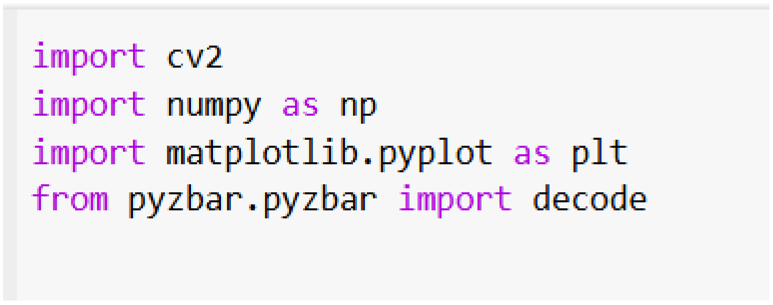
Barcode Localization:
The exact location of the barcode is identified, often by detecting repetitive line patterns for 1D barcodes

Bounding Box Extraction:
The region containing the barcode is extracted, and perspective distortions

are corrected to simplify the decoding process.

The Code Workflow:

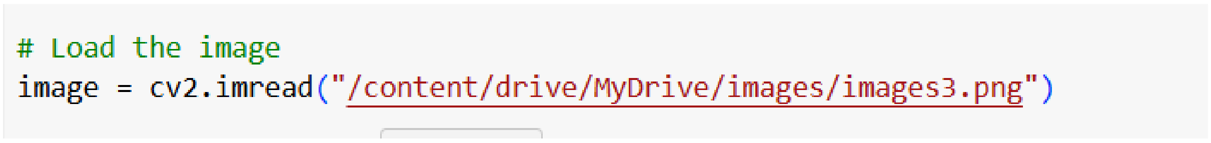
1- Import Libraries



```
import cv2
import numpy as np
import matplotlib.pyplot as plt
from pyzbar.pyzbar import decode
```

cv2: OpenCV library is used for image processing tasks like loading images, converting color spaces, and performing transformations such as blurring, thresholding, and morphological operations.
numpy: Used for numerical operations on arrays (manipulating image matrices).
matplotlib.pyplot: For visualizing and plotting images and intermediate results.
pyzbar: A Python library for decoding barcodes and QR codes in images.

2-Load the Image



```
# Load the image
image = cv2.imread("/content/drive/MyDrive/images/images3.png")
```

3-Convert Image to Grayscale

```
# Convert to grayscale
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
```

cv2.cvtColor: Converts the image from its original color format (BGR, used by OpenCV) to grayscale (single channel).

4- Apply Gradient Calculation (Edge Detection)

```
# Step 1: Apply Gradient Calculation
gradX = cv2.Sobel(gray, ddepth=cv2.CV_32F, dx=1, dy=0, ksize=-1)
gradY = cv2.Sobel(gray, ddepth=cv2.CV_32F, dx=0, dy=1, ksize=-1)
gradient = cv2.subtract(gradX, gradY)
gradient = cv2.convertScaleAbs(gradient)
```

Sobel Filter: Used to calculate the gradient of the image in both the X and Y directions.

cv2.Sobel: Computes the first derivative of the image with respect to the X-axis (dx=1, dy=0) and Y-axis (dx=0, dy=1).

ksize=-1: Specifies the kernel size for the Sobel filter.
The default value of -1 means the kernel size will be calculated automatically.

ddepth=cv2.CV_32F: Specifies the depth of the output image (32-bit floating point in this case).

Gradient Calculation:

gradX: Stores the gradient along the X-axis (horizontal edges).

gradY: Stores the gradient along the Y-axis (vertical edges).

cv2.subtract: Subtracts the Y-gradient from the X-gradient, enhancing edges by emphasizing the horizontal edges over the vertical ones.

cv2.convertScaleAbs: Converts the gradient image into an 8-bit image suitable for display by scaling and taking the absolute value of the gradients.

5- Blur the Image

```python
# Step 2: Blur the Image
blurred = cv2.GaussianBlur(gradient, (9, 9), 0)
```

6- Apply Binary Thresholding

```python
# Step 3: Apply Binary Thresholding
_, binary = cv2.threshold(blurred, 127, 255, cv2.THRESH_BINARY)
```

Converts the image into a binary image, where pixel values are either 0 (black) or 255 (white), depending on whether they are below or above the threshold.

7- Define the Kernel for Morphological Operations

```python
# Step 3: Define the Kernel
kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (5, 5))
```

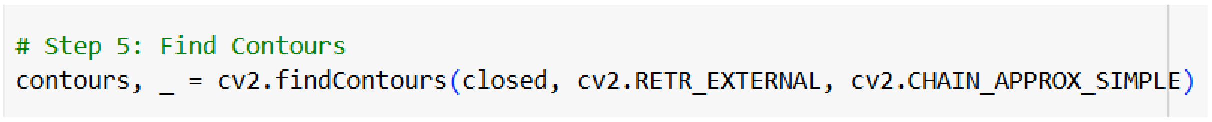
Defines a kernel (a small matrix) used for morphological operations such as dilation, erosion, opening, and closing.

8- Morphological Transformations

```python
# Apply morphological transformations (Erosion, Dilation, Opening, Closing)
eroded = cv2.erode(binary, kernel, iterations=1)
dilated = cv2.dilate(binary, kernel, iterations=1)
# Opening: Removes noise
opened = cv2.morphologyEx(binary, cv2.MORPH_OPEN, kernel)
# Closing: Fills gaps
closed = cv2.morphologyEx(binary, cv2.MORPH_CLOSE, kernel)
```

Erosion:
cv2.erode: Shrinks white regions and removes small noise points by applying the kernel to the binary image.
Dilation:
cv2.dilate: Expands white regions and helps close small gaps in objects.
Opening:
cv2.morphologyEx: which removes small noise or objects from the binary image.
Opening is a combination of erosion followed by dilation.
Closing:
cv2.morphologyEx: which fills small gaps in the foreground objects.
Closing is a combination of dilation followed by erosion.

9- Find Contours



```
# Step 5: Find Contours
contours, _ = cv2.findContours(closed, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
```

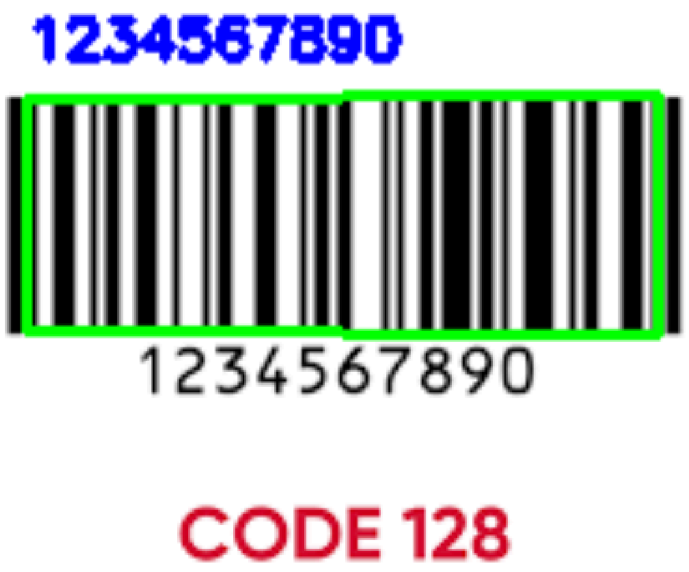
The contours will represent the outlines of objects or potential barcodes in the image.

10- Decoding the Barcode

Decode the barcode using the pyzbar library.

11- Results

```
Detected CODE128 barcode: 1234567890
```



Conclusion

The code demonstrates the power of image processing techniques in solving practical problems, such as automating barcode detection in industrial, retail, and healthcare environments.