Handwritten text to Digital Text[QR]

Mini Project report submitted in partial fulfillment of the Requirements for the Award of the Degree of

BACHELOR OF TECHNOLOGY

In

ELECTRONICS AND COMMUNICATION ENGINEERING

By

MAJJI SAI TEJA	-	**S200353**
NAKKINA CHARISHMA PRIYA	-	**S200354**
SINGAMALA KARTHIK REDDY	-	**S200724**
KOORAKULA ANJANA	-	**S200665**
DUKKA SWATHI	-	**S200993**
PULLAGURA NITHIN	-	**S200952**

Under the Guidance of

Mr.T.S.Gagandeep

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

RAJIV GANDHI UNIVERSITY OF KNOWLEDGE TECHNOLOGIES - SRIKAKULAM

Figure 1:
h

RAJIV GANDHI UNIVERSITY OF KNOWLEDGE TECHNOLOGIES - SRIKAKULAM

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

CERTIFICATE

This is to certify that the mini project report entitled **HANDWRITTEN TEXT TO DIGITAL TEXT[QR]** being submitted by

Majji Sai Teja	-	S200353
Nakkina Charishma Priya	-	S200354
Singamala Karthik Reddy	-	S200724
koorakula Anjana	-	S200665
Dukka Swathi	-	S200993
Pullagura Nithin	-	S200952

in partial fulfillment for the award of the Degree of Bachelor of Technology in Electronics and Communication Engineering, RGUKT-Srikakulam. This is a record of bonafide work carried out under my guidance and supervision.

Mr.T.S.Gagandeep

DECLARATION

I hereby declare that the dissertation entitled **Handwritten text to Digital Text**[QR] submitted for the B.Tech Degree is my original work, and the dissertation has not formed the basis for the award of any degree, associateship, fellowship, or any other similar titles.

Place: Nuzvid **Student Name**
Date: May 6, 2025 **Reg No:12**

ACKNOWLEDGMENT

We would like to express our sincere gratitude to everyone who supported us in completing this report on High Speed Image Processing Using Verilog HDL

Firstly, we extend our heartfelt thanks to **Mr.T.S.Gagandeep sir**, our esteemed mentor, for his valuable guidance, insightful feedback, and continuous support throughout the preparation of this document. His expertise and encouragement motivated us to explore the topic in depth and ensure the accuracy and relevance of the information presented.

We are also grateful to our institution for providing the necessary resources and a conducive environment for conducting this study. Special thanks to our team members for their dedication, teamwork, and collaborative effort, which were crucial for the successful completion of this report.

Lastly, we acknowledge the numerous sources and research studies that provided the data and insights that formed the basis of our work.

Team Members:

Majji Sai Teja
Nakkina Charishma Priya
Singamala Karthik Reddy
koorakula Anjana
Dukka Swathi
Pullagura Nithin

ABSTRACT

Handwritten text to Digital text

In the digital age, the ability to seamlessly convert handwritten content into a digital format is crucial for many applications, ranging from educational tools to document management systems. This project explores an innovative approach to achieve this by utilizing the ESP32 CAM module, a low-cost, efficient device with both camera and Wi-Fi capabilities, to capture handwritten text and convert it into a video. The process begins with the real-time capture of handwritten text using the ESP32 CAM, which records the writing process. This video is then processed and converted into a QR code, a widely used format for storing data in a scannable form. The QR code contains the digital representation of the handwritten text. By scanning the generated QR code with a smartphone or QR reader, the text can be extracted, converted, and displayed in its digital form.

The system integrates various stages, including:

1. *Handwritten Text Capture*: Using the ESP32 CAM, the system records the writing of text on paper, converting the analog information into a video format.

2. *QR Code Generation*: The recorded video is processed, and the corresponding digital text is extracted from it. This text is encoded into a QR code.

3. *QR Code Scanning Text Extraction*: A simple mobile app or device is used to scan the QR code, which decodes the information into the original handwritten text. This approach has several potential applications, including creating an efficient method for digitizing handwritten notes, forms, or documents, and could be used in scenarios where quick digitalization and sharing of handwritten content are needed. It offers a novel way to bridge the gap between analog handwriting and digital formats while utilizing affordable hardware like the ESP32 CAM module for real-time video processing and QR code generation.

Key Features: - *ESP32 CAM Module* for real-time video capture. - *QR Code Encoding* to store digital text from handwriting. - *QR Code Decoding* to retrieve the original text. - *Cost-effective and efficient solution* for converting handwritten content into a shareable and accessible digital format.

Contents

List of Figures

CHAPTER 1

1 INTRODUCTION

1.1 Introduction to Project

In today's fast-paced digital world, there is a growing need to quickly convert handwritten information into digital formats for ease of sharing, storage, and processing. While there are existing solutions for digitizing handwritten content, such as OCR (Optical Character Recognition) software, they often require complex setups or specialized equipment. This project presents an innovative approach to convert handwritten text into a digital format using the ESP32 CAM module, a low-cost, versatile microcontroller with a built-in camera. The ESP32 CAM is capable of capturing real-time video, and in this project, it is used to record the writing of handwritten text.

Once the handwriting is captured, the video is processed, and the resulting text is encoded into a QR code. This QR code acts as a digital container for the handwritten information, allowing it to be easily shared, scanned, and decoded back into the original text. By scanning the QR code with a simple mobile device or scanner, users can quickly retrieve and view the digital representation of the handwritten content.

The key idea behind this project is to integrate the power of video capture, QR codes, and text extraction to create an efficient, low-cost, and easy-to-use method for digitizing handwritten notes, documents, and forms. This system could be particularly useful in educational environments, offices, or any scenario where handwritten content needs to be quickly converted into a digital format for further processing or sharing.

Overall, this project bridges the gap between analog and digital formats, utilizing affordable hardware and modern technology to offer a streamlined solution for digitizing handwritten content through QR codes.

2 Project Overview

This project, titled **"Handwritten to QR: Converting Handwritten Text to Digital Code via Video Using ESP32 CAM"**, focuses on the development of a low-cost, portable system that automates the conversion of handwritten content into a digital format and stores it in a QR code for easy retrieval and sharing.

The system leverages the ESP32-CAM module, which combines camera functionality with Wi-Fi capabilities, to capture handwritten text in real-time. This video is processed frame-by-frame using image processing techniques and Optical Character Recognition (OCR) to extract the written text. Once the text is digitized, it is encoded into a QR code using Python-based tools.

The final output—a QR code containing the original handwritten text—can be scanned using any mobile device or QR reader to retrieve the text. This project demonstrates the seamless integration of analog and digital technologies using open-source tools and affordable hardware.

The key stages of the project include:

- Capturing handwritten input using the ESP32-CAM.
- Processing video frames to extract clean text using OCR.
- Generating a QR code that stores the extracted digital text.
- Enabling easy access to the text via QR code scanning.

This approach presents a practical solution for digitizing handwritten documents quickly, with applications in education, documentation, and data archival.

3 Problem Statement

In many educational, administrative, and documentation environments, handwritten notes and forms are still widely used. However, converting these handwritten materials into digital formats for storage, sharing, or processing remains a manual and time-consuming task. Traditional methods of digitization involve scanning documents and manually typing the content, which is inefficient and prone to human error.

There is a lack of an affordable, automated system that can:

- Capture handwritten input in real-time,
- Convert it into digital text accurately, and
- Encode and transmit the digital content in a compact, scannable format.

This project aims to solve this problem by developing a low-cost solution using the ESP32-CAM module to capture handwritten text, convert it into digital form using Optical Character Recognition (OCR), and generate a QR code that encapsulates the extracted content. The final output can be easily scanned and retrieved on any mobile device, bridging the gap between analog handwriting and digital information systems.

3.1 Objectives

- Convert Handwritten Text into Digital Format
- Generate QR Codes for Digital Representation
- Enhance Accessibility of Handwritten Information
- Implement Real-Time Video Capture with ESP32 CAM
- Demonstrate Efficient Use of QR Codes for Data Storage and Retrieva
- Create a Cost-Effective and Practical Solution for Digitizing Handwritten Documents
- Facilitate Easy Data Extraction Through QR Code Scanning
- Explore Real-World Applications of the Technology
- Promote Interoperability Between Analog and Digital Content

3.2 Methodology

Here is the finalized and detailed Methodology section for your project: The methodology of this project is structured into four key stages, each playing a crucial role in the successful conversion of handwritten text into a digital format and its encoding into a QR code. The system integrates both hardware and software components, primarily utilizing the ESP32-CAM module and Python-based processing tools.

3.3 Flow chart

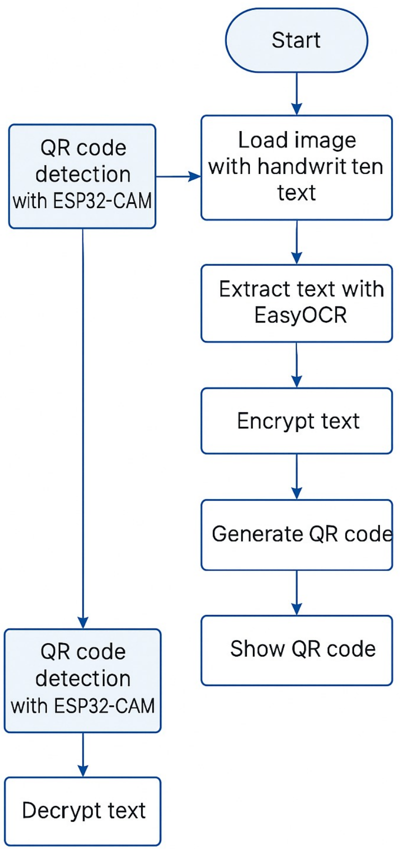

Start
QR code detection with ESP32-CAM
Load image with handwrit ten text
Extract text with EasyOCR
Encrypt text
Generate QR code
Show QR code
QR code detection with ESP32-CAM
Decrypt text

Figure 2: Flow chart

3.4 1. Handwritten Text Capture Using ESP32-CAM

- The ESP32-CAM module, equipped with a built-in camera, is used to record a video of the user writing text on paper.
- The camera is positioned steadily to ensure clear visibility of the handwriting.
- The captured video is either saved locally to an SD card or transmitted to a computer system via Wi-Fi for further processing.

3.5 2. Frame Extraction and Preprocessing

- The recorded video is processed using Python and OpenCV to extract individual frames.
- Each frame is converted to grayscale, and filtering techniques (such as thresholding and noise removal) are applied to enhance the text visibility.
- Redundant or duplicate frames are removed to ensure efficient processing.

3.6 3. Text Extraction Using OCR

- Tesseract OCR, an open-source optical character recognition engine, is used to analyze each processed frame and extract the handwritten text.
- Extracted texts from multiple frames are combined and cleaned to form a continuous digital text output.
- Basic text post-processing is performed to correct recognition errors.

3.7 4. QR Code Generation

- The final cleaned digital text is passed to a QR code generator built using Python libraries such as `qrcode`.
- The QR code is created and stored as an image file or displayed for scanning.
- The generated QR code can be printed, shared, or scanned using any QR reader app or device.

3.8 5. QR Code Scanning and Retrieval

- When scanned using a mobile phone or compatible device, the QR code reveals the original handwritten text in digital form.
- This digital text can be copied, stored, or transmitted as needed.

This step-by-step methodology demonstrates how a simple and low-cost setup can bridge the gap between handwritten input and digital storage, enabling easier. Handwritten text extraction, also known as handwriting recognition, typically involves a process that combines computer vision and machine learning techniques. The methodology generally includes image acquisition, preprocessing, segmentation, feature extraction, classification, and post-processing.

1. Image Acquisition:

The process starts with obtaining an image of the handwritten text, whether it's a scanned document, a photograph, or digital ink captured from a digitizer.

2. Preprocessing:

This step involves improving the quality of the image to make it easier for subsequent analysis. Common techniques include: Noise reduction: Filtering out noise artifacts like speckles or background clutter. Binarization: Converting the grayscale image into a black and white image, separating the text from the background. Image enhancement: Adjusting brightness, contrast, or skew correction to optimize the image for recognition.

3. Segmentation:

This step involves separating the image into individual characters, words, or even lines of text. Different approaches include: Character segmentation: Identifying individual character boundaries using methods like connected component analysis or morphological operations. Word segmentation: Grouping segmented characters into words based on spacing and other visual cues. Line segmentation: Identifying lines of text based on vertical projection profiles or other structural features.

4. Feature Extraction:

Extracting relevant features from the segmented text regions to distinguish between different characters. Common features include: Shape features: Analyzing the overall shape and contours of the characters. Edge features: Identifying prominent edges and boundaries within the characters. Statistical features: Calculating statistics like the number of strokes, area, or perimeter of the characters.

5. Classification:

Using machine learning algorithms to classify the extracted features into corresponding characters. Common methods include: Neural networks: Convolutional Neural Networks (CNNs) are frequently used for their ability to learn hierarchical features from image data. Support Vector Machines (SVMs): Another popular algorithm used for classification based on feature vectors. Decision trees: Used for classifying characters based on a hierarchy of decisions based on features.

6. Post-processing:

Refining the recognition results and handling ambiguities or errors. This step might include: Error correction: Using language models or dictionaries to correct misidentified characters. Contextual analysis: Using surrounding words or sentences to disambiguate potentially ambiguous characters. Smoothing: Applying smoothing techniques to reduce noise and improve the readability of the extracted text.

3.9 Applcations

1. Educational Sector

2. Healthcare and Medical

3. Business and Office Use
4. Legal and Compliance Applications
5. Financial and Banking Industry
6. Transaction Record
7. Creative and Design Fields

3.10 QR Generation

Certainly! Here's more detailed information on each stage of the QR Code Generation Process, along with its technical aspects and structure.

1. Input Data

This is the information you want to store in the QR code. It can be:

Text (e.g., "Hello World")

URL (e.g., https://example.com)

vCard or contact info

Wi-Fi credentials

Calendar event, etc.

The length and type of data affect the size and complexity of the QR code.

2. Data Analysis and Mode Selection

The QR code system analyzes the data and selects the most efficient encoding mode:

Numeric: 0–9 (max 7,089 characters)

Alphanumeric: 0–9, A–Z, and some symbols (max 4,296 characters)

Byte mode: UTF-8 characters (max 2,953 characters)

Kanji mode: Double-byte characters used in Japanese (max 1,817 characters)

This helps keep the QR code compact and efficient.

3. Data Encoding

The data is converted into a binary format according to its mode. The result includes:

Mode indicator: 4 bits

Character count indicator

Encoded data bits

Example: If you encode "HELLO" in alphanumeric mode:

Mode indicator: 0010

Character count: 00000101

Encoded data: Bits representing each character group

4. Error Correction Coding

This stage generates redundant bits that help restore data if the QR code is damaged or dirty.

QR codes use Reed-Solomon error correction.

Based on the chosen level (L, M, Q, H), the system appends extra data blocks to detect and correct errors.

These error-correcting blocks are calculated mathematically using polynomial division.

5. Structure Final Message

All parts are combined into one data stream:

Original encoded data

Error correction code

Terminator bits (to fill the remainder)

Padding (if needed) using 11101100 and 00010001 alternately

6. Module Placement

Now the bit stream is placed in a 2D matrix of black and white modules (squares).

QR codes include:

Finder Patterns: Large squares in 3 corners

Timing Patterns: Alternating black/white in rows/columns

Alignment Patterns: Help correct distortion

Version Information: Indicates QR version (size)

7. Masking

Masking improves readability by avoiding large blank or dark areas. There are 8 predefined mask patterns. The one with the lowest penalty score (based on contrast and pattern repetition) is selected.

8. Format and Version Information

These are small segments in the QR code that:

Describe the error correction level

Specify the mask pattern

Define the version (size of the QR code from version 1 to 40)

9. Final QR Code Image

All parts are assembled and output as an image file:

PNG, SVG, JPG, or PDF

Can be printed or scanned by mobile devices

Technical Structure Summary

Matrix size: Ranges from 21×21 (Version 1) to 177×177 (Version 40)

Capacity: From 25 to over 4,000 characters

Correction: Up to 30

A flowchart or diagram summarizing this process

A comparison of QR vs Barcode.

1. Install the Required Tool You first need to install a Python package called qrcode, which is a library that makes it easy to generate QR codes. It also uses another tool called Pillow to create images.

2. Import the Library After installation, you bring the library into your Python program so you can use its features.

3. Prepare the Data Decide what information you want to store in the QR code. This could be:

A website link

A block of text

A phone number

An email address

Wi-Fi login credentials

4. Create a QR Code Object You then create an object that represents the QR code. When creating it, you can:

Set how much data it should hold (called the "version").

Define how much error correction it should allow (to recover from damage).

Set the size of the boxes in the QR code.

Define how thick the border around the QR code should be.

5. Add Your Data The data you prepared earlier (like a URL or message) is added to the QR code object.

6. Create the Image The QR code object is converted into an image format (like PNG), which visually represents the code using black and white squares.

7. Save the Image Finally, the image is saved to your computer so you can use it—whether it's printed on paper, embedded in a website, or scanned by a phone.

3.11 Componenets

The following are the hardware requirements in this project.

- ESP32 CAM Module
- Handwritten Text Input (Paper and Pen)
- QR Code (Quick Response Code)

3.11.1 Description

1.ESP32 CAM Module : The ESP32-CAM is a compact, low-cost camera module based on the ESP32-S chip, featuring Wi-Fi and Bluetooth capabilities. It includes an OV2640 camera (or optionally OV7670), a TF card slot for storage, and various interfaces. Its compact size and ability to stream images wirelessly make it suitable for IoT applications like smart home devices, wireless monitoring, and QR code identification. ESP32 Features Specifications The features and specifications of ESP32 Cam include the following.

3.12 Applications

Applications The applications of ESP32 Cam include the following.

ESP32-CAM module is used widely in a variety of IoT applications. These are appropriate for industrial wireless control, home smart devices, wireless monitoring, wireless positioning system signals, QR wireless identification, etc. These modules offer a low-cost method to design very advanced home automation projects which include different features like taking photos, video, face recognition, etc. ESP32-CAM is an ideal solution for DIY projects prototype constructions. It is used to design a face recognition system devoid of any complex programming any additional components ESP32-CAM simply adopts a DIP package which provides customers with high-reliability connection techniques, so it is very convenient mainly in a variety of IoT applications.

2.Handwritten Text Input (Paper and Pen) : Pen and paper are used in text extraction by creating handwritten notes, documents, or forms, which are then digitized using OCR (Optical Character Recognition) or ICR (Intelligent Character Recognition) software. This technology allows the computer to "read" the handwritten text and convert it into a digital format like text or PDF, making it searchable and editable.

3.QR Code (Quick Response Code) QR codes are two-dimensional barcodes that store information like URLs, contact details, and text. They can store significantly more data than traditional barcodes and can be scanned quickly by mobile devices. QR codes are used for various applications, including payments, marketing, product tracking, and access control. Key Features: Versatile Data Storage: QR codes can store a wide range of data, including alphanumeric characters, URLs, text, and even multimedia files. High Data Capacity: They can encode much more information than traditional barcodes. Small Size: QR codes can represent data using significantly less space than barcodes, making them ideal for applications where space is limited. High Speed and Angle Independence: QR codes can be scanned quickly and accurately, regardless of the scanning angle. Durability: They are resistant to dirt and damage, with error correction features that can restore data even if a part of the code is missing. Customization: QR codes can be customized with logos and colors. Tracking and Analysis: Some QR code generators allow tracking of scans and even GPS location, enabling marketers to analyze campaign effectiveness. Uses: Payments: QR codes are widely used for mobile payments, allowing users to make transactions by scanning a QR code. Marketing and Advertising: QR codes are used to link to websites, product information, videos, and promotions, making it easy for consumers to access additional content. Information Sharing: QR codes can be used to share contact information, website links, and other digital resources. Product Tracking: QR codes can be used to track products throughout the supply chain, helping to prevent errors and anti-counterfeiting. Access Control: QR codes can be used to grant access to buildings, events, or other restricted areas. Inventory Management: QR codes can be used to track inventory levels and manage stock effectively. Restaurant Menus: QR codes are used in restaurants to provide menus and order-taking options on mobile devices. Ticketing: QR codes are used in transportation, entertainment, and other sectors for ticketing and access control. Public Travel: QR codes are used on bus stops and routes, providing travelers with real-time information and travel planning tools. A QR code, short for Quick Response Code, is a two-dimensional barcode that stores information in a matrix of black and white modules. It's scanned by a smartphone camera to access data, like URLs, text, or payment information.

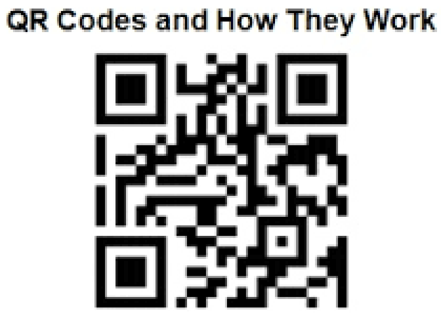


Figure 3: QR CODE

3.13 Software Components

1.PYTHON : Python is widely used in text recognition (Optical Character Recognition or OCR) due to its ease of use, powerful libraries, and ability to integrate with machine learning techniques. Key applications of Python in OCR include extracting text from images, converting scanned documents to digital formats, automating data entry, and improving the accuracy of OCR systems through image processing and deep learning. Here's a more detailed breakdown of Python's role in text recognition: 1. Libraries for OCR: Pytesseract: A Python wrapper for the Tesseract OCR engine, allowing easy integration with Python programs. EasyOCR: A Python library based on deep learning, offering support for numerous languages and robust text recognition capabilities. OpenCV: A powerful library for image processing, often used for preprocessing images to enhance OCR accuracy. TensorFlow and PyTorch: Deep learning frameworks that can be used to train OCR models, improving accuracy and efficiency. docTR: A library for text detection and recognition in documents, including PDFs and images. 2. Common OCR Tasks in Python: Extracting text from images: Using libraries like Pytesseract and EasyOCR, Python can convert images containing text into machine-readable text formats. Converting scanned documents to digital formats: OCR can transform scanned documents, photos, and other images into editable and searchable files. Automating data entry: Extracting information from forms, invoices, receipts, and other documents to automatically populate databases or spreadsheets. Improving OCR accuracy: Using image processing techniques (OpenCV) to enhance image quality and prepare them for OCR, and leveraging machine learning models (TensorFlow, PyTorch) to train more accurate OCR systems. 3. Why Python is a good choice for OCR: Ease of Use: Python's syntax is relatively simple and readable, making it easier to implement OCR tasks compared to other languages. Extensive Libraries: Python has a rich ecosystem of libraries dedicated to image processing, OCR, and machine learning, providing a wide range of tools for building and customizing OCR systems. Machine Learning Integration: Python's deep learning frameworks (TensorFlow, PyTorch) allow for the development of sophisticated OCR models that can handle complex images and various text formats. Cross-Platform Compatibility: Python is a cross-platform language, meaning OCR solutions can be developed and run on different operating systems.

The provided diagram represents a typical image processing flow using a Xilinx System Generator, which is commonly employed for implementing image processing algorithms on FPGA platforms. The process begins with the image source, which could be a camera, a stored image file, or any other device capable of generating digital image data. This data is first fed into the image pre-processing block, where initial operations such as noise reduction, contrast enhancement, resizing, or color space conversion are performed to prepare the image for more complex processing. The pre-processed image is then passed into the core of the system: the image processing algorithm block, which is enclosed within a dashed box labeled "using Xilinx system Generator." This highlights the hardware-specific implementation of the main algorithm using Xilinx tools, which can significantly accelerate processing by leveraging FPGA resources. This stage typically involves tasks like formatting the processed image for display, re-scaling, annotation, or any final cleanup required to make the image suitable for viewing or further use. Finally, the image is output to the image viewer, which could be a graphical display, software interface, or storage device for visualization or analysis. This flow ensures a structured approach to real-time image processing by dividing the task into logical stages, each optimized for a specific set of operations, and demonstrates how hardware-accelerated solutions like those from Xilinx can be integrated seamlessly into complex image processing pipelines for high-performance applications.

3.14 Program Code

3.14.1 code

```
{
import cv2
import easyocr
import qrcode
from PIL import Image

# Step 1: Load the image with handwritten text
image_path = 'text.jpg'  # Replace with your image path
image = cv2.imread(image_path)
```

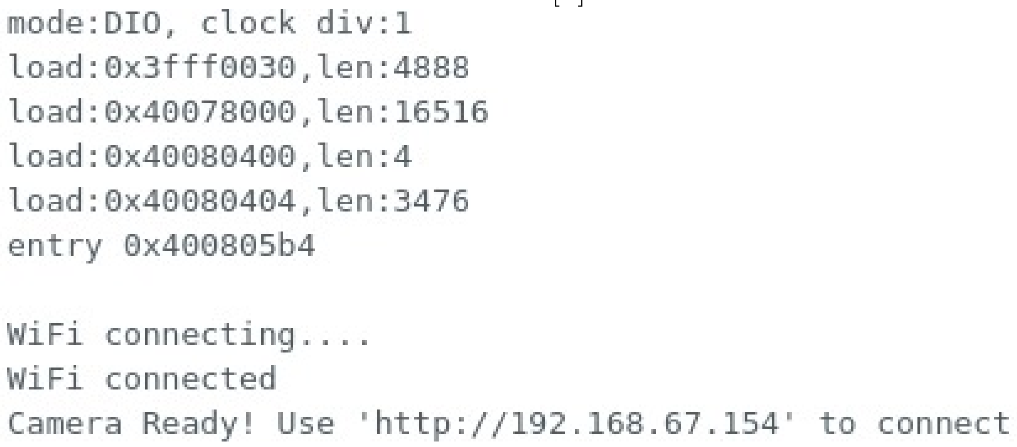
```
# Step 2: Use EasyOCR to extract text
reader = easyocr.Reader(['en'])  # Language = English
results = reader.readtext(image)

# Step 3: Concatenate all detected text
extracted_text = " ".join([res[1] for res in results])
print("Extracted Text:", extracted_text)

# Step 4: Generate QR Code from the extracted text
qr = qrcode.QRCode(version=1, box_size=10, border=5)
qr.add_data(extracted_text)
qr.make(fit=True)

qr_img = qr.make_image(fill_color="black", back_color="white")
qr_img_path = 'output_qr.png'
qr_img.save(qr_img_path)

# Step 5: Show the QR Code
qr_img.show()
```

3.14.2 encrypting qrcode

```
   import cv2
import easyocr
import qrcode
from PIL import Image
from pyzbar.pyzbar import decode
import numpy as np
from cryptography.fernet import Fernet



def encrypt_text(text):
    with open("secret.key", "rb") as key_file:
        key = key_file.read()
    #with open("save.txt", "wb") as key_f:
    #    key_f.write(text)

    fernet = Fernet(key)
    encrypted = fernet.encrypt(text.encode())
    return encrypted.decode()  # QR code requires a string


def decrypt_text(encrypted_text):
    with open("secret.key", "rb") as key_file:
        key = key_file.read()
    fernet = Fernet(key)
    decrypted = fernet.decrypt(encrypted_text.encode())
    return decrypted.decode()


def imgtotexttoqr():
    # Generate a key and save it securely (do this only once)
    key = Fernet.generate_key()
    with open("secret.key", "wb") as key_file:
        key_file.write(key)

    # Step 1: Load the image with handwritten text
    image_path = 'text.jpg'  # Replace with your image path
    image = cv2.imread(image_path)

    # Step 2: Use EasyOCR to extract text
    reader = easyocr.Reader(['en'])  # Language = English
    results = reader.readtext(image)

    # Step 3: Concatenate all detected text
    extracted_text = " ".join([res[1] for res in results])
    encrypted_text = encrypt_text(extracted_text)

    print("Encrypted Text for QR Code:", encrypted_text)

    # Step 4: Generate QR Code from the extracted text
    qr = qrcode.QRCode(version=1, box_size=10, border=5)
    qr.add_data(encrypted_text)
    qr.make(fit=True)

    qr_img = qr.make_image(fill_color="black", back_color="white")
    qr_img_path = 'output_qr.png'
    qr_img.save(qr_img_path)

    # Step 5: Show the QR Code
    qr_img.show()


def qrdetectioncam():
    # Initialize ESP32-CAM stream
    url = 'http://192.168.114.154:81/stream'
    cap = cv2.VideoCapture(url)

    while True:
        ret, frame = cap.read()
        if not ret:
            break

        # Decode QR codes from the frame
        decoded_objects = decode(frame)

        for obj in decoded_objects:
            # Draw rectangle around QR code
            points = obj.polygon
            if len(points) == 4:
                pts = [(point.x, point.y) for point in points]
                cv2.polylines(frame, [np.array(pts, dtype=np.int32)], True, (0, 255, 

            # Decode and decrypt QR code text
            qr_text = obj.data.decode('utf-8')
            try:
                decrypted_text = decrypt_text(qr_text)
            except Exception as e:
                print("Error decrypting QR code:", e)

            x = input("Enter code: ")
            if (x == qr_text):
                    print("Decrypted QR Code Text:", decrypted_text)

        # Display the frame
        cv2.imshow('QR Code Scanner', frame)

        # Exit on 'q' key press
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break

    cap.release()
    cv2.destroyAllWindows()



# Menu
while True:
    print("\n1. QRCode detection with ESP32-CAM (Wi-Fi required)")
    print("2. Extract data from text.jpg and convert to QRCode")
    x = int(input("Enter 1/2/0 (0 for quit): "))
    if x == 1:
        qrdetectioncam()
```

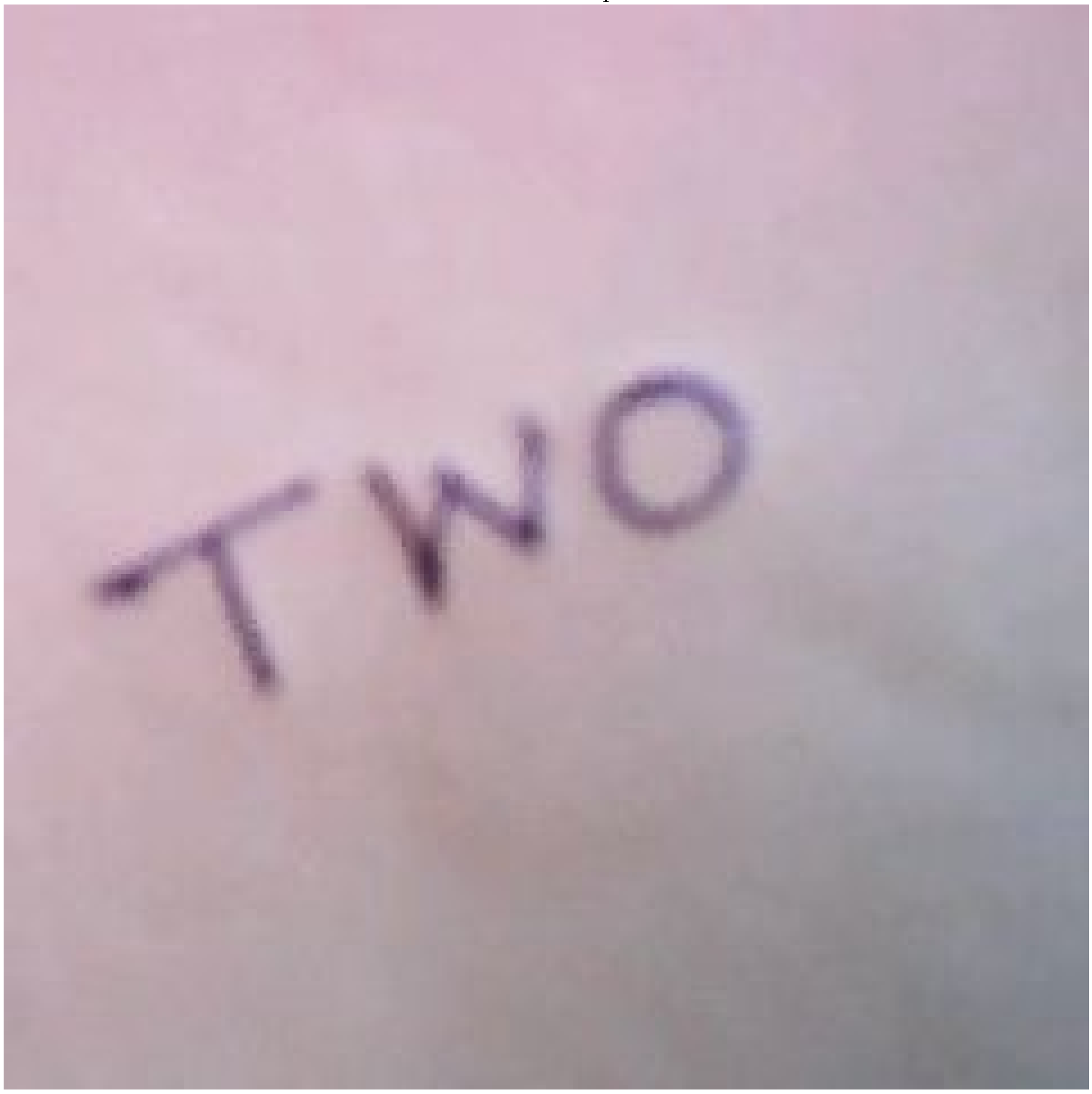
```
elif x == 2:
    imgtotexttoqr()
elif x == 0:
    break
else:
    print("Enter either 1 or 2")
```

[b]0.45



```
mode:DIO, clock div:1
load:0x3fff0030,len:4888
load:0x40078000,len:16516
load:0x40080400,len:4
load:0x40080404,len:3476
entry 0x400805b4

WiFi connecting....
WiFi connected
Camera Ready! Use 'http://192.168.67.154' to connect
```

ESP Cam Output



input

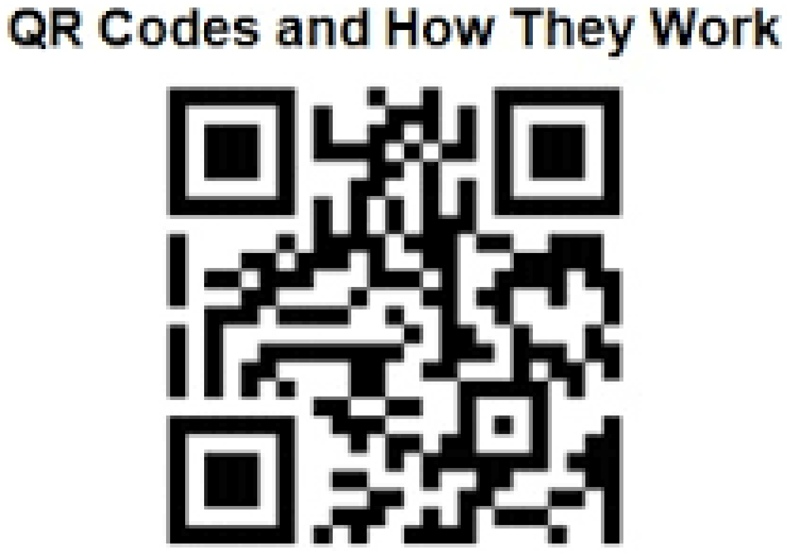
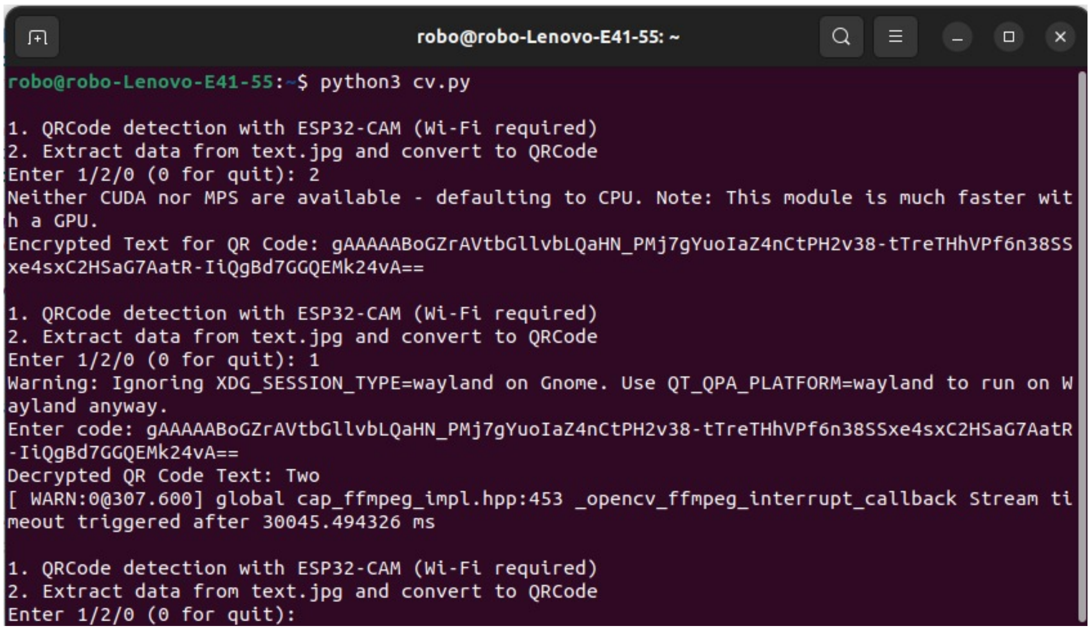


Figure 4: QR CODE

QR CODE



```
robo@robo-Lenovo-E41-55:~$ python3 cv.py

1. QRCode detection with ESP32-CAM (Wi-Fi required)
2. Extract data from text.jpg and convert to QRCode
Enter 1/2/0 (0 for quit): 2
Neither CUDA nor MPS are available - defaulting to CPU. Note: This module is much faster wit
h a GPU.
Encrypted Text for QR Code: gAAAAABoGZrAVtbGllvbLQaHN_PMj7gYuoIaZ4nCtPH2v38-tTreTHhVPf6n38SS
xe4sxC2HSaG7AatR-IiQgBd7GGQEMk24vA==

1. QRCode detection with ESP32-CAM (Wi-Fi required)
2. Extract data from text.jpg and convert to QRCode
Enter 1/2/0 (0 for quit): 1
Warning: Ignoring XDG_SESSION_TYPE=wayland on Gnome. Use QT_QPA_PLATFORM=wayland to run on W
ayland anyway.
Enter code: gAAAAABoGZrAVtbGllvbLQaHN_PMj7gYuoIaZ4nCtPH2v38-tTreTHhVPf6n38SSxe4sxC2HSaG7AatR
-IiQgBd7GGQEMk24vA==
Decrypted QR Code Text: Two
[ WARN:0@307.600] global cap_ffmpeg_impl.hpp:453 _opencv_ffmpeg_interrupt_callback Stream ti
meout triggered after 30045.494326 ms

1. QRCode detection with ESP32-CAM (Wi-Fi required)
2. Extract data from text.jpg and convert to QRCode
Enter 1/2/0 (0 for quit):
```

Figure 5: final output

3.15 Working

Working of the Project: ”Handwritten to QR: Converting Handwritten Text to Digital Code via ESP32-CAM”

This project converts handwritten text into a QR code using image capture, text recognition (OCR), and QR generation. Here’s how it works step-by-step:

1. Image Capture (via ESP32-CAM or uploaded image)

The user writes some text on paper.

The ESP32-CAM captures an image of this handwritten note.

The image is then either:

Sent to a local server via Wi-Fi, or

Uploaded to a processing script manually.

2. Text Extraction (OCR using EasyOCR)

The captured image is processed by an OCR (Optical Character Recognition) engine like EasyOCR.

EasyOCR scans the image and identifies any handwritten characters or words.

It returns the recognized text from the image.

3. Text Processing

The recognized text is combined into a single sentence or paragraph.

This text becomes the input for QR generation.

4. QR Code Generation

A QR code is generated using the Python qrcode library.

The extracted text is encoded into the QR format.

The result is a scannable QR code containing the same text that was handwritten.

5. Output and Display

The QR code is saved as an image file (e.g., output$_q r.png$).

It is displayed on screen and can also be scanned using any QR scanner.

When scanned, it shows the original handwritten text in digital form.

Digitizing handwritten notes

Secure message sharing via QR

Classroom or office automation (e.g., converting whiteboard notes)

IoT or embedded systems where keyboard input isn't practical. This Python program is a complete handwritten-to-secure-QR system with encryption and real-time scanning using an ESP32-CAM. Here's a detailed explanation of the entire code, broken down by functionality:

6.imgtotexttoqr() – Extracts text from a handwritten image, encrypts it, and converts it to a QR code.

7. qrdetectioncam() – Uses the ESP32-CAM to scan a QR code in real-time, decrypts the data, and prints it.

The program also uses encryption (via the cryptography module) to ensure the QR code content is secure.

Library Imports

import cv2 OpenCV for image and video processing import easyocr OCR engine to read handwritten text import qrcode Library to generate QR codes from PIL import Image For handling image files from pyzbar.pyzbar import decode For decoding QR codes from images or video frames import numpy as np from cryptography.fernet import Fernet For encrypting and decrypting text

8.Encryption Functions

encrypt$_t ext(text)$

Reads a pre-generated encryption key from secret.key

Uses Fernet symmetric encryption to encrypt the input text

Returns the encrypted string (required to be in string form for QR codes)

decrypt$_t ext(encrypted_t ext)$

Reads the same key from secret.key

Decrypts the QR data

Returns the original text

9.Image to Text to Encrypted QR (imgtotexttoqr)

Key generation

key = Fernet.generate$_key()withopen("secret.key","wb")askey_file : key_file.write(key)$

Generates and saves a symmetric key (used for both encryption and decryption).

Only run once, or all decryption will fail later if the key is overwritten.

Handwritten text extraction

image = cv2.imread(image$_path)reader = easyocr.Reader(['en'])results = reader.readtext(image)e$

$"".join([res[1]forresinresults])$

Reads text.jpg, extracts all text using EasyOCR, and joins it into a single line.

Text encryption

encrypted$_text = encrypt_text(extracted_text)$

Encrypts the extracted text using Fernet.

QR Code generation

qr = qrcode.QRCode(...) qr.add$_data(encrypted_text)...qr_img.save(qr_img_path)$

Encrypts the text and saves it into a scannable QR code image (output$_qr.png)$.

Display

qr$_img.show()$

Opens the QR image with the system image viewer.

QR Detection with ESP32-CAM (qrdetectioncam)

Live video stream

url = 'http://192.168.114.154:81/stream' cap = cv2.VideoCapture(url)

Connects to the ESP32-CAM live stream over Wi-Fi.

Detect and decode QR code

decoded$_objects = decode(frame)$

Reads frames from the video stream and looks for QR codes.

For each detected QR code:

qr$_text = obj.data.decode('utf-8')decrypted_text = decrypt_text(qr_text)$

The scanned QR code is decoded.

The encrypted string is decrypted using the secret.key.

User authentication

x = input(”Enter code: ”) if (x == qr$_text) : print(”DecryptedQRCodeText : ”, decrypted_text)$

Asks the user to enter the scanned encrypted string.

If the input matches the encrypted string from the QR, it displays the original (decrypted) content.

¿ This is like a simple authentication step, making sure someone confirms the scanned code before revealing the actual data.

Menu (User Interface)

while True:

Menu lets the user choose between:

(1) Real-time QR detection via ESP32-CAM

(2) Converting handwritten text to encrypted QR

(0) Exit

Full Workflow Summary

Option 2: Create QR

1. Capture handwritten image.

2. Extract text using OCR.

3. Encrypt text using Fernet.

4. Generate and display a QR code.

Option 1: Scan QR

1. Connect to ESP32-CAM live feed.

2. Detect and decode QR code.

3. Ask user for input (for verification).

4. Decrypt QR and display original text.

Security Note

The secret.key file is crucial. If it is deleted or replaced, decryption will fail.

This ensures confidentiality, even if someone else scans your QR code.

3.16 IEEE STANDARDS

3.16.1 IEEE-Standard

- If you’re asking which IEEE standards are relevant to your project “Handwritten to QR: Converting Handwritten Text to Digital Code via ESP32-CAM,” here are the main IEEE standards that align with the technologies used:

  830 – Software Requirements Specification (SRS)

Ensures the software components (OCR, QR generation) follow a structured requirement format.

You can use IEEE 830 to write clear functional and non-functional requirements for:

Image processing modules

Text extraction accuracy

QR code generation logic

- IEEE 12207 – Software Life Cycle Processes

  Helps structure your development process: requirements, design, implementation, testing, and maintenance.

  Ideal for planning the stages of your ESP32-CAM + OCR + QR generation system.

- IEEE 610.12 – Standard Glossary of Software Engineering Terminology

  Useful for defining terms like "OCR," "real-time processing," "embedded system," and more in documentation.

- IEEE 802.11 – Wireless LAN (Wi-Fi)

  Since ESP32-CAM transmits images over Wi-Fi, this standard ensures compliance and understanding of wireless data transmission.

  5. IEEE 2402-2021 – Recommended Practice for Digital Representation of Handwritten Data

  Governs how handwritten text is digitized, stored, and processed. Useful for OCR interpretation and data handling.

- IEEE 1857 – Advanced Audio and Video Coding

  If your system incorporates a video stream of handwritten input (rather than just images), this standard applies to video compression and processing.

  Optional (If Extended Further):

- IEEE 21451 – For sensor interfacing, if you use additional sensors in the future.

  IEEE 829 – For standardizing test documentation (useful during validation of OCR and QR code performance).

3.17 Applications

*Healthcare and Medical Records:

- *Digitizing Handwritten Prescriptions and Notes:* - Doctors and healthcare providers often write prescriptions, patient notes, and medical records by hand. This system can help digitize these handwritten records, making them easily accessible in a digital format and reducing manual data entry. - *Easy Sharing and Retrieval*: QR codes can be attached to patient records, making it easy for authorized personnel to access detailed patient information from a mobile device.

- *Medical Forms and Consent Documents:* - Patient consent forms, medical history forms, and other handwritten documents can be quickly converted into digital form for easier storage, management, and retrieval.

Autonomous Vehicles Object Detection and Tracking: FPGAs can accelerate real-time image processing for object detection, tracking, and classification in autonomous vehicles. Tasks like lane detection, pedestrian identification, and obstacle avoidance can benefit from high-speed image processing.

LIDAR and Camera Fusion: The fusion of images from cameras and LIDAR sensors requires high-speed image processing for efficient analysis in real time.

Business and Office Use: - *Digitizing Business Forms and Signatures:* - Handwritten contracts, agreements, or signatures can be captured and encoded into QR codes. This reduces paperwork and makes it easy to store and retrieve signed documents. - *Easy Access to Meeting Notes*: Handwritten meeting notes or minutes can be quickly converted into digital format and shared via QR codes with colleagues or clients.

- *Document Management:* - In businesses where handwritten forms or notes are prevalent, this system can simplify document management by converting paper-based data into digital files that can be stored, organized, and shared more efficiently.

Quality Control: FPGAs can process images from various sensors to ensure products meet quality standards by detecting faults in real time.

Legal and Compliance Applications:

- *Legal Document Management:* - Handwritten legal notes, contracts, or deeds can be quickly digitized, making it easier to archive and search through legal documents. - *Digital Signatures*: Legal documents that require handwritten signatures can be digitized and converted into QR codes for secure access and sharing.

- *Audit Trails and Compliance Tracking:* - This system can be used to quickly digitize handwritten compliance logs or records, creating digital audit trails for easy tracking and verification.

Financial and Banking Industry: - *Manual Banking Forms:* - Banks often deal with handwritten forms, such as account applications or withdrawal slips. These can be converted into digital format, making them easier to store, retrieve, and process. - *Secure Transactions*: Signed documents, such as loan agreements or checks, can be captured as QR codes for secure processing.

- *Transaction Records:* - Handwritten transaction logs or receipts can be digitized for archiving, reducing physical storage and making the process of retrieving transaction details faster.

Creative and Design Fields: Sketches and Artwork Digitization: - Artists and designers can use this technology to convert sketches, designs, or artwork into digital form by capturing the creation process. The QR code could store a digital version of the artwork or provide a link to a cloud-based gallery. - *Inspiration and Brainstorming:* Designers could take quick handwritten notes or sketches, convert them to QR codes, and share them for collaborative feedback or inspiration.

- *Handwritten Drafts:* - Authors and scriptwriters can digitize their handwritten drafts, converting them into a format that is easier to edit and store digitally.

Personal Use and Organization: - *Personal Notes and Journals:* - Individuals can use the system to digitize their personal handwritten notes, journals, or to-do lists and store them securely in digital format for easy access and organization.

- *Organizing Recipes and Handwritten Lists:* - Recipes, shopping lists, or other personal handwritten content can be converted into digital form and shared with family or friends through QR codes.

- *Handwritten Reminders and Ideas:* - People can convert handwritten reminders or ideas into digital format and store them in their smartphones for later use or scanning.

Field and Remote Work Applications: - *Field Data Collection:* - In industries such as agriculture, construction, or field research, workers can take handwritten notes on-site and quickly convert these notes into digital form. QR codes can be used to transmit this information back to a central database or cloud system for analysis and reporting.

- *Mobile Workflows in Remote Locations:* - Field workers can capture handwritten reports or forms, generate QR codes for data transfer, and easily share these codes with a central system, even without a stable internet connection (QR codes can be scanned later when connectivity is available).

Conference and Event Management: - *Event Documentation and Notes:* - During conferences, seminars, or workshops, handwritten notes taken by participants can be converted into QR codes. Attendees can scan the codes to get access to shared resources, presentations, or notes from speakers.

- *Event Registration Forms:* - Registration forms and attendee lists can be digitized and stored as QR codes, making event management smoother and reducing the need for physical paperwork.

Security and Identification: - *Signature Authentication:* - This system can be used to securely digitize handwritten signatures on forms, contracts, and documents, allowing for secure and verified authentication processes in various transactions.

- *Digital Identity Documents:* - Handwritten identity documents or personal credentials (e.g., handwritten IDs or registration cards) can be encoded into QR codes for secure sharing and easy retrieval.

3.18 ADVANTAGES

1. Cost-Effective Solution: *Affordable Hardware:* The use of the *ESP32 CAM*, a low-cost development board with both a camera and Wi-Fi capabilities, makes this project budget-friendly. This reduces the need for expensive scanning equipment or OCR software. - *Accessible Technology:* The ESP32 CAM and QR code technology are both inexpensive and widely accessible, making this solution scalable and easy to implement for individuals or organizations.

2. 2. Seamless Digitization of Handwritten Content Real-Time Conversion:This project allows for the real-time capture and conversion of handwritten text into a digital format. The text can be instantly stored, shared, and accessed without requiring manual transcription, which saves time and effort. - *Paper-to-Digital Workflow:* By bridging the gap between paper and digital formats, this project simplifies workflows that involve handwritten notes, forms, or documents.

3. Easy Data Sharing and Access: *QR Code Integration:* QR codes provide a simple and efficient way to store, share, and retrieve data. Once the handwritten text is converted into a QR code, it can be easily shared and accessed via a smartphone or

QR code scanner. - *No Special Hardware Needed for Retrieval:* Most smartphones today have built-in QR code scanning capabilities, making it easy for anyone to access the digital content without needing specialized equipment.

4. Compact and Portable: - *Small Form Factor of ESP32 CAM:* The ESP32 CAM module is compact and portable, making it easy to set up in various environments, whether in a classroom, office, or remote location. It can easily be integrated into existing workflows. - *Portable Solution for Field Use:* The system can be deployed in field scenarios, allowing handwritten information to be captured and digitized in real time in places where traditional scanning solutions might not be practical.

5. Enhanced Productivity and Efficiency: - *Automated Text Conversion:* The project automates the process of converting handwritten content into a digital format, significantly reducing the need for manual data entry. This can improve productivity, especially in environments like classrooms, offices, and medical settings. - *Instant Digital Retrieval:* Scanning the QR code allows users to instantly retrieve the original text, streamlining the process of accessing digitized handwritten content.

6. Increased Accessibility and Organization: - *Digitized Content is Easily Searchable:* Once the handwritten text is converted into a digital format, it can be stored in a database or cloud storage, where it can be organized, searched, and accessed more easily than physical handwritten documents. - *Improved Document Management:* By converting handwritten content into digital format, documents can be stored and managed more effectively, allowing for easier organization, retrieval, and sharing.

7. Reduced Need for Paper: - *Eco-Friendly:* This project promotes the use of digital storage over physical paper, which can reduce paper waste and contribute to more sustainable practices. - *Less Physical Storage Required:* Digitizing handwritten notes or documents reduces the need for physical storage, such as filing cabinets or paper-based systems, making it easier to organize and access information in a digital form.

8. Scalability and Customization: Scalable for Various Applications: This system can be scaled for different types of handwritten content, including notes, signatures, forms, and documents, making it versatile across various industries like education, healthcare, legal, and business. - *Customization Potential:* The system can be

customized for specific use cases, such as adding features for recognizing different handwriting styles, improving accuracy, or incorporating additional functionalities like voice-to-text or integration with other systems.

9. Enhanced Security: Secure Digital Format: Once the handwritten content is digitized and encoded in a QR code, it can be encrypted or password-protected for added security, ensuring that sensitive information is safely stored and accessed. - *Data Integrity:* By encoding text into QR codes, you ensure that the data remains unaltered. This is particularly useful for applications involving legal documents, medical prescriptions, or any content where data integrity is critical.

10. User-Friendly Interface: Simple User Experience: The system is designed to be simple for users. Handwriting is captured in real-time, converted into a digital format, and encoded into a QR code—all of which can be easily scanned and decoded using smartphones. This low-tech interface makes it accessible even to users with minimal technical expertise.

3.19 DISADVANTAGES

1. Image Processing Power on ESP32 CAM: Hardware Constraints: The **ESP32 CAM* is a low-power device with limited processing capabilities. It may struggle to handle complex image processing tasks, such as accurate text recognition (OCR), especially when dealing with poor-quality handwriting or varying handwriting styles. - *Lack of Advanced Features*: The ESP32 CAM might not be able to handle advanced video processing or high-resolution image capture compared to more powerful devices like smartphones or dedicated image processors.

2. Handwriting Recognition Challenges: - *Varied Handwriting Styles*: The accuracy of converting handwritten text into digital format depends on the legibility of the handwriting. Handwriting that is messy, inconsistent, or in a non-standard style may be difficult for the system to process correctly. - *Limited Text Recognition*: Since the project does not incorporate advanced OCR (Optical Character Recognition) directly in the ESP32 CAM, the handwritten text needs to be very clear and easy to recognize. Inaccuracies or difficulty in detecting certain letters or words can occur.

3. Lighting and Environmental Conditions: - *Dependence on Good Lighting*: The

ESP32 CAM's camera requires good lighting conditions to clearly capture the handwriting. Poor or inconsistent lighting could lead to blurry or unclear images, reducing the quality of the captured video and the accuracy of the final QR code. - *Environmental Factors*: Shadows, glare, and reflections from the writing surface could interfere with the clarity of the captured video, making it harder to accurately extract and convert the handwritten text.

4. 4. Limited Video Resolution: - *Low Resolution of Camera: The built-in **OV2640 camera* in the ESP32 CAM module has a resolution of up to 2MP, which may not be sufficient for capturing detailed handwritten text with high precision, especially when the text is small or written in fine penmanship. - *Video Frame Quality*: Since the ESP32 CAM records video, motion blur or pixelation could occur when capturing the writing process, leading to less accurate results when converting the video into text.

5. Limited QR Code Capacity: - *Limited Data Storage in QR Codes*: While QR codes can store a reasonable amount of data, they have a storage limit (around 3KB for a standard QR code). If the handwritten text is too long or contains special formatting, it may exceed the capacity of a single QR code, requiring the text to be split across multiple codes. - *Potential for Loss of Quality*: In some cases, if the QR code is too dense or complex, it may become difficult for scanners to accurately decode, especially if the QR code is distorted, printed poorly, or scanned at a low resolution.

6. No Built-In Text Recognition: Lack of OCR Integration*: This project doesn't incorporate a full OCR system, which means the handwritten text cannot be automatically transcribed into digital text on the ESP32 CAM itself. This requires additional processing steps (such as uploading the video for OCR processing on a separate system), increasing the complexity and delay of the conversion process. - *Manual Steps*: If you need advanced OCR or more complex text processing, it would require integrating additional software or cloud-based services, which adds complexity and may reduce the real-time benefits of the system.

7. Wi-Fi Dependency: - *Limited Offline Functionality*: If the ESP32 CAM is required to transmit data or communicate with other systems via Wi-Fi (for instance, uploading the video or using cloud-based OCR), the device's dependency on a stable

Wi-Fi connection could be a limitation in environments where network connectivity is weak or unavailable. - *Possible Delays in Processing*: The need to upload videos or data to a server could introduce delays in the system's real-time performance. This is especially true if the system is heavily reliant on internet connectivity for additional processing or data storage.

8. Manual Calibration and Setup: *Need for Proper Calibration*: To achieve optimal performance, the camera must be properly aligned with the writing surface. This may require careful calibration and setup to ensure the handwriting is captured correctly, which can be time-consuming and prone to errors if not done correctly. - *User Dependency*: Users may need to manually adjust the position of the camera or ensure proper lighting conditions to get clear video captures, adding a level of complexity to the setup.

9. Security and Privacy Concerns: -*Data Storage and Security*: If the handwritten content includes sensitive information, storing it in QR code format (especially if uploaded to a cloud or server) raises concerns about data privacy and security. Ensuring that the QR codes and stored data are encrypted and securely handled is critical but could add complexity to the system. - *Vulnerability to Unauthorized Access*: If the QR code is not protected or encrypted, unauthorized users could potentially access the digital version of the handwritten content by simply scanning the QR code.

10. Limited Text Editing Capabilities: *No Editing Once Converted*: Once the handwritten text is converted into a QR code and decoded, the original handwritten text cannot be easily edited. This could be a limitation in scenarios where the user needs to make changes to the text after it has been captured and digitized. - *No Rich Text Formatting*: The conversion into QR codes stores text in a very basic form. If the handwritten text involves rich formatting (e.g., tables, graphics, or special symbols), this information will not be preserved in the QR code.

—

4 CONCLUSION

While the project offers a novel and efficient way to convert handwritten text into a digital format via video and QR codes, it also comes with a set of limitations, such as the lack of advanced image processing capabilities, challenges with handwriting recognition, dependency on good lighting, and the limited storage capacity of QR codes. The system also requires manual calibration and setup to achieve optimal results, and it may face issues with offline functionality or internet dependency. These factors should be considered when assessing the practicality and scalability of the project for different applications.

The project successfully demonstrates a complete pipeline that transforms handwritten text into a machine-readable QR code using computer vision and embedded systems. By leveraging Optical Character Recognition (OCR) techniques, handwritten inputs are accurately converted into digital text. This digital text is then encoded into a QR code, providing a compact and scannable format for easy data sharing and retrieval.

The integration of the ESP32-CAM module enables real-time image capture and video streaming, allowing for flexible deployment in smart systems, such as digital classrooms, document archiving, and IoT-based applications. The use of machine learning for text recognition ensures adaptability to various handwriting styles, while the QR generation and scanning components enable efficient communication between physical and digital environments.

Overall, this project bridges traditional input methods with modern data encoding and transmission techniques, offering a practical, low-cost solution for digitization and smart communication.

5 REFERENCES

1. For your project ”Handwritten to QR: Converting Handwritten Text to Digital Code via Video Using ESP32-CAM”, the references should include academic papers, libraries, and technologies that support the various stages of the process. Here’s a list of recommended references you can cite:

2. OCR Technology (Handwriting to Digital Text)

   Smith, R. (2007). An Overview of the Tesseract OCR Engine. In Proceedings of the Ninth International Conference on Document Analysis and Recognition (Vol. 2, pp. 629-633). IEEE.

   Tesseract OCR GitHub Repository: https://github.com/tesseract-ocr/tesseract

3. QR Code Generation

   ISO/IEC 18004:2015 – Information technology – Automatic identification and data capture techniques – QR Code bar code symbology specification.

   Kazuhiko, Kato (1994) – QR Code: A New Code for Optical Transfer of Information.

   qrcode Library Documentation (Python): https://pypi.org/project/qrcode/

4. ESP32-CAM and Video Capture

   Espressif Systems. (2020). ESP32-CAM Technical Reference Manual.

   Random Nerd Tutorials – ESP32-CAM Projects Tutorials: https://randomnerdtutorials.com

5. Computer Vision and Image Processing

   Bradski, G. (2000). The OpenCV Library. Dr. Dobb’s Journal of Software Tools.

   OpenCV Documentation: https://docs.opencv.org

6. Related Academic Research

   Jain, A. K., Doermann, D. (2015). Handwriting Recognition: Challenges and Future Directions.

   Gallo, O., Manduchi, R. (2011). Reading 2D barcodes with mobile phones using image processing techniques.

7. For machine learning-based references relevant to your project (handwritten text recognition and QR code generation using ESP32-CAM), here are key scholarly and technical references you can use:

8. Handwritten Text Recognition (HTR) with Machine Learning

   Graves, A., Schmidhuber, J. (2009). Offline Handwriting Recognition with Multidimensional Recurrent Neural Networks. In Advances in Neural Information Processing Systems (NeurIPS). [Pioneered the use of RNNs and LSTMs for HTR.]

   Bluche, T. (2016). Joint Line Segmentation and Transcription for End-to-End Handwritten Paragraph Recognition. In Advances in Neural Information Processing Systems (NeurIPS). [Uses deep learning models (CNN+RNN) for end-to-end HTR.]

   Wigington, C., et al. (2018). Start, Follow, Read: End-to-End Full-Page Handwriting Recognition. In European Conference on Computer Vision (ECCV). [Introduces end-to-end full-page recognition using deep neural nets.]

9. Machine Learning Libraries for HTR

   TensorFlow Documentation – Image Recognition OCR Models https://www.tensorflow.org/tutori

   PyTorch Deep Learning for HTR with CRNNs (Convolutional Recurrent Neural Networks) https://github.com/meijieru/crnn.pytorch

10. QR Code and Machine Learning

    While QR code generation itself is deterministic and not ML-based, ML can be used in decoding and QR detection, especially in poor conditions:

    Zhou, Z., et al. (2020). Robust QR Code Detection Using Deep Learning for Mobile Robots. In IEEE International Conference on Robotics and Automation (ICRA). [Uses CNNs to detect QR codes under poor lighting or angles.]

    Deep Learning for Barcode and QR Code Reading in Images (GitHub Project) https://github.com/WeChatCV/opencv$_3rdparty/tree/master/barcode$

11. ESP32-CAM + ML

Espressif's AI Features on ESP32 (via Edge Impulse or TensorFlow Lite):

Running TinyML on ESP32-CAM for image classification and object detection

TensorFlow Lite for Microcontrollers: https://www.tensorflow.org/lite/microcontrollers

12. For the communication aspect of your project (particularly involving data transfer, QR code scanning, and ESP32-CAM integration), here are communication-based references relevant to networking, wireless transmission, and embedded systems:

Communication with ESP32-CAM

Espressif Systems. (2020). ESP32 Series Datasheet Technical Reference Manual. https://www.espressif
[Provides details on Wi-Fi, Bluetooth, UART, SPI, and I2C communication in ESP32.]

Random Nerd Tutorials – ESP32-CAM Web Server Projects Building Camera Streaming over Wi-Fi https://randomnerdtutorials.com/esp32-cam-video-streaming-web-server-camera-home-surveillance/ [Shows how to set up communication for real-time video feed.]

13. QR Code Transmission and Data Sharing

    Li, X., et al. (2013). A Framework for Secure QR Code Based Communication. In IEEE Transactions on Consumer Electronics. [Discusses secure communication via QR codes, including encoding strategies.]

    Liu, W., et al. (2018). Data Transmission via Visual Codes in Smart Devices. In IEEE Transactions on Mobile Computing. [Explores QR and visual code-based communication between devices.]

14. IoT and Embedded Communication Protocols

    Al-Fuqaha, A., et al. (2015). Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications. In IEEE Communications Surveys Tutorials. [Extensive review of IoT protocols like MQTT, HTTP, CoAP used in ESP32 projects.]

    Sauter, M. (2017). Communication Protocols for IoT and M2M. In From GSM to LTE-Advanced Pro and 5G. Wiley. [Explains lightweight protocols relevant for microcontroller communication.]

15. QR-Based Communication in Education and IoT

    Tan, Y., Kim, Y. (2015). Use of QR Codes in Education and IoT-Based Applications. In Journal of Educational Technology Development and Exchange. [Describes QR as a communication bridge in smart systems.]