Software Development Practice 1

Instructor: RSP <rawat.s@eng.kmutnb.ac.th>

Exploring QR Code Generation and Detection

Objectives

- Learn how to write code in different programming languages to generate, detect or scan QR codes.

Expected Learning Outcomes

After completing all tasks, students will be able to:

- Write Python, Node.js and C++ code with different libraries to generate, detect or scan QR codes.
- Use a Web camera to detect or scan QR codes.

Task 1: Generating QR code image files with Python

The provided **Python** script (generate_qrcode_jpg.py) demonstrates how to generate a JPEG image file with the following properties:

- It contains a QR code representing a random short UUID code of fixed length (e.g., 8 characters).
- Below each QR code, the associated UUID text is displayed.

1. Install the necessary **Python packages** on **WSL 2 Ubuntu** and **Raspberry Pi Desktop** (remote):

```
$ pip install Pillow qrcode[pil]
```

2. Run the Python script and view the generated QR code file. To view the QR code file on a Linux desktop, you may use a program like '**feh**':

```
$ sudo apt install feh
$ feh qrcode.jpg
```

An example of a generated QR code image

3. Explore different parameters for generating QR code image files such as:

- **qrcode version**: a number from 1 to 40. The higher the version, the bigger the QR code and the more data it can store.
- **box_size**: the size (in pixels) of each individual box (black or white square) in the QR code grid.
- **border**: the blank margin around the QR code, measured in boxes (not pixels). Using too small a border makes scanning harder.
- **error_correction**: the level of error correction (Low, Medium, Quartile, High).

4. Write a **Python** script that generates a JPEG image file with the following properties.

- It contains four QR codes that are arranged vertically and center-aligned horizontally.
- Each QR code represents a random short UUID code (varying between 8 to 16 characters).
- Below each QR code, the associated UUID should be displayed.
- Each QR code should be generated using different parameters such as box size, border width and level error correction.
- The width of the QR codes should be chosen properly for clear viewing on a Smartphone display.
- A timestamp text is also added near the bottom of the JPG image.

5. Run the **Python** script to generate QR Code JPEG files. Add the generated JPEG files to your report.

6. Test the generated QR codes with different QR code reader devices.

5. Write a **Python** script that operates as a **Flask Web server** which generates and provides a JPEG file for a single QR code (**JPEG content, and not an HTML page**).

- The JPEG content is available at the '**/qrcode**' endpoint.
- Every reload will generate a new JPEG image.
- Use your Smartphone to load and view the JPEG image from your web server running on either **Windows**, or **Raspberry Pi** computer in the same local network.
- Specify which QR code parameters are used and best for viewing on your smartphone screen.

6. Capture your smartphone screen displaying the QR code image and include it in your report.

An example of a generated QR code image displayed on smartphone

File: generate_qrcode_jpg.py

```
import uuid  # uuid is a built-in Python module.
import qrcode
import PIL
from PIL import Image, ImageDraw, ImageFont

# Show the version of Pillow
#print("Pillow version:", PIL.__version__)

# 1. Generate a short UUID (only 8 chars), using UUID version 4
num_chars = 8
short_uuid = str(uuid.uuid4())[:num_chars]
print("Generated UUID:", short_uuid)

# 2. Create QR code
# qrcode version = 2, box size = 12 pixels
# border = 4 boxes, error correction = high
qr = qrcode.QRCode(
    version=2,
    error_correction=qrcode.constants.ERROR_CORRECT_H,
    box_size=12,
    border=4,
)

# Make the QR code image
qr.add_data(short_uuid)
qr.make(fit=True)
qr_img = qr.make_image(fill_color="black", back_color="white").convert("RGB")

count = qr.modules_count
print( f"QR size: {count} x {count}" )
print( f"QR dimension: {qr_img.size} pixels" )

# 3. Prepare final image with text below
qr_width, qr_height = qr_img.size

# Load a TrueType font, fallback to default if not found
font_list = ["DejaVuSansMono-Bold.ttf","LiberationMono-Regular-Bold.ttf"]
font_size = 32
font = None
for f in font_list:
    try:
        font = ImageFont.truetype(f, font_size)
        break
    except IOError:
        pass
if not font:
    print("Use default font")
    font = ImageFont.load_default()

# Calculate text size using textbbox
dummy_img  = Image.new("RGB", (1, 1))
draw_dummy = ImageDraw.Draw(dummy_img)

bbox = draw_dummy.textbbox((0, 0), short_uuid, font=font)
text_width  = bbox[2] - bbox[0]
text_height = bbox[3] - bbox[1]

# Create new image for QR + text + padding
padding = 40
total_height = qr_height + text_height + padding
img = Image.new("RGB", (qr_width, total_height), color="white")

# Paste QR code  image into the final image
```

```
img.paste(qr_img, (0, 0))

# Draw a text centered below the QR on the image
draw = ImageDraw.Draw(img)
text_x = (qr_width - text_width) // 2
text_y = qr_height + (padding // 4)
draw.text((text_x, text_y), short_uuid, fill="black", font=font)

# 4. Save as JPG
img.save("qrcode.jpg", "JPEG")
print("Done…")
```

Task 2: Generating QR code image files with Node.js

1. **Node.js** code (gen_qy_Code.js) is provided as an example. It uses libraries such as '**qrcode**', '**uuid**', '**canvas**' and '**sharp**' to generate a JPEG image file with the following properties.

    • It contains a single QR code which represents a random short UUID code (varying between 8 to 16 characters).
    • Below the QR code, the associated UUID should be displayed.
    • A timestamp text is also added near the bottom of the JPEG image.

2. Build the project from the source code and run the executable to generate a QR code JPEG file.

Note:

- Create a new folder for your Node.js project.
- Initialize the Node.js project:

  $ npm init -y
- Install necessary modules using the following command:

  $ npm install qrcode uuid sharp canvas

3. Write **Node.js** code that uses the **Express framework** to implement a **Web server**. It generates and provides a JPEG file for a single QR code (**JPEG content, and not an HTML page**).

- The JPEG content is available at the '**/qrcode**' endpoint.
- Every reload will generate a new JPEG image.
- Use your Smartphone to load and view the JPEG image from your web server running on either **Windows**, or **Raspberry Pi** computer in the same local network.

4. Capture your smartphone screen displaying the QR code image and include it in your report.

5. Test the generated QR codes with different QR code reader devices.

File: gen_qy_Code.js

```
const { v4: uuidv4 } = require('uuid');
const QRCode = require('qrcode');
const sharp = require('sharp');
const { createCanvas, loadImage } = require('canvas');
const fs = require('fs');

// 1. Generate a random short UUID (8-16 characters)
function generateShortUUID(m,n) {
  const min = Math.min(m, n);
  const max = Math.max(m, n);
  const uuid = uuidv4().replace(/-/g, '');
  const len = Math.floor(Math.random() * (max - min + 1)) + min;
  return uuid.substring(0, len);
}

// 2. Create QR code image buffer
async function generateQRCode(text) {
  return await QRCode.toBuffer(text, {
    type: 'png', width: 400, margin: 2 });
}

// 3. Combine QR code and text into final image
async function createFinalImage() {
  const shortUUID = generateShortUUID(8, 16);
  const qrBuffer = await generateQRCode(shortUUID);
  const qrImage  = await sharp(qrBuffer).toBuffer();
  // Prepare canvas
  const margin = 40;
  const canvasWidth = 500;
  const canvasHeight = canvasWidth + 2*margin;
  const canvas = createCanvas(canvasWidth, canvasHeight);
  const ctx = canvas.getContext('2d');
  // Fill background white
  ctx.fillStyle = '#FFFFFF';
  ctx.fillRect(0, 0, canvasWidth, canvasHeight);
  // Load QR code PNG into canvas
  const img = await loadImage(qrImage);
  const qrX = (canvasWidth - img.width) / 2;
  ctx.drawImage(img, qrX, margin);
  // Draw UUID text
  ctx.fillStyle = '#000000';
  ctx.font = 'bold 30px Sans';
  ctx.textAlign = 'center';
  ctx.fillText(shortUUID, canvasWidth / 2, img.height + 2*margin);
  // Draw timestamp
  const timestamp = new Date().toISOString();
  ctx.font = '20px Sans';
  ctx.fillText(`Generated: ${timestamp}`, canvasWidth / 2, canvasHeight - margin);
  // Convert to buffer and output as JPG
  const finalBuffer = canvas.toBuffer('image/jpeg', { quality: 90 });
  // Save to file
  fs.writeFileSync(`./qrcode.jpg`, finalBuffer);
  console.log(`QR image created successully...`);
}

createFinalImage().catch(console.error);
```

Task 3: Generating QR code image files with C++.

1. A **C++** source file (main.cpp) and a **CMakeLists.txt** file (to be used with **cmake**) are provided as an example. This project uses libraries such as '**libqrencode-dev**', '**libopencv-dev**', and '**uuid-dev**' to generate a JPEG image file with the following properties:

- It contains a single QR code representing a random short UUID (between 8 and 16 characters).
- Below the QR code, the associated UUID is displayed.
- A timestamp is added near the bottom of the JPEG image.

2. Revise the C++ code to generate a JPEG file containing an (m × n) grid of QR codes with random UUID strings arranged in an (m × n) tile (e.g., m = 3, n = 4).

- Only one timestamp string should be added near the bottom of the JPEG image.

3. Build the project from the source code and run the executable to generate a QR code JPEG file. Add the generated JPEG files to your report.

4. Test the generated QR codes with different QR code reader devices.

File: main.cpp

```cpp
#include <iostream>             // for std::cout, std::cerr
#include <string>               // for std::string
#include <ctime>                // for std::time, std::strftime
#include <random>               // for std::rand
#include <algorithm>            // for std::remove (used to strip '-')
#include <uuid/uuid.h>          // for libuuid (UUID generation)
#include <qrencode.h>           // for QR code generation (libqrencode)
#include <opencv2/opencv.hpp>   // for image processing and rendering


// Generate a short UUID (8–16 characters), hyphens removed
std::string generateUUID(int min = 8, int max = 16) {
    uuid_t uuid;
    char uuidStr[37]; // Full UUID = 36 chars + null terminator
    uuid_generate_random(uuid);             // Generate random UUID v4
    uuid_unparse_lower(uuid, uuidStr);      // Convert to string format

    // Remove '-' in the UUID string
    std::string cleaned(uuidStr);
    cleaned.erase(std::remove(cleaned.begin(), cleaned.end(), '-'), cleaned.end());

    int len = min + (std::rand() % (max - min + 1));  // Choose random length
    return cleaned.substr(0, len);                    // Return truncated UUID
}


// Return current timestamp string (formatted as YYYY-MM-DD HH:MM:SS)
std::string getTimestamp() {
    std::time_t now = std::time(nullptr);
    char buf[64];
    std::strftime(buf, sizeof(buf), "%Y-%m-%d %H:%M:%S", std::localtime(&now));
    return buf;
}


// Render a QR code for the given data using libqrencode and OpenCV
cv::Mat renderQRCode(const std::string& data, int scale = 10) {
    QRcode* qr = QRcode_encodeString(data.c_str(), 0, QR_ECLEVEL_Q, QR_MODE_8, 1);
    if (!qr) {
        std::cerr << "Failed to encode QR code\n";
        exit(1);
    }

    auto WHITE = cv::Scalar(255, 255, 255); // Background color
    int size = qr->width * scale;           // Output image size
    cv::Mat qrImage(size, size, CV_8UC3, WHITE); // Initialize white canvas

    // Draw each module (black square) in the QR matrix
    for (int y = 0; y < qr->width; y++) {
        for (int x = 0; x < qr->width; x++) {
            if (qr->data[y * qr->width + x] & 0x01) {
                cv::rectangle(qrImage,
                      cv::Point(x * scale, y * scale),
                      cv::Point((x + 1) * scale - 1, (y + 1) * scale - 1),
                      cv::Scalar(0, 0, 0), cv::FILLED);  // Black pixel
            }
        }
    }

    QRcode_free(qr); // Free memory allocated by QRcode_encodeString
    return qrImage;
}
```

```
// Draws text horizontally centered at (centerX, baselineY) on the image
void drawCenteredText(
    cv::Mat& image,
    const std::string& text,
    int centerX, int baselineY,
    int fontFace = cv::FONT_HERSHEY_SIMPLEX,
    double fontScale = 0.8,
    cv::Scalar color = cv::Scalar(0, 0, 0),
    int thickness = 2 )
{
    int baseline = 0;
    cv::Size textSize = cv::getTextSize(
                            text, fontFace, fontScale, thickness, &baseline);
    int textX = centerX - textSize.width / 2;  // Shift to center
    cv::putText(image, text, cv::Point(textX, baselineY),
                fontFace, fontScale, color, thickness);
}

int main() {
    std::srand(static_cast<unsigned int>(std::time(nullptr))); // Seed RNG
    // Generate data and timestamp
    std::string uuid = generateUUID();
    std::string timestamp = getTimestamp();

    // Color constants
    auto WHITE = cv::Scalar(255, 255, 255);
    auto BLACK = cv::Scalar(0, 0, 0);
    auto GRAY  = cv::Scalar(80, 80, 80);

    // Generate QR image from UUID
    int qrScale = 14;  // Scaling factor for QR resolution
    cv::Mat qr = renderQRCode(uuid, qrScale);

    // Define canvas dimensions with padding
    int padding = 40;
    int canvasWidth  = qr.cols + 2 * padding;
    int canvasHeight = qr.rows + 4 * padding;

    // Create output canvas and place QR in center with padding
    cv::Mat canvas(canvasHeight, canvasWidth, CV_8UC3, WHITE);
    cv::Rect roi(padding, padding, qr.cols, qr.rows);
    qr.copyTo(canvas(roi));

    std::cout << "QR rows: " << qr.rows << ", cols: " << qr.cols << std::endl;

    // Coordinates for text (below the QR code)
    int centerX = canvas.cols / 2;
    int textY = padding + qr.rows + 3 * padding / 2;
    // Draw centered UUID and timestamp under QR
    drawCenteredText(canvas, uuid, centerX, textY,
                     cv::FONT_HERSHEY_SIMPLEX, 1.0, BLACK, 2);
    drawCenteredText(canvas, timestamp, centerX, textY + padding,
                     cv::FONT_HERSHEY_SIMPLEX, 0.7, GRAY, 1);
    // Save output image as JPEG
    std::string filename = "qrcode.jpg";
    if (cv::imwrite(filename, canvas)) {
        std::cout << "Saved image: " << filename << std::endl;
    } else {
        std::cerr << "Failed to save image." << std::endl;
    }
    return 0;
}
```

File: CMakeLists.txt

```
cmake_minimum_required(VERSION 3.20)
project(QRCODE_JPG_GEN)

set(CMAKE_CXX_STANDARD 17)
set(CMAKE_CXX_STANDARD_REQUIRED ON)

# Find OpenCV
find_package(OpenCV REQUIRED)

# Find libqrencode
find_path(QRENCODE_INCLUDE_DIR qrencode.h)
find_library(QRENCODE_LIBRARY qrencode)

# Find uuid-dev
find_library(UUID_LIB uuid REQUIRED)

if (NOT QRENCODE_INCLUDE_DIR OR NOT QRENCODE_LIBRARY)
    message(FATAL_ERROR "libqrencode not found. Please install libqrencode-dev.")
endif()

include_directories(${OpenCV_INCLUDE_DIRS} ${QRENCODE_INCLUDE_DIR})
link_libraries(${OpenCV_LIBS} ${QRENCODE_LIBRARY})

# Build executable
add_executable(gen_qrcode_jpg main.cpp)
target_link_libraries(gen_qrcode_jpg ${OpenCV_LIBS} ${QRENCODE_LIBRARY} ${UUID_LIB})
```

Task 4: Generating QR code image files with PyQt5.

1. The following **Python** code (pyqt5_qrcode_gen.py) is provided as an example. It implements a **PyQt5-based GUI application** that does the following:

   - It has an input text field (single-line), a "UUID" button, and a "Generate" button to generate a QR code.
   - If the "Generate" button is clicked and the input text is not empty, the QR code is generated and displayed.
   - Below the QR code, a timestamp shows when the QR code was generated.

2. Run the example code on wSL 2 Ubuntu and on Raspberry Pi Desktop (remote).

3. Rewrite the code to implement the following functions.

   - If the "UUID" button is clicked, a **random UUID string of length between 8 and 16 characters (no hyphen)** is generated and placed in the input text field.
   - If the "Generate" button is clicked and the input text is not empty, the QR code is generated and displayed with a random rotation angle.

4. Rewrite the code to add input field for a text line from a USB-HID QR code reader.

5. Test the generated QR codes with different QR code reader devices.

An example of a generated QR code image displayed PyQt5 app

An example of a generated QR code image rotated and displayed PyQt5 app

An example of PyQt5 App that shows a generated QR code with input texts from a USB-HID QR code reader

File: pyqt5_qrcode_gen.py

```
import sys
import qrcode
import io
import random
from datetime import datetime

from PyQt5.QtWidgets import (
    QApplication, QWidget, QLabel, QLineEdit, QPushButton,
    QVBoxLayout, QHBoxLayout, QSizePolicy
)
from PyQt5.QtGui import QPixmap, QImage, QFont
from PyQt5.QtCore import Qt

class QRCodeApp(QWidget):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("QR Code Generator")

        # Fonts
        font = QFont()
        font.setPointSize(14)
        small_font = QFont()
        small_font.setPointSize(12)
        # Input field
        self.input_field = QLineEdit()
        self.input_field.setFont(font)
        self.input_field.setPlaceholderText("Enter text to generate QR code")
        # UUID button
        self.uuid_button = QPushButton("UUID")
        self.uuid_button.setFont(font)
        self.uuid_button.clicked.connect(self.generate_uuid)
        # Generate button
        self.generate_button = QPushButton("Generate")
        self.generate_button.setFont(font)
        self.generate_button.clicked.connect(self.generate_qr)
        # Layout for input + buttons
        input_layout = QHBoxLayout()
        input_layout.addWidget(self.input_field)
        input_layout.addWidget(self.uuid_button)
        input_layout.addWidget(self.generate_button)
        # QR Code display
        self.qr_label = QLabel()
        self.qr_label.setAlignment(Qt.AlignCenter)
        self.qr_label.setSizePolicy(QSizePolicy.Expanding, QSizePolicy.Expanding)
        # Timestamp display
        self.timestamp_label = QLabel()
        self.timestamp_label.setFont(small_font)
        self.timestamp_label.setAlignment(Qt.AlignCenter)
        # Main layout
        main_layout = QVBoxLayout()
        main_layout.addLayout(input_layout)
        main_layout.addWidget(self.qr_label)
        main_layout.addWidget(self.timestamp_label)

        self.setLayout(main_layout)
        self.resize(500, 550)
        self.last_text = "" # Use to kepe the last text input

    def generate_uuid(self):
        print( "To be implemented!!!" )
```

```
    def generate_qr(self):
        text = self.input_field.text().strip()
        if not text:
            return

        self.last_text = text

        # Generate QR code as PIL image
        qr = qrcode.make(text)
        buf = io.BytesIO()
        qr.save(buf, format='PNG')
        buf.seek(0)

        # Convert PIL image to QPixmap
        qimage = QImage.fromData(buf.read())
        pixmap = QPixmap.fromImage(qimage)

        # Resize pixmap to fit the label while keeping aspect ratio
        self.qr_label.setPixmap(pixmap.scaled(
            self.qr_label.size(),
            Qt.KeepAspectRatio,
            Qt.SmoothTransformation
        ))

        self.qr_label.setPixmap(pixmap.scaled(
          self.qr_label.size(),
          Qt.KeepAspectRatio,
          Qt.SmoothTransformation
        ))

        # Set timestamp
        timestamp = datetime.now().strftime("Generated: %Y-%m-%d %H:%M:%S")
        self.timestamp_label.setText(timestamp)

    def resizeEvent(self, event):
        # Regenerate the QR code when the window is resized
        if self.last_text:
            self.generate_qr()
        super().resizeEvent(event)

if __name__ == "__main__":
    app = QApplication(sys.argv)
    window = QRCodeApp()
    window.show()
    sys.exit(app.exec_())
```

Task 5: QR code scanning with Python and Webcam

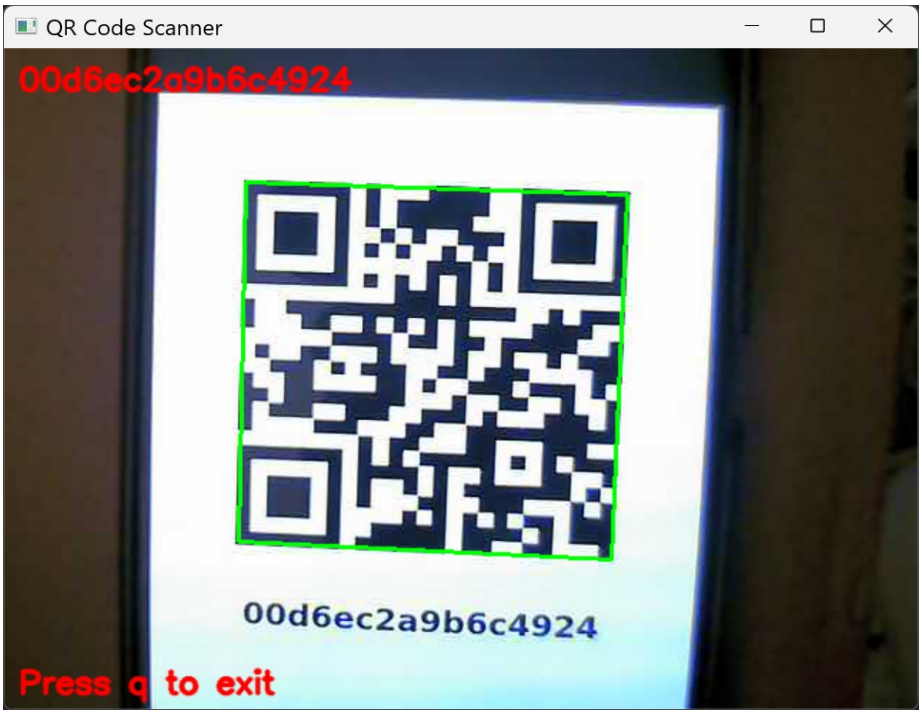
1. This **Python** code (py_opencv_qrcode_detect.py) uses **OpenCV** to access the webcam, detect multiple QR codes in real time, and display their decoded text on the video feed. It also prints detected QR codes to the console. The program runs in a loop until the user presses 'q' to exit.

2. Install all necessary Python packages in a **Python virtual environment** and run this code on **Window** computer and on **Raspberry Pi Desktop** computer (remote).

- Use the built-in camera for a Windows Notebook.
- Use a USB Webcam for Raspberry Pi Desktop.

3. Revise the code to show detected each detected QR code and draw a bounding box around it on the video frame. Try to detect multiple QR codes simulataneously.

4. Write Python code that uses the **pyzbar** library to detect QR codes.



An example of Python-OpenCV App that shows a detected QR code

File: py_opencv_qrcode_detect.py

```
       import cv2
       import numpy as np

       # Note: Tested with Python OpenCV v4.12.0
       # Show OpenCV version
       print( f"Python OpenCV version: {cv2.__version__}" )

       # Define a function to draw text on the frame at (x, y)
       def drawText( x, y, text, color=(0,0,255) ):
           font = cv2.FONT_HERSHEY_SIMPLEX
           font_scale = 0.75
           thickness = 2
           _, _ = cv2.getTextSize(text, font, font_scale, thickness)
           cv2.putText(frame, text, (x, y),
                       font, font_scale, color, thickness, cv2.LINE_AA)

       # Open the camera (use the default device index 0)
       cap = cv2.VideoCapture(0)

       # Create a QRCodeDetector instance
       qr = cv2.QRCodeDetector()

       # Main loop which reads camera frames repeatedly
       while True:
           # Grab a frame from the camera
           ret, frame = cap.read()
           if not ret:
               break # If capture failed, exit loop

           # Try to detect multiple QR codes in the frame
           retval, decoded_info, points, _ = qr.detectAndDecodeMulti(frame)
           if retval:
               # If detected, print all detected QR code strings to console
               for qrcode in decoded_info:
                   print( f'QR Code: {qrcode}' )

           # Draw a text near the bottom of the frame
           text_x = 10
           text_y = frame.shape[0] - 12
           drawText( text_x, text_y, "Press q to exit" )

           cv2.imshow('QR Code Scanner', frame)
           if cv2.waitKey(1) == ord('q'):
         break

# Cleanup: release camera and close windows
cap.release()
cv2.destroyAllWindows()
```

Last update: 2025-07-26