Software Development Practice 1

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

GUI App Development on Ubuntu / Raspberry Pi OS

Objectives

- Learn how to build a basic MJPEG video streaming server and a screensaver using OpenCV and Qt5 software libraries, in Python and C++ programming languages.
- Explore cross-platform development and testing (Windows, WSL2 Ubuntu, Raspberry Pi Desktop).

Expected Learning Outcomes

After completing all tasks, students will be able to:

- Set up a Python-based MJPEG streaming server using Flask and OpenCV.
- View and process MJPEG video streams using OpenCV, Qt5 in both Python and C++ programming languages.
- Install and manage required dependencies on Windows and Linux-based systems.
- Compile and run C++ OpenCV applications using standard development tools.
- Troubleshoot common issues related to video capture, HTTP streaming, and client-server interaction.
- Implement a screensaver application to visualize / animate a specific algorithm.

Task 1: MJPEG Video Streaming with Python & C++

1. Implement a simple **Python-based MJPEG stream server on Windows**.

1.1 Open Windows PowerShell, install Python packages (in a Python virtual environment). We use **Python-OpenCV** to capture video frames from a USB WebCam or a built-in camera on a Windows notebook. In addition, we use the **Python-Flask** to implement a simple HTTP server (port 8080) for streaming an MJPEG video.

```
# Create and activate a Python virtual environment.
> python -m venv venv
> .\venv\Scripts\Activate

# Install OpenCV (e.g., opencv-python-4.12.0).
> pip install opencv-python

# Install Flask (e.g., flask-3.1.1).
> pip install flask
```

1.2 Create a new directory for the **Python project** (py_cv2_mjpeg_server/) and use the provided Python code (py_cv2_mjpeg_server/server.py) to run the **MJPEG server on Window**.

2. Implement an **MJPEG client on WSL2 Ubuntu and Raspberry Pi Desktop** (via Remote Desktop / VNC Client) to view the MJPEG video stream. We use **OpenCV and Qt5** for this purpose.

2.1. Run the following commands to install the required Python packages for **OpenCV** (in a Python virtual environment).

```
# Install OpenCV (e.g., opencv-python-4.12.0)
$ pip install opencv-python
```

2.2 Create a new directory for the Python project (py_cv2_mjpeg_client). Use the provided Python code (cv2_mjpeg_client/client.py) to run a MJPEG viewer.

2.3. Install the required **OpenCV libraries for C/C++**.

```
# Install OpenCV development packages (e.g. 4.6.0).
$ sudo apt install libopencv-dev

# Check the OpenCV version.
$ pkg-config --modversion opencv4
```

2.4 Create a new directory for the **OpenCV / C++ project** (cpp_cv2_mjpeg_client) and use the provided C++ code (cpp_cv2_mjpeg_client/main.cpp) to run a MJPEG viewer. Use the following command line to compile the source code.

```
$ g++ main.cpp -o ./cv2_mjpeg_client `pkg-config --cflags --libs opencv4`
```

2.5 Create a new directory for the **Qt5 / C++ project** (cpp_qt_mjpeg_client) and use the provided C++ code (cpp_qt_mjpeg_client/main.cpp) to run a MJPEG viewer.

```
# Install Qt5 packages.
$ sudo apt install qtbase5-dev qtchooser qt5-qmake \
  qtbase5-examples qtcreator qttools5-dev-tools qttools5-dev

# Check the version of qmake
$ qmake --version

# Create a Qt project file (.pro) under the project directory.
$ nano qt5_cv2_mjpeg_client.pro
```

```
QT += core gui widgets

CONFIG += c++11 link_pkgconfig

PKGCONFIG += opencv4

SOURCES += main.cpp
```

```
# Run the qmake command to build the project.
$ qmake && make
```

2.6 Create a new directory for **PyQt5 project** (py_qt_mjpeg_client) and use the provided Python code (py_qt_mjpeg_client/client.py) to run a MJPEG viewer.

```
# Install the necessary Python packages (in a Python virtual environment).
$ pip install pyqt5 opencv-python
```

Note: On **Raspberry Pi**, use the following command to install **PyQt5**.

```
# Install PyQt5 package (e.g., v5.15.9).
$ sudo apt-get install python3-pyqt5

# Check the version of the installed PyQt5.
$ /usr/bin/python3 \
  -c "import PyQt5.QtCore; print(PyQt5.QtCore.PYQT_VERSION_STR)"
```

Extra Assignments for Task 1

1. Extend the example code for the MJPEG viewer (**for all Python, OpenCV, and Qt5 versions**) to display the following overlay texts:

- The server’s IP address and port, along with a short description (e.g., the camera name or ID).
- The current date and time near the bottom edge of the viewer screen. Use the Asia/Bangkok timezone.

2. Extend the example code (**PyQt5 version only**) to add a scrolling text line that displays useful information. The text should move from right to left at an appropriate speed and repeat continuously.

3. Create a new project for (**Qt5 C++ version only**) an MJPEG viewer that displays two video feeds from different servers side by side simultaneously.

File 1: py_cv2_mjpeg_server/server.py

```
import cv2
from flask import Flask, Response

# Create an instance of the Flask web application.
app = Flask(__name__)

# Open default webcam (e.g., /dev/video0).
cap = cv2.VideoCapture(0)

# Set the width and height for the camera's video frames.
FRAME_W, FRAME_H = (640,480)
cap.set(cv2.CAP_PROP_FRAME_WIDTH, FRAME_W)
cap.set(cv2.CAP_PROP_FRAME_HEIGHT, FRAME_H)

# Define a generator function that yields video frames.
def generate_mjpeg():
    while True:
        # Read the next frame from the capture.
        ret, frame = cap.read()
        if not ret:
            break
        # Encode the frame as JPEG.
        ret, jpeg = cv2.imencode('.jpg', frame)
        if not ret:
            continue
        # Yield the frame in multipart MJPEG format.
        yield (b'--frame\r\n'
               b'Content-Type: image/jpeg\r\n\r\n'
               + jpeg.tobytes() + b'\r\n')

@app.route('/')
def video_feed():
    # Create an HTTP response (a streaming response).
    return Response(generate_mjpeg(),
             mimetype='multipart/x-mixed-replace; boundary=frame')

if __name__ == '__main__':
    # Start the HTTP server with multi-threading enabled.
    app.run(host='0.0.0.0', port=8080, threaded=True)
```

File 2: py_cv2_mjpeg_client/client.py

```
import cv2

def main():
    # Replace with your Windows IP and port.
    URL = 'http://192.168.100.38:8080'
    # Open the video stream as specified by the URL.
    cap = cv2.VideoCapture(URL)
    if not cap.isOpened():
        print("Cannot open video stream!")
        return
    while True:
        # Read the next frame.
        ret, frame = cap.read()
        if not ret:
            print("Video frame not received")
            break
        # Update the image.
        cv2.imshow("Webcam Video Stream", frame)
        # Press 'q' to quit.
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break
    # Release the camera.
    cap.release()
    # Close the OpenCV window.
    cv2.destroyAllWindows()

if __name__ == "__main__":
    main()
```

File 3: cpp_cv2_mjpeg_client/main.cpp

```
#include <opencv2/opencv.hpp>
#include <iostream>

using namespace std;

int main(int argc, char** argv) {
    // Print the OpenCV version.
    cout << "OpenCV version: " << CV_VERSION << endl;
    // Change the URL to match your MJPEG server.
    string URL = "http://192.168.100.38:8080";
    // Open the MJPEG stream as a video capture.
    cv::VideoCapture cap(URL);
    if (!cap.isOpened()) {
        cerr << "Error: Cannot open video stream!" << endl;
        return -1;
    }
    cv::Mat frame;
    while (true) {
        bool ret = cap.read(frame);
        if (!ret) {
            cerr << "Error: Cannot read video frame!" << endl;
            break;
        }
        cv::imshow("MJPEG Stream", frame); // Show the frame.
        if (cv::waitKey(1) == 'q') { // Press 'q' to exit.
            break;
        }
    }
    cap.release(); // Release the camera.
    cv::destroyAllWindows(); // Close all OpenCV windows.
    return 0;
}
```

File 4: cpp_qt_mjpeg_client/main.cpp

```cpp
#include <QApplication>  // Qt application framework
#include <QLabel>        // Widget to display images
#include <QTimer>        // Timer for periodic updates
#include <QImage>        // Image format class
#include <QPixmap>       // Pixmap to show images in QLabel
#include <opencv2/opencv.hpp> // OpenCV for video stream

using namespace std;

int main(int argc, char *argv[]) {
    // Create the Qt application.
    QApplication app(argc, argv);

    string URL = "http://192.168.100.38:8080";
    // OpenCV VideoCapture: open the MJPEG stream URL.
    cv::VideoCapture cap(URL);
    if (!cap.isOpened()) {
        qWarning("Failed to open the stream!");
        return -1;
    }

    // Create a QLabel to show the video frames.
    QLabel label("Stream Viewer");
    label.resize(640, 480); // Set the window size.
    label.setAlignment(Qt::AlignCenter); // Center the image.
    label.show(); // Show the window.

    // Create a QTimer to grab frames periodically.
    QTimer timer;
    QObject::connect(&timer, &QTimer::timeout, [&]() {
        cv::Mat frame;
        // Read the next frame from the stream.
        if (cap.read(frame)) {
            // Convert BGR (OpenCV default) to RGB.
            cv::cvtColor(frame, frame, cv::COLOR_BGR2RGB);
            // Wrap the OpenCV frame data in a QImage.
            QImage img(frame.data, frame.cols, frame.rows,
                       frame.step, QImage::Format_RGB888);
            // Update the QLabel pixmap with the new frame.
            label.setPixmap(QPixmap::fromImage(img));
        }
    });
    timer.start( 30 /*msec*/ ); // Start the Qt Timer.

    // Run the Qt event loop.
    return app.exec();
}
```

File 5: py_qt_mjpeg_client/client.py

```
import sys
import cv2
from PyQt5.QtCore import QTimer, Qt
from PyQt5.QtGui import QImage, QPixmap
from PyQt5.QtWidgets import QApplication, QLabel

def main():
    app = QApplication(sys.argv)

    # Set the MJPEG stream URL.
    url = "http://192.168.100.38:8080"
    cap = cv2.VideoCapture(url)
    if not cap.isOpened():
        print("Failed to open the stream!")
        return -1

    # Create QLabel to display video.
    label = QLabel("Stream Viewer")
    label.resize(640, 480)
    label.setAlignment(Qt.AlignCenter)
    label.show()

    # Create a QTimer to periodically fetch frames.
    timer = QTimer()
    def update_frame():
        ret, frame = cap.read()
        if ret:
            # Convert BGR (OpenCV) to RGB (Qt)
            frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
            h, w, channels = frame.shape
            bytes_per_line = channels * w
            img = QImage(frame.data, w, h, bytes_per_line,
                         QImage.Format_RGB888)
            label.setPixmap(QPixmap.fromImage(img))

    timer.timeout.connect(update_frame)
    timer.start(30)  # Update every 30 ms

    # Start Qt event loop
    sys.exit(app.exec_())

if __name__ == "__main__":
    main()
```

Task 2: Matrix Rain Screensaver with Qt5

1. Create a new C++ project (cpp_qt_matrix_rain) for a simpple screensaver, which is expected to run in a fullscreen mode on Ubuntu and Raspberry Pi Desktop.

2. Use the provided source code (*.cpp and *.h) and the **Qt project file** with the **qmake** command to build the project and run the executable file.

Extra Assignments for Task 2

1. Implement your own **screensaver application with Qt5** that visualizes an algorithm of your choice (→ **algorithm visualization**), such as graph or tree-based, recursion-based algorithms (Tower of Hanoi, recursion trees, fractals), or basic 2D/3D graphics animation (e.g., bouncing balls).

2. Write code comments and explain how your code works, with a focus on the algorithm you visualized and how it is represented or animated on screen.

File 1: cpp_qt_matrix_rain.pro

```
QT += core gui widgets

TARGET = MatrixRain

TEMPLATE = app

SOURCES += \
    main.cpp \
    MatrixRain.cpp

HEADERS += \
    MatrixRain.h
```

File 2: cpp_qt_matrix_rain/main.cpp

```
#include <QApplication>
#include "MatrixRain.h"

int main(int argc, char *argv[]) {
    QApplication app(argc, argv);
    MatrixRain window;
    window.show();
    return app.exec();
}
```

File 3: cpp_qt_matrix_rain/MatrixRain.h

```
#ifndef MATRIXRAIN_H
#define MATRIXRAIN_H

#include <QWidget>
#include <QVector>
#include <QColor>
#include <QString>

struct TextDrop {
    int id;
    float x, y;
    int speed;
    QString text;
    QColor color;
    int fontSize;
};

class MatrixRain : public QWidget {
    Q_OBJECT

public:
    MatrixRain(QWidget *parent = nullptr);

protected:
    void paintEvent(QPaintEvent *) override;
    void resizeEvent(QResizeEvent *) override;
    void keyPressEvent(QKeyEvent *event) override;

private slots:
    void updateDrops();

private:
    QVector<TextDrop> drops;

    void generateDrops();
    void initializeDrop(TextDrop &drop);
};

#endif // MATRIXRAIN_H
```

File 4: cpp_qt_matrix_rain/MatrixRain.cpp

```
#include "MatrixRain.h"

#include <QPainter>
#include <QTimer>
#include <QRandomGenerator>
#include <QKeyEvent>
#include <QApplication>
#include <iostream>

using namespace std;

#define TEXT_FONT "Liberation Serif"

MatrixRain::MatrixRain(QWidget *parent) : QWidget(parent) {
    setWindowState(Qt::WindowFullScreen);
    setStyleSheet("background-color: black;");
    generateDrops();
    QTimer *timer = new QTimer(this);
    connect(timer, &QTimer::timeout, this, &MatrixRain::updateDrops);
    timer->start(25);
}

void MatrixRain::paintEvent(QPaintEvent *) {
    QPainter painter(this);
    painter.setRenderHint(QPainter::TextAntialiasing);
    for (const TextDrop &drop : drops) {
        QFont font(TEXT_FONT, drop.fontSize, QFont::Bold);
        painter.setFont(font);
        painter.setPen(drop.color);
        painter.drawText(drop.x, static_cast<int>(drop.y), drop.text);
    }
}

void MatrixRain::resizeEvent(QResizeEvent *) {
    generateDrops();
}

void MatrixRain::keyPressEvent(QKeyEvent *event) {
    if (event->key() == Qt::Key_Escape) {
        QApplication::quit();
    }
}

void MatrixRain::updateDrops() {
    for (TextDrop &drop : drops) {
        if (drop.y > height()) {
            initializeDrop(drop);
        } else {
            drop.y += drop.speed;
        }
    }
    update();
}

void MatrixRain::initializeDrop(TextDrop &drop) {
    drop.text = QRandomGenerator::global()->bounded(2) ? "0" : "1";
    drop.speed = QRandomGenerator::global()->bounded(4, 12);
    drop.fontSize = QRandomGenerator::global()->bounded(32, 96);
    int value = QRandomGenerator::global()->bounded(64, 64 + drop.id);
    int green = qBound(0, value, 255);
    drop.color = QColor(0, green, 0);
    drop.x = QRandomGenerator::global()->bounded(width() - drop.fontSize);
    drop.y = QRandomGenerator::global()->bounded(-height(), 0);
}
```

```
void MatrixRain::generateDrops() {
    drops.clear();
    int numDrops = QRandomGenerator::global()->bounded(width()/10, width()/4);
    cout << "Number of drops: " << numDrops << endl;
    for (int i = 0; i < numDrops; ++i) {
        TextDrop drop;
        drop.id = i;
        initializeDrop(drop);
        drops.append(drop);
    }
}
```

Last update: 2025-07-15