Python Exercises

Zenoh - OpenCV - MCAP

This worksheet introduces three tools for building distributed and multimedia applications in Python:

- **Zenoh** – A communication middleware that unifies data in motion, data at rest, and computations. It supports publish/subscribe, queries, and storage in an efficient and flexible way.
- **OpenCV** – A widely used computer vision library for image and video processing, machine learning, and AI applications.
- **MCAP** – A flexible, performant, and well-structured container format for recording and replaying message data (similar to ROS bag files but more general-purpose).

The goal of these exercises is to gain hands-on experience in integrating these technologies. By the end, you will be able to:

- Publish and subscribe to messages using Zenoh.
- Capture and process video frames with OpenCV.
- Record and replay data using MCAP.

Resources:

- **Zenoh:** https://zenoh.io
- **OpenCV** https://docs.opencv.org/4.12.0/
- **MCAP:** https://mcap.dev/docs/python/

1. Quick Start: Structuring Your Python Project

To keep your code organized, use the following folder structure:

```
zenoh_opencv_mcap/
├── README.md
├── requirements.txt
├── venv/                # Python virtual environment
├── src/
│    ├── pub_sub.py       # Source files goes here
│    ├── ...
```

- `src/` will hold your source code.
- `requirements.txt` will specify dependencies.
- `venv/` is the isolated Python environment (so you don't pollute system packages).

1.1. Create and Activate a Virtual Environment

```bash
# Create a project folder
mkdir zenoh_opencv_mcap
cd zenoh_opencv_mcap

# Create a virtual environment
python3 -m venv venv

# Activate it
source venv/bin/activate
```

When activated, your terminal prompt should show (venv) at the start.

1.2. Install Required Packages

We need Zenoh Python API, OpenCV, and MCAP.

```bash
pip install eclipse-zenoh opencv-python mcap
```

Optionally, create a `requirements.txt` file:

```txt
pyzenoh
opencv-python
mcap
```

Then install all with:

```bash
pip install -r requirements.txt
```

1.3. Verify Installations

In Python:

```python
import zenoh
import cv2
import mcap
print("Zenoh module loaded successfully:")
print("OpenCV version:", cv2.__version__)
print("MCAP version:", mcap.__version__)
```

If no errors occur, you are ready to start the exercises.

2. Zenoh Basics

Zenoh provides a unified way to handle publish/subscribe (pub/sub) and query/reply. These exercises will help you learn the fundamentals of using Zenoh in Python.

2.1. First Publisher

Write a Python program that publishes a message every second. Create a file `src/publisher.py`:

```python
import time
import zenoh

def main():
    z = zenoh.open(zenoh.Config())  # Open a Zenoh session
    pub = z.declare_publisher("demo/example/hello") # Declare a publisher
    for i in range(10): # Publish messages every second
        msg = f"Hello {i}"
        print("[Publisher] Publishing:", msg)
        pub.put(msg)
        time.sleep(1)
    z.close() # Close session
if __name__ == "__main__":
    main()
```

Run it in one terminal:

```bash
python src/publisher.py
```

2.2. First Subscriber

```python
import time
import zenoh

def main():
    z = zenoh.open(zenoh.Config())  # Open a Zenoh session
    # Declare a subscriber
    def listener(sample):
        print(f"[Subscriber] Received: {sample.payload.to_string()}")

    sub = z.declare_subscriber("demo/example/hello", listener)
    print("Listening... Press Ctrl+C to stop.")
    try:
        while True:
            time.sleep(1)
    except KeyboardInterrupt:
        pass
    z.close() # Close session

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

Run it in another terminal at the same time as the publisher. You should see the subscriber printing messages.

2.3. Request/Reply

Zenoh also supports a query/reply model.

Create a file `src/responder.py`:

```python
import time
import zenoh

def main():
    z = zenoh.open(zenoh.Config())
    # Register a queryable resource
    def callback(query):
        print("[Responder] Received query:", query.selector)
        query.reply(query.selector.key_expr, "This is PSR")

    qable = z.declare_queryable("demo/example/service", callback)
    print("Responder ready. Ctrl+C to stop.")
    try:
        while True:
```

```
            time.sleep(1)
    except KeyboardInterrupt:
        pass

cv2.destroyAllWindows()
    z.close()

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

`src/requester.py`

```python
import zenoh

def main():
    z = zenoh.open(zenoh.Config())

    q = z.declare_querier("demo/example/service")
    replies = q.get()
    for reply in replies:
        print("[Requester] Got reply:", reply.ok.payload.to_string())

    z.close()

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

2.4. Challengs

2.4.1. Countdown Publisher

Write a new program called countdown_publisher.py that:

- Publishes the numbers from 10 down to 1.
- Waits one second between each publish.
- Publishes the final message “Lift-off!”.

Run it together with your subscriber from Exercise 2.

2.4.2. Math Service

Implement a Zenoh service called “demo/math/square” that:

- Waits for a query containing a number.
- Replies with the square of that number.
- For example, a query with “7” should reply “49”.

Run your `math_responder.py` in one terminal, and write a `math_requester.py` that asks for numbers 1 through 5 and prints the replies.

3. Combining Zenoh and OpenCV

In these exercises, you will learn how to publish video frames captured with OpenCV and subscribe to them using Zenoh.

3.1. Publish Camera Frames

Create a new file `src/video_publisher.py`:

```python
import cv2
import numpy as np
import zenoh
import time

def main():
    z = zenoh.open(zenoh.Config())

    def listener(sample):
        # Convert received bytes back into an image
        np_arr = np.frombuffer(sample.payload.to_bytes(), np.uint8)
        frame = cv2.imdecode(np_arr, cv2.IMREAD_COLOR)

        # Show frame
        cv2.imshow("Subscriber Video", frame)
        if cv2.waitKey(1) & 0xFF == ord("q"):
            exit(0)

    sub = z.declare_subscriber("demo/video/frame", listener)
    print("Listening for frames... Press 'q' to quit")
    try:
        while True:
            time.sleep(1)
    except KeyboardInterrupt:
        pass
    cv2.destroyAllWindows()
    z.close()

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

Run it, and it will continuously capture frames and publish them as JPEG-encoded bytes.

3.2. Subscribe and Display Frames

Create `src/video_subscriber.py`:

```python
import cv2
import zenoh
import time

def main():
    z = zenoh.open(zenoh.Config())
    pub = z.declare_publisher("demo/video/frame")
    cap = cv2.VideoCapture(0) # Open default camera (0 = webcam)
```

```
    if not cap.isOpened():
        print("❌ Cannot open camera")
        return
    while True:
        ret, frame = cap.read()
        if not ret:
            print("❌ Failed to grab frame")
            break
        _, buffer = cv2.imencode(".jpg", frame) # Encode frame as JPEG
        pub.put(buffer.tobytes()) # Publish frame
        print("[Publisher] Sent frame")
    cap.release()
    z.close()

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

Run this in a separate terminal (or even on a different machine in the same network). You should see the video feed from the publisher.

3.3. Apply OpenCV Processing Before Publishing

Modify the publisher to grayscale the frames before sending.

In `src/video_publisher.py`, inside the loop:

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

# Encode as JPEG
_, buffer = cv2.imencode(".jpg", gray)

# Publish
pub.put(buffer.tobytes())
print("[Publisher] Sent grayscale frame")
```

3.4. Face Detection with OpenCV + Zenoh

Let's process frames with a Haar cascade classifier for face detection.

Update publisher loop:

```python

face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades +
"haarcascade_frontalface_default.xml")

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

```

```
9       faces = face_cascade.detectMultiScale(gray, 1.3, 5)
10      # Draw rectangles around detected faces
11      for (x, y, w, h) in faces:
12          cv2.rectangle(frame, (x, y), (x+w, y+h), (255, 0, 0), 2)
13
14      _, buffer = cv2.imencode(".jpg", frame)
15      pub.put(buffer.tobytes())
16      print(f"[Publisher] Sent frame with {len(faces)} faces detected")
```

The subscriber will now show a live video stream with bounding boxes around faces.

3.5. Challenges

3.5.1. Edge Detection

Update your video publisher to apply an edge detection filter (Canny) before publishing. The subscriber should display a live video showing only the detected edges.

3.5.2. Moving Circle

Draw a red circle that moves horizontally across the video (like an animation) and resets when it reaches the edge. Overlay it on top of the live camera feed before publishing.

4. Recording and Replaying with MCAP

4.1. Record Frames into MCAP

We'll subscribe to the Zenoh video stream and write each received frame into an MCAP file.

Create `src/video_recorder.py`:

```python
import zenoh
import time
from mcap.writer import Writer

def main():
    z = zenoh.open(zenoh.Config())
    # Open MCAP file for writing
    with open("video_stream.mcap", "wb") as f:
        writer = Writer(f)
        start_time = int(time.time() * 1e9)  # nanoseconds

        # Create channel for video frames
        channel_id = writer.register_channel(
            topic="demo/video/frame",
            message_encoding="opencv_jpg",
            schema_id=0,
        )

        def listener(sample):
            ts = int(time.time() * 1e9)  # timestamp in ns
            data = sample.payload.to_bytes()
```

```
22              writer.add_message(
23                  channel_id=channel_id,
24                  log_time=ts,
25                  publish_time=ts,
26                  data=data,
27              )
28              print(f"[Recorder] Wrote frame at {ts}")
29
30          writer.start()
31          sub = z.declare_subscriber("demo/video/frame", listener)
32
33          print("Recording... Press Ctrl+C to stop")
34          try:
35              while True:
36                  time.sleep(1)
37          except KeyboardInterrupt:
38              pass
39
40          # Finish writing MCAP file
41          writer.finish()
42          print("Saved to video_stream.mcap")
43
44  if __name__ == "__main__":
45      main()
```

Run the publisher, the subscriber, and then run video_recorder.py. It will save all frames to `video_stream.mcap`.

4.2. Replay Frames from MCAP

Now, we'll replay the recorded frames from the MCAP file and display them.

Create `src/video_player.py`

```python
1   import cv2
2   import numpy as np
3   from mcap.reader import make_reader
4
5   def main():
6       with open("video_stream.mcap", "rb") as f:
7           reader = make_reader(f)
8
9           for schema, channel, message in reader.iter_messages():
10              frame = cv2.imdecode(
11                  np.frombuffer(message.data, np.uint8), cv2.IMREAD_COLOR
12              )
13              cv2.imshow("MCAP Replay", frame)
14
```

```
15              # Wait ~33 ms for ~30 FPS replay
16              if cv2.waitKey(33) & 0xFF == ord("q"):
17                  break
18
19      cv2.destroyAllWindows()
20
21  if __name__ == "__main__":
22      main()
```

Run `video_player.py`and you should see the replay of the previously recorded video.

4.3. Challenges

4.3.1. Replay into Zenoh

Instead of just displaying frames locally, let's replay them back into Zenoh so that other subscribers (e.g., video_subscriber.py) can see them as if they were live.

4.3.2. Adjustable Replay Speed

Modify your `video_replayer.py`so that it accepts a command-line argument for speed multiplier:

- 1.0 → real-time speed
- 0.5 → half speed (slower)
- 2.0 → double speed (faster)

Test with your previously recorded file.