ROS2 Camera Web Streaming Implementation Guide

This guide walks you through implementing a web-based video streaming solution for your ROS2 camera publisher on Raspberry Pi 4.

Prerequisites

Before starting, ensure you have the following installed on your Pi4:

bash

```bash
# Install required Python packages
pip3 install flask opencv-python cv-bridge

# Verify ROS2 installation
ros2 --version

# Check if your camera topic is active
ros2 topic list | grep camera
```

Step 1: Create the Web Streamer Node

Create a new Python file for your web streamer:

bash

```bash
# Create a new file
nano ~/ros2_web_streamer.py
```

Copy the following code into the file:

python

```python
#!/usr/bin/env python3
import rclpy
from rclpy.node import Node
from sensor_msgs.msg import Image
from cv_bridge import CvBridge
import cv2
import threading
from flask import Flask, Response
import time
import logging

class WebStreamer(Node):
    def __init__(self):
        super().__init__('web_streamer')
        self.bridge = CvBridge()
        self.latest_frame = None
        self.frame_lock = threading.Lock()

        # Subscribe to camera topic
        self.subscription = self.create_subscription(
            Image,
            '/camera/image_raw',
            self.image_callback,
            10)

        self.get_logger().info('Web streamer node started. Waiting for camera data...')

        # Flask app setup
        self.app = Flask(__name__)
        self.app.logger.setLevel(logging.WARNING)  # Reduce Flask logging
        self.setup_routes()

    def setup_routes(self):
        """Set up Flask routes"""
        self.app.add_url_rule('/video_feed', 'video_feed', self.video_feed)
        self.app.add_url_rule('/', 'index', self.index)
        self.app.add_url_rule('/status', 'status', self.status)

    def image_callback(self, msg):
        """Callback for receiving camera images"""
        try:
            # Convert ROS Image to OpenCV format
            cv_image = self.bridge.imgmsg_to_cv2(msg, 'bgr8')
```

```
            with self.frame_lock:
                self.latest_frame = cv_image

        except Exception as e:
            self.get_logger().error(f'Error processing image: {str(e)}')

    def generate_frames(self):
        """Generate frames for video streaming"""
        while True:
            with self.frame_lock:
                if self.latest_frame is not None:
                    # Encode frame as JPEG
                    ret, buffer = cv2.imencode('.jpg', self.latest_frame,
                                               [cv2.IMWRITE_JPEG_QUALITY, 80])
                    if ret:
                        frame = buffer.tobytes()
                        yield (b'--frame\r\n'
                               b'Content-Type: image/jpeg\r\n\r\n' + frame + b'\r\n')
                else:
                    # Send a placeholder frame if no camera data
                    placeholder = self.create_placeholder_frame()
                    ret, buffer = cv2.imencode('.jpg', placeholder)
                    if ret:
                        frame = buffer.tobytes()
                        yield (b'--frame\r\n'
                               b'Content-Type: image/jpeg\r\n\r\n' + frame + b'\r\n')

            time.sleep(0.033)  # ~30 FPS

    def create_placeholder_frame(self):
        """Create a placeholder frame when no camera data is available"""
        frame = cv2.imread('/dev/null')  # This will be None
        if frame is None:
            # Create a simple placeholder
            frame = cv2.zeros((480, 640, 3), dtype=cv2.uint8)
            cv2.putText(frame, 'Waiting for camera...', (150, 240),
                        cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 255, 255), 2)
        return frame

    def video_feed(self):
        """Video streaming route"""
        return Response(self.generate_frames(),
                        mimetype='multipart/x-mixed-replace; boundary=frame')
```

```
    def status(self):
        """Status endpoint"""
        with self.frame_lock:
            has_frame = self.latest_frame is not None

        return {
            'status': 'active' if has_frame else 'waiting',
            'message': 'Camera feed active' if has_frame else 'Waiting for camera data'
        }

    def index(self):
        """Main page with video stream"""
        return '''
        <!DOCTYPE html>
        <html>
        <head>
            <title>ROS2 Camera Stream</title>
            <style>
                body {
                    font-family: Arial, sans-serif;
                    background-color: #f0f0f0;
                    margin: 0;
                    padding: 20px;
                }
                .container {
                    max-width: 800px;
                    margin: 0 auto;
                    background: white;
                    padding: 20px;
                    border-radius: 10px;
                    box-shadow: 0 2px 10px rgba(0,0,0,0.1);
                }
                h1 {
                    color: #333;
                    text-align: center;
                }
                .video-container {
                    text-align: center;
                    margin: 20px 0;
                }
                img {
                    max-width: 100%;
                    height: auto;
```

```
            border: 2px solid #ddd;
            border-radius: 5px;
        }
        .controls {
            text-align: center;
            margin: 20px 0;
        }
        button {
            background-color: #4CAF50;
            color: white;
            padding: 10px 20px;
            border: none;
            border-radius: 5px;
            cursor: pointer;
            margin: 0 10px;
        }
        button:hover {
            background-color: #45a049;
        }
        .status {
            text-align: center;
            margin: 10px 0;
            padding: 10px;
            background-color: #e8f5e8;
            border-radius: 5px;
        }
    </style>
</head>
<body>
    <div class="container">
        <h1>ROS2 Camera Stream</h1>
        <div class="status" id="status">Loading...</div>
        <div class="video-container">
            <img src="/video_feed" alt="Camera Feed" id="videoStream">
        </div>
        <div class="controls">
            <button onclick="refreshStream()">Refresh Stream</button>
            <button onclick="checkStatus()">Check Status</button>
        </div>
    </div>

    <script>
        function refreshStream() {
            const img = document.getElementById('videoStream');
```

```
            const timestamp = new Date().getTime();
            img.src = '/video_feed?' + timestamp;
        }

        function checkStatus() {
            fetch('/status')
                .then(response => response.json())
                .then(data => {
                    document.getElementById('status').textContent = data.message;
                })
                .catch(error => {
                    document.getElementById('status').textContent = 'Error checking status';
                });
        }

        // Check status on page load
        window.onload = function() {
            checkStatus();
        };
    </script>
</body>
</html>
'''

def main(args=None):
    rclpy.init(args=args)
    node = WebStreamer()

    # Run Flask in a separate thread
    flask_thread = threading.Thread(
        target=lambda: node.app.run(host='0.0.0.0', port=5000, debug=False)
    )
    flask_thread.daemon = True
    flask_thread.start()

    print("Web streamer started!")
    print("Access the stream at: http://your_pi_ip:5000")

    try:
        rclpy.spin(node)
    except KeyboardInterrupt:
        print("\nShutting down web streamer...")
    finally:
        rclpy.shutdown()
```

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

Step 2: Make the Script Executable

bash

```
# Make the script executable
chmod +x ~/ros2_web_streamer.py
```

Step 3: Start Your Camera Publisher

First, ensure your camera publisher node is running:

bash

```
# Start your camera publisher (adjust the command based on your setup)
ros2 run your_package camera_publisher
```

Step 4: Run the Web Streamer

In a new terminal (or SSH session):

bash

```
# Navigate to the script location
cd ~

# Run the web streamer
python3 ros2_web_streamer.py
```

You should see output like:

```
Web streamer started!
Access the stream at: http://your_pi_ip:5000
```

Step 5: Access the Stream

From any device on your network:

1. Open a web browser
2. Navigate to `http://your_pi_ip:5000` (replace `your_pi_ip` with your Pi's IP address)

3. You should see the camera feed in your browser

Troubleshooting

Common Issues and Solutions:

1. **"No module named 'flask'"**

   bash

   ```
   pip3 install flask
   ```

2. **"No module named 'cv_bridge'"**

   bash

   ```
   sudo apt install ros-humble-cv-bridge python3-opencv
   ```

3. **Can't access the web page**

   bash

   ```
   # Check if the Pi's firewall is blocking port 5000
   sudo ufw allow 5000

   # Or find your Pi's IP address
   hostname -I
   ```

4. **Camera topic not found**

   bash

   ```
   # Check available topics
   ros2 topic list

   # Check if your camera node is publishing
   ros2 topic hz /camera/image_raw
   ```

5. **Port already in use**

   bash

   ```
   # Kill any existing process on port 5000
   sudo lsof -ti:5000 | xargs kill -9
   ```

Advanced Configuration

Changing the Port

To use a different port, modify line in the script:

```python
target=lambda: node.app.run(host='0.0.0.0', port=8080, debug=False)  # Change 5000 to 8080
```

Adjusting Frame Rate

Modify the sleep time in the `generate_frames` method:

```python
time.sleep(0.033)  # 30 FPS
time.sleep(0.066)  # 15 FPS
time.sleep(0.1)    # 10 FPS
```

Improving Performance

For better performance on Pi4:

```python
# In the generate_frames method, reduce JPEG quality
ret, buffer = cv2.imencode('.jpg', self.latest_frame,
                           [cv2.IMWRITE_JPEG_QUALITY, 60])  # Lower quality = faster
```

Security Note

This implementation runs on all network interfaces (`0.0.0.0`). For production use, consider:

- Adding authentication
- Using HTTPS
- Restricting to specific network interfaces
- Adding rate limiting

Next Steps

Once you have the basic streaming working, you can:

- Add recording functionality
- Implement motion detection
- Add multiple camera support

- Create a mobile-responsive interface
- Add zoom and pan controls