Tutorial 7 ROS Programming - Computer Vision

Objective

This tutorial is aimed at learning to develop ROS2 Computer Vision Python programs with OpenCV and YOLO to scan QR codes and identify objects.

1 Package Installation and Setup

1. Open a new terminal and install the required packages using the following commands:

```
$ sudo apt update
$ sudo apt install -y ros-jazzy-vision-opencv ros-jazzy-image-transport
ros-jazzy-image-transport-plugins python3-opencv
```

2. Source the ROS2 Jazzy setup bash file

```
$ source /opt/ros/jazzy/setup.bash
```

3. Create a new workspace:

```
$ mkdir -p ~/cv_ws/src
$ cd ~/cv_ws/src
$ ros2 pkg create --build-type ament_python cv_robotics_tutorial
--dependencies rclpy sensor_msgs cv_bridge image_transport std_msgs
$ cd ~/cv_ws
$ colcon build
$ source install/setup.bash
```

2 QR Code Scanning

1. Close all the terminals opened previously.
2. Connect the camera to the virtual machine by: VM => Removable Devices => Find the camera) => Connect
3. Set the USB controller to USB 3.1 by: VM => Settings => USB Controller => USB 3.1
4. Open the workspace folder by File => Open Folder "~/cv_ws" in VSCode
5. Create the image_publisher.py in the path "~/cv_ws/src/cv_robotics_tutorial/cv_robotics_tutorial/":

```
from rclpy.qos import QoSProfile, QoSReliabilityPolicy, QoSHistoryPolicy
import rclpy
from rclpy.node import Node
from sensor_msgs.msg import Image
from cv_bridge import CvBridge
import cv2
import numpy as np

from sensor_msgs.msg import CompressedImage

class ImagePublisher(Node):
    def __init__(self):
            super().__init__('image_publisher')
            qos_profile = QoSProfile(
                depth=1,
                reliability=QoSReliabilityPolicy.BEST_EFFORT,
                history=QoSHistoryPolicy.KEEP_LAST
            )
            self.publisher_ = self.create_publisher(
                CompressedImage,
                '/camera/image_raw/compressed', qos_profile)
            self.bridge = CvBridge()
            # 0 for default webcam
            self.cap = cv2.VideoCapture(0)
            # Fix 1: Minimal buffer
            self.cap.set(cv2.CAP_PROP_BUFFERSIZE, 1)
            if not self.cap.isOpened():
                self.get_logger().error('Cannot open camera')
                rclpy.shutdown()
            # Publish at 10 Hz
            self.timer = self.create_timer(0.1, self.timer_callback)

    def timer_callback(self):
        ret, frame = self.cap.read()
```

```
        if ret:
            msg = CompressedImage()
            msg.header.stamp = self.get_clock().now().to_msg()
            msg.format = "jpeg"
            msg.data = np.array(cv2.imencode('.jpg', frame)[1]).tobytes()
            self.publisher_.publish(msg)
        else:
            self.get_logger().warn('Failed to capture frame')


def main(args=None):
    rclpy.init(args=args)
    node = ImagePublisher()
    rclpy.spin(node)
    node.cap.release()
    rclpy.shutdown()


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

6. Modify the setup.py's "console_scripts" by adding this line inside the bracket:

```
'image_publisher = cv_robotics_tutorial.image_publisher:main'
```

```
    entry_points={
        'console_scripts': [
            'image_publisher = cv_robotics_tutorial.image_publisher:main'
        ],
    },
```

7. Open a new terminal to build the package:

```
$ cd ~/cv_ws
$ colcon build --packages-select cv_robotics_tutorial
$ source install/setup.bash
```

8. Try to run the image_publisher node:

```
$ ros2 run cv_robotics_tutorial image_publisher
```

A "/camera/image_raw/compressed" topic should be found in the ROS2 topic list.

```
$ ros2 topic list
```

9. Create the qr_scanner.py in the path "~/cv_ws/src/cv_robotics_tutorial/cv_robotics_tutorial/":

```
from rclpy.qos import QoSProfile, QoSReliabilityPolicy, QoSHistoryPolicy
import rclpy
from rclpy.node import Node
from sensor_msgs.msg import Image
from cv_bridge import CvBridge
import cv2


from sensor_msgs.msg import CompressedImage


class QRScanner(Node):
    def __init__(self):
        super().__init__('qr_scanner')
        qos_profile = QoSProfile(
            depth=1,
            reliability=QoSReliabilityPolicy.BEST_EFFORT,
            history=QoSHistoryPolicy.KEEP_LAST
        )
        self.subscription = self.create_subscription(
            CompressedImage, '/camera/image_raw/compressed',
            self.listener_callback, qos_profile
        )
        self.bridge = CvBridge()
        self.detector = cv2.QRCodeDetector()


    def listener_callback(self, msg):
            try:
                cv_image = self.bridge.compressed_imgmsg_to_cv2(
                   msg, desired_encoding='bgr8')
            except Exception as e:
                self.get_logger().error(
                   f'Image conversion failed: {str(e)}')
                return
```

```
            data, bbox, _ = self.detector.detectAndDecode(cv_image)
            if data:
                self.get_logger().info(f'QR Code detected: {data}')
                if bbox is not None:
                    n = len(bbox)
                    for i in range(n):
                        # Convert to int
                        pt1 = (int(bbox[i][0][0]), int(bbox[i][0][1]))
                        pt2 = (int(bbox[(i + 1) % n][0][0]),
                           int(bbox[(i + 1) % n][0][1]))
                        cv2.line(cv_image, pt1, pt2, (0, 255, 0), 2)
            else:
                self.get_logger().info('No QR Code detected')
            # Always show the window
            cv2.imshow('QR Detection', cv_image)
            cv2.waitKey(1)


def main(args=None):
    rclpy.init(args=args)
    node = QRScanner()
    rclpy.spin(node)
    cv2.destroyAllWindows()
    rclpy.shutdown()


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

10. Modify the setup.py's "console_scripts" by adding this line inside the bracket:

```
'qr_scanner = cv_robotics_tutorial.qr_scanner:main'
```

```
    entry_points={
        'console_scripts': [
            'image_publisher = cv_robotics_tutorial.image_publisher:main',
            'qr_scanner = cv_robotics_tutorial.qr_scanner:main'
        ],
    },
```

11. Try the QR code scanning function:
    Build the package and run the qr_scanner node:

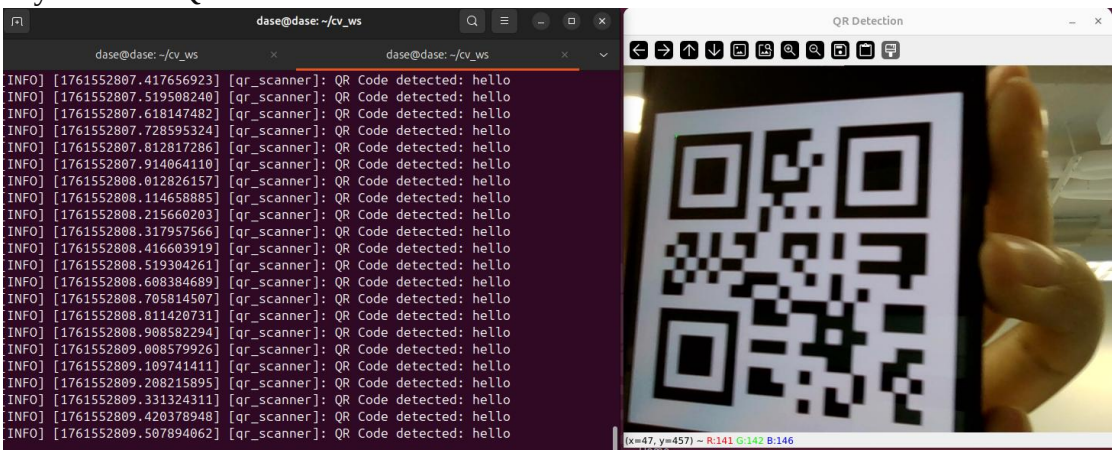
```
$ cd ~/cv_ws

$ colcon build

$ source install/setup.bash

$ ros2 run cv_robotics_tutorial qr_scanner
```

Try to scan a QR code!



A QR Code with specified content can be generated from this link: https://cli.im/

3 Object identification

1. Close all the terminals opened previously.
2. Open a new terminal and install the required packages:

```
$ pip3 install onnxruntime opencv-python --break-system-packages

$ pip3 install numpy==1.26.4 --break-system-packages
```

3. Create a folder inside "~/cv_ws/src/cv_robotics_tutorial/cv_robotics_tutorial" with the name "models" and put the file "yolov8n.onnx" in it.
4. Create the coco_classes.py in path "~/cv_ws/src/cv_robotics_tutorial/cv_robotics_tutorial"

```
#!/usr/bin/env python3


COCO_CLASSES = [
    'person', 'bicycle', 'car', 'motorcycle', 'airplane', 'bus', 'train',
    'truck', 'boat', 'traffic light', 'fire hydrant', 'stop sign',
    'parking meter', 'bench', 'bird', 'cat', 'dog', 'horse', 'sheep',
```

```
    'cow', 'elephant', 'bear', 'zebra', 'giraffe', 'backpack', 'umbrella',
    'handbag', 'tie', 'suitcase', 'frisbee', 'skis', 'snowboard',
    'sports ball', 'kite', 'baseball bat', 'baseball glove', 'skateboard',
    'surfboard', 'tennis racket', 'bottle', 'wine glass', 'cup', 'fork',
    'knife', 'spoon', 'bowl', 'banana', 'apple', 'sandwich', 'orange',
    'broccoli', 'carrot', 'hot dog', 'pizza', 'donut', 'cake', 'chair',
    'couch', 'potted plant', 'bed', 'dining table', 'toilet', 'tv',
    'laptop', 'mouse', 'remote', 'keyboard', 'cell phone', 'microwave',
    'oven', 'toaster', 'sink', 'refrigerator', 'book', 'clock', 'vase',
    'scissors', 'teddy bear', 'hair drier', 'toothbrush'
]
```

5. Create the yolo_onnx_detector.py in path "~/cv_ws/src/cv_robotics_tutorial/cv_robotics_tutorial"

```
#!/usr/bin/env python3


import rclpy
from rclpy.node import Node
from sensor_msgs.msg import CompressedImage
from cv_bridge import CvBridge
import cv2
import numpy as np
import onnxruntime as ort
from rclpy.qos import QoSProfile, QoSReliabilityPolicy, QoSHistoryPolicy
import os
from ament_index_python.packages import get_package_share_directory
from .coco_classes import COCO_CLASSES


class YOLOONNXDetector(Node):
    def __init__(self):
        super().__init__('yolo_onnx_detector')


        # Declare parameters
        self.declare_parameter('model_path', '')
```

```
        self.declare_parameter('confidence_threshold', 0.5)
        self.declare_parameter('iou_threshold', 0.45)
        self.declare_parameter('show_window', True)
        self.declare_parameter('input_size', 640)

        # Get parameters
        model_path = self.get_parameter('model_path').value
        self.conf_threshold = self.get_parameter(
           'confidence_threshold').value
        self.iou_threshold = self.get_parameter('iou_threshold').value
        self.show_window = self.get_parameter('show_window').value
        self.input_size = self.get_parameter('input_size').value

        # Set default model path if not provided
        # Set default model path if not provided
        if not model_path:
            # Try to find the model in the source directory
            try:
                # First try: source workspace location
                workspace_path = os.path.expanduser(
                    '~/cv_ws/src/cv_robotics_tutorial')
                model_path = os.path.join(
                     workspace_path, 'cv_robotics_tutorial',
                     'models', 'yolov8n.onnx')

                if not os.path.exists(model_path):
                    # Second try: installed share directory
                    package_path = get_package_share_directory(
                       'cv_robotics_tutorial')
                    model_path = os.path.join(
                       package_path, 'models', 'yolov8n.onnx')
            except Exception as e:
                self.get_logger().warn(f'Error finding package: {e}')
                # Fallback to absolute path
```

```
                model_path = os.path.expanduser(
'~/cv_ws/src/cv_robotics_tutorial/cv_robotics_tutorial/models/yolov8n.onnx'
)


        # Check if model exists
        if not os.path.exists(model_path):
            self.get_logger().error(
                f'Model file not found: {model_path}')
            self.get_logger().error(
                'Please ensure yolov8n.onnx is in the models directory')
            raise FileNotFoundError(f'Model not found: {model_path}')


        # Load ONNX model
        self.get_logger().info(f'Loading ONNX model from: {model_path}')
        self.session = ort.InferenceSession(
            model_path, providers=['CPUExecutionProvider'])
        self.get_logger().info('ONNX model loaded successfully')


        # Get model input details
        self.input_name = self.session.get_inputs()[0].name
        self.output_names = [
            output.name for output in self.session.get_outputs()]


        # Class names
        self.class_names = COCO_CLASSES


        # Initialize CV Bridge
        self.bridge = CvBridge()


        # QoS Profile
        qos_profile = QoSProfile(
            depth=1,
            reliability=QoSReliabilityPolicy.BEST_EFFORT,
            history=QoSHistoryPolicy.KEEP_LAST
```

```
        )

        # Subscriber to compressed camera images
        self.subscription = self.create_subscription(
            CompressedImage,
            '/camera/image_raw/compressed',
            self.image_callback,
            qos_profile
        )

        # Publisher for detection results
        self.detection_publisher = self.create_publisher(
            CompressedImage,
            '/yolo/detections/compressed',
            qos_profile
        )

        self.get_logger().info('YOLO ONNX Detector Node initialized')

    def preprocess(self, image):
        """Preprocess image for YOLO model"""
        # Resize image to input size
        input_img = cv2.resize(image, (self.input_size, self.input_size))

        # Convert BGR to RGB
        input_img = cv2.cvtColor(input_img, cv2.COLOR_BGR2RGB)

        # Normalize to [0, 1] and transpose to (1, 3, H, W)
        input_img = input_img.astype(np.float32) / 255.0
        input_img = np.transpose(input_img, (2, 0, 1))
        input_img = np.expand_dims(input_img, axis=0)

        return input_img
```

```
    def postprocess(self, outputs, original_shape):
        # Process YOLO outputs to get bounding boxes
        predictions = outputs[0]

        # predictions shape: (1, 84, 8400) for YOLOv8
        # Transpose to (8400, 84)
        predictions = np.squeeze(predictions).T

        # Extract boxes and scores
        boxes = predictions[:, :4]
        scores = predictions[:, 4:]

        # Get class with highest score for each detection
        class_ids = np.argmax(scores, axis=1)
        confidences = np.max(scores, axis=1)

        # Filter by confidence threshold
        mask = confidences > self.conf_threshold
        boxes = boxes[mask]
        confidences = confidences[mask]
        class_ids = class_ids[mask]

        # Convert from center format to corner format
        # and scale to original image
        orig_h, orig_w = original_shape[:2]
        scale_x = orig_w / self.input_size
        scale_y = orig_h / self.input_size

        boxes_xyxy = []
        for box in boxes:
            x_center, y_center, width, height = box
            x1 = int((x_center - width / 2) * scale_x)
            y1 = int((y_center - height / 2) * scale_y)
            x2 = int((x_center + width / 2) * scale_x)
```

```
            y2 = int((y_center + height / 2) * scale_y)
            boxes_xyxy.append([x1, y1, x2, y2])

        # Apply Non-Maximum Suppression
        if len(boxes_xyxy) > 0:
            indices = cv2.dnn.NMSBoxes(
                boxes_xyxy,
                confidences.tolist(),
                self.conf_threshold,
                self.iou_threshold
            )

            if len(indices) > 0:
                indices = indices.flatten()
                boxes_xyxy = [boxes_xyxy[i] for i in indices]
                confidences = confidences[indices]
                class_ids = class_ids[indices]
            else:
                boxes_xyxy = []
                confidences = np.array([])
                class_ids = np.array([])

        return boxes_xyxy, confidences, class_ids

    def draw_detections(self, image, boxes, confidences, class_ids):
        # Draw bounding boxes and labels on image
        for box, conf, cls_id in zip(boxes, confidences, class_ids):
            x1, y1, x2, y2 = box

            # Draw rectangle
            color = (0, 255, 0)  # Green
            cv2.rectangle(image, (x1, y1), (x2, y2), color, 2)

            # Draw label
```

```
            label = f'{self.class_names[cls_id]}: {conf:.2f}'
            label_size, _ = cv2.getTextSize(
                label, cv2.FONT_HERSHEY_SIMPLEX, 0.5, 2)

            # Draw label background
            cv2.rectangle(image, (x1, y1 - label_size[1] - 10),
                          (x1 + label_size[0], y1), color, -1)

            # Draw label text
            cv2.putText(image, label, (x1, y1 - 5),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 0), 2)

        return image

    def image_callback(self, msg):
        try:
            # Convert compressed image to OpenCV format
            np_arr = np.frombuffer(msg.data, np.uint8)
            frame = cv2.imdecode(np_arr, cv2.IMREAD_COLOR)
            original_shape = frame.shape

            # Preprocess image
            input_img = self.preprocess(frame)

            # Run inference
            outputs = self.session.run(
                self.output_names, {self.input_name: input_img})

            # Postprocess outputs
            boxes, confidences, class_ids = self.postprocess(
                outputs, original_shape)

            # Draw detections
            annotated_frame = self.draw_detections(
```

```
                frame.copy(), boxes, confidences, class_ids)

            # Log detections
            if len(boxes) > 0:
                detection_info = []
                for conf, cls_id in zip(confidences, class_ids):
                    label = self.class_names[cls_id]
                    detection_info.append(f"{label}: {conf:.2f}")

                self.get_logger().info(
                    f'Detected: {", ".join(detection_info)}')

            # Publish annotated image
            annotated_msg = CompressedImage()
            annotated_msg.header = msg.header
            annotated_msg.format = "jpeg"
            annotated_msg.data = np.array(
                cv2.imencode('.jpg', annotated_frame)[1]).tobytes()
            self.detection_publisher.publish(annotated_msg)

            # Show window if enabled
            if self.show_window:
                cv2.imshow('YOLO ONNX Detection', annotated_frame)
                cv2.waitKey(1)

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

    def destroy_node(self):
        if self.show_window:
            cv2.destroyAllWindows()
        super().destroy_node()
```

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

    try:
        rclpy.spin(node)
    except KeyboardInterrupt:
        pass
    finally:
        node.destroy_node()
        rclpy.shutdown()

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

6. Modify the setup.py's "console_scripts" by adding this line inside the bracket:

```
'yolo_detector = cv_robotics_tutorial.yolo_onnx_detector:main'
```

```
    entry_points={
        'console_scripts': [
            'image_publisher = cv_robotics_tutorial.image_publisher:main',
            'qr_scanner = cv_robotics_tutorial.qr_scanner:main',
            'yolo_detector = cv_robotics_tutorial.yolo_onnx_detector:main'
        ],
    },
```

7. Open a new terminal and build the package

```
$ cd ~/cv_ws
$ colcon build --packages-select cv_robotics_tutorial
$ source install/setup.bash
```

8. Open a new terminal and run the image_publisher node

```
$ cd ~/cv_ws
$ source install/setup.bash
$ ros2 run cv_robotics_tutorial image_publisher
```

9. Open a new terminal and run the yolo_detector node

```
$ cd ~/cv_ws

$ source install/setup.bash

$ ros2 run cv_robotics_tutorial yolo_detector
```







Acknowledgement

1. Micro-ROS Project. https://micro.ros.org/

2. QR Code Generator. https://cli.im/

3. OpenCV. https://opencv.org/

4. YOLO. https://docs.ultralytics.com/

References

1. Micro-ROS for Arduino GitHub project.
https://github.com/micro-ROS/micro_ros_arduino

2. ROS2 Jazzy Tutorial.
https://docs.ros.org/en/jazzy/index.html