深度学习

问题

我现在打算用yolo模型来做巡线到十字路口根据数字选择分支，但我不大会，该怎么部署这个模型

回答

好的，当然。这是一个使用 Markdown 格式化后的版本，结构更清晰，便于阅读。

这是一个非常棒且非常典型的机器人综合项目，它结合了传统的计算机视觉（巡线）和深度学习（目标检测），是学习和实践的绝佳案例。

我们不建议自始至终都只用一个复杂的YOLO模型来完成所有事情（包括巡线和数字识别），因为这样效率很低，而且调试困难。

我们推荐一种更稳定、更高效的**混合策略**：

- **平时**：使用简单、快速的OpenCV颜色处理方法进行巡线。
- **遇到十字路口时**：**激活**YOLO模型，专门用来识别数字，并根据结果做出决策。

模型部署与开发分步指南

整个过程分为三大步：**1. 模型训练（在PC上）** -> **2. 模型转换（在PC上）** -> **3. 板端部署（在机器人上）**。

第一步：数据准备与模型训练（在PC上完成）

您的首要任务是训练一个能认识数字的YOLO模型。

1. **收集数据**：
   - 在您要比赛或运行的场地上，用机器人的摄像头**拍摄大量包含数字（例如 "1", "2", "3"）的图片**。
   - **关键**：图片要包含各种角度、光照条件、距离远近、甚至轻微模糊的情况，数据越多，模型越鲁棒。至少为每个数字类别准备100-200张图片。
2. **标注数据**：

- 使用标注工具（如 `labelimg` 或在线的 `CVAT`、`Roboflow`）为您的图片打上标签。
- 操作很简单：在每张图片的数字周围画一个框（bounding box），并给它指定一个类别（例如，框住数字"1"，就给它打上"1"的标签）。

3. **选择并训练模型**：
   - **选择模型**：为了在边缘设备上流畅运行，请选择轻量级的YOLO模型，例如 **YOLOv5s** 或 **YOLOv8n**。
   - **开始训练**：使用您标注好的数据集，在带有GPU的电脑上开始训练。YOLOv5和YOLOv8都有非常成熟的开源代码库，您只需要按照它们的官方文档指引，配置好数据集路径，就可以一键开始训练。
   - **训练结果**：训练完成后，您会得到一个模型权重文件，通常是 `.pt` 格式。这个文件就是您定制的、能认识数字的AI模型。

第二步：模型转换（在PC上完成）

这一步是将您在PC上训练好的 `.pt` 模型，转换成能在地平线BPU上高效运行的格式。

1. **准备工具**：下载并安装地平线官方的 **“天工开物”（Togetheo）AIoT DDK** 开发工具包。
2. **执行转换**：使用DDK中提供的模型转换工具，对您的 `.pt` 文件进行转换和**量化**。这个过程会生成一个后缀为 `.bin` 或 `.hbm` 的文件。
   - 这个过程是整个部署流程中最具技术挑战性的一环，您需要**严格遵循地平线官方的模型转换文档**来操作。

第三步：板端部署（机器人上的代码）

这是项目的核心，我们需要编写一个**状态机（State Machine）**来控制机器人的行为。

1. **定义机器人的三种状态：**
   - `FOLLOWING_LINE` (正在巡线)
   - `AT_CROSSROADS` (到达路口，准备识别)
   - `TURNING` (正在转弯)
2. **编写代码逻辑** 在您的ROS2节点中（可以基于之前的`follower`节点修改），`image_callback` 函数需要根据当前状态执行不同操作。

   Python

```
# 这是一个简化的伪代码结构
class AutonomousNavigator(Node):
    def __init__(self):
        # ... 此处进行各种初始化 ...
        super().__init__('autonomous_navigator')
        self.state = "FOLLOWING_LINE"
```

```
        # 在这里加载转换好的 .hbm 模型
        self.yolo_model = bpu_infer_lib.Infer(...)
        self.yolo_model.load_model("您的数字识别模型.hbm")
        # ... 其他初始化 ...

    def image_callback(self, msg):
        image = self.bridge.imgmsg_to_cv2(msg, 'bgr8')

        if self.state == "FOLLOWING_LINE":
            # --- 执行OpenCV颜色巡线 ---
            # 使用您之前写的基于颜色阈值的巡线代码。
            # 它简单、快速、可靠。

            # --- 判断是否到达十字路口 ---
            # 判断条件可以是：
            # 1. 检测到的黄线突然消失。
            # 2. 检测到一条横向的黑线（路口停止线）。
            if self.is_at_crossroads(image):
                self.get_logger().info("到达十字路口，切换到识别状态！")
                self.stop_robot() # 让机器人停下或减速
                self.state = "AT_CROSSROADS"

        elif self.state == "AT_CROSSROADS":
            # --- 在这个状态下，激活YOLO模型 ---
            # 1. 对图像进行预处理（缩放、填充等），使其符合模型输入要求。
            # 2. 调用 self.yolo_model.forward() 进行推理。
            # 3. 解析模型的输出，找到识别到的数字和它的置信度。

            recognized_number = self.run_yolo_inference(image)

            if recognized_number is not None:
                self.get_logger().info(f"成功识别数字: {recognized_number}，
准备转弯！")
                self.target_direction =
self.decide_direction(recognized_number)
                self.state = "TURNING"

        elif self.state == "TURNING":
            # --- 根据识别结果执行转弯动作 ---
            # 例如：
```

```
            # if self.target_direction == "left":
            #     self.turn_left()
            # elif self.target_direction == "right":
            #     self.turn_right()

            # 在转弯动作完成后（可以通过延时或里程计判断）
            self.get_logger().info("转弯完成，恢复巡线状态！")
            self.state = "FOLLOWING_LINE"
```

3. **调试建议**
   - **分步测试**：先确保您的纯OpenCV巡线模块能稳定工作。再单独测试YOLO识别模块，看看它能否在您把数字卡片放到摄像头前时正确识别。
   - **打印日志**：在代码的关键位置（如状态切换、识别出数字时）使用 `self.get_logger().info()` 打印日志，方便您实时了解机器人当前的状态。
   - **可视化**：使用 `rqt_image_view` 查看机器人摄像头传回的图像，以及您处理后（画上检测框）的图像，这对于调试至关重要。

这个方案将一个复杂问题分解成了多个简单、可控的子任务，是实现您目标的稳健路径。祝您项目顺利！