프로젝트 개요

1. **Yolov8 기반 데이터 수집/학습/배포**
    - 감시용 데이터 수집 및 라벨링
    - Yolov8 기반 오브젝트 검출(Object Detection)
2. **Flask를 이용한 웹서버 구축**
3. **SQLite3 데이터베이스 구축 및 연동**
4. **Jetson Nano 기반 카메라 인식 시스템 구축**
5. **감시 시스템 통합 구현 및 Jetson Nano 기반 물체 추적 기능 구현**

프로젝트 시나리오

- **caller(rc카)**가 위험 상황이라고 판단되면 **standing_camera**를 통해 Yolov8으로 객체를 인식(detecting)한다.
- Protector(**turtlebot3**)가 먼저 caller의 위치로 이동한다.
- 이후 turtlebot3에 달린 **amr_camera**를 통해, caller(= rc카)를 일정 거리를 유지하며 목적지까지 동행(추적)하는 것이 최종 목표이다.

문제 제기 및 해결 아이디어

- **강사님 요구사항**: rc카 옆에 있는 dummy를, 같은 클래스(caller)로 묶어서 학습하라는 것
  → “rc카와 dummy를 같은 클래스로 묶으면 두 객체는 어떻게 구별해야 할까?”라는 의문이 생김.
- **해결 방법**:
    - **standing_camera 코드**에 ID 번호를 부여하여 GUI와 연동
        - 예: standing_camera에서 caller(class)로 인식된 객체가 2개 있다면, ID 1과 ID 2를 부여
        - GUI에서 ‘1번’ 객체를 선택하면 turtlebot3가 caller(ID=1)를 인식 후 해당 방향으로 이동
    - standing_camera의 역할이 끝난 뒤에는 **amr_camera**가 caller를 계속 추적
        - rc카(=caller)를 잘 인식하고 따라가도록 Yolov8 학습을 꼼꼼히 진행

데이터 수집 및 학습 과정

- **데이터 준비**:
    - rc카와 dummy 사진 약 100장 직접 촬영
    - 데이터 증강(augmentation)으로 추가 900장 생성
    - 총 1000장 규모로 Yolov8 학습 진행
- **학습 설정**:
    - Epoch: 100
    - Batch size: 32
    - Google Colab(T4 GPU) 사용
    - 최종적으로 **standing_best.pt**와 **amr_best.pt** 파일 획득
    - 95에포크 시점부터 성능이 잘 나왔으며, 과적합 우려가 있을 정도로 정확도가 높았음
        - 하지만 에포크보다 먼저(95에포크) 학습이 수렴한 것을 보고, 선택한 파라미터(100에포크/32배치)가 오히려 적절했다고 판단

Jetson Nano 환경에서의 구현

- **amr_camera 사용**:
    - turtlebot3에 '핫스팟' 설정을 이용
    - .pt 파일을 바로 쓰면 문제가 발생하여, **.engine 파일**로 변환 후 사용
        - (이유에 대해선 TensorRT 변환이나 Jetson 환경과의 호환성 문제로 추정)
- **파일 전송 명령어**
    - 예:

      ```
      bash
      CopyEdit
      scp /home/seungroknam/Downloads/inteligence2/exproject15.py
      rokey3@192.168.1.3:~
      ```

    - 위 명령어로 PC에서 Jetson Nano로 파일을 전송

마무리

- 프로젝트 첫날, 브레인스토밍을 통해 세부적인 프로젝트 구조를 잡아본 것은 처음이다.
- Yolov8 학습 과정부터 Flask 서버, DB 연동, Jetson Nano 시스템 구현까지 모두 통합하여, 최종적으로 caller(rc카)와 dummy를 같은 클래스 처리 후 ID 분류를 통해 turtlebot3가 caller를 정확히 추적하도록 하는 것이 주요 목표다.