[README] ROS Settings & Autonomous Driving

순번	115
구분	Basic
개발 상태	완료
front 연결	완료

자율주행 구현

하드웨어 및 개발 환경

- **메인 보드**: Jetson Orin Nano
  - JetPack 5.1.3
  - Ubuntu 20.04 (기본 제공)
  - Python 3.8
- **추가 라이브러리 및 환경**
  - OpenCV 4.10.0
  - PyTorch 2.1.0a
  - CUDA, cuDNN, TensorRT(JetPack 설치 시 포함)

> Note: 위 라이브러리들의 설치 단계는 본 매뉴얼에서 별도로 다루지 않음.

ROS Noetic 환경 구성

- **ROS Noetic 설치**
  - `sudo apt-get update` 후, `ros-noetic-desktop-full` 패키지 설치
  - `rosdep`, `rosinstall` 등 필수 툴 설치 후 `sudo rosdep init && rosdep update` 수행

- **환경 변수 등록**
  - `echo "source /opt/ros/noetic/setup.bash" >> ~/.bashrc`
  - `source ~/.bashrc`
- **Catkin 워크스페이스 생성**
  - `mkdir -p ~/catkin_ws/src && cd ~/catkin_ws && catkin_make`
  - `echo "source ~/catkin_ws/devel/setup.bash" >> ~/.bashrc && source ~/.bashrc`
- **의존 패키지 설치**
  - `rosdep install --from-paths src --ignore-src -r -y`
  - 부족한 ROS 패키지나 의존 라이브러리를 자동으로 설치

워크스페이스 구조 및 주요 패키지

- **backoff_recovery**
  - ROS Navigation 스택 내 `move_base`의 Recovery Behavior 플러그인
  - 장애물 등 특수 상황에서 로봇을 후진하여 회피
- **hector_slam**
  - 라이다 기반 실시간 SLAM
  - 오도메트리 센서(엔코더 등) 없이도 맵 생성 가능
- **hybrid_astar_planner**
  - 전진·후진을 고려한 하이브리드 A* 경로 계획 알고리즘
  - 복잡한 지형에서도 효율적 경로 생성
- **lidar_camera_calibration**
  - LiDAR와 카메라 간 좌표계(Extrinsic) 정렬을 위한 캘리브레이션 도구
  - 체커보드나 패턴을 이용한 자동/반자동 스크립트 포함
- **my_slam**
  - 커스텀 SLAM/Navigation 설정 및 런치 파일을 담은 패키지로 추정
  - 예: `my_slam/launch/navigation.launch`에서 핵심 노드 일괄 실행 가능
- **robot_motor_controller**
  - DC 모터 및 서보모터 제어 로직 담당

- `/cmd_vel` 등 토픽 수신 후 실제 PWM 제어 수행

- **`ydlidar_ros_driver`**
  - YDLiDAR 계열(예: X4) 센서를 위한 ROS 드라이버
  - `/scan` 토픽 퍼블리시
- **`yolo_pkg`**
  - 딥러닝 기반 객체인식 기능을 담당하는 모듈로 추정
  - Torch, OpenCV 등을 활용해 영상/라이다 데이터 처리 가능

---

udev 규칙 설정 (시리얼/USB 포트 고정)

- **이유**: 라이다나 모터 드라이버 등 시리얼(USB) 장치를 여러 개 사용할 경우 `/dev/ttyUSB0`, `/dev/ttyUSB1` 등이 재부팅마다 달라질 수 있음
- **설정 예시**
  1. `sudo nano /etc/udev/rules.d/99-ydlidar.rules`
  2. 아래 내용 작성 (Vendor/Product ID는 실제 장치에 맞게 수정)

     ```
     SUBSYSTEM=="tty", ATTRS{idVendor}=="XXXX", ATTRS{idProduct}=="YYYY", MODE:="0666", SYMLINK+="ydlidar"
     ```

  3. `sudo udevadm control --reload-rules && sudo udevadm trigger`로 반영
  4. 재부팅 후 `/dev/ydlidar`로 고정된 포트명을 사용할 수 있음
- **퍼미션 설정**
  - 시리얼 포트 접근을 위해 `dialout` 그룹에 사용자 추가
    - `sudo adduser $USER dialout`
  - 혹은 `sudo chmod 666 /dev/ttyUSB0` 등으로 임시 권한 부여(재부팅 시 리셋됨)

하드웨어 연결 (DC 모터·서보모터·라이다)

- **DC 모터 & 서보모터**
  - 후륜(DC 모터, 엔코더 미장착) → 속도 제어
  - 전륜(서보모터) → 조향 제어

    - 연결 방식: PWM 보드 또는 모터 드라이버
- **라이다 (YDLiDAR X4)**
    - USB 포트로 Jetson Orin Nano와 연결
    - `ydlidar_ros_driver` 패키지 통해 `/scan` 토픽 발행
- **테스트 방법**
    - 라이다: `rostopic echo /scan`으로 데이터 유입 여부 확인
    - 모터 제어: `/cmd_vel` 퍼블리시 후 실제 모터 회전 상태 점검

실행 방법

1. **워크스페이스 빌드 후 환경설정**
    - `cd ~/catkin_ws && catkin_make`
    - `source devel/setup.bash`
2. **런치 예시**
    - `roslaunch my_slam navigation.launch`
        - hector_slam / hybrid_astar_planner / move_base 등 연동 실행
        - 필요 시 `backoff_recovery` 플러그인 활성화(예: `move_base` 파라미터 yaml 설정)
3. **TF(Transform) 트리 확인**
    - `rqt_tf_tree` 등을 사용해 `map → odom → base_link` 연결 여부 확인
    - `camera_link`, `laser_frame` 등 센서별 TF가 올바르게 설정되어야 SLAM 및 Object Detection 정확도 향상
4. **RViz 2D Nav Goal 선택**
    - GUI를 활용해 목적지 선택 시 경로 생성 및 추종 시작
5. **Autonomous Exploration**
    - 독거노인 페이지의 주행버튼 or 조이스틱의 X버튼 선택 시 자율 탐사 시작

런치 파일 개요

- `my_slam.launch` :

    - 정적 TF 퍼블리셔, YDLidar X4 드라이버, Hector SLAM, (옵션) RViz 실행
    - 주로 SLAM 관련 노드와 라이다 센서 초기화를 담당
- `navigation.launch`:
    - SLAM 및 모터 컨트롤러 포함 실행
    - ROS Navigation Stack(`move_base`), 조이스틱 제어, YOLO 객체인식, 자율 탐색 등 포괄적 기능 구동
    - 전체 시스템 통합 및 시각화를 용이하게 함

---

1. `my_slam.launch` 파일

1.1 정적 TF 퍼블리셔

- **형식**: `<node pkg="tf" type="static_transform_publisher" ...>`
- **역할**:
    - 로봇 좌표계(`base_link` 또는 `base_footprint`)와 라이다 센서 좌표계(`laser_frame`) 간 상대적 위치·자세를 고정
    - 로봇 구동 시 TF가 일정하게 유지될 수 있도록 퍼블리시

1.2 YDLidar X4 실행

- **형식**: `<include file="$(find ydlidar_ros_driver)/launch/X4.launch" />`
- **역할**:
    - YDLidar X4 드라이버 노드 실행
    - `/scan` 토픽으로 라이다 스캔 데이터 퍼블리시

1.3 Hector SLAM 노드

- **형식**: `<node pkg="hector_mapping" type="hector_mapping" name="hector_mapping" ...>`
- **역할**:
    - 라이다 데이터만으로 실시간 SLAM 수행
    - 로봇의 위치 추정 및 맵(`map`) 생성·갱신
- **주요 파라미터**
    - `map_frame`: 맵 좌표계 명(기본 `"map"`)

- `base_frame`: 로봇 기준 프레임(보통 `"base_footprint"`)
- `odom_frame`: 오도메트리 프레임(헥터에서는 `"base_footprint"` 사용 가능)
- `scan_topic`: 라이다 데이터 토픽(`/scan`)
- `map_update_distance_thresh`, `map_update_angle_thresh`: 맵 갱신 최소 거리지표/각도지표
- `map_resolution`: 맵 해상도(ex. 0.05 → 5cm)
- `pub_odometry`: 헥터 SLAM이 `/odom`을 추가 발행할지 여부

1.4 RViz 실행 (옵션)

- **형식**: `<node pkg="rviz" ...>`
- **역할**:
  - Hector SLAM에서 생성되는 맵, 라이다 스캔, 로봇 위치를 시각적으로 모니터링
  - 필요 시 주석 해제 후 활용

---

2. `navigation.launch` 파일

2.1 SLAM 및 모터 컨트롤러 포함 실행

- **형식**:
  - `<include file="$(find my_slam)/launch/my_slam.launch" />`
    - Hector SLAM 및 라이다 드라이버 자동 실행
  - `<include file="$(find robot_motor_controller)/launch/motor_controller.launch" />`
    - DC 모터, 서보모터 노드 실행
    - `/cmd_vel` 구독 후 PWM 또는 시리얼 통신을 통해 실제 구동 명령 수행

2.2 move_base 노드

- **형식**: `<node pkg="move_base" type="move_base" name="move_base" ...>`
- **역할**:
  - ROS Navigation Stack의 중심
  - 전역·지역 Costmap 구성 및 경로 계획 수행
  - 로봇을 지정 목표 지점까지 이동
- **주요 파라미터**

- **공통 설정 파일:** `costmap_common_params.yaml`, `global_costmap_params.yaml`, `local_costmap_params.yaml`, `dwa_local_planner.yaml` 등
- `recovery_behavior_enabled`: 후진·회전 등을 통한 장애물 회피 허용 여부
- `clearing_rotation_allowed`: 회전으로 장애물 클리어 가능 여부
- `controller_frequency`: 초당 로컬 플래너 업데이트 빈도
- `controller_patience`: 유효한 제어 명령 부재 시 대기 시간
- `shutdown_costmaps`: 목표 도달 후 costmap 종료 여부
- `base_global_planner`: 전역 플래너(예: `hybrid_astar_planner/HybridAStarPlanner` or `navfn/NavfnROS`)
- `base_local_planner`: 로컬 플래너(예: `dwa_local_planner/DWAPlannerROS`)

2.3 조이스틱 노드 및 Joy Teleop

- **형식:**
  - `<node pkg="joy" type="joy_node" ...>`
    - Joystick 입력 `/joy` 토픽 발행
  - `<node pkg="robot_motor_controller" type="joy_teleop.py" ...>`
    - `/joy` 구독 후 축(`axis_linear`, `axis_angular`), 버튼(`mode_button`, `stop_button`) 정보를 `/cmd_vel`에 반영
    - `priority` 옵션을 통해 자율주행 명령보다 조이스틱 명령을 우선 적용 가능

2.4 YOLO Pose 노드

- **형식:** `<include file="$(find yolo_pkg)/launch/zz.launch" />`
- **역할:**
  - YOLO 기반 객체 검출
  - 카메라 토픽 처리 후 2D/3D 위치 추정 가능

2.5 주행 모드 매니저 / 자율 탐색 / 사람 추적 노드

- **예시 노드:**
  - `<node pkg="my_slam" type="driving_mode_manager.py" ...>`: 주행 모드 관리(수동, 자율, 추적 등)

- `<node pkg="my_slam" type="autonomous_exploration.py" ...>`: 목적지 없이 맵 전체를 탐색하는 로직
- `<node pkg="my_slam" type="person_following.py" ...>`: 특정 인물 추적 로직(이미지·라이다 기반 추적)

2.6 소켓 클라이언트 노드

- **형식**: `<node pkg="my_slam" type="socket_client.py" ...>`
- **역할**:
  - 외부 서버(예: 70.12.247.214:12345)에 소켓 통신
  - 로봇 상태·센서 데이터 전송 및 명령 수신

2.7 RViz 시각화

- **형식**: `<node pkg="rviz" type="rviz" ...>`
- **역할**:
  - `d $(find my_slam)/rviz/nav.rviz` 파일 로드
  - SLAM 맵·경로·객체 검출 결과 등을 한 화면에서 모니터링

실행 순서 요약

1. **빌드 및 환경 설정**
   - `cd ~/catkin_ws && catkin_make && source devel/setup.bash`
2. **런치 파일 실행**
   - `roslaunch my_slam navigation.launch`
   - SLAM, 네비게이션, 모터 컨트롤러, YOLO, 자율 탐색, 조이스틱, 소켓 통신 등 통합 구동
3. **RViz 확인**
   - 맵, 로봇 위치, 장애물, 경로, 인식 객체 등 모니터링
   - 2D Goal 지정 또는 조이스틱으로 주행 모드 전환

> 위 두 런치 파일(my_slam.launch, navigation.launch)로 라이다 기반 SLAM(hector_mapping), Navigation Stack(move_base), 다양한 사용자 노드(자율 탐색·추적·소켓 통신)를 일괄 관리 가능. TF 트

> 리, Costmap 설정, 모터 구동 및 네트워크 연결 등을 종합 검증하여 안정적 로봇 운용을 구현할 수 있음.