제8회 국민대학교 자율주행 경진대회
자율주차 관련 개발계획서

□ 개발 개요

□ 서론 (Introduction)

■ 1. 자율주차 기술에 대한 관심 증가

최근 자율주행 기술의 발전에 따라 차량의 주차 자동화 기능에 대한 관심이 높아지고 있다.
특히 좁은 공간에서의 정밀한 주차 수행 능력은 자율주행 시스템의 완성도를 평가하는 핵심 요소 중 하나로 여겨진다.이를 위해 차량 주변 환경에 대한 정확한 인식과 정밀한 위치 추정 기술이 필수적이며, 다양한 센서 및 인식 기법이 연구되고 있다.

■ 2. 개발 목표 및 적용 기술

이러한 맥락에서 본 개발계획서는 AprilTag 기반 시각 마커 시스템을 활용하여 차량과 주차구역 사이의 상대 위치(Position) 및 자세(Orientation) 를 실시간으로 파악하고, 이를 바탕으로 경로를 생성(Path Planning) 하며, 생성된 경로를 따라 차량이 안정적으로 주차할 수 있도록 하는 경로 추종(Path Following) 알고리즘을 구현하는 과정을 다룬다.

■ 3. AprilTag의 기술적 특성과 장점

AprilTag는 낮은 계산 비용에도 불구하고 높은 인식 정확도를 제공하며, 실내외 다양한 환경에서도 강인하게 작동하는 특성을 가진 시각 마커이다. 이를 차량 자율주차 시스템에 적용함으로써, 기존의 GPS 기반 위치 인식이 어려운 공간이나 센서 오차가 큰 상황에서도 신뢰도 높은 주차 성능을 기대할 수 있다.

■ 4. 개발 계획서의 구성 및 방향성

본 계획서에서는 이러한 AprilTag 기반 주차 시스템의 구조, 사용 알고리즘, 구현 방법 및 전체 개발 일정 등을 구체적으로 제시하며, 실제 자율주차 기술에 적용 가능한 수준의 정밀한 시스템을 개발하고자 한다.

□ 프로젝트 조직 (Project Organizaion)

본 프로젝트는 효율적인 개발 진행과 체계적인 업무 분담을 위해 다음과 같이 기능 중심의 조직 체계를 구성한다. 각 구성원은 자신의 역할에 책임을 가지며 유기적인 협업을 통해 전체 시스템의 통합 및 성능 최적화를 목표로 한다.

○ 팀 역할 체계

-프로젝트 총괄 (PM): 전체 프로젝트의 일정 관리 및 개발 방향성을 조율하며, 팀 내부 및 외부 커뮤니케이션을 총괄하는 역할을 담당한다. (1명)

-알고리즘 개발: AprilTag 기반 인식 알고리즘을 구현하고, 이를 통해 차량과 주차구역 간의 상대 위치 및 자세를 추정하는 로직을 개발한다. (1명)

-경로 계획 및 제어: 차량의 현재 위치를 기반으로 주차 목표점까지의 경로를 생성하는 알고리즘(Path Planning)과, 이를 따라 안정적으로 주차를 수행하기 위한 경로 추종 제어(Path Following) 알고리즘을 설계 및 구현한다. (1명)

-시스템 통합: 알고리즘, 경로 계획, 모터 제어 등 각 기능을 ROS 환경에서 하나의 플랫폼으로 통합하고 노드 연동 및 디버깅 수행 (1명)

-하드웨어 운영: 1/10 자율주행 차량 플랫폼 구성 및 하드웨어 세팅(카메라, 센서 장착 등)을 수행하며, 실제 주차 환경 구축 및 필드 테스트를 지원한다. (1명)

○ 팀 간 협력 구조

-프로젝트 총괄은 모든 팀의 진행 상황을 주기적으로 점검하고, 개발 우선순위 및 이슈 해결 방향을 조율한다.

-알고리즘 개발팀과 경로 계획 및 제어팀은 밀접하게 협업하며, 인식 결과를 경로 생성에 활용하기 위한 데이터 인터페이스를 설계한다.

-시스템 통합팀은 전체 기능이 하나의 플랫폼에서 통합되어 동작할 수 있도록 ROS 환경에서 노드 연동 및 디버깅을 진행한다.

-하드웨어 운영팀은 실험 환경을 유지 관리하고, 실주행 테스트 시 차량의 상태를 점검하며 이상 여부를 모니터링한다.

이러한 조직 구조를 바탕으로 각 구성원이 자신의 역할에 집중하며, 상호 간 협력을 통해 정확하고 신뢰성 있는 자율주차 시스템 개발을 추진한다.

□ 기술적 접근 (Technical Approach)

○문제 정의 및 분석

-단일 USB 카메라(OV2710 센서, 170도 어안렌즈)로부터 640x480 60fps 영상 데이터를 ROS2 토픽(/image_raw)으로 수신.

-주차구역 벽면에 부착된 AprilTag를 인식하여 차량의 상대 위치 및 자세 추정.

-추정한 위치 정보를 기반으로 경로 계획 수립 및 차량 제어(조향 각도, 속도) 명령을 ROS2 토픽(/xycar_motor)으로 발행.

-VESC 기반 모터 제어기로부터 조향 및 주행 제어 신호 생성, 차량의 주차구역 진입 및 경로 추종 수행.

-ROS2 Humble (Ubuntu 22.04) 기반 경량 시스템 구현, AMD CPU 환경에서 실시간 처리를 목표로 함.

○해결 방법 도출 및 선정

문제 영역	후보 기술 및 방법	선정 및 이유
영상 획득 및 처리	- xycar_cam 패키지를 통해 /image_raw 토픽 구독 - cv_bridge + OpenCV로 영상 처리 -영상 처리 순서 1.그레이스케일 2.가우시안 블러 3.Canny 엣지	- ROS2 표준 토픽 기반 실시간 처리 가능 - OpenCV 기반으로 구조 단순화 및 성능 최적화 용이
AprilTag 인식 및 위치 추정	- apriltag_ros 패키지를 활용해 6자유도(pose) 추정 - PnP 알고리즘 병행 적용	- 검증된 ROS 패키지로 안정적 위치 추정 - Tag ID별 위치 맵핑으로 정확도 향상
Path Planning (경로 계획)	- A* 알고리즘으로 격자 기반 전역 경로 생성 - 주차장 맵 상에서 시작점부터 목표점까지 최단 경로 탐색	- 계산 효율이 높고 장애물 회피에 유리 - 정적 환경에서 효과적인 경로 생성 가능
Path Following (경로 추종)	- Pure Pursuit 알고리즘으로 Look-ahead point 추적 - 실시간 조향각 계산 방식 적용	- 계산이 단순해 실시간 제어에 적합 - 경로를 부드럽게 추종하며 주행 안정성 확보
모터 제어	- xycar_motor 패키지를 통해 VESC에 speed/angle 제어값 발행 - 조향 범위 ±20도, 속도 ±100 범위 제어	- ROS2 토픽 기반으로 구현 간단 - 2축(구동/조향) 제어로 정밀 제어 가능

○ROS 노드 및 토픽 설계표

노드 이름	구독 토픽	발행 토픽	메시지 타입	주요 기능	비고
xycar_cam	–	/image_raw	sensor_msgs/Image	카메라 영상 송출	USB 카메라
image_processor	/image_raw	/tag_detections	geometry_msgs/PoseStamped 또는 apriltag_msgs/AprilTagDetectionArray	AprilTag 검출 및 영상 전처리	OpenCV 사용
pose_estimator	/tag_detections	/target_pose	geometry_msgs/PoseStamped	6DOF 포즈 추정	PnP 병행
a_star_planner	/target_pose	/trajectory_path	nav_msgs/Path	A* 기반 경로 생성	Spline 적용
pure_pursuit_controller	/trajectory_path, /vehicle_pose	/xycar_motor	xycar_msgs/XycarMotor	경로 추종(Pure Pursuit) 및 모터 제어 명령 계산	–
vesc_driver	/xycar_motor	–	xycar_msgs/XycarMotor	실제 바퀴 구동 (조향/속도)	VESC 사용

○ 시스템 아키텍처 및 구현 계획

□ 카메라 노드

xycar_cam 패키지를 실행하여 /image_raw 토픽 발행
my_cam 노드에서 /image_raw를 구독하여 OpenCV 기반 전처리 수행
영상 필터링(그레이스케일, 블러, 엣지 검출) 후 AprilTag 인식 및 6DOF pose 추정 수행
인식된 Tag ID 및 상대 위치 정보는 경로 계획 노드로 전달

□ 경로 계획 노드 (Path Planning)

AprilTag 기반 위치 정보를 바탕으로 A* 알고리즘으로 전역 경로 탐색
탐색된 경로를 부드럽게 연결하기 위해 Cubic Spline 기반의 trajectory 생성
생성된 trajectory는 path following 노드로 전달되어 실제 주행 제어에 사용

□ 경로 추종 노드 (Path Following / Tracking)

경로 계획 노드에서 생성된 trajectory를 실시간으로 구독
차량 현재 위치와 경로를 비교하여 Pure Pursuit 알고리즘으로 steering angle 계산
/xycar_motor 토픽으로 조향각(angle)과 속도(speed) 명령 발행 → VESC 모터 제어 수행

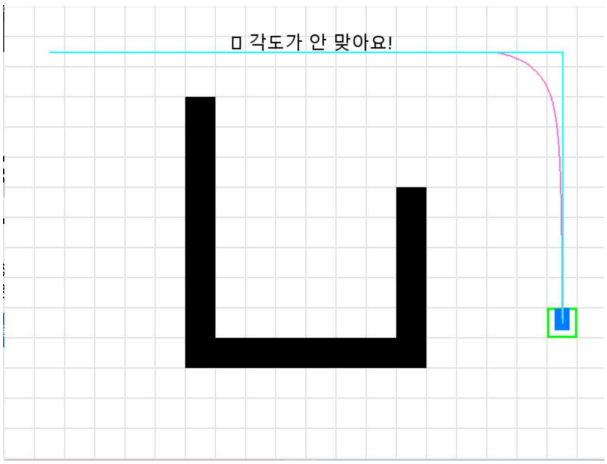
□ 하드웨어 적용

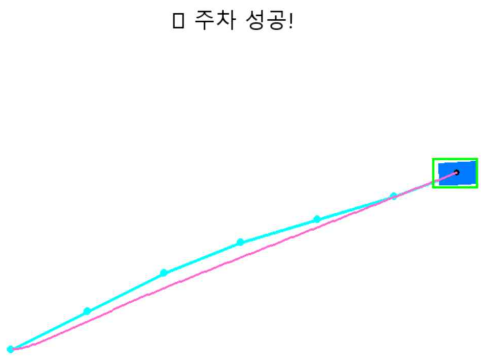
Ubuntu 22.04 + ROS2 Humble 환경에서 전체 시스템 실행
AMD 기반 CPU가 탑재된 대회 차량 내에서 구동
ROS2 런치 파일을 통해 카메라, Path Planning, Path Following, 모터 제어 노드 동시 실행 가능

□ 디버깅 및 시각화

Pygame 또는 RViz를 활용하여 다음 항목 시각화:
실시간 카메라 영상
인식된 AprilTag 위치 및 pose
A* 경로 및 spline trajectory
차량의 현재 위치와 경로 추종 상태

□ Pygame 기반 시뮬레이터 활용:
본 프로젝트에서는 실제 차량 주행 테스트 이전 단계에서 알고리즘의 논리적 정확성과 안정성을 사전 검증하기 위해 Pygame 기반 시각화 시뮬레이터를 활용하였다.
A* 알고리즘을 사용한 전역 경로 계획 로직과 Pure Pursuit 기반 경로 추종 알고리즘을 각각 Python으로 구현하였으며, Pygame 환경 내에서 시각적으로 주행 경로 생성 및 추적 결과를 확인할 수 있도록 구성하였다.
시뮬레이터 상에서는 시작점, 목표점, 장애물 등을 마우스로 직접 설정할 수 있으며, 생성된 경로가 실제 차량 경로처럼 시각적으로 재현된다.
이를 통해 알고리즘 튜닝, 회피 경로 확인, 조향 반응 테스트 등의 기능 검증을 반복 수행함으로써, 실차 테스트 전 알고리즘의 동작 정확도를 확보하였다.





(A* 알고리즘)
path = a_star(start, goal, obstacle_map)
trajectory = generate_cubic_spline(path)

추후 경로 생성 로직과 실제 차량 동작 간의 오차를 비교 분석하는 데도 유용하게 활용되었으며, 다양한 장애물 환경과 경로 조건에서도 일관된 성능을 보임을 확인하였다.
Pygame 시뮬레이터는 개발 초반 알고리즘의 안정성과 경로 생성 패턴을 반복 검증하는 데 효과적인 도구로 활용되었고, 추후 시각화 자료로도 재활용이 가능하다.

□ 일정 계획 (Schedule Plan)

본 프로젝트는 총 10주 동안 수행되며, ROS2 기반 자율주차 기능의 단계별 구현 및 통합을 목표로 한다. 각 주차별 주요 개발 및 검증 항목은 다음과 같다.

주차	주요 활동 내용
1주차	프로젝트 계획 수립, 개발 환경 세팅 (Ubuntu 22.04 + ROS2 Humble), 팀 역할 분담
2주차	카메라 `/image_raw` 토픽 수신 및 OpenCV 기반 Apriltag 인식 테스트
3주차	Apriltag 위치정보 기반 차량 Pose 추정 알고리즘 구현 및 RViz 시각화 연동
4주차	주차 공간 탐지 및 상대 좌표계 정렬 로직 개발
5주차	경로 생성 (Path Planning, 예: A*) 알고리즘 구현 및 Pygame 시뮬레이터 검증
6주차	경로 추종 (Path Tracking, 예: Pure Pursuit or Stanley) 알고리즘 구현
7주차	모터 제어 노드 개발 및 `/xycar_motor` 토픽 발행을 통한 자이카 차량 구동 실험
8주차	전체 시스템 통합 테스트 (카메라 인식 〉 위치추정 〉 경로 생성/추종 〉 주차)
9주차	성능 개선 및 경량화 테스트 (CPU 환경 최적화, ROS 노드 경량화)
10주차	시연 준비, 보고서 및 발표자료 작성, 최종 검증 및 결과 제출

□ 자원 계획 (Resource Plan)

1) 인적 자원

역할	담당 업무	인원
프로젝트 매니저 (PM)	전체 일정 조율, 문서 관리, 대회 대응	1명
ROS2 개발자	ROS 노드 구성, launch, 토픽/서비스 연동	1명
영상 처리 담당	카메라 토픽 처리, Apriltag 인식 및 디버깅	1명
경로 계획 담당	Path Planning/Tracking 알고리즘 개발	1명
하드웨어 운영 및 실험	자이카 차량 제어, 실차 주행 실험	1명

2) 소프트웨어 자원

항목	상세 내용
운영체제	Ubuntu 22.04 LTS
ROS 버전	ROS2 Humble Hawksbill
개발 언어	Python 3 (주 개발), C++ (보조 또는 고속 처리 시)
주요 라이브러리	OpenCV, CvBridge, apriltag_ros2, NumPy
시각화 도구	RViz2, rqt_graph, rqt_plot, rqt_image_view, Pygame, Foxglove Studio
시뮬레이션/로깅	rosbag2, Matplotlib (실험 데이터 후처리)
버전 관리	Git + GitHub 또는 GitLab
런치 관리	`launch/` 디렉토리 및 `.launch.py` 스크립트 기반 구성

3) 하드웨어 자원

항목	상세 스펙
모형 차량	자이카 (1/10 스케일 EV, 브러시리스 모터, 8.4V 배터리)
제어 컴퓨터	AMD CPU 기반 미니 PC (GPU 없음, 저전력 시스템)
카메라	USB UVC 지원 OV2710 카메라, 170도 어안렌즈
모터 제어기	VESC (Wheel Motor + Steering Motor 제어)
주행환경	직각형 주차공간이 포함된 실내 시험장 (AprilTag 부착)
네트워크	로컬 LAN 또는 라즈베리파이와 USB 직결 환경

□ 품질 관리 계획 (Quality Assurance Plan)

품질 관리 계획서

1) 품질 목표

- 카메라 영상 인식률 95% 이상 (Apriltag 미인식 상황 최소화)
- 계획 경로와 실제 주행 궤적의 오차 ≤ 10cm
- 차량 주차 정밀도: 주차선 내 진입 성공률 ≥ 90%
- 시스템 응답 시간: 0.5초 이내 제어 피드백 루프 동작
- ROS2 노드 및 토픽 안정성 확보 (1초 이상 토픽 미발행 시 자동 정지 포함)

2) 주요 품질 관리 항목 및 관리 기준

항목	관리 내용	확인 방법
노드 통신	ROS2 토픽 정상 송수신 여부 확인	`rqt_graph`, `ros2 topic echo` 등으로 수동 점검
카메라 인식 정확도	Apriltag 인식률 ≥ 95%	다양한 조도/각도에서 테스트 후 누락률 측정
경로 생성 정확도	장애물 회피, 주차구역 중심 정렬 정확도	RViz/Foxglove 시각화로 실시간 비교 검토
경로 추종 정확도	차량이 계획 경로를 따라 주행하는가	`rosbag2` 기록 및 실제 영상 비교 분석
모터 제어 응답성	조향 및 속도 제어 신속 반응	1초 이상 미발행 시 자동 정지 기능 테스트
시스템 안정성	CPU 사용량, 메모리 점유율 등 시스템 부하	`htop`, `rqt_runtime_monitor` 등으로 지속 모니터링

3) 테스트 계획 및 검증 절차

단계	내용	기준
단위 테스트	각 노드별 개별 기능 동작 확인 (카메라, 제어, 경로 등)	정상 동작 여부 판단
통합 테스트	전체 파이프라인 구성 후 실제 주행 시험	주차 성공률 ≥ 90%
성능 테스트	다양한 조건에서 반복 주차 테스트 (조명, 각도 등)	최소 10회 반복, 오류율 기록
예외 상황 테스트	Tag 미인식, 토픽 중단, 급선회 등 예외 상황 시나리오	안전 정지 및 재시도 동작 여부 확인

4) 지속적 품질 관리 방안

- 매 테스트 이후 `rosbag2` 기록 및 결과 분석
- `Foxglove Studio`를 통한 실시간 경로 오차 시각화 및 로그 추적
- Git 기반 형상관리와 함께 버전별 성능 리포트 유지
- 매 주 리뷰 회의 통해 이전 주 품질 이슈 피드백 반영

5) 실패 기준 (Fail Criteria)

- 시스템이 경로를 벗어나거나 주차 실패 ≥ 3회 이상 연속 발생
- 모터 제어 지연 또는 오동작 발생 ≥ 2회 이상
- ROS2 토픽 오류로 전체 시스템 정지 시

□ 위험 관리 계획 (Risk Management Plan)

위험 관리 계획서

1) 위험 관리 목표

- 시스템 장애, 실차 사고, 개발 일정 지연 등의 발생률을 최소화

- 위험 요소의 사전 예측 및 선제적 대응 방안 수립

- 각 단계별 책임자 지정 및 비상 대응 프로세스 마련2) 위험 요소 식별 및 평가

위험 요소	유형	영향도	발생 가능성	대응 전략
카메라 인식 실패	기술적	높음	보통	조명 조건 개선, 다중 Tag 배치, 인식 실패 시 정지 및 재시도 로직 구현
ROS2 노드 충돌	기술적	높음	보통	launch 구성 검토, `rclpy` 예외처리, 다중 노드 병렬 실행 테스트
토픽 누락/지연	기술적	높음	보통	1초 이상 미수신 시 정지하는 safety mechanism 유지
모터 제어 오류	하드웨어	매우 높음	낮음	조향각·속도값 검증 후 발행, 비상 정지 조건 설정
시각화 디버깅 실패	기술적	중간	보통	Foxglove, RViz, rosbag2 등 복수 시각화 수단 병행
주행 중 차량 파손	물리적	높음	낮음	테스트 환경 보호벽 설치, 제한 속도 설정
개발 일정 지연	운영	중간	중간	각 단계 완료 기준 미리 정의, 매주 리뷰 및 리스크 재조정
GPU 없음으로 인한 부하	시스템	중간	높음	영상처리 알고리즘 경량화, CPU 병렬처리 및 해상도 축소

3) 위험 대응 전략

• 예방 중심 전략 (Preventive Action):

- 코드 단위에서 예외 처리 및 로그 기록을 철저히 수행

- 주요 모듈별 단위 테스트 통과 후에만 통합

- 실내 실험장 환경 제어 (조명, Tag 위치 등)

• 모니터링 전략 (Monitoring Action):

- `ros2 topic hz`, `rqt_runtime_monitor` 등으로 실시간 모니터링

- `rosbag2` 기록 및 사후 분석 (성능/실패 요인 추적)

• 비상 대응 전략 (Fallback/Recovery):

- 인식 실패 시 차량 정지 후 재탐색 루틴 수행

- 모터/센서 오작동 시 수동 조작 전환

- 시스템 전체 정지 시 launch 파일 자동 재시작 스크립트 준비

4) 책임 분담 및 보고 체계

위험 영역	담당자	보고 방식
인식 오류 / 시각화	영상 처리 담당	테스트 후 Slack 공유, 이슈 트래커 등록
모터 및 ROS 통신	ROS 개발자	GitHub Issue 또는 위클리 리포트
전체 일정/성능	PM	주간 회의 및 품질 로그 보고

5) 주간 위험 리뷰 프로세스

- 매주 주차별 회의에서 위험 발생 사례 리뷰

- 신규 위험 식별 시 위험 목록 업데이트 및 대응자 지정

- 위협 우선순위(High → Critical)에 따라 우선 대응

□ 형상 관리 계획(Configuration Management Plan)

형상 관리는 소스 코드, 설정 파일, 노드 구조, 하드웨어 배치 등의 변경 사항을 체계적으로 관리함으로써 프로젝트 일관성과 안정성을 유지하는 데 중요한 역할을 한다. 본 프로젝트에서는 ROS2 기반 자율주차 시스템의 다양한 구성 요소에 대해 형상 관리를 적용한다.

1) 형상 관리 대상

구분	관리 항목
소프트웨어 코드	ROS2 노드 소스코드 (`.py`, `.cpp`), launch 파일, CMakeLists.txt, package.xml 등
시각화 설정	RViz 설정 파일 (`.rviz`), Pygame 시뮬레이터 파라미터
토픽/노드 구조	노드 간 통신 구조, 메시지 타입, QoS 설정 등
모터 파라미터	속도/조향각 설정값, VESC 관련 설정
카메라 설정	해상도, fps, 이미지 변환 파라미터, 카메라 캘리브레이션 결과
환경 구성 정보	ROS2 환경변수, 의존성 패키지 목록 (`requirements.txt`, `apt install`, `rosdep`)
하드웨어 배치	자이카 차량 내 장치 배치도, 센서 장착 위치 및 방향

2) 형상 관리 도구 및 체계

관리 수단	세부 내용
버전 관리 시스템	Git (GitHub 또는 GitLab) 사용, branch 전략 도입 (main/dev/feature)
이슈 추적 시스템	GitHub Issues 또는 Notion/Trello 등으로 버그 및 기능 요청 관리
커밋 규칙	기능 단위 커밋, `[node]`, `[topic]`, `[fix]` 등의 prefix 사용 권장
백업 체계	주요 milestone 시점마다 tag 생성, 로컬 + 원격 저장소 백업 병행
문서화 도구	README, Wiki, .md 파일로 각 노드 및 설정 설명 문서화
파라미터 관리	YAML 파일 또는 ROS2 parameter server에 의한 설정 자동화

3) 형상 변경 관리 절차

1. 변경 요청 발생: 팀원 또는 테스트 결과로부터 변경 요구가 발생하면 GitHub Issue 또는 내부 Wiki에 등록한다.

2. 변경 영향 분석: 다른 노드, 설정 또는 물리적 장치에 영향이 있는지 확인하고 우선순위 판단

3. 변경 수행 및 커밋: 변경 내용을 브랜치 분기 후 반영하고, 기능별 커밋 및 Pull Request(PR) 생성

4. 변경 리뷰 및 승인: 팀원 또는 담당자가 코드 리뷰를 진행하고 main 브랜치에 merge

5. 변경 문서화: 변경된 기능과 설정을 Wiki 또는 버전 기록 문서에 업데이트

4) ROS2 환경에서의 특별 고려사항

- `launch/` 폴더 내 `.launch.py` 구성 변경은 반드시 리뷰 후 반영
- 파라미터 서버 사용 시, 동일 노드 이름 중복으로 인한 충돌 방지
- `ros2 topic list`, `ros2 node list` 결과의 스냅샷 저장을 통해 시스템 구성 추적
- `rosbag2`로 실험 데이터 기록 시, 해당 버전의 코드와 함께 태그 연동

□ 유지보수 계획(Maintenance Plan)

1) 유지보수 목적

- 프로젝트 종료 후에도 안정적으로 시스템을 운용할 수 있도록 보장
- ROS2 및 Python 버전 변화, 하드웨어 교체 등에 유연하게 대응
- 시스템 확장성(다중 Tag 인식, 후진 주차 등)을 고려한 구조 유지

2) 유지보수 내용

항목	유지보수 내용	관리 방법
ROS2 패키지	주요 노드 업데이트, deprecated API 대응	Git branch로 업데이트 버전 관리
Launch 파일	센서 추가 시 재구성, 파라미터 구조화	YAML 설정 파일화 및 주석 관리
카메라/모터 설정	해상도, 제어 속도 등 환경별 보정	설정파일 버전 기록 및 적용 일자 관리
시각화 도구	RViz, Foxglove 구성 변경	설정 템플릿 (.rviz/.json) 백업
시뮬레이터	알고리즘 테스트 환경 유지	Pygame 코어 모듈 분리 및 문서화
사용 설명서	설치/실행/테스트 매뉴얼 제공	README, Wiki, Markdown 문서 관리

3) 유지보수 책임 분담

구분	담당자	방식
코드 유지	ROS 개발자	GitHub 브랜치 유지 및 PR 검토
하드웨어 유지	실험 담당자	센서 상태 점검 및 재설치 기록
문서 유지	PM	매뉴얼, 시각화 템플릿 최신화

4) 유지보수 주기

- 단기: 테스트 이후 문제 발생 시 즉각 패치 (1~2일 내)
- 중기: 기능 추가 요청 발생 시 (주 단위)
- 장기: ROS 버전 변화, 차량 플랫폼 변경 발생 시 (분기 단위)

□ 기대효과(Expected Outcomes)

1) 기술적 기대 효과

항목	내용
ROS2 실차 적용	RViz, Foxglove, rosbag 등을 활용한 실시간 자율주차 구조 구현 경험 확보
모듈화 구조 습득	카메라 → 인식 → 경로 생성 → 추종 → 제어로 이어지는 파이프라인 개발 능력 향상
경량화 기술 습득	GPU 없이도 작동 가능한 CPU 기반 최적화 알고리즘 구성
실시간 시각화 경험	디버깅 중심 RViz 외에도 Foxglove, Pygame 등 다양한 시각화 도구 실전 경험

2) 실용적 기대 효과

항목	내용
자율주행 테스트베드로 활용	자이카 플랫폼 기반 자율주행 실험에 활용 가능
교육 콘텐츠 활용	ROS2 기반 실습 자료로 재구성하여 교육 목적으로 사용 가능
기능 확장 기반 확보	후진 주차, 다중 차량 협력 등 향후 프로젝트 기반 마련
대회 및 발표 활용	기술 시연, 데모 영상 제작, 학술 포스터 제출 등에 활용 가능

3) 팀원 역량 강화

- ROS2 실전 경험 → 향후 산업용 자율주행, AMR, 로보틱스 개발 참여 가능
- Git 기반 협업과 이슈 추적 경험 → 소프트웨어 개발 실무 능력 강화
- 실차 실험 → 물리 환경에서의 예외 처리 능력 및 문제 해결 능력 향상

□ 부록(APPENDIX)

ROS2 기반 자율주차 시스템 요약 문서

A. 사용 토픽 및 메시지 타입 요약

카메라

토픽명	메시지 타입	설명
/image_raw	sensor_msgs/Image	USB 카메라 영상 데이터 스트리밍
/tag_detections	apriltag_msgs/TagDetectionArray	Apriltag 인식 결과 (ID, 위치, 회전)

모터 제어

토픽명	메시지 타입	설명
/xycar_motor	xycar_msgs/XycarMotor	조향각(`float32 angle`), 속도(`float32 speed`) 제어 명령

B. 주요 ROS2 노드 구성 예시

Nodes:
/xycar_cam # 카메라 영상 제공
/tag_detector # Apriltag 인식 처리
/path_planner # 주차 경로 생성
/path_tracker # 경로 추종 및 제어
/xycar_motor # 모터 제어 노드

C. 실험 환경 구성

실내 주차 트랙 제작 (1/10 스케일, 직각 주차 공간 포함)

Apriltag 부착: 16h5 태그 사용, 각도 및 거리별 다수 설치

조명 조건: LED 고정광 사용, 그림자 최소화 조정

노트북 환경:

- Ubuntu 22.04 LTS
- ROS2 Humble 설치
- AMD Ryzen CPU, 16GB RAM, GPU 없음
- RViz2, Foxglove Studio, rqt 시리즈 설치

D. 주요 명령어 모음

카메라 노드 실행

ros2 launch xycar_cam xycar_cam.launch.py

Apriltag 인식 노드 실행

ros2 run apriltag_ros2 apriltag_node

경로 생성 및 추종 노드 실행

ros2 run path_planner planner_node

ros2 run path_tracker tracker_node

모터 제어 실행

ros2 launch xycar_motor go.launch.py

Foxglove Studio 연결

ros2 launch foxglove_bridge foxglove_bridge.launch.py

F. 참고 자료

- ROS2 공식 문서
- Apriltag 공식 GitHub: https://github.com/AprilRobotics/apriltag
- Foxglove Studio: https://foxglove.dev/
- 자이카 매뉴얼 및 기술 사양 (제공된 PDF 참조)