**SW 개발계획서**

**- 목 차 -**

1. 개요
   1. 프로젝트 명
   2. 프로젝트 목표
   3. 주요 기능
   4. 기대 효과
2. 팀 구성 및 역할분담
   1. 팀 구성
   2. 역할 분담
   3. 의사소통 방법
3. 기술적 접근
   1. 문제 정의
   2. 기술적 목표
   3. 사용 기술 및 구현계획
4. AprilTag 분석
   1. 사용 기술
   2. 칼리브레이션
   3. AprilTag 라이브러리 사용
   4. pose 행렬 분석
   5. 태그 인식
5. Path planning
   1. 사용 기술
   2. Dubins path
   3. Reeds-Shepp path
   4. Bezier curve
   5. 비교 분석
   6. Bezier curve 구현
6. Path tracking
   1. 사용 기술
   2. Pure pursuit control
   3. Model predictive control (MPC)
   4. PID 제어
   5. Stanley controller
   6. 비교 분석
7. 시각화 도구
   1. 사용기술
   2. pygame
   3. RViz
   4. Matplotlib
   5. 종합 비교
   6. 시각화 구현
8. 자원관리 및 품질관리
   1. 자원관리 – HW/SW
   2. 테스트 방안
   3. 품질관리 방안
   4. 위험관리 방안
   5. 유지보수 계획
   6. 형상관리 계획
   7. 일정 계획
9. 예상되는 문제점 및 해결 방안
   1. 태그 인식 문제와 해결 방안
   2. 시뮬레이션과 현실의 스케일 차이 문제와 해결 방안
   3. 차량 제어 문제와 해결 방안
   4. 카메라의 왜곡 문제와 해결 방안
   5. 차량 프로세스의 연산량 문제와 해결 방안
10. 결과
    1. 예상 결과
    2. 기대 효과
    3. 평과 계획
    4. 한계 및 개선 계획

**1. 개요**

* 1. 프로젝트 명

국민대학교 제8회 자율주행 경진대회

* 1. 프로젝트 목표

ROS2 기반 1/10 모형차에서 실행가능한 경량시스템을 개발한다.

주어진 하드웨어에서 이용 가능한 자율주차 알고리즘을 작성하고 이를 검증한다.

* 1. 주요 기능
* 카메라 하나만을 사용하여 주차구역의 AprilTag 인식
* AprilTag를 이용하여 차량과 주차구역 사이의 상대적인 위치와 자세 파악
* 차량의 상대위치 추정하여 차량의 이동경로 계획(path planning)
* 차량을 계획된 경로대로 잘 이동시켜 주차구역에 진입하는 자율주행 시스템 구현(path tracking)
* 시각화 도구를 이용해 계획한 주차경로의 테스트 및 검증
  1. 기대 효과
* 본선 대회의 주차미션에 주차 경로 계획 알고리즘 사용 가능
* 본선 대회 차량에 사용되는 ROS2 기반 시스템 설계를 통해 최신 로봇 운영체제 기술의 활용 능력을 강화
* 여러 가지 파이썬 라이브러리 및 모듈 활용 능력 강화

1. **팀 구성 및 역할분담**
   1. 팀 구성

프로젝트의 목표, 범위, 기술 스택을 파악하고 팀원들의 관심분야와 주력분야에 따라 역할들을 분배한다.

* 1. 역할 분담

프로젝트 역할 분담은 다음과 같다.

팀장 : ROS2 환경구성, 팀원간 조율 및 총괄, 자료조사, 문서화

팀원1 : AprilTag 분석 및 활용방안 모색

팀원2 : path planning 조사 및 적용

팀원3 : path tracking 조사 및 적용

팀원4 : 시각화 도구 개발 및 적용

* 1. 의사소통 방법
* Git 및 GitHub를 활용한 소프트웨어 버전 관리와 팀원 간 코드 공유
* 카카오톡, 전화 등을 통한 실시간 연락망 구축
* 대면 회의를 통한 프로젝트 진행 상황 공유, 문제점 논의, 의사소통 강화
* 회의록 작성을 통한 업무 이력 관리 및 정보 공유
* 적극적인 질문과 피드백을 통한 신속하고 효율적인 의사소통
* Docker 및 Docker Hub를 활용하여 컨테이너 기반의 동일한 개발 환경 구축 및 공유
* 철저한 README 작성을 통해 코드 구조 및 사용 방법을 명확히 전달
* 역할 분담 명확화 – 누가, 무엇을, 언제 할 것인지 분명히 소통
* 즉각적인 자료 공유로 중복 작업을 방지하고 효율적인 협업 유도

1. **기술적 접근**
   1. 문제 정의

본 프로젝트는 ROS2 기반으로 자율주행 차량 시스템을 구현하며 주요 기술적 접근은 다음과 같다.

* 차량은 주차 구역을 인식한 후, 목표 지점까지의 최적 경로를 탐색하고 이동해야 한다.
* 차량은 카메라를 통해 환경 데이터를 수집하며, 경로 계획 및 제어 알고리즘을 통해 자율적으로 주행한다.
  1. 기술적 목표

본 프로젝트는 ROS2 기반으로 자율주행 차량 시스템을 구현하며 주요 기술적 접근은 다음과 같다

* ROS2를 활용하여 모듈 간 통신을 구현하고, OpenCV를 통해 카메라 데이터를 처리한다.
* 처리된 카메라 데이터를 기반으로, 경로 생성(path planning) 알고리즘을 통해 최적 경로를 설정하고, 경로 추종(path tracking) 알고리즘을 통해 차량이 해당 경로를 따라 이동하도록 한다.
* 구현한 알고리즘의 작동 여부는 pygame, RViz 등의 시각화 도구를 통해 확인한다.
* 프로젝트는 효율적인 개발과 관리를 위해 총 네 개의 세션으로 분리하여 진행한다.

1. AprilTag 처리
2. path planning
3. path tracking
4. 시각화
   1. 사용 기술 및 구현계획
      1. 사용 기술

* Python - 3.10
* ROS2 - humble

기본 개발 환경 설정과 모듈 간 통신 및 데이터 처리를 한다.

* cv2 (OpenCV)

실시간 컴퓨터 비전을 위한 오픈소스 라이브러리로, 이미지와 영상 처리, 물체 인식, 추적, 3D 재구성 등 다양한 기능을 제공한다.

* NumPy (Numerical Python)

고성능 수치 계산을 위한 파이썬 라이브러리로 배열 연산 및 벡터 계산에 사용할 수 있다.

* AprilTag

AprilTag 인식에 사용되는 대표적인 파이썬 라이브러리로 태그의 위치 및 자세 계산에 주로 사용된다.

* cv\_bridge

cv\_bridge는 ROS2 이미지 메시지를 OpenCV 이미지 형식 (np.array)으로 변환하거나, 그 반대로 변환해주는 패키지이다.

* SciPy

수학, 과학, 공학 계산에 필요한 고급 기능들을 모아놓은 NumPy 기반의 Python 라이브러리로 회전 행렬을 오일러 각도로 변환하기 위해 사용하였다.

* os

운영 체제 관련 작업 (파일 경로, 시스템 호출 등)에 사용하는 파이썬 라이브러리이다.

* rclpy

ROS2를 Python으로 제어할 수 있게 하는 클라이언트 라이브러리이다. 또한 노드 생성, 토픽 발행/구독 및 ROS2 메시지 처리 등을 할 수 있다.

* sensor\_msgs.msg

센서 데이터 메시지 타입이다.

* xycar\_msgs.msg

모터 제어에 사용되는 메시지 타입이다.

* + 1. 주요 노드 및 토픽
* 노드

|  |  |
| --- | --- |
| 노드 이름 | Track\_Driver |
| 역할 | 메인 노드로, 카메라 데이터를 처리하고 AprilTag 검출 및 모터 제어를 수행 |

* 구독 토픽

|  |  |
| --- | --- |
| 토픽 이름 | /usb\_cam/image\_raw |
| 메시지 타입 | sensor\_msgs/Image |
| 역할 | 카메라로부터 실시간으로 이미지를 수신하여 OpenCV 형식(np.array)으로 변환. |

* 발행 토픽

|  |  |
| --- | --- |
| 토픽 이름 | xycar\_motor |
| 메시지 타입 | xycar\_msgs/XycarMotor |
| 역할 | 차량의 조향각(angle)과 속도(speed)를 설정하여 모터를 제어 |

* + 1. 주요 함수
* 콜백 함수

|  |  |
| --- | --- |
| 함수 이름 | usbcam\_callback |
| 역할 | /usb\_cam/image\_raw 토픽에서 수신된 카메라 데이터를 OpenCV 형식(np.array)으로 변환하여 image 변수에 저장 |

* 모터 제어 함수

|  |  |
| --- | --- |
| 함수 이름 | drive |
| 입력 | angle: 조향각, speed: 속도 |
| 역할 | 차량의 조향각(angle)과 속도(speed)를 설정하여 모터를 제어 |

* AprilTag 가공 함수

|  |  |
| --- | --- |
| 함수 이름 | apriltag\_pose() |
| 입력 | Image, 칼리브레이션 정보 |
| 역할 | 칼리브레이션 정보를 바탕으로 image로부터 태그의 상대좌표, 각도, 태그의 중심좌표 등과 같은 정보를 구함 |

|  |  |
| --- | --- |
| 함수 이름 | path\_planning() |
| 입력 | 태그의 상대좌표, 각도 |
| 역할 | 입력 받은 태그의 상대좌표, 각도를 이용해 Bezier curve 알고리즘 등으로 path 생성 |

* path\_planning 함수
* path\_tracking 함수

|  |  |
| --- | --- |
| 함수 이름 | path\_tracking() |
| 역할 | Path planning에서 구한 path 따라가게 함 |

* 시각화 함수

|  |  |
| --- | --- |
| 함수 이름 | visualizer() |
| 입력 | 시작 좌표, 끝점 좌표, 초기 방향각, yaw 값, 차량의 현재 위치 |
| 역할 | 1. 시작점 NumPy 배열로 변환 2. 메인 화면 생성 & 기본 설정 3. path 생성 4. 초기 위치 업데이트 5. 화면 실행 6. 차량의 위치 변화를 감지하여 차량의 현재 위치 업데이트 |

* 메인 함수

|  |  |
| --- | --- |
| 함수 이름 | start() |
| 역할 | ROS2 노드를 초기화하고, 카메라 데이터를 처리하여 주요 메소드들을 실행시킬 준비를 한 뒤 이를 실행시킴. |

* + 1. 데이터 흐름

1. 노드 초기화

ROS2 노드(Track\_Driver)를 생성하고, 카메라 데이터 (/usb\_cam/image\_raw)를 구독하고 차량 모터 제어 토픽(xycar\_motor)을 발행한다.

1. 카메라 입력

카메라에서 이미지를 수신하고, OpenCV로 처리하여 흑백 이미지 생성한다.

1. AprilTag 검출

AprilTag 라이브러리를 사용하여 태그의 ID, 중심 좌표, 그리고 pose를 계산한다. 과정은 다음과 같다.

1. 카메라에서 수신된 이미지를 흑백으로 변환 (cv2.cvtColor).
2. apriltag.detector를 사용하여 이미지에서 태그를 검출.
3. 검출된 태그의 ID, 중심 좌표, 모서리 좌표를 출력 및 시각화.
4. 태그의 pose를 계산하여 위치와 회전 행렬을 추출.
5. 거리 및 pose 계산

태그의 위치와 회전 정보를 기반으로 차량과 태그 간의 거리 및 상대적인 pose를 계산하고, SciPy를 활용하여 회전 행렬을 오일러 각도로 변환하여 Yaw(회전 각도)를 추출한다.

1. path planning

path planning 알고리즘을 조사하고 적용해 태그의 위치와 회전 정보로부터 차량이 주행할 경로를 만들어낸다.

1. path tracking

path tracking 알고리즘을 조사하고 적용해 차량이 path planning에서 설정한 경로를 추종하게 한다.

1. 모터 제어  
   차량의 조향각(angle)과 속도(speed)를 계산하여 xycar\_motor 토픽으로 발행한다.
2. 시각화 및 디버깅

RViz, pygame 등을 이용해 차량의 path tracking 과정 및 결과를 지켜보고 보완한다.

1. **AprilTag 분석**
   1. 사용 기술

차량에 대한 주차구역의 상대적인 위치 측정을 위해 AprilTag를 인식하고 이를 가공해 상대적인 좌표와 각도 정보를 얻는 단계이다

* AprilTag : AprilTag 태그를 인식하고 이를 분석해 여러 가지 정보를 도출할 수 있는 라이브러리이다.
* OpenCV : 칼리브레이션, 행렬계산 등 여러 유용한 계산들을 제공하는 라이브러리이다.
  1. 칼리브레이션  
      칼리브레이션(calibration)이란 센서(카메라)의 내부적 오류나 왜곡을 모델링하고 보정해 측정값과 실제 물리 값의 차이를 줄이는 과정이다. 이는 우리가 수행할 프로젝트인 AprilTag를 이용한 자율주행 경로 계획 및 추종에서 카메라 렌즈의 왜곡을 보정하고 보다 이상에 가까운 이미지를 차량에 넘겨줄 수 있게 한다. 이러한 관점에서 칼리브레이션은 반 필수적인 과정이다.
     1. 칼리브레이션 과정

1. 평면의 AprilTag 보드를 준비

일반적으로 칼리브레이션은 체커보드 등으로 진행하나 해당 프로젝트는 AprilTag를 사용하는 프로젝트이므로 이를 중심으로 계획을 작성하였다. 일단 정확하고 안정적인 칼리브레이션을 위해 하나의 태그보다는 태그의 크기와 간격이 정확히 정의된 평면의 AprilTag 보드를 준비한다.

하나의 태그가 아니라 여러 개의 태그가 들어간 보드를 사용함으로 더 많은 대응점을 확보하고, 한 번에 중앙, 가장자리 등 더 넓은 화면 영역을 사용할 수 있고, 한 번에 다양한 각도에서 관측하여 보정 정확도가 높아진다.

1. 해당 보드를 여러 각도에서 여러 장 촬영한 뒤 AprilTag 라이브러리에서 태그를 검출

|  |
| --- |
| **import apriltag**  **detector = apriltag.Detector()**  **results = detector.detect(gray\_image)** |

1. 각 태그를 현실의 위치와 매핑시켜 리스트로 저장
2. 리스트들을 여러 데이터를 OpenCV 라이브러리의 calibrateCamera 메소드에 넣어 칼리브레이션을 실행

|  |
| --- |
| **import cv2**  **ret, K, dist, rvecs, tvecs = cv2.calibrateCamera(**  **object\_points\_list, # 각 이미지마다의 object points**  **image\_points\_list, # 각 이미지마다의 image points**  **image\_size, # (width, height)**  **None, None)** |

* + 1. 칼리브레이션 통해 얻은 정보

이를 통해 얻을 수 있는 정보는 다음과 같다.

|  |  |  |
| --- | --- | --- |
| 반환값 | 자료형 | 의미 |
| Retval | float | 평균 재투영 오차 |
| cameraMatrix | 3x3 NumPy array | 내부 파라미터 |
| distCoeffs | 1x5 or 1x8 NumPy array | 렌즈 왜곡 계수 |
| Rvecs | 3x1 벡터 리스트 | 회전 벡터 |
| tvecs | 3x1 벡터 리스트 | 이동벡터 |

1. distCoeffs

카메라의 왜곡을 보정하는 상수이다.

[k1, k2, p1, p2, k3] (일반적으로 5개)

|  |  |  |
| --- | --- | --- |
| k1, k2, k3 | 방사 왜곡 | 원형 패턴이 퍼지거나 오그라드는 왜곡 |
| p1, p2 | 접선 왜곡 | 렌즈가 정확히 센터에 놓이지 않은 경우 발생 |

1. cameraMatrix

3x3 행렬로 카메라의 초점거리와 중앙 좌표에 대한 정보이다. 이를 사용하여 태그의 상대적인 위치와 각도 정보를 알아낼 수 있다.

와 는 초점거리에 대한 정보이고 와 는 주점(이미지의 중심)에 대한 정보이다.

* 1. AprilTag 라이브러리 사용

다음과 같은 코드를 사용하여 AprilTag 라이브러리를 사용할 수 있다.

|  |
| --- |
| **# 카메라 파라미터 (fx, fy, cx, cy) (초점거리 좌표 / 센서 중앙 좌표)**  **camera\_params = (371.42821, 372.60371, 310.49805, 235.74201)**  **# distortion coefficients**  **dist\_coeffs = np.array([-0.325278, 0.082082, 0.000997, -0.000955, 0.0])**  **undistorted = cv2.undistort(image, camera\_matrix, dist\_coeffs)**  **# 감지할 태그 옵션지정 "tag36h11"**  **options = apriltag.DetectorOptions(families="tag36h11")**  **# 태그 검출 객체 생성**  **detector = apriltag.Detector(options)**  **#태그 크기(m)**  **tag\_size=8**  **# 흑백 변환**  **gray = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)**  **#AprilTag 검출**  **detections = detector.detect(gray)**  **pose, e0, e1 = detector.detection\_pose(det,camera\_params,tag\_size)** |

이러한 과정을 거치면 pose 행렬, AprilTag의 정보를 담은 리스트인 detections를 얻을 수 있다. detections의 속성들은 다음과 같다. 이중 중요한 정보는 중심점과 네 꼭짓점 정보이다.

|  |  |
| --- | --- |
| 이름 | 역할 |
| d.tag\_id | 태그 ID (예: 0, 1, 2, ...) |
| d.center | 중심점 (x, y) 좌표 |
| d.corners | 네 꼭짓점 [(x0, y0), (x1, y1), (x2, y2), (x3, y3)] |
| d.hamming | 해밍 거리 (인식 오류 정도) |
| d.decision\_margin | 디코딩 신뢰도 |

* 1. pose 행렬 분석

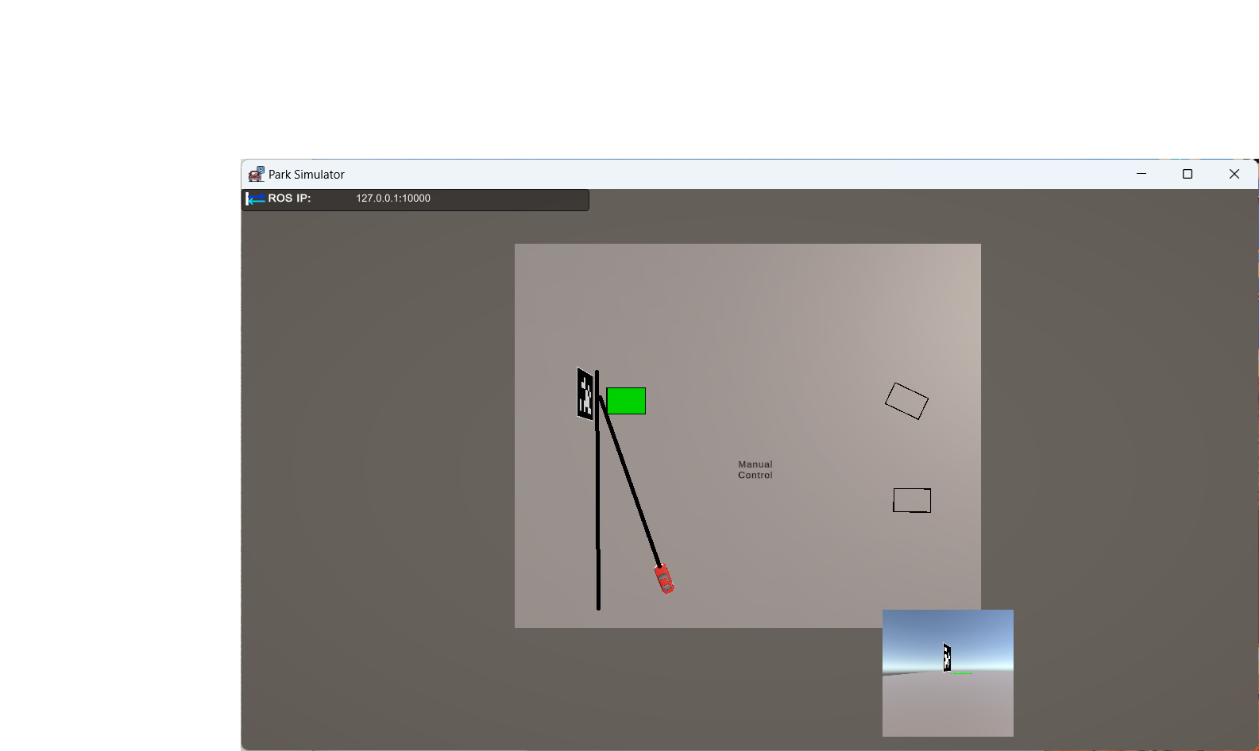
pose 행렬은 거리, 방향에 대한 행렬로 다음과 같은 구조를 가진다.

위치 벡터는 라고도 나타내며 차량에 대한 태그의 상대적인 위치이다. 일반적으로 x좌표가 클수록 차량 우측에 위치하고 y좌표가 클수록 차량 위쪽에 위치하고 z좌표가 클수록 차량 앞쪽에 위치한다.

회전 행렬은 라고도 나타내며 차량과 태그의 상대적인 각도 차이에 관한 행렬이다. 이는 다음과 같이 가공하여 xz 평면 상의 각도 차이 정보를 알아낼 수 있다.

|  |
| --- |
| **# 회전 행렬 객체 생성**  **rot = R.from\_matrix(pose\_r)**  **# 객체에 대해 오일러 각을 z-y-x 순서로 도 단위로 반환**  **euler\_angles = rot.as\_euler('zyx', degrees=True)**  **# yaw만 필요하므로 나머지 버림**  **\_, yaw, \_ = euler\_angles** |

* 1. 태그 인식

위에서 살펴본 AprilTag를 국민대 자율주행 시뮬레이션에 적용해보았다. 시뮬레이션 결과 좌우로 70도 정도까지 차량이 AprilTag를 인식할 수 있었다. 시뮬레이션 상에서 거리에 따른 꼭짓점 좌표 등의 오차가 존재할 것으로 예측했으나 화면에 태그가 들어오면 대부분 인식할 수 있는 것을 확인할 수 있었다.

1. **Path planning**
   1. 사용 기술

AprilTag를 분석하여 얻은 정보를 바탕으로 차량의 주행에 사용될 경로를 계획하는 단계이다. 여러 가지 알고리즘들을 사용하여 진행할 수 있으며 고려해야 할 사항은 연산량, 연산속도, 안정성 등이다.

* 1. Dubins path

Dubins path 알고리즘의 원리는 최소 회전 반경을 가지는 차량이 출발점에서 목표점까지 이동할 때 최단 경로를 계산하는 알고리즘이다.

* + 1. 기하학적 원칙

1. 차량은 직선으로 이동하거나 일정한 반경으로 좌회전(Curve Left, L) 또는 우회전(Curve Right, R)만 할 수 있다.
2. 차량의 방향은 출발점과 목표점에서 각각 고정되어 있다.
3. 최단 경로는 항상 직선(S)과 곡선(L 또는 R)의 조합으로 이루어진다.
   * 1. 가능한 경로 조합
4. LSL (좌회전-직선-좌회전)
5. LSR (좌회전-직선-우회전)
6. RSL (우회전-직선-좌회전)
7. RSR (우회전-직선-우회전)
8. LRL (좌회전-우회전-좌회전)
9. RLR (우회전-좌회전-우회전)

총 6가지 경로 조합 가능하다.

* + 1. Dubins path 이론

1. 출발점과 목표점 설정
2. 가능한 경로 조합에 대한 path 계산
3. 최단 거리 계산된 경로 선택

(Python의 dubins 라이브러리 사용해서 구현)

* 1. Reeds-Shepp path  
     Reeds-Shepp path는 Dubins path의 확장된 형태로, 차량이 후진을 포함하여 이동할 수 있는 경우에 사용된다. Reeds-Shepp path의 경우 좁은 주차공간이나 복잡한 환경에서 유리하다.
     1. 기하학적 원칙

1. 차량은 직선으로 이동하거나 일정한 반경으로 좌회전 또는 우회전 할 수 있으며, 전진 또는 후진 이동을 포함할 수 있다.
2. 차량의 방향은 출발점과 목표점에서 각각 고정되어 있다.
3. 최단 경로는 항상 직선과 곡선의 조합으로 이루어진다.
   * 1. 가능한 경로 조합

전진(F), 후진(B), 좌회전(L), 우회전(R)을 조합하여 가능한 모든 경로를 생성한다. 총 48가지 경로 조합 가능하다.

* + 1. Reeds-Shepp path 이론

1. 출발점과 목표점 설정
2. 가능한 경로 조합에 대한 path 계산
3. 최단 거리 계산된 경로 선택

(Python의 reeds\_shepp 라이브러리 사용해서 구현)

* 1. Bezier curve

Bezier curve은 제어점들을 이용해 부드러운 곡선을 생성하는 수학적 알고리즘이다. 주로 n개의 점으로부터 얻어지는 n-1차 곡선으로 표현된다. 이번 주차 경로 계획을 위해서는 3차 Bezier curve을 활용했다. 정해진 시작점과 끝점을 이용해 제어점 2개를 추가적으로 구한 후 Bezier curve을 구한다.

* + 1. Bezier curve 이론

(초기점: P0, 끝점: P3, 제어점 1: P1, 제어점 2: P2)

1. 제어점 1 계산

제어점 1의 경우 초기점에서의 방향과 임의로 설정한 초기점에서 제어점 사이 거리를 이용해 계산한다.

* : 제어점 1의 x 좌표
* : 제어점 1의 y 좌표
* : 초기점과 제어점 사이 거리
* : 초기점에서의 방향

1. 제어점 2 계산

제어점 2의 경우 끝점에서의 방향과 임의로 설정한 초기점에서 제어점 사이 거리를 이용해 계산한다.

* : 제어점 2의 x 좌표
* : 제어점 2의 y 좌표
* : 끝점과 제어점 사이 거리
* : 끝점에서의 방향

1. Bezier curve 그리기
   1. 비교 분석

|  |  |  |  |
| --- | --- | --- | --- |
| 알고리즘 | 특징 | 장점 | 단점 |
| Dubins path | 최소 회전 반경을 가지는 차량이 전진만 가능한 경우의 최단 경로 계산. 직선과 원호의 조합 | 차량의 회전 반경 반영한 현실적인 경로 생성. | 후진 불가능. |
| Reeds-Shepp path | Dubins path의 확장. 후진과 전진 모두 가능 | 차량의 회전 반경과 후진을 반영한 경로 생성. 앞선 두 알고리즘보다 복잡한 상황에서 주차 가능. | 후진 가능. 계산 복잡(48가지 경우 고려해야 함) |
| Bezier curve | 제어점을 기반으로 부드러운 곡선 생성. 경로 부드러움 | 경로 매우 부드럽고 연속적. 계산 비교적 간단(하나의 경로 계산) | 차량의 회전 반경을 직접적으로 반영하기 어려움. 후진 불가능 |

* + 1. Bezier curve 선택 이유

위의 세 알고리즘 중 Bezier curve 알고리즘을 선택한 이유는 다음과 같다

1. 계산 비교적 간단함

Dubins path, Reeds-Shepp path 알고리즘의 경우 차량의 경로를 구하기 위해 가능한 여러 경로를 계산한 후 최단거리인 경로를 선택하는 방식으로 구현된다. 반면 Bezier curve의 경우 주어진 시작점과 끝점의 정보로 최적의 path를 바로 구현한다. 간단한 알고리즘은 곧 짧은 수행시간을 의미하고, 이는 실시간 시각화에 있어 굉장한 이점이라 할 수 있다.

1. 주행 목적 (차량 주차)

차선 주행이 아닌 차량 주차가 주행 목적인 상황인 만큼 보다 더 정교한 경로를 짤 필요가 있다고 판단하여 위 곡선들 중 가장 자유도가 높은 Bezier curve를 채택하였다. 가장 이상적인 경로를 계획한 다음, 경로 추종 알고리즘을 정교화하는 접근으로 주차 알고리즘을 설계할 예정이다.

* 1. Bezier curve 구현
* 곡선 연산 함수

|  |  |
| --- | --- |
| 함수 이름 | bezier() |
| 입력 | 입력 위치 행렬, 제어점 2개, 시작점, 도착점 |
| 역할 | 입력 위치에 따른 베지에 곡선 좌표들을 계산하여 해당하는 점을 반환한다. |
| 예시 코드 | **def** bezier(t, P0, P1, P2, P3):      return (1 - t)\*\*3 \* P0 + 3\*(1 - t)\*\*2 \* t \* P1 + 3\*(1 - t) \* t\*\*2 \* P2 + t\*\*3 \* P3 |

* 제어점 연산 함수

|  |  |
| --- | --- |
| 함수 이름 | compute\_control\_points() |
| 입력 | 시작 좌표, 끝점 좌표, yaw 값, 제어점 거리 |
| 역할 | 제어점의 위치와 각도를 연산한다. |
| 예시 코드 | def compute\_control\_points(P0, P3, theta0\_deg, yaw\_deg, d0=20, d1=30):          theta0 = np.deg2rad(theta0\_deg)          theta1 = np.deg2rad(theta0\_deg - yaw\_deg)          P1 = P0 + d0 \* np.array([np.cos(theta0), np.sin(theta0)])          P2 = P3 - d1 \* np.array([np.cos(theta1), np.sin(theta1)])          return P1, P2, theta0, theta1 |

* 본 함수

|  |  |
| --- | --- |
| 함수 이름 | path\_planning() |
| 입력 | pose x좌표, pose z 좌표, yaw |
| 역할 | 위의 메소드들을 바탕으로 베지에 곡선 점들의 array를 생성한다. |
| 예시 코드 | P0 = np.array([0.0, 0.0])      theta0\_deg = 90      P3 = np.array([latest\_pose\_x, latest\_pose\_z])      P1, P2, theta0, theta1 = compute\_control\_points(P0, P3, theta0\_deg, latest\_yaw\_deg)      offset\_distance = -10      direction\_vector = np.array([np.cos(theta1), np.sin(theta1)])      offset\_point = P3 + offset\_distance \* direction\_vector      ts = np.linspace(0, 1, 100)      return np.array([bezier(t, P0, P1, P2, offset\_point) for t in ts]) |

1. **Path tracking**
   1. 사용기술  
      path planning 단계에서 설계한 경로를 바탕으로 이를 추종하여 자율주행을 직접 실행하는 단계이다. 해당 알고리즘은 차량의 정보와 곡선 정보를 받아서 차량의 조향각과 속도를 변화시키고 차량의 현재 위치를 업데이트 하도록 설계되어야 한다. 여러 가지 알고리즘들을 사용하여 진행할 수 있으며 고려해야 할 사항은 연산량, 연산속도, 안정성 등이다. 조사한 알고리즘은 다음과 같다.
   2. Pure pursuit control

자율주행에서 사용되는 알고리즘 중 하나로 차량에서 일정거리 떨어진 점을 목표로 계속 추종하게 하는 알고리즘이다. 알고리즘의 개요는 다음과 같다.

1. lookahead point설정

차량의 현재 위치에서 일정 거리 떨어진 경로 상의 목표점 선택한다.

1. 목표점 방향 계산

차량의 현재 위치와 lookahead point를 사용해 회전 각도 계산한다.

1. 조향각 계산

: 조향각

: 차량의 휠 베이스 (앞바퀴-뒷바퀴 거리)

: 차량과 목표점 사이의 각도

: lookahead 거리

1. 차량 이동

계산된 조향각을 이용해 차량이 목표점을 향해 움직이도록 한다.

* 1. Model predictive control (MPC)

Model predictive control (MPC)은 최적화 기반 제어 기법으로, 시스템의 동적 모델을 사용하여 일정 시간 동안의 제어 입력을 예측하고 최적의 제어 입력을 계산한다. 자율주행 차량의 트랙 주행에서 MPC는 차량이 주어진 경로(트랙)를 따라가면서 속도와 방향을 제어하는 데 매우 효과적이다.

1. 현재 상태 측정

차량의 현재 상태(위치, 방향, 속도) 측정한다.

1. 미래 상태 예측

차량의 동적 모델을 통해 미래 상태 예측한다.

1. 비용 함수

차량이 중심선을 따라가며 부드럽게 주행하도록 비용 함수(cost function) 정의한다.

1. 최적화

비용 함수를 최소로 만든다

1. 제어 입력

위의 내용을 바탕으로 제어 입력을 도출한다.

* 1. PID 제어

제어 대상을 원하는 값으로 맞추기 위하여 오차를 조정하는 제어 방식이다.

: 제어 입력

: 현재 오차

: 비례 게인. 현재 상태에서의 오차값 크기에 비례

: 적분 게인. 정상상태의 오차 제어

: 미분 게인. 오버슈팅을 줄이고 안정성을 향상

세 가지 파라미터들을 수학적, 경험적 방법으로 계산하는 과정을 거쳐 원하고자 하는 출력값에 부드럽게 수렴하게 한다. 대표적인 파라미터 튜닝 방법으로는 Ziegler–Nichols 방법 등이 있다.

1. 오차 계산

시스템의 목표값과 현재값 비교해 오차()를 계산한다

1. 제어 신호 생성

위에서 계산한 오차를 기반으로 비례, 적분, 미분 요소 계산 후 제어 신호 를 생성한다.

1. 목표 값 도달

를 이용해 시스템이 목표 값에 도달하도록 한다.

1. 반복

위의 과정 반복해 오차를 줄인다.

* 1. Stanley controller

아래의 관계식을 통해 path tracking을 구현한다.

* : 출력 조향각 (steering angle)
* : heading error (차량의 진행 방향과 경로 방향의 차이)
* : cross track error (차량의 앞차축 중심에서 경로까지 수직 거리
* : 차량의 현재 속도
* : gain 값 (경로로 끌어당기는 강도)

1. 현재 상태 측정

차량의 현재 위치와 진행 방향 측정 한다.

1. 목표점

경로 상에서 차량과 가장 가까운 점 찾는다.

1. heading error계산

heading error는 차량의 진행 방향과 경로의 접선 방향 간의 각도 차이를 조정한다.

1. Cross track error (CTE) 계산

CTE는 차량의 현재 위치와 목표 경로 사이의 수직 거리 차이를 조정한다.

1. 조향 각도 계산

위의 출력 조향각 식을 이용해 조향 각도를 계산한다.

1. 차량 이동

위의 내용을 바탕으로 차량을 이동시킨다.

* 1. 비교 분석

|  |  |  |  |
| --- | --- | --- | --- |
| 알고리즘 | 특징 | 장점 | 단점 |
| Pure pursuit | 차량의 현재 위치에서 look-ahead point를 추종함 | 구현 간단  실시간성 우수  비교적 안정적 | 곡선에서의 오버슈팅  고속 주행과 좁은 공간에 부적합 |
| Model predictive control (MPC) | 역학 모델링으로 미래를 예측해 조향 | 고정밀 제어  다양한 제약사항 처리 가능 | 큰 계산량  복잡한 튜닝 |
| Stanley controller | 현재 경로와의 가로편차와 헤딩 차이를 보정함 | 고속 주행에서 안정적  경로 정렬 정밀  빠른 계산 | 저속에서 불안정  노이즈에 민감 |
| PID 제어 | Lateral error에 비례하는 단순 피드백 제어 | 구현이 간단 다양한 시스템에 적용 가능  목표 값에 빠르게 도달 가능 | 튜닝이 어려움  노이즈에 민감  적분 포화 발생 가능성 있음 |
| Pure pursuit + PID | Pure pursuit으로 조향, PID로 속도 제어 결합 | 경량 구조에 비교적 우수  튜닝으로 조건 대응 가능 | 튜닝 중요  경로 복잡 시 불안정 |

* + 1. Pure pursuit 및 PID 선택 이유

해당 프로젝트의 주요 요구사항은 간단한 구현, 낮은 계산량, 그리고 실시간성 확보이다. 또한 주행 환경이 비교적 단순하며, 경로 상에 급격한 커브나 복잡한 장애물이 적기 때문에, 고정밀 제어나 복잡한 경로 계획이 반드시 필요한 상황은 아니다. 이러한 조건에서는 고차원 모델 기반 제어(MPC)나 LQR 등의 복잡한 알고리즘보다는, 구현이 간단하고 계산 속도가 빠른 경량형 추종 제어 방식이 적합하다.

이러한 배경에서 Pure pursuit 알고리즘과 PID 제어기의 결합 방식은 본 프로젝트의 목적에 매우 부합한다.

* Pure pursuit는 경로 상의 지정된 look-ahead point를 따라가는 방식으로, 구조가 직관적이며 실시간 주행 제어에 적합하다.
* PID 제어기는 차량 속도에 대한 정밀 제어가 가능해, 목표 속도를 안정적으로 유지하며 주행 성능을 향상시킬 수 있다.

두 제어기를 결합함으로써, 조향 제어는 Pure pursuit가 담당하고 속도 제어는 PID가 담당하게 되어, 경량화된 구조로도 안정적이고 부드러운 경로 추종이 가능해진다. 또한 각 알고리즘은 비교적 소수의 파라미터만 조정하면 되므로, 튜닝과 디버깅도 용이하다.

따라서 본 프로젝트의 목적과 환경을 고려할 때, Pure pursuit + PID 제어 방식은 단순성과 성능을 모두 만족시키는 최적의 선택지로 판단된다.

1. **시각화**
   1. 사용기술  
      해당 단계는 path tracking을 실행하는 차량의 움직임을 분석하고 피드백하는 단계이다. 해당 단계에서 구현해야 할 것은 주차 경로, 이동하는 자동차, 주차구역, path planning에서 설정한 경로이다. 또한 path tracking 과 연계되어 실시간으로 차량의 위치를 계산하여 나타내야 할 것이다. 이는 여러가지 시각화 도구를 통해 이루어질 수 있으며 사용한 도구는 다음과 같다.

* pygame : 파이썬 기반의 게임 개발 라이브러리로, 2D 그래픽, 소리, 사용자 입력 처리를 쉽게 할 수 있게 한다.
* RViz : ROS2환경에서 사용하는 3D 시각화 도구이다.
* Matplotlib : 파이썬의 2D 그래프 시각화 라이브러리로 과학, 공학, 데이터 분석 분야에서 그래프, 차트, 플롯을 쉽게 만들 수 있도록 한다.
  1. 텍스트, 스크린샷, 소프트웨어, 멀티미디어 소프트웨어이(가) 표시된 사진

     AI가 생성한 콘텐츠는 부정확할 수 있습니다.pygame

Matplotlib로 그린 그래프를 바로 옮기는 데 유용하다. 예시 실행 화면은 다음과 같다.

* 1. RViz

현재 자율주행 시스템을 ROS2 기반으로 제작하고 있다는 점에서 ROS2와 호환성이 가장 좋은 시각화 도구라고 할 수 있는 RViz 또한 활용을 고려하였다. 그럼에도 RViz는 주로 3D 공간 내 시뮬레이션을 다룬다는 점과 그로 인해 연산 자원이 많이 필요하다는 점에서 차량이 2D 공간을 주행하고 있는 것으로 상정하고 있는 현 상황에 맞지 않다고 판단하였다. 이러한 환경에서는 차량을 기준으로 한 Top-view 시점으로 경로를 나타내는 것으로 경로 계획 시각화는 충분할 것으로 보여 RViz 대신 pygame을 시각화 도구로 채택하였다.

* 1. Matplotlib

Bezier curve 등 그래프를 그리는 과정에서 그래프가 잘 그려지는지 확인하는 용도로 사용할 수 있다. 예시 실행 화면은 다음과 같다. 텍스트, 스크린샷, 소프트웨어, 디스플레이이(가) 표시된 사진

AI 생성 콘텐츠는 정확하지 않을 수 있습니다.

* 1. 종합 비교

|  |  |  |  |
| --- | --- | --- | --- |
| 시각화 도구 | 특징 | 장점 | 단점 |
| pygame | 파이썬 기반의 게임 개발 라이브러리 | 시각화에 적합함  간단한 조작  고성능 렌더링  멀티미디어 통합 기능  파이썬 친화적 | 고급 기능 부족  윈도우 관리 기능 부족  해상도 이슈  다중 스레드 제한 |
| RViz | ROS2 기반의 시각화 도구임 | 실시간 데이터 시각화  다양한 플러그인  ROS2와의 호환성 | 높은 시스템 자원 요구  높은 진입장벽 |
| Matplotlib | 그래프 및 차트 출력에 특화됨 | 과학 및 공학적 시각화에 최적화  정확하고 정밀  다양한 출력 지원  높은 호환성  비교적 쉬운 사용법 | 성능 문제  3d 환경에서의 한계  복잡한 애니메이션 제한 |

* + 1. pygame 선택 이유

가장 성능이 뛰어난 시각화 도구는 ROS2와의 호환성이 우수하고 3D 모델링을 지원하는 RViz라고 할 수 있다. 그러나 현재 차량의 움직임이 2D 공간에 한정되어 있으며, 차량의 경로와 실제 움직임 간의 정합성을 확인하기 위해 Top-view 시점이 필요한 점을 고려하여, 본 프로젝트에서는 pygame을 시각화 도구로 사용하기로 결정하였다. 시각화 과정에서 필요한 수학적 연산과 그래프 출력은 Matplotlib을 활용하되, 시각화 프레임의 안정성과 실시간성 확보를 위해 pygame을 통해 최종 시각화를 구현하였다.

* 1. 시각화 구현

텍스트, 스크린샷, 소프트웨어, 멀티미디어 소프트웨어이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.아래 코드는 베지에 곡선을 따라 주행하는 차량의 위치를 실시간으로 추적하고, pygame을 사용해 그 궤적을 시각화하는 예시이다. Path planning 방식은 위에서 논의한대로 베지에 곡선 방식을 채택하였다.

|  |  |
| --- | --- |
| 함수 이름 | visualizer() |
| 입력 | 곡선 데이터, 차량 위치 데이터, 차량 속도, 차량 방향 |
| 역할 | 1) pygame을 띄우기 위한 초기 설정  2) path tracking 함수로부터 곡선 데이터 받아오기  3) path, 주차위치, 차량위치 그리기  4) path tracking 을 이용한 차량 위치 제어 및 위치 업데이트  5) 프로그램 종료 시 까지 3~4의 과정을 반복 |
| 예시 코드 | pygame.init()      width, height = 800, 600      screen = pygame.display.set\_mode((width, height))      pygame.display.set\_caption("Bezier Curve Visualization")      clock = pygame.time.Clock()  curve = path\_planning()  *# 전체 범위 계산 및 스케일링*      all\_x, all\_y = curve[:, 0], curve[:, 1]      min\_x, max\_x = np.min(all\_x), np.max(all\_x)      min\_y, max\_y = np.min(all\_y), np.max(all\_y)      margin = 1.5      range\_x, range\_y = max\_x - min\_x, max\_y - min\_y      scale = min(width / (range\_x \* margin), height / (range\_y \* margin))      center\_x = (min\_x + max\_x) / 2      center\_y = (min\_y + max\_y) / 2  **def** world\_to\_screen(x, y):          sx = int(width / 2 + (x - center\_x) \* scale)          sy = int(height / 2 - (y - center\_y) \* scale)          return sx, sy      while not stop\_visualization:          screen.fill((255, 255, 255))          for event in pygame.event.get():              if event.type == pygame.QUIT:                  stop\_visualization = True  *# 곡선 그리기*          for i in range(len(curve) - 1):              start = world\_to\_screen(curve[i][0], curve[i][1])              end = world\_to\_screen(curve[i + 1][0], curve[i + 1][1])              pygame.draw.line(screen, (0, 0, 255), start, end, 2)  *# 제어점 원 그리기*          for point, color in zip([P0, P1, P2, P3], [(0, 255, 0), (255, 0, 0), (255, 0, 0), (255, 0, 255)]):              pygame.draw.circle(screen, color, world\_to\_screen(point[0], point[1]), 6)  *# 시작점 방향 화살표 (진한 초록색)*          p0\_dir = P0 + 5 \* np.array([np.cos(theta0), np.sin(theta0)])          pygame.draw.line(screen, (0, 150, 0), world\_to\_screen(\*P0), world\_to\_screen(\*p0\_dir), 3)  *# 도착점 방향 화살표 (보라색)*          p3\_dir = P3 + 5 \* np.array([np.cos(theta1), np.sin(theta1)])          pygame.draw.line(screen, (255, 0, 255), world\_to\_screen(\*P3), world\_to\_screen(\*p3\_dir), 3)  *# 방향에 맞는 속이 빈 직사각형 그리기*          rect\_width = 3   *# 진행 방향 길이 (world 단위)*          rect\_height = 5   *# 수직 방향 길이*          half\_w = rect\_width / 2          half\_h = rect\_height / 2          angle\_rad = theta1  *# 진행 방향*  *# 직사각형 꼭짓점 (중심 기준, 회전 전)*          corners = np.array([              [-half\_w, -half\_h],              [ half\_w, -half\_h],              [ half\_w,  half\_h],              [-half\_w,  half\_h]          ])  *# 회전 행렬 적용*          rotation\_matrix = np.array([              [np.cos(angle\_rad), -np.sin(angle\_rad)],              [np.sin(angle\_rad),  np.cos(angle\_rad)]          ])          rotated\_corners = np.dot(corners, rotation\_matrix)  *# offset\_point 기준으로 이동*          world\_corners = rotated\_corners + offset\_point  *# 화면 좌표 변환*          screen\_corners = [world\_to\_screen(x, y) for x, y in world\_corners]          pygame.draw.polygon(screen, (0, 0, 0), screen\_corners, width=1)  *# 회전 행렬 적용*          rotation\_matrix = np.array([              [np.cos(angle\_rad), -np.sin(angle\_rad)],              [np.sin(angle\_rad),  np.cos(angle\_rad)]          ])          rotated\_corners = np.dot(corners, rotation\_matrix)  *# 중심 좌표 더하기*          world\_corners = rotated\_corners + offset\_point  *# 화면 좌표로 변환*          screen\_corners = [world\_to\_screen(x, y) for x, y in world\_corners]          pygame.draw.polygon(screen, (0, 0, 0), screen\_corners, width=1)          path\_tracking()          pygame.display.flip()          clock.tick(10)      pygame.quit() |

1. **자원관리 및 품질관리**
   1. 자원 관리 – HW/SW
      1. 하드웨어(HW) 자원

* 개발용 PC
* 170도 어안렌즈 카메라
  + 1/3’’ CMOS OV2710 센서
  + 640x480 60fps, 1080p 30fps
  + USB 비디오 클래스 – UVC 1.1
* 자이카 모터
  + Wheel Motor, 구동모터 제어신호 – 전진/후진, 속도 조종
  + Steering Motor, 조향모터 제어신호 – 좌회전/우회전
  + 조향범위 -100 ~ 100, 좌/우 20도
  + 속도 -100 ~ 100
    1. 소프트웨어(SW)
* Visual Studio Code
* GitHub
* Notion
* ROS2

|  |  |
| --- | --- |
| 자원 항목 | 계획 및 관리 내용 |
| 개발자 인력 | 주요 기능을 구현할 수 있는 Python 및 ROS2 숙련자 확보 |
| 테스터 인력 | 시스템을 실제 환경에서 검증할 인력 배치 |
| 로봇 하드웨어 | 자이카 및 USB 카메라, 모터 제어 장치 확보 및 점검 |
| 소프트웨어 라이브러리 | OpenCV, ROS2, Matplotlib 등 라이브러리 설치 및 의존성 관리 |
| 작업 공간 | 안전하고 충분한 공간에서 테스트 수행 |
| 컴퓨팅 자원 | 고성능 노트북 또는 워크스테이션 확보 |
| 네트워크 연결 | ROS2 노드 통신을 위한 안정적인 로컬 네트워크 환경 유지 |
| 전원 공급 장치 | 지속적인 테스트를 위한 안정적 전원 확보 |
| 백업 시스템 | 코드 및 데이터의 정기적 백업 수행 |
| 시뮬레이션 도구 | Gazebo 또는 RViz와 같은 시뮬레이터 활용 |
| 교통 표지판 모델 | 테스트용 AprilTag 또는 시각 표지판 확보 |
| 데이터 저장 장치 | 로그, 이미지, 센서 데이터를 위한 저장 공간 확보 |
| 디버깅 도구 | 실시간 로깅 및 디버깅을 위한 rqt, rostopic 등 도구 사용 |
| 스페어 부품 | 모터, 휠, 센서 등 소모성 부품 예비 확보 |
| 교육 자료 | 참여자 교육을 위한 매뉴얼 및 가이드 문서 준비 |
| 외부 컨설턴트 | 필요 시 외부 전문가 자문 확보 |
| 보안 관리 | 코드 접근 제어 및 장비 보안 조치 수행 |

* 1. 테스트 방안

본 프로젝트에서 개발한 자율주차 시스템은 AprilTag을 활용한 인식 기반 경로 계획(path planning)과 경로 추종(path tracking) 알고리즘을 결합하여 차량이 자동으로 주차할 수 있도록 구성된다. 이를 검증하기 위해 다양한 단계의 테스트 방안을 마련하였다.

* + 1. 단위 테스트  
        단위 테스트(unit test)를 통해 AprilTag 인식, 경로 생성, 차량 제어 등 개별 모듈의 정상 동작을 검증한다. ROS2 환경에서는 launch\_testing을 활용하여 노드 및 시스템 통합 테스트를 수행하고, Python 모듈의 경우 unittest 또는 pytest와 같은 표준 테스트 프레임워크를 사용하여 입력과 출력의 정합성을 확인한다.
    2. 통합 테스트  
        통합 테스트(integration test)를 통해 전체 시스템의 데이터 흐름과 연동 상태를 점검한다. 카메라 영상 수신부터 태그 인식, 경로 생성, 제어 명령 송신까지의 일련의 과정을 ROS2 기반 시뮬레이터 및 실시간 시각화 도구인 RViz를 활용해 확인하며, 시스템이 실제 환경에서도 안정적으로 작동하는지를 가상 환경에서 미리 검증한다.
    3. 시각화 시뮬레이션  
        pygame을 활용한 시각화 시뮬레이션을 통해 자율주행 알고리즘의 경로 생성 및 추종 결과를 직관적으로 확인한다. 차량의 궤적, 조향 각도, 목표 위치 등이 2D 시각적으로 표현되며, 알고리즘의 동작 상태를 실시간으로 분석할 수 있어 주차 알고리즘의 디버깅과 최적화에 유리하다.
    4. 성능 테스트  
        성능 테스트(performance test) 에서는 알고리즘의 실시간성 및 반응 속도를 평가한다. 태그 인식 지연, 경로 재계산 시간, 모터 제어 명령의 전송 지연 등 주요 지표를 측정하고, 시스템 전체가 프레임 손실 없이 주행을 유지할 수 있는지를 검토한다. 저사양 장비에서도 안정적인 동작을 보장하기 위해 비동기 처리와 병렬 연산을 활용한 최적화 전략도 함께 고려된다.
    5. 현장 테스트  
        현장 테스트(field test)를 통해 실제 주차 환경에서 알고리즘을 검증한다. 다양한 조명 조건, 바닥 반사, 태그의 인식 거리 등 실제 상황에서 발생할 수 있는 변수들을 고려하여 테스트를 수행하며, 테스트 결과를 바탕으로 알고리즘의 신뢰성과 견고성을 개선한다.  
        이와 같은 다단계 테스트를 통해 자율주차 시스템의 정확도와 안정성을 확보하고, 실제 환경에 적용 가능한 수준의 완성도를 갖춘 소프트웨어를 구현하는 것을 목표로 한다.
  1. 품질관리 방안
* 카메라가 AprilTag 인식하는 범위 구하기 위해 가상 환경에서 시뮬레이션을 진행해 AprilTag threshold를 구한다. 관련 내용은 ‘4.5 태그 인식’에서 확인할 수 있다.
* 기능 정확성 판단을 위해 여러 시뮬레이션 가상환경에서 테스트 진행한다.
* 정확한 AprilTag의 거리 및 방향 측정을 위해 실제 카메라 초점거리에 대한 정보 적용해 시뮬레이션을 진행해 측정값의 안정성을 높인다.
* GitHub를 이용해 팀의 코드 공유 및 관리한다.
* 차 이동에 대한 시각적 알고리즘을 만들어 주차 알고리즘의 동작을 확인한다.

|  |  |
| --- | --- |
| 품질 항목 | 관리 방안 |
| 기능 정확성 | 요구사항에 따른 테스트 케이스 통과 여부확인 |
| 성능 최적화 | 주행 지연 최소화를 위한 연산 최적화 |
| 에러 처리 | 예외 발생 시 시스템 안전 정지 및 로그 출력 |
| 시각화 품질 | 플롯 갱신 및 직관적 표현 확인 |
| 센서 신뢰도 | 태그 인식 정확도 및 거리 추정 정확도 점검 |
| 시스템 반응성 | 실시간 반응 시간 측정 및 튜닝 |
| UI/UX | 터미널 및 시각 피드백 구성 점검 |
| 테스트 범위 | 다양한 환경 및 각도에서의 인식 정확성 평가 |
| 회귀 테스트 | 업데이트 시 기존 기능 재검증 |
| 버전 관리 | Git 등을 활용한 안정적 코드 관리 |
| 문서화 | 개발/설치/운영 문서 제공 |
| 라이브러리 검증 | 사용된 오픈소스 라이브러리의 안정성 확인 |
| 하드웨어 연동 검증 | 모터 제어, 센서 연결 테스트 |
| 시나리오 기반 검증 | 실제 주행 조건 기반 테스트 |
| 환경 적응성 | 조도, 위치 변화 등에 따른 안정성 검증 |
| 결함 추적 | 버그 발생 시 즉시 기록 및 해결 추적 |
| 정기 리뷰 | 주간 코드 리뷰 및 품질 회의 |
| 자동화 테스트 | 가능한 테스트는 스크립트화 하여 반복 수행 |

* 1. 위험관리 방안  
      위험관리 방안은 예기치 못한 사고나 손실을 사전에 예방하고, 발생 시 피해를 최소화하기 위해 반드시 필요하다. 프로젝트나 시스템 운영 중 발생할 수 있는 다양한 위험 요소를 미리 파악하고 대비함으로써, 업무의 안정성과 연속성을 확보할 수 있다. 또한, 구성원 간의 역할과 대응 방안을 명확히 정해 위기 상황에서의 혼란을 줄이고 신속한 대응을 가능하게 한다. 결과적으로 위험관리는 목표 달성의 신뢰도를 높이고, 자산과 인명을 보호하는 중요한 수단이다.
     1. 카메라 신호 미수신

차량 카메라에서 송출되는 영상 스트림이 ROS2 이미지 토픽을 통해 정상적으로 수신되지 않을 경우, 태그 인식 알고리즘이 동작하지 않아 주차 진행이 중단될 수 있다. 특히 네트워크 지연, 하드웨어 오류, ROS2 노드 간 연결 문제 등이 원인일 수 있다. 이러한 문제에 대한 대응 전략은 다음과 같다.

* rclpy.wait\_for\_message() 또는 QoS 설정을 활용해 이미지 토픽이 정상적으로 들어오는지 주기적으로 확인한다.
* 이미지 토픽 수신이 일정 시간 이상 지연될 경우, 노드 재시작 또는 재접속을 시도하는 자동 복구 로직을 구현한다.
* ROS2의 라이프사이클 노드(lifecycle node) 또는 헬스 체크(heartbeat) 메커니즘을 도입해 연결 상태를 모니터링하고, 영상 신호 이상 시 운영자에게 알림을 발생시킨다.
* 하드웨어 상태 점검 및 케이블, 카메라 모듈 재부팅 절차도 포함한다.
  + 1. 카메라 프레임 지연 및 유실

카메라 영상이 프레임 단위로 전송되지만, 저사양 하드웨어나 네트워크 부하로 인해 프레임 손실이나 지연이 발생할 수 있다. 이로 인해 태그 인식 주기가 불규칙해져 경로 추종 오류가 발생할 위험이 있다. 이러한 문제에 대한 대응 전략은 다음과 같다.

* 영상 수신 및 처리 코드를 비동기 또는 멀티스레드 방식으로 설계해 프레임 처리를 병렬화한다.
* 프레임 드롭 감지 로직을 추가해 유실 프레임 수가 임계치를 넘으면 시스템 경고 또는 프레임 재요청을 수행한다.
* 카메라 해상도 및 프레임 레이트를 최적화하여 처리 부담을 줄인다.
* 네트워크 대역폭 모니터링 및 품질 보증(QoS) 설정을 통해 영상 데이터 안정성을 확보한다.  
  + 1. 태그 인식 실패

태그 인식은 조명 상태, 태그와 카메라 간 거리, 태그의 회전 각도 등에 민감하다. 불충분한 조명, 태그 훼손, 카메라 위치 변화 등으로 인식 실패가 발생할 수 있다. 이러한 문제에 대한 대응 전략은 다음과 같다.

* 태그 설치 위치 및 각도를 최적화하여 조명 조건과 시야각을 개선한다.
* 소프트웨어 측면에서 태그 검출 알고리즘의 민감도 및 임계값을 조절하여 다양한 환경에 적응할 수 있도록 한다.
* 인식 실패 시에는 마지막으로 인식한 태그 위치를 기반으로 차량의 상대 위치를 추정하는 보간 및 예측 알고리즘을 적용해 일시적 태그 부재에도 주행이 가능하도록 한다.
* 태그 주변에 보조 조명 장치 설치를 고려하고, 태그가 훼손될 경우 교체 절차를 마련한다.  
  + 1. 태그 수신 지연

태그를 인식한 후 해당 데이터가 처리 및 주행 명령에 반영되기까지 지연이 발생하면, 주행 경로 추종에 영향을 주어 부정확한 제어가 일어날 수 있다. 이러한 문제에 대한 대응 전략은 다음과 같다.

* 태그 데이터 처리 파이프라인을 최적화하여 병렬 처리와 큐 관리를 강화한다.
* 실시간 처리 시스템에서는 처리 지연 모니터링을 추가해 임계치 초과 시 경고 및 비상 정지 신호를 발송한다.
* 우선순위 기반 태그 데이터 처리 및 메시지 필터링 기법을 적용해 최신 데이터를 빠르게 반영한다.  
  + 1. 모터 통신 불량

차량의 모터 제어 명령이 ROS2 메시지로 전달될 때 전송 실패 또는 지연이 발생하면 주행 제어 신뢰성이 떨어지고 차량이 예상치 못한 행동을 할 수 있다. 이러한 문제에 대한 대응 전략은 다음과 같다.

* 메시지 송신 시 확인 응답(ACK) 메커니즘을 도입하거나, 발행 재시도 로직을 적용해 실패 시 자동 재전송한다.
* 통신 채널 상태를 주기적으로 점검하며 이상 발생 시 즉시 로그를 기록하고 운영자에게 알림을 보낸다.
* 통신 하드웨어(케이블, 무선 모듈 등) 상태 점검 절차를 마련한다.
* 모터 컨트롤러의 펌웨어와 소프트웨어 간 호환성을 지속적으로 검증한다.  
  + 1. 쓰레드 충돌  
        주차 경로 시각화 등 GUI 플롯과 차량 제어를 병렬로 처리할 때 쓰레드 간 자원 접근 충돌이 발생할 수 있다. 이러한 문제에 대한 대응 전략은 다음과 같다.
* Python `threading.Lock`이나 `mutex` 같은 동기화 기법을 활용해 데이터 접근 시점의 충돌을 방지한다.
* 충돌 발생 시 로그를 기록하고, 복구 가능한 경우 재시도한다.
* GUI 플롯 업데이트는 메인 루프와 분리해 이벤트 큐 기반으로 처리하여 블로킹 현상을 줄인다.  
  + 1. Pose 값의 왜곡으로 인한 거리 추정 오차

태그로부터 차량까지의 거리를 추정할 때 카메라 pose 값이 왜곡될 수 있어 실제 거리와 차이가 발생할 위험이 있다. 이러한 문제에 대한 대응 전략은 다음과 같다.

* 여러 프레임에서 측정된 거리 값을 평균화하거나 보정함으로써 단일 데이터 왜곡 영향을 줄인다.
* 카메라 렌즈 왜곡 보정 및 캘리브레이션 값을 정기적으로 갱신한다.
* 필요 시 별도의 거리 센서(LiDAR, 초음파 센서 등)를 추가해 교차 검증한다.  
  + 1. 코드 호환성 문제

개발 시 사용한 Python, ROS2 버전 차이로 인해 배포 환경에서 오류가 발생할 수 있다. 특히 ROS2, Python 3.7과 3.10 이상 사이 호환성 문제를 주의해야 한다. 이러한 문제에 대한 대응 전략은 다음과 같다.

* 개발 및 배포 환경의 Python과 ROS 버전을 명확히 명시하고, 해당 버전에서만 동작하도록 제한한다.
* CI/CD 도구를 활용해 여러 버전에서 자동 테스트를 수행한다.
* 주요 외부 라이브러리 버전을 고정하고, 의존성 충돌을 미리 방지한다.  
  + 1. 배터리 전압 강하  
        테스트 환경이나 실제 주행 시 배터리 전압이 낮아지면 시스템 전체가 불안정해지고 갑작스러운 장애가 발생할 수 있다. 이러한 문제에서 대응 전략은 다음과 같다.  
                
       - 배터리 전압 상태를 실시간 모니터링하는 로거를 도입하여 임계 전압 이하 시 경고 및 안전 정지를 수행한다.  
       - 무정전 전원공급장치(UPS)를 활용해 갑작스러운 전원 차단에 대비한다.  
       - 배터리 용량 및 상태를 주기적으로 점검하고, 주행 테스트 전 충분한 충전을 보장한다.
  1. 유지보수 계획
     1. 유지 보수 목적
* AprilTag 기반 자율주차 시스템의 지속적인 안정성과 신뢰성 확보, 그리고 사용자 경험 향상을 위한 기능 개선
* 실제 주차 환경의 조도, 장애물, 노면 상태 등의 변화에 능동적으로 대응하기 위해 알고리즘과 파라미터의 지속적인 점검 및 조정이 필요
* ROS2, OpenCV, AprilTag 등 주요 외부 라이브러리의 버전 업데이트 또는 API 변경에 따른 시스템 호환성 유지 및 리스크 최소화
* 하드웨어 구성 요소(카메라, 센서 등)의 노후화 또는 교체에 따른 소프트웨어 측 보정 및 적응 능력 확보
* 장기적으로는 시스템의 확장성 및 타 차량 또는 플랫폼과의 호환성 확보를 위한 구조적 유연성 유지
  + 1. 유지보수 내용

|  |  |  |
| --- | --- | --- |
| 항목 | 내용 | 점검 주기 |
| Tag 인식 정확도 | 인식률 감소 시 원인 분석 (조도, 카메라 문제 등) | 월 1회 |
| Path planning 알고리즘 | Bezier curve/리드셉스 등 알고리즘의 파라미터 튜닝 | 변경시 |
| 실차 동작 테스트 | 자율주차 정확도, 충돌 여부 점검 | 주 1회 |
| 카메라 보정 상태 | 내부 파라미터 및 왜곡계수 재측정 여부 확인 | 3개월 1회 |
| 모듈 간 통신 상태 | ROS2 topic/service 간 딜레이 및 오류 체크 | 주 1회 |
| 환경 변화 반영 | |  | | --- | | Tag 위치 변경, 주차선 위치 변경 대응 | | 변경시 |

* 1. 형상관리 계획
     1. 형상관리 목적
* **시스템 안정성 및 신뢰성 확보**

AprilTag 기반 자율주차 시스템의 지속적인 동작 안정성과 신뢰성을 확보하기 위해 정기적인 성능 테스트 및 기능 점검을 수행한다.

* **환경 변화 대응력 강화**

조명 변화, 임시 장애물, 노면 상태 등 다양한 실외·실내 주차 환경 변화에 대응할 수 있도록 알고리즘의 주요 파라미터를 지속적으로 조정하며, 인식 정확도 향상을 위한 업데이트와 데이터셋 보강 작업도 병행할 예정이다.

* **외부 라이브러리 및 의존성 버전 관리**

ROS2, OpenCV, AprilTag 등의 핵심 라이브러리에 대해 버전 업데이트 주기를 모니터링하고, 새로운 버전에 따른 API 변경사항이나 기능 호환성 문제를 사전에 검토하여 코드 수정 및 테스트를 체계적으로 수행한다. 버전 고정(Tagging) 및 변경 이력 관리를 통해 형상 안정성을 유지한다.

* **하드웨어 구성요소의 주기적 점검**

카메라, 센서, 모터 등 핵심 하드웨어 부품의 수명 주기를 고려한 정기적인 점검과 유지보수를 실시한다. 하드웨어 교체 시에는 소프트웨어 보정 및 재캘리브레이션 절차를 통해 기존 성능과의 일관성을 유지하며, 테스트 자동화 도구를 활용해 신속하게 적응할 수 있는 체계를 구축한다.

* **미래 확장성과 이식성 확보**

향후 다양한 차량 플랫폼이나 다른 시스템 구조와의 연계를 고려하여, 소스코드는 모듈화 및 인터페이스 기반으로 설계한다. 이를 통해 기능 추가 및 수정 시 시스템 전체에 미치는 영향을 최소화하고, 장기적인 유지보수와 확장에 유리한 구조를 유지한다

* + 1. 형상관리 내용

|  |  |  |
| --- | --- | --- |
| 항목 | 도구 / 방식 | 설명 |
| VCS | Git | 분산 버전 관리 시스템 |
| 원격 저장소 | GitHub | |  | | --- | | 형상관리 저장소 및 협업 플랫폼 |  |  | | --- | |  | |
| 이슈 관리 | GitHub issues | |  | | --- | | 모듈별 작업 분류 및 상태 관리 |  |  | | --- | |  | |
| CI/CD | GitHub actions | 테스트/빌드 자동화 구성 (선택) |

브랜치 관리

|  |  |
| --- | --- |
| 항목 | 설명 |
| Main | ROS2 프레임워크 관리 |
| AprilTag | AprilTag 관련 소스코드 관리 |
| Path planning | path planning 알고리즘 관리 |
| Path tracking | path tracking 알고리즘 관리 |
| visualizing | 시각화 도구 관리 |

* 1. 일정 계획

 7월 한 달간 프로젝트를 집중적으로 수행한다는 전제 하에, 전체 일정과 주요 목표를 체계적으로 수립하였다. 각 단계별로 요구되는 시간과 자원을 종합적으로 고려하여 주차별 세부 계획을 마련하였으며, 구성원 간의 효율적인 역할 분담과 업무 흐름의 연속성을 확보하고자 일정의 구체화를 병행하였다. 이러한 계획을 바탕으로 프로젝트의 진행 상황을 주기적으로 점검하며, 각 단계별 산출물이 실제 구현과 검증으로 이어질 수 있도록 추진할 예정이다. 이를 통해 실질적인 성과를 도출하고, 최종 결과물의 완성도와 신뢰성을 높이고자 한다.

|  |  |
| --- | --- |
| 1주차 | Ros2 개발환경 설정 및 역할 분담   1. ROS2 설치 및 테스트 환경 구축 2. AprilTag 인식 테스트 3. 개발 역할 분배 및 협업 체계 수립 |
| 2주차~3주차 | 팀원 별 알고리즘 개발 수행   1. Reeds-Shepp 및 Bezier 기반 경로 알고리즘 구현 2. Pure pursuit 기반 경로 추종 알고리즘 개발 3. pygame, RViz를 활용한 시각화 기능 개발 |
| 4주차 | 피드백 및 알고리즘 결함   1. 각 모듈 통합 테스트 2. 인식률 저하, 경로 오류 등 알고리즘 결함 수정 3. 피드백 반영을 통한 성능 개선 |
| 5주차 | 프로그램 최종 수정 및 디버깅   1. 전체 시스템 디버깅 및 안정화 2. 실주행 테스트 및 성능 검증 3. 발표 및 결과 문서 정리 |

1. **예상되는 문제점 및 해결 방안**
   1. 태그의 인식 문제와 해결 방안
      1. 문제  
          실제 주차를 위한 주행을 계획하고 실행하는 과정에서는 AprilTag를 인식하여, 태그와 차량 간의 상대 위치를 기반으로 주행이 이루어진다. 그러나 이 과정에서 태그를 인식하지 못하거나 인식 가능 범위를 벗어나게 될 경우, 주행에 심각한 문제가 발생할 수 있다. 특히, 주행 계획 수립 단계에서 태그가 인식되지 않는다면 계획 자체가 불가능해지고, 주차의 마지막 단계에서는 태그 상단이 카메라에 잡히지 않아 차량이 태그를 인식하지 못하게 되는 문제도 발생할 수 있다.
      2. 해결 방안

첫 번째로, 차량이 주행 도중 태그를 놓치는 경우에는 마지막으로 인식한 태그의 위치를 기반으로 차량의 상대 위치를 추정하고, 태그 없이 일정 구간 동안 주행할 수 있도록 하는 알고리즘을 구성할 수 있다. 태그 인식 가능 범위에 대한 사전 조사를 바탕으로, 차량이 인식 범위에서 벗어나는 경우를 예측하여 주행 경로를 사전에 계획하거나 보정할 수도 있다.  
 두 번째로, 주행 초기 경로 작성 단계에서 태그가 인식되지 않는 상황은 두 가지로 나눌 수 있다. 첫 번째는 태그가 차량의 인식 범위를 물리적으로 벗어난 경우이다. 이 경우 차량을 이동시키며 태그를 찾는 알고리즘을 설계할 수도 있으나, 이는 본 프로젝트의 주제와 맞지 않으므로 제외하겠다. 두 번째는 태그가 인식 범위 내에 있음에도 일시적으로 인식되지 않는 경우이다. 이러한 경우에는 차량이 태그를 인식할 수 있을 때까지 충분한 시간을 확보해 주는 방식으로 문제를 해결할 수 있다. 이는 주행 중 실시간 경로 수정이 어렵다는 프로젝트 특성상 더욱 중요하다.  
 마지막으로, 주차의 마무리 단계에서 태그의 상단 부분이 잘려 인식되지 않는 문제는, 태그 자체를 도착 지점으로 설정하는 대신, 태그가 바라보는 방향으로 일정 거리 떨어진 지점을 목적지로 설정함으로써 해결할 수 있다. 이 경우 차량은 해당 지점까지 주행한 후, 마지막에 직선 주행을 통해 정확한 위치에 주차하게 된다. 이 과정을 알고리즘에 반영함으로써 주차 정확도를 높일 수 있다.  
 단, 이 방식이 효과적으로 작동하려면, 설정된 목적지에서 차량의 방향이 태그의 방향과 일치하며, 차량 렌즈 중심과 태그 중심이 정확히 정렬되어 있어야 한다. 둘 중 하나라도 맞지 않을 경우 올바른 주차가 이루어지지 않기 때문에, 이를 사전에 검사하고, 필요시 후진이나 조향을 통한 미세 조정을 수행하는 보정 알고리즘이 추가로 필요하다.

* 1. 시뮬레이션과 현실의 스케일 차이 문제와 해결 방안
     1. 문제  
         우리가 사용하는 Bezier curve 기반의 시뮬레이션에서는 좌표 스케일이 현실 세계의 좌표계와 구조적·환경적 요인으로 인해 차이를 보일 가능성이 크다. 예를 들어, 시뮬레이터 상에서 1픽셀 혹은 1단위는 단순한 가상 거리로 표현되지만, 실제 환경에서는 동일한 단위가 1미터 또는 수 센티미터에 해당하는 물리적 거리로 변환되어야 한다. 이러한 좌표계 간의 스케일 차이가 존재할 경우, 시뮬레이션에서는 정확한 궤적을 따라 주행하는 것처럼 보이더라도, 실제 차량은 목표 위치보다 앞서거나 뒤처지게 도착하는 등 주행 정확도에 문제가 발생할 수 있다.
     2. 해결방안

이러한 문제를 해결하기 위해서는 시뮬레이션 좌표계와 현실 좌표계 간의 스케일 정합성을 확보하는 과정이 필수적이다. 즉, 두 좌표계의 거리 단위 간의 비율과 기준점을 정확히 보정하여, 시뮬레이션 상에서 생성된 경로가 현실에서도 동일하게 구현될 수 있도록 해야 한다.

이를 위한 첫 번째 해결 방법은 스케일 보정이다. 두 좌표계의 비례 관계를 파악한 후, 일정한 상수를 곱하여 거리 단위를 일치시키는 방식이다. 예를 들어, 시뮬레이션에서의 1단위가 현실 세계의 20cm에 해당한다면, 모든 시뮬레이션 좌표에 0.2를 곱함으로써 현실 거리로 변환할 수 있다. 이 방법은 거리 비율이 일정한 경우 매우 효율적이며 계산이 간단하다는 장점이 있다.  
 두 번째 방법은 오프셋(offset) 보정이다. 시뮬레이션의 좌표 원점과 실제 차량의 출발점이 다를 경우, 일정한 값을 더하거나 빼는 방식으로 기준점을 일치시킬 수 있다. 이 방법은 특히 시작 위치나 기준 좌표가 물리적으로 다른 상황에서 유효하며, 주행 경로의 전체적인 위치를 현실 환경에 정확히 맞추는 데 도움이 된다.  
 실제로는 이 두 가지 방법을 함께 적용해야 보다 정확한 좌표 변환이 가능하다. 먼저 스케일을 맞추기 위해 좌표에 상수를 곱한 뒤, 기준점 차이를 보정하기 위해 일정한 값을 더하거나 빼는 방식으로 오프셋을 조정해야 한다. 이러한 좌표 정합 작업은 시뮬레이터 기반의 경로를 현실에서도 정밀하게 재현하는 데 핵심적인 역할을 하며, 자율주행 경진대회와 같은 실시간 고정밀 주행이 요구되는 환경에서 특히 중요한 요소로 작용한다.

* 1. 차량 제어 문제와 해결 방안
     1. 문제  
         ROS2 기반 제어 시스템에서는 XycarMotor 메시지 객체를 생성한 뒤, 차량의 속도(speed)와 조향각(angle) 값을 motor\_msg에 담아 publish하면, 다른 노드가 이를 구독하여 실제 차량의 모터 제어에 반영하는 구조로 작동한다. 이러한 방식은 저수준의 입출력(I/O) 처리를 ROS2가 추상화하여 담당하기 때문에, 사용자는 하드웨어 세부 사항까지 직접 다루지 않아도 되어 전체 시스템을 보다 안정적이고 효율적으로 개발할 수 있다. 이로 인해 하드웨어를 잘못 다루어 발생할 수 있는 물리적인 문제들도 사전에 예방할 수 있다.  
         그러나 반대로, 저수준의 입출력이 필요한 특수한 상황이 발생했을 때에는 ROS2의 추상화 구조로 인해 문제를 직접적으로 파악하거나 해결하기 어려울 수 있으며, ROS2 내부에서 예기치 못한 문제가 발생했을 경우에도 원인을 빠르게 진단하고 능동적으로 대응하기가 쉽지 않다. 또한 ROS2에서 publish되는 speed와 angle 값이 현실의 차량 움직임과 정확히 일치하지 않을 수 있으며, 이는 제어 신호와 실제 하드웨어의 반응 간의 차이에서 비롯된다. 이러한 괴리를 줄이기 위해서는 ROS2에서 사용하는 제어 신호에 대해 스케일링(scaling) 작업을 적용하고, 현실 차량의 특성에 맞춰 보정하는 과정이 필요하다.
     2. 해결 방안

ROS2를 효과적으로 활용하기 위해서는 단순히 제공되는 인터페이스를 사용하는 것에 그치지 않고, 메시지 구조와 토픽, 그리고 내부적인 입출력 흐름에 대한 충분한 이해가 필수적이다. ROS2에 전적으로 의존하기보다는 ROS2가 어떻게 작동하는지, 실제 차량과 어떤 방식으로 연결되어 있는지를 이해하는 것이 현실과의 괴리를 줄이고 더 신뢰성 높은 시스템을 구현하는 데 큰 도움이 된다.

* 1. 카메라의 왜곡 문제와 해결 방안
     1. 문제  
         대회에서 사용하게 될 차량을 포함하여, 현실 세계의 대부분의 카메라 렌즈는 빛의 굴절이라는 구조적 한계로 인해 다양한 형태의 왜곡을 유발한다. 이로 인해, 원래는 직선으로 이루어진 사각형 형태로 인식되어야 할 태그가 왜곡된 곡선 형태로 식별되거나, 차량이 주행해야 할 직선 경로가 휘어져 보이는 등 여러 시각적 오류가 발생할 수 있다.
     2. 해결 방안

이러한 문제를 해결하기 위한 대표적인 방법은 카메라 캘리브레이션(calibration)이다. 이는 초기 설정 단계에서 카메라 렌즈의 특성을 수치적으로 추정하고, 이를 프로그램에 반영하여 왜곡을 보정하는 과정이다. 초점 거리, 광학 중심, 왜곡 계수 등 렌즈의 내부 파라미터를 측정하여, 렌즈가 제공하는 이미지와 실제 세계 간의 오차를 최소화할 수 있다. 앞서 설명한 계획 절차에 따라 진행되며, 이 과정을 통해 카메라 입력과 현실 세계 간의 괴리를 줄일 수 있다.  
 또 다른 보완 방법으로는 프로그램 상의 상수 조정을 들 수 있다. 예를 들어, 태그의 실제 크기나 사전 측정된 캘리브레이션 값을 상수로 도입한 뒤, 여러 번의 주행 실험을 통해 거리 스케일, 방향 스케일 등의 보정 상수를 점진적으로 조정할 수 있다. 이러한 반복적인 튜닝을 통해 초기 주행 결과보다 더욱 정확하고 안정적인 경로 추종이 가능해진다.  
 결론적으로, 현실 세계에서 발생하는 렌즈 왜곡 문제는 단순히 하드웨어의 문제가 아닌, 소프트웨어적으로 충분히 예측하고 보정할 수 있는 영역이다. 따라서 정확한 캘리브레이션과 경험 기반의 상수 조정을 병행함으로써, 보다 신뢰성 있는 주행 시스템을 구축할 수 있을 것이다.

* 1. 차량 프로세스의 연산량 문제와 해결 방안
     1. 문제  
         모든 하드웨어는 메모리 용량과 연산 속도에 한계를 가지고 있다. 이러한 한계를 넘는 연산이나 과도한 메모리 사용을 소프트웨어에서 하드웨어에 요구하게 되면, 하드웨어가 의도한 대로 동작하지 않거나, 현재 상태를 정확히 파악할 수 없게 되는 문제가 발생할 수 있다. 이러한 원리는 이번 프로젝트에서 사용하는 차량에도 동일하게 적용된다.

이 문제는 단순히 중앙처리장치(CPU)에만 국한되지 않는다. 예를 들어, 프로그램이 모터의 물리적인 성능을 초과하는 출력이나 조향을 요구하거나, 센서의 입출력 범위를 초과하는 데이터를 처리하려 할 경우에도 심각한 오류나 시스템 불안정이 발생할 수 있다.

* + 1. 해결

따라서 이번 프로젝트에서 소프트웨어를 작성할 때는 차량에 탑재된 하드웨어의 사양을 철저히 파악하고, 그 한계를 고려하여 개발을 진행해야 한다. 특히 차량의 중앙처리장치가 감당할 수 있는 연산량과 메모리 사용량을 기준으로 코드의 구조와 알고리즘을 설계해야 하며, 현실적으로 정확성, 성능, 속도를 모두 최상으로 끌어올리는 것은 불가능하기 때문에, 성능과 속도 사이에서 적절한 타협점을 찾는 것이 중요하다.  
 또한 디버깅 단계에서 사용하던 코드(예를 들어 센서의 원시 이미지 출력, `pygame`을 통한 실시간 시각화, 로그 출력 등)은 테스트 이후에는 반드시 비활성화하거나 제거해야 한다. 이는 전체 시스템의 부하를 줄이고, 보다 안정적이고 빠른 성능을 확보하는 데 큰 도움이 된다.

1. **결과**
   1. 예상 결과

본 프로젝트를 통해 개발되는 자율주차 시스템은 AprilTag를 이용한 상대 위치 인식과 Bezier curve을 기반으로 한 경로 계획(path planning) 알고리즘을 통합하여 차량의 실제 주차 작업을 자동으로 수행할 수 있게 될 것이다.

차량은 카메라를 통해 주차장 내에 부착된 태그를 인식하고, 태그와의 상대 위치 및 방향 정보를 바탕으로 차량의 현재 위치를 정확히 파악한다. 이를 기반으로 최적의 주차 경로를 생성하며, 경로는 차량의 조향 및 속도 제한을 반영해 현실적인 주행 경로로 설계된다. 또한, 경로를 따라 차량이 정확히 주행할 수 있도록 Pure pursuit 제어 알고리즘 등을 적용하여 경로 추종(path tracking)을 구현함으로써 부드럽고 안정적인 주행이 가능하도록 한다.

실제 차량에 적용 시, 경로에 따른 모터와 조향 제어 신호가 ROS2 메시지로 전달되어 원활한 주차 동작을 보장할 것이다. 주차 성공 시 차량은 목표 위치에 안정적으로 정지하며, 위치 및 방향 모두 허용 오차 범위 내에 수렴하여 정밀한 주차 결과가 기대된다. 또한, 태그 인식 실패 또는 일시적인 시야 이탈 상황에서도 보간 알고리즘을 활용해 주행 연속성을 유지하는 견고한 성능을 발휘할 것으로 예상된다.

한편, 주차 과정과 경로 추종 성능 검증을 위해 pygame과 RViz를 활용하여 시뮬레이션과 시각화 환경을 구축함으로써 실시간으로 주행 궤적을 모니터링하고, 알고리즘의 동작을 직관적으로 확인할 수 있도록 하였다. 이를 통해 개발 과정에서 발생할 수 있는 문제점을 신속히 파악하고 개선하는 데 도움을 줄 것으로 기대된다.

* 1. 기대 효과

이 시스템은 태그 인식, 경로 계획, 제어 신호 발행까지의 전 과정을 하나의 통합된 파이프라인으로 구현하였다. ROS2 기반의 통신 구조를 통해 하드웨어 제어를 효과적으로 추상화하면서도, 저수준 입출력에 대한 실질적인 이해를 병행할 수 있어 실무 역량 향상에 크게 기여한다. 또한, 차량 하드웨어와 소프트웨어 간의 효율적인 인터페이스 설계 경험과, 실시간 주행 환경에 최적화된 성능 튜닝 경험은 향후 더 복잡하고 다양한 자율주행 프로젝트에서도 폭넓게 활용될 수 있다.

* 1. 평가 계획

시스템 성능은 다각도의 정량적 지표를 통해 체계적으로 평가할 예정이다. 우선, 전체 주차 시도 중 성공적으로 목표 위치에 도달한 비율인 주차 성공률을 측정한다. 성공률은 시스템의 신뢰도를 대표하는 핵심 수치이며, 여러 환경 조건에서 반복 실험하여 통계적 유의성을 확보할 계획이다. 다음으로, 태그 인식 후부터 차량이 완전히 정차할 때까지의 평균 주차 소요 시간을 측정하여 주행 효율성을 평가한다. 또한, 주차 완료 시 차량 중심점과 목표 위치 간 거리 차이인 위치 오차, 그리고 차량 방향과 목표 방향 간 각도 차이인 방향 오차를 분석하여 주차 정확도를 세밀하게 검증할 것이다. 오류 유형별 발생 빈도도 기록하는데, 태그 인식 실패, 주행 궤도 이탈, 제어 신호 불일치 등 문제 상황을 체계적으로 분류하여 문제 원인 파악과 개선점을 도출할 수 있다. 이러한 정량적 평가는 개발 중간 및 최종 단계 모두에서 진행하여 지속적인 품질 개선에 활용될 것이다.

* 1. 한계 및 개선 계획

본 프로젝트는 실내 평탄한 주차 환경과 단일 비전 센서를 전제로 설계 및 구현되었기에, 현실 세계의 다양한 변수와 복잡한 환경에는 아직 완전한 대응이 어렵다. 예컨대, 조명 변화, 태그 표면 상태, 시야 각도 제한 등으로 인한 태그 인식률 저하는 시스템 신뢰도를 저해할 수 있다. 또한, 경사면 주차, 좁은 공간, 장애물 밀집 지역 등 복잡한 주차 환경에서는 현 경로 계획 및 제어 알고리즘의 한계가 분명히 존재한다. 이에 따라 향후에는 LiDAR, 초음파 센서, IMU 등 다양한 센서 데이터를 융합해 인식과 위치 추정을 보강하는 다중 센서 융합 기술이 도입되어야 한다. 알고리즘 측면에서는 주차 시나리오를 확장하고, 장애물 회피와 동적 환경 대응 기능을 추가하는 등 보다 범용적이고 실용적인 주차 시스템으로 진화시켜야 한다.