**SW 개발계획서**

**- 목 차 -**

1. 개요
   1. 프로젝트 명
   2. 프로젝트 목표
   3. 주요 기능
   4. 기대 효과
2. 프로젝트 조직 (project organization)
   1. 팀 구성
   2. 역할 분담
   3. 의사소통 방법
3. 기술적 접근
   1. 문제 정의
   2. 기술적 목표
   3. 개발 방법론
4. Apriltag 분석
5. Path planning
6. Path tracking
7. 시각화 도구
8. 일정 계획
   1. 개발 단계/일정
9. 자원관리 및 품질관리
   1. 사용 기술 스택
   2. 자원관리 – HW/SW
   3. 테스트 방안
   4. 품질관리 방안
10. 예상되는 어려움
    1. 예상되는 어려움
    2. 대응 방안
11. 결과
    1. 결과물 발표 내용
    2. 결과 보고서

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 주자구역 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

팀장 : ros 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 과 git hub를 이용한 소프트웨어 버전 관리 및 팀원 간의 공유

카카오톡, 전화 등을 통한 연락망 구축

대면을 통한 프로젝트 공유 및 의사소통, 문제점 토의

회의록 작성을 통한 이력 관리 및 정보공유

적극적인 질문과 피드백을 통해 빠른 의사소통

컨테이너 및 도커를 사용하고 이를 docker hub를 이용해 공유하여 팀원 간에 동일한 작업 공간 확보

철저한 readme 파일 작성을 통한 코드 정보 전달

누가 할 건지, 무엇을 할 건지 명확하게 소통

팀원 간에 즉각적인 자료 공유로 중복 작업 방지

1. **기술적 접근**
   1. 문제 정의

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

* 차량이 주차구역 인식 후 목표 지점까지 최적 경로 탐색하고 이동해야 한다.
* 차량은 카메라를 통해 데이터를 수집하고 경로 계획 및 제어 알고리즘을 통해 자율적으로 이동한다.
  1. 기술적 목표

ROS2를 사용해 모듈 간의 통신을 구현하고, OpenCV를 통해 카메라 데이터를 처리한다. 또한 처리한 카메라 데이터 활용하여 최적의 경로를 path planning 알고리즘으로 구현하고 path tracking 알고리즘으로 이를 추종하게 한다. 그리고 구현한 알고리즘의 작동을 pygame, Rviz 등 시각화 도구로 확인한다. 이를 위해 해당 프로젝트를 네 개의 세션으로 분리하였다.

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

Python

* 기본 프로그래밍 언어

ROS2

* 기본 개발 환경 설정
* 모듈 간 통신 및 데이터 처리

cv2 (OpenCV)

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

NumPy (Numerical Python)

* 고성능 수치 계산을 위한 파이썬 라이브러리로 배열 연산 및 벡터 계산에 사용할 수 있음

Apriltag

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

cv\_bridge

* ROS 이미지 데이터를 OpenCV 형식으로 변환해주는 패키지

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 라이브러리

Time

* 시간 지연 및 루프 제어

Rclpy

* **ROS 2를 Python으로 제어할 수 있게 하는 클라이언트 라이브러리**입니다.
* 노드 생성, 토픽 발행/구독 및 ROS 메시지 처리 등을 할 수 있다.
* sensor\_msgs.msg는 센서 데이터 메시지 타입이다.
* xycar\_msgs.msg는 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 형식으로 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 형식으로 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 |

에이프릴 태그 가공 함수

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

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

* Path\_planning 함수
* Path\_tracking 함수

|  |  |
| --- | --- |
| 함수 이름 | Path\_tracking() |
| 입력 |  |
| 역할 |  |

* 메인 함수

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

* + 1. 데이터 흐름

1. 노드 초기화

ROS 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 생성.

1. Apriltag 검출

Apriltag 라이브러리를 사용하여 태그의 ID, 중심 좌표, 그리고 자세(pose)를 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 자세(pose)를 계산하여 위치와 회전 행렬을 추출.
5. 거리 및 자세 계산

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

1. Path planning
2. Path tracking
3. 모터 제어  
   차량의 조향각(angle)과 속도(speed)를 계산하여 xycar\_motor 토픽으로 발행한다.
4. 시각화 및 디버깅
   1. 시스템 구현 및 Test

베지에 곡선 알고리즘을 활용해 차량의 주차 경로 계산한다. 계산한 주차 경로를 시뮬레이션 할 수 있는 시각적 알고리즘을 작성한다.

* + 1. 사용 라이브러리

|  |  |
| --- | --- |
| 라이브러리 | 역할 |
| numpy | 수치 계산 및 배열 연산 |
| Matplotlib.pyplot | 그래프 시각화 |
| Matplotlib.widgets | GUI 위젯 제공 |

* + 1. 주요 클래스 (DraggableControlPoints)

DraggableControlPoints는 베지에 곡선 이용한 시각적 알고리즘의 핵심 클래스로, 사용자가 마우스 드래그와 슬라이더를 통해 베지에 곡선을 실시간으로 조작할 수 있게 해주는 기능을 제공한다.

* 초기화 메소드

|  |  |
| --- | --- |
| 메소드 이름 | \_\_init\_\_() |
| 입력 | 축 객체, 시작/끝점, 초기각도, 초기거리 |
| 역할 | 객체 초기화 및 그래픽 요소 생성   1. 멤버변수 초기화 2. 곡선 라인 생성 3. 제어점 scatter plot 생성 4. 라벨 텍스트 생성 5. 방향벡터 화살표 생성 6. 마우스 이벤트 연결 |

* 업데이트 메소드

|  |  |
| --- | --- |
| 메소드 이름 | update\_dirs\_and\_points() |
| 역할 | 방향벡터와 제어점 위치 계산  아래 수식을 통해 계산  • dir = [cos(θ), sin(θ)] • P1 = P0 + d0 × dir0 • P2 = P3 + d1 × dir1 |

|  |  |
| --- | --- |
| 메소드 이름 | update\_curve() |
| 역할 | 베지에 곡선 및 모든 그래픽 요소 갱신   1. 0부터 1까지 100등분한 점에 대해 베지에 메소드를 적용해 점들을 계산 2. 곡선 데이터 업데이트 3. 제어점 위치 업데이트 4. 라벨 위치 업데이트 5. 방향벡터 화살표 재생성 6. 화면 다시 그리기 |

* 이벤트 처리 메소드

|  |  |
| --- | --- |
| 메소드 이름 | on\_press(event) |
| 역할 | 마우스 클릭 시 드래그 대상 결정   1. 클릭 위치와 제어점 간 거리 계산 2. Threshold(0.5) 이하면 드래그 시작   (P1 => P2순으로 검사) |

|  |  |
| --- | --- |
| 메소드 이름 | on\_motion(event) |
| 역할 | 마우스 이동 시 제어점 위치 업데이트   1. 마우스 위치를 방향벡터에 정사영 2. 정사영 길이로 거리 업데이트 3. 실시간 곡선 갱신 |

|  |  |
| --- | --- |
| 메소드 이름 | on\_release(event) |
| 역할 | 드래그 종료 처리  dragging\_point를 None으로 초기화 |

* + 1. 주요 함수
* 메인 함수

|  |  |
| --- | --- |
| 함수 이름 | plot\_bezier\_interactive() |
| 입력 | 시작 좌표, 끝점 좌표, 시작점에서 초기 방향각, 시작점에서 제어점까지 거리, 끝점 제어점까지 거리 |
| 역할 | 1. 시작점 numpy 배열로 변환 2. 메인 화면 생성 & 기본 설정(차트) 3. 베지에 곡선 객체 생성(위에서 정의한 DraggableControlPoints 활용) 4. 슬라이더 UI 생성 (시작점에서 제어점, 끝점에서 제어점, 방향각) 5. 초기 위치 조절 슬라이드 생성(x좌표 조절, y 좌표 조절) 6. 초기 위치 업데이트 함수 사용해 초기 위치 업데이트 7. 화면 실행 |

* + 1. 데이터 흐름

1. 초기화

메인 함수plot\_bezier\_interactive() 실행 후 시작점 numpy 배열로 변환

메인 화면 생성 & 기본 설정(차트)

1. 베지에 곡선 객체 생성

위에서 정의한 주요 클래스 DraggableControlPoints 활용

1. 슬라이더 UI 생성 (시작점에서 제어점, 끝점에서 제어점, 방향각)

초기 위치 조절 슬라이드 생성(x좌표 조절, y 좌표 조절)

1. 초기 위치 업데이트 함수 사용해 초기 위치 업데이트
2. 화면 실행
3. **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.

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 여러개의 태그가 들어간 보드를 사용함으로 더 많은 대응점을 확보하고, 한번에 중앙, 가장자리 등 더 넓은 화면 영역을 사용할 수 있고, 한번에 다양한 각도에서 관측하여 보정 정확도가 높아진다.

이후 해당 보드를 여러 각도에서 여러 장 촬영한 뒤 apriltag 라이브러리에서 태그를 검출한다. 예시 코드는 다음과 같다.

import apriltag

detector = apriltag.Detector()

results = detector.detect(gray\_image)

그 후 각 태그를 현실의 위치와 매핑시켜 리스트로 저장한다.

그후 리스트들을 과 여러 데이터를 OpenCV 라이브러리의 calibrateCamera 메소드에 넣어 칼리브레이션을 실행한다. 예시 코드는 다음과 같다.

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

)

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 |

이중 특히 중요하게 보아야 할 정보는 cameraMatrix와 distCoeffs이다.

distCoeffs는 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 |

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

|  |  |  |
| --- | --- | --- |
| fx | 0 | cx |
| 0 | fy | cy |
| 0 | 0 | 1 |

fx와 fy는 초점거리에 대한 정보이고 cx와 cy는 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, apriltag 의 정보를 담은 리스트인 detections를 얻을 수 있다.

detections의 속성들은 다음과 같다.

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

이중 우리가 중요하게 보아야 할 정보는 중심점과 네 꼭짓점 정보이다.

* 1. pose 행렬 분석

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

|  |  |
| --- | --- |
| 4x4 회전 행렬 | 3x1 위치벡터 |
| 0 | 1 |

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

회전 행렬은 pose\_r라고도 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. **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. 베지에 곡선 알고리즘
* 베지에 곡선은 제어점들을 이용해 부드러운 곡선을 생성하는 수학적 알고리즘이다. 주로 n개의 점으로부터 얻어지는 n-1차 곡선으로 표현된다.
* 이번 주차 경로 계획을 위해서는 3차 베지에 곡선을 활용했다. 정해진 시작점과 끝점을 이용해 제어점 2개를 추가적으로 구한 후 베지에 곡선을 구한다.

베지에 곡선 구현

(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.

P1 = P0 + d0 \* np.array([np.cos(theta0), np.sin(theta0)])

1. 제어점 2 계산

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

P2 = P3 - d1 \* np.array([np.cos(theta1), np.sin(theta1)])

1. 베지에 곡선 그리기
   1. Dubins path
   2. Reeds-shepp
   3. 비교 분석
2. **Path tracking**
   1. 사용기술  
      path planning 단계에서 설계한 경로를 바탕으로 이를 추종하여 자율주행을 직접 실행하는 단계이다. 마찬가지로 여러 가지 알고리즘들을 사용하여 진행할 수 있으며 고려해야 할 사항은 연산량, 연산속도, 안정성 등이다. 조사한 알고리즘은 다음과 같다.
   2. Pure pursuit control
   3. Model predictive control(MPC)
   4. PID 제어?
3. **시각화**
   1. 사용기술  
      해당 단계는 path tracking을 실행하는 차량의 움직임을 분석하고 피드백하는 단계이다. 여러가지 시각화 도구를 통해 이루어질 수 있으며 사용한 도구는 다음과 같다.

Pygame :

Rviz :

Matplotlib : 그래프를 작성하여 시각화하는 라이브러리로 ~~

* 1. pygame
  2. Rviz
  3. Matplotlib

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

* 개발용 PC
* 170도 어안렌즈 카메라
* 자이카 모터
* 라이다(Lidar)
  + 1. 소프트웨어(SW)
* visual studio
* GitHub
* Notion
* ROS2

|  |  |
| --- | --- |
| 자원 항목 | 계획 및 관리 내용 |
| 개발자 인력 | 주요 기능을 구현할 수 있는 Python 및 ROS 숙련자 확보 |
| 테스터 인력 | 시스템을 실제 환경에서 검증할 인력 배치 |
| 로봇 하드웨어 | XYCAR 및 USB 카메라, 모터 제어 장치 확보 및 점검 |
| 소프트웨어 라이브러리 | OpenCV, ROS, Matplotlib 등 라이브러리 설치 및 의존성 관리 |
| 작업 공간 | 안전하고 충분한 공간에서 테스트 수행 |
| 컴퓨팅 자원 | 고성능 노트북 또는 워크스테이션 확보 |
| 네트워크 연결 | ROS 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 |
| 문서화 도구 | Sphinx, Google Docs, Word 등 문서화 도구 활용 |
| 프로젝트 관리 도구 | Trello, Jira 등을 통한 작업 분배 및 일정 관리 |
| 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 |
| 안전 장비 | 테스트 중 안전 확보를 위한 보호 장비 구비 |
| 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

가상 환경 시뮬레이션 사용해 테스트 진행

Pygame 활용해 시각적 알고리즘 구현 후 주차 알고리즘 테스트

* 1. 품질관리 방안
* 기능 정확성 판단 위해 여러 시뮬레이션 가상환경에서 테스트 진행
* 정확한 Apriltag의 거리 및 방향 측정을 위해 실제 카메라 초점거리에 대한 정보 적용해 시뮬레이션 진행(측정값의 안정성 높임)
* Github 이용해 팀의 코드 공유 및 관리
* 차 이동에 대한 시각적 알고리즘 만들어 주차 알고리즘의 동작 확인

디스플레이 연결

도커 이용

|  |  |
| --- | --- |
| 품질 항목 | 관리 방안 |
| 코드 정합성 | PEP8 등 코딩 표준 준수 |
| 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 |
| 사용자 피드백 | 사용자 관찰 및 의견 수집 |
| 자동화 테스트 | 가능한 테스트는 스크립트화하여 반복 수행 |

* 1. 위험관리 방안

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| No | 위험 요소 | 설명 | 영향도 | 발생 가능성 | 대응 전략 |
| 1 | 카메라 신호 미수신 | ROS 이미지 토픽 수신 실패 | 높음 | 중간 | `wait\_for\_message()` 및 연결 체크 |
| 2 | 카메라 프레임 지연 | 저사양에서 프레임 유실 | 중간 | 중간 | 비동기 처리, 프레임 드롭 감지 |
| 3 | 태그 인식 실패 | 조명/거리/각도로 인식률 저하 | 높음 | 중간 | 태그 재배치, 감도 조절 |
| 4 | 베지어 시각화 오류 | GUI 환경 미지원으로 예외 발생 | 중간 | 높음 | `Agg` 백엔드로 전환 |
| 5 | 쓰레드 충돌 | 플롯 쓰레드와 메인 충돌 | 중간 | 낮음 | `Lock` 사용으로 동기화 |
| 6 | ROS 노드 다운 | 예기치 않은 충돌 발생 | 높음 | 낮음 | `try-except`, 로그 기록 |
| 7 | 모터 통신 불량 | 메시지 송신 누락, 지연 | 높음 | 낮음 | 발행 재시도 로직 |
| 8 | 잘못된 경로 생성 | yaw 계산 오류로 경로 왜곡 | 높음 | 중간 | 각도 제한 및 로그 점검 |
| 9 | 곡률 급변 | 급커브로 차량 제어 불안정 | 높음 | 낮음 | 곡선 평활화 보정 |
| 10 | OpenCV 예외 | 이미지 처리 중 타입 오류 | 중간 | 낮음 | `try-except` 처리 |
| 11 | 태그 수신 지연 | 태그 인식 이후 처리 지연 | 중간 | 중간 | 병렬 처리 최적화 |
| 12 | 카메라 위치 오차 | 물리적 설치 위치 오차 | 중간 | 낮음 | 보정값 적용 |
| 13 | 거리 추정 부정확 | pose.z 사용 시 왜곡 | 중간 | 중간 | 정규화 또는 평균 거리 사용 |
| 14 | 시각화 딜레이 | 플롯 갱신 지연으로 실시간성 저하 | 중간 | 높음 | 업데이트 주기 최적화 |
| 15 | 라벨 누락 | 텍스트 위치 중첩/누락 | 낮음 | 중간 | 겹침 검사 또는 offset 조절 |
| 16 | 플롯 튕김 현상 | `plt.pause()` 문제로 UI 튕김 | 중간 | 낮음 | 예외처리 및 백업 저장 방식 |
| 17 | 데이터 동기화 누락 | `P0`, `P3`, `yaw` 갱신 타이밍 오류 | 높음 | 중간 | `data\_lock` 강제 적용 |
| 18 | 시뮬레이션 불일치 | 실제 동작과 결과 차이 | 중간 | 중간 | 시뮬 기반 경로 검증 병행 |
| 19 | 터미널 출력 누락 | print() 로그 누락 | 낮음 | 중간 | `rospy.loginfo()` 병행 사용 |
| 20 | GUI 메모리 누수 | 반복 그림 삭제 누락 시 누수 발생 | 중간 | 낮음 | `ax.clear()` 및 `plt.close()` 보장 |
| 21 | 코드 호환성 문제 | Python 3.10 이상에서 오류 | 중간 | 낮음 | 버전 명시 및 테스트 |
| 22 | 파라미터 튜닝 미흡 | `camera\_params` 부정확 | 중간 | 중간 | 테스트 데이터로 튜닝 |
| 23 | ROS 토픽 혼동 | 잘못된 토픽명 사용 | 높음 | 낮음 | 런타임 토픽 검사 스크립트 활용 |
| 24 | 회전행렬 오류 | `R.from\_matrix()` NaN 발생 가능 | 높음 | 낮음 | 유효성 검토 후 사용 |
| 25 | 태그 ID 중복 인식 | 두 태그 간 경계 애매할 경우 | 중간 | 낮음 | 최소 거리 조건으로 필터링 |
| 26 | 시각화 중단 | `plt.show()` 호출로 block 발생 | 높음 | 중간 | `plt.ion()` 보장 |
| 27 | ROS 큐 누락 | Subscriber 큐 크기 부족 | 중간 | 중간 | `queue\_size` 증가 |
| 28 | 배터리 전압 강하 | 테스트 중 시스템 불안정 | 높음 | 중간 | UPS 또는 전압 로거 사용 |
| 29 | 예외 미처리 | `imgmsg\_to\_cv2()` 등 오류 방치 | 중간 | 중간 | `rospy.logerr()` 처리 |
| 30 | 비정상 종료 시 플롯 잔존 | `Ctrl+C` 등 종료 시 `plot\_thread` 미정리 | 낮음 | 중간 | `finally:`에서 종료 보장 |

* 1. 유지보수 계획

**유지보수 목표**

* AprilTag 기반 자율주차 시스템의 **지속적인 안정성 확보** 및 **기능 개선**을 위함
* 실제 주차 환경 변화에 대한 **유연한 대응**
* **ROS, OpenCV, Tag 라이브러리 등 외부 의존성 업데이트**에 따른 적응

|  |  |  |
| --- | --- | --- |
| 항목 | 내용 | 점검 주기 |
| Tag 인식 정확도 | 인식률 감소 시 원인 분석 (조도, 카메라 문제 등) | 월 1회 |
| Pose 추정 정확성 | solvePnP 결과값 검증 및 보정 여부 확인 | 월 1회 |
| Path planning 알고리즘 | 베지에 곡선/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 | ROS topic/service 간 딜레이 및 오류 체크 | 주 1회 |
| 환경 변화 반영 | |  | | --- | | Tag 위치 변경, 주차선 위치 변경 대응 |  |  | | --- | |  | | 변경시 |

* 1. 형상관리 계획  
     **형상관리 목적**

시스템의 **버전 이력 추적**, **기능 분기 관리**, **협업 안정성 확보**

개발자 간 충돌 최소화 및 변경 내역 투명성 확보

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

브랜치 관리

|  |  |
| --- | --- |
| Main | Ros 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 진행한다고 가정하고 일정을 계획하였다.

|  |  |
| --- | --- |
| 7.1~7.7 | Ros 개발환경 설정 및 역할 배분 |
| 7.8~7.18 | 팀원 별 알고리즘 개발 수행 |
| 7.19~7.25 | 피드백 및 알고리즘 결함 |
| 7.26~7.30 | 프로그램 최종 수정 및 디버깅 |

1. **예상되는 문제점 및 해결 방안**
   1. 태그의 인식 문제  
      실제 주차를 위하여 주행을 계획하고 실행하는 과정에서 태그를 이용해 태그와 차량 간의 상대위치를 바탕으로 주행한다. 그러나 이 과정에서 태그를 인식하지 못하거나 태그를 인식할 수 있는 범위를 벗어나게 된다면 주행에 큰 문제가 생길 수도 있다. 또한 최초에 주행을 계획할 때 태그가 인식되지 않는다면 주행 계획 수립에 문제가 발생한다. 마지막으로 주차를 마치기 직전에는 태그의 상단 부분이 카메라에 인식되지 않아 차량이 태그를 인식하지 못하게 되는 문제 또한 발생한다.

이에 따른 해결 방안으로는 알고리즘의 보강 및 보완을 들 수 있다.

첫번째로 차량이 태그의 위치를 중간에 놓친다면 마지막으로 포착된 태그의 위치를 바탕으로 상대위치를 추정해 태그 없이 잠깐 주행이 가능하게 하는 알고리즘을 작성할 수 있을 것이다. 태그의 인식 범위에 대한 조사는 위의 자료를 참고하면 될 것이다. 또한 주행 경로를 미리 작성해보는 path planning 과정에서 차량이 인식 범위를 벗어나는지 미리 판별하여 대비할 수도 있을 것이다.

두번째로 최초 경로 작성 시 태그가 인식되지 않는 상황은 두 가지로 분류할 수 있다. 먼저 태그가 차량의 인식 범위 바깥에 있는 경우로 차량을 조정해 가며 태그를 인식하는 알고리즘을 작성할 수는 있겠으나 이는 해당 프로젝트의 취지와 맞지 않다고 판단하여 넘어가도록 하겠다.

다른 경우는 태그가 차량의 인식 범위 내에 있으나 이를 인식하지 못하는 경우이다. 이때는 차량이 태그를 정확히 인식할 때까지 기다려주면 대부분 해결될 문제이다. 따라서 최초 경로 작성 시에 카메라가 태그를 인식하고 알고리즘을 작성하기까지 충분한 시간을 주어야 할 것이다. 이는 해당 프로젝트 특성 상 중간에 경로를 수정하기 어려운 까닭도 있다.

마지막으로 주차의 마무리 단계에서 태그 상단이 잘리는 문제는 주행 경로를 작성할 때 태그 자체의 위치를 목적지로 설정해 경로를 작성하는 것이 아니라 태그에서 태그가 바라보는 방향으로 약간 떨어진 지점을 목적지로 작성한다면 이 문제를 해결할 수 있다. 이 경우 목적지에서 설정한 거리만큼 직선 주행하면 주차가 완료되기 때문에 이를 알고리즘 상에 반영하면 될 것이다.

이때 설정한 목적지에서 태그의 중심이 렌즈의 중심에 위치하는지, 차량의 방향과 태그의 방향이 일치하는지 검사해야 할 것이며 둘 중 하나라도 일치하지 않는다면 올바른 주차를 할 수 없다. 따라서 이들을 검사한 뒤 맞지 않다면 후진 등으로 올바르게 조정하는 알고리즘 또한 필요할 것이다.

* 1. 시뮬레이션과 현실의 스케일 차이 문제
  2. 차량 제어 문제  
     우리가 사용할 ROS 코드에서는 msg의 형태로 XycarMotor의 객체를 생성한 뒤 차량의 속도와 방향 값을 motor\_msg에 담아서 publish 하면 다른 노드가 이를 읽어 실제로 반영하는 방식으로 모터가 제어된다.

이에 따라 우리는 차량을 다룰 때 저 수준의 입출력들은 ROS상에서 추상화되어 처리되기 때문에 신경 쓸 부분이 줄어든다. 이에 따라 하드웨어를 잘못 사용해 문제를 발생시킬 여지도 줄어든다.

그러나 저 수준의 입출력이 필요한 경우가 발생했을 때 이를 해결하기 힘들 수도 있으며 ROS 상에서 예기치 못한 문제가 발생하였을 경우에도 이를 능동적으로 대처하기 힘들다.

그리고 ROS 상에서의 speed 및 angle 토픽 값과 현실 차량에서의 방향, 속도의 괴리가 발생할 수 있다.

이와 같은 문제를 예방하고 해결하기 위해 ROS 상에서의 변수와 토픽, 메시지에 대한 이해가 필요할 것이다. 또한 ROS에 전적으로 의지하기 보다는 ROS 상에서의 입출력이 이루어지는 과정을 이해하는 것도 도움이 될 것이다.  
또한 ROS 상에서 publish 하는 speed와 angle의 값은 기본적으로 현실의 차량에 기반하지만 어디까지나 이상적인 값이므로 scaling을 통해 현실과의 괴리를 줄일 필요가 있다.

* 1. 카메라의 왜곡 문제  
     대회에서 사용하게 될 차량을 비롯해 현실세계의 대부분의 렌즈들은 빛을 굴절시키는 구조적 한계에서 비롯된 왜곡을 유발한다. 이에 따라 이상적으로는 직선들로 이루어진 사각형으로 보여야 할 태그가 둥근 변을 가진 도형으로 식별되기도 하고 차량이 주행해야 할 경로가 휘어져 보이는 등 여러 문제가 발생하게 된다.

이에 따른 해결 방안으로는 차량의 캘리브레이션을 들 수 있다. 이는 초기에 카메라 렌즈의 특성을 프로그램 상에 적용하는 과정으로 렌즈가 측정하는 값과 실제 값 사이의 오차를 보정해 줄 수 있다. 이는 위의 계획 내용에서 설명한 방법을 따라 진행되며 초점거리, 왜곡 계수 등을 측정하여 현실과 렌즈 사이의 괴리와 왜곡을 줄여 줄 수 있다.

다른 방안으로는 프로그램 작성 상에서의 상수의 조정이 있을 수 있겠다. 초기에 태그의 크기, 캘리브레이션 값 등의 상수를 도입할 수 있다면 주행을 여러 번 시도하는 과정에서는 거리 스케일, 방향 스케일 등의 상수를 조정할 수 있다. 이 과정에서 초기의 주행보다 더 나은 주행 결과를 도출할 수 있다.

* 1. 차량 프로세스의 연산량 문제  
     모든 하드웨어는 메모리와 연산 속도의 한계를 가진다. 이때 소프트웨어 상에서 한계를 넘는 연산이나 메모리를 하드웨어에 요구하게 된다면 하드웨어가 우리의 의도와 다른 동작을 하게 되거나 하드웨어의 현재 상태를 정확히 알 수 없게 될 수 있다. 이는 프로젝트에서의 차량에도 똑같이 적용된다.

그리고 이는 비단 중앙처리장치에만 한정된 문제가 아니다. 이를테면 프로그램이 주어진 모터의 성능을 뛰어넘는 출력이나 조향을 요구하거나 센서의 입출력보다 더 큰 입출력을 요구하게 된다면 큰 문제가 발생할 수 있다.

이에 따라 해당 프로젝트에서 프로그램을 작성할 때는 하드웨어의 스펙을 철저히 알아본 뒤 이를 고려하여 작성해야 할 것이다. 또한 프로그램은 차량의 중앙처리장치의 스펙, 성능을 고려하여 작성되어야 할 것이다. 정확성, 성능, 속도 모두를 챙길 수 있다면 더할 나위 없겠지만 이는 현실적으로 불가능하므로 우리는 프로그램의 성능과 속도 사이에서 적절히 타협하여야 한다.

또한 디버깅 단계에서 사용하였던 디버깅용 코드, 센서 이미지, pygame 이미지 등을 비활성화 한다면 더 나은 성능을 보일 것이다.

1. **결과**