제22회 임베디드SW경진대회 개발계획서

[자율주행 레이싱]

□ 개발 개요

○ 요약 설명

팀 명	장덕동왕족발보쌈			
팀장	이민규			
팀원	고광현	유명렬	전규훈	정진솔
목 표	- Adaptive AutoSAR의 구조와 작동 원리, 플랫폼 설정과 통신 스택 학습 - DeepRacer의 카메라와 라이다 센서를 활용, 강화학습을 통해 최단 시간 내에 트랙을 완주			
개발 내용 (요약)	- Adaptive AUTOSAR 표준에 맞춘 Application 개발 - 강화학습 모델을 최적화하여 주행 성능을 향상시키고, 테스트 및 검증한다.			
추가 동영상 여부	있음(　　) / 없음 (　○　)			

○ 개발 목표

- Adaptive AUTOSAR의 구조와 작동 원리를 이해하고, 플랫폼 설정 및 통신 스택 학습을 통한 자율주행 소프트웨어 개발의 기초를 마련한다.
- Amazon SageMaker를 활용해 강화학습 모델의 파라미터 수정을 통해 자율주행 성능을 향상시킨다.
- Amazon DeepRacer와 AA기반 애플리케이션을 통합해 상황 인지, 객체 탐지, 최적 경로 탐색을 구현한다.
- Cloud Native 기반 환경에서 Application을 개발하고, 시뮬레이션을 통해 테스트를 거쳐 AWS DeepRacer에 성공적으로 적용시키는 것을 목표로 한다.

□ 개발 방향 및 전략

○ 기술적 요구 사항

0. 개발 환경

Adaptive AUTOSAR 기반 자율 주행 차량 개발을 위한 개발 환경을 준비한다.

다양한 센서 데이터를 처리하고, 강화 학습을 통해 자율 주행을 구현하며, 서비스 지향 통신 프로토콜을 사용하여 시스템 간의 통신을 관리한다.

운영체제	Ubuntu 20.04
통합 개발 환경 (IDE)	SAAS 기반 팝콘사 Adaptive AUTOSAR 솔루션
프로젝트 형상 관리 시스템	GIT, GitHub를 활용한 버전 관리
프로젝트 관리 도구	JIRA를 통한 애자일 방법론 도입
시뮬레이션 및 테스트 도구	ROS 기반 시뮬레이터
강화 학습 및 모델 관리 도구	Amazon SageMaker, AWS RoboMaker,Amazon S3

1. Adaptive AUTOSAR 이해

1.1 공식 문서 및 표준 문서

1.1.1 Adaptive AUTOSAR 기본 개념과 구조를 설명하는 공식 문서를 준비한다.

1.1.2 Adaptive AUTOSAR에서 제공하는 API와 서비스 목록 및 기능을 학습한다.

1.2 실시간 운영 체제 및 Configuration Management(CM)

1.2.1 POSIX 기반 실시간 운영 체제의 API 활용 방법을 습득한다.

1.2.1 멀티코어 프로세서에서의 멀티스레딩과 동기화 관리 방법을 습득한다.

1.3 Execution Management(EM)

1.3.1 EM 구성 및 애플리케이션 실행 환경 설정 방법을 습득한다.

1.3.2 다양한 리소스 할당 정책 적용을 위한 기법을 학습한다.

1.3.3 성능 데이터 분석 및 피드백 반영을 위한 모니터링 데이터 분석 기술을 습득한다.

1.4 실제 적용 사례

1.4.2 자동차 산업에서의 적용 사례 (예: Bosch, Volvo, BMW)를 파악한다.

1.4.3 AWS RoboMaker의 기술 문서와 API 이해 및 활용 방법을 숙지한다.

2. 서비스 지향 통신 프로토콜에 대한 이해 및 활용

본 프로젝트에서는 DeepRacer의 자율 주행에 필요한 Application간의 실시간 통신을 위해 Adaptive AUTOSAR의 SOME/IP 통신 프로토콜을 활용하여 개발한다.

2.1 SOME/IP (Scalable service-Oriented MiddlewarE over IP)

2.1.1 차량 내 정보들을 실 수요자 위주의 유니캐스트 방식으로 관리한다.

2.1.2 서비스 요청 및 응답, 이벤트 통지 등의 다양한 메시지 유형을 지원하여 서비스 간의 통신을 표준화한다.

2.2 DDS (Data Distribution Service)

2.2.1 QoS 설정, 데이터 분배 전략 기능 제공

2.2.2 실시간 데이터 전송을 위한 신뢰성 및 지연 시간 관리 기능 제공

2.2.3 Adaptive AUTOSAR 환경에서 다양한 애플리케이션 간의 실시간 데이터 교환을 가능하게 한다.

2.3 Service discovery

2.3.1 서비스 제공자는 네트워크에 자신의 서비스를 등록하고, 클라이언트는 이를 검색하여 필요한 서비스를 발견하고 활용한다.

2.3.2 네트워크에서 동적으로 서비스 연결을 관리하는 프로토콜

2.3.3 다양한 메시지 유형과 유니캐스트 방식을 통한 네트워크 효율성을 극대화한다.

3. 센서 데이터 처리 능력

자율 주행 차량의 주행 및 제어를 위해 다양한 센서 데이터를 처리하고 결합하는 능력이 필요하다.

3.1 카메라 및 라이다 센서: 이미지 및 거리 데이터 처리 방법 학습. 카메라 및 라이다 센서의 캘리브레이션을 통해 데이터 정합성을 확보한다.

3.2 OpenCV를 활용한 카메라 이미지 처리: OpenCV 라이브러리를 활용해 이미지 전처리, 객체 탐지 및 추적, 캘리브레이션 기능을 활용한다.
이미지 변환 및 필터링 기술을 사용해 객체를 식별하고 추적한다.

3.3. PCL(Point Cloud Library): PCL을 통해 라이다 센서를 통해 얻은 포인트 클라우드 데이터의 전처리, 필터링, 세그멘테이션 및 특징을 추출한다. 포인트 클라우드 데이터를 활용해 현실과 같은 3D 공간에서 물체의 위치와 형태를 분석할 수 있다.

3.4 센서 퓨전: 라이다와 카메라 센서 데이터를 통합하여 더 정확한 환경 정보를 생성한다. 이를 위해 칼만 필터, 파티클 필터 등 센서 퓨전 알고리즘을 활용할 수 있다.

4. 객체 탐지 및 강화 학습

차량이 차선에 맞춰 주행하고, 장애물을 회피하기 위해 객체 탐지 및 차선 인식 기술과 효율적으로 주행하기 위한 강화 학습이 필요하다.

4.1 객체 탐지

4.1.1 Python 및 OpenCV를 사용한 이미지 처리: OpenCV를 사용하여 이미지 전처리, 객체 탐지 및 추적 기능을 구현해 이미지 변환, 필터링을 통해 객체를 식별하고 추적한다.

4.1.2 Lidar 센서와 함께 전방 장애물 탐지: 라이다 센서를 활용하여 주변 환경의 3D 포인트 클라우드를 생성하고, 이를 기반으로 장애물의 위치와 크기를 파악한다.
PCL(Point Cloud Library)을 사용하여 라이다 데이터의 전처리, 필터링,
세그멘테이션 및 특징 추출을 수행한다.

4.1.3 차선 인식 알고리즘 및 변환 기법을 사용하여 정확한 차선을 인식하고 주행: Hough 변환, Canny 엣지 검출 등의 알고리즘을 사용하여 차선을 인식한다.
이미지 변환 기법을 통해 주행 경로를 추정하고 차량이 차선을 따라 주행할 수 있도록 한다.

4.1.4 멀티스레딩 및 병렬 처리 기법: 실시간 데이터 처리 기법을 적용하여 여러 센서에서 동시에 데이터를 처리하고, 빠르게 반응할 수 있도록 한다.

4.2 강화 학습

4.2.1 Amazon SageMaker: Amazon SageMaker를 사용하여 모델을 학습시키고, 학습된 모델을 Amazon S3에 저장한다. 배치 크기, 탐색 계수 등 하이퍼 파라미터를 조정하여 모델의 성능을 최적화한다.

4.2.2 강화 학습의 실제 적용: AWS DeepRacer를 통해 강화 학습 모델을 구축해 시뮬레이션 환경에서 주행 데이터를 기반으로 모델을 학습한다.
실제 차량 환경에서 Adaptive Application에 빌드해 학습된 모델을 테스트하고 성능을 평가한다.
실제 주행 테스트를 통해 강화 학습 모델의 성능을 최적화한다.

○ 개발 방법

이 프로젝트는 Amazon DeepRacer의 하드웨어와 ROS2, Adaptive AUTOSAR 등의 소프트웨어 플랫폼을 통합하여 자율 주행 차량의 주행 성능을 최적화하는 것을 목표로 하며,
우리 팀은 설계-개발-검증 단계로 나눠 각 스프린트 단위로 진행할 예정이다.

1. 설계

1.1 시스템 아키텍처

1.1.1 하드웨어 및 소프트웨어 통합: 카메라, 라이다 센서 등 DeepRacer의 구성 요소와 ROS2, SaaS 기반 Adaptive AUTOSAR 등의 소프트웨어 플랫폼을 통합하는 방법을 설계한다.

1.2 모듈 및 인터페이스 정의

1.2.1 모듈화 설계: 센서 데이터 처리, 강화 학습, 데이터 통신, 차량 제어 등의 기능을 독립적인 모듈로 분리하여 설계한다.

1.2.2 인터페이스 정의: 모듈 간 데이터 교환을 위한 인터페이스를 명확히 정의한다. 각 모듈이 주고받을 데이터 형식과 프로토콜을 설계한다.

1.2.3 API 설계: 각 모듈의 기능을 호출할 수 있는 API를 설계하고, 이를 통해 모듈 간 상호 작용을 구현한다.

1.3 데이터 흐름 및 통신 구조 설계

1.3.1 데이터 흐름 설계: 센서에서 수집된 데이터가 각 모듈을 거쳐 처리되는 과정을 시각적으로 표현한다.
데이터의 흐름을 명확히 하여 각 모듈의 역할을 정의한다.

1.3.2 통신 구조 설계: 모듈 간 데이터 통신을 위한 구조를 설계한다.
SOME/IP, DDS 등의 통신 프로토콜을 사용하여 안정적인 데이터 전송을 구현한다.

1.3.3 실시간 데이터 처리: 실시간 데이터 처리를 위한 멀티스레딩 및 병렬 처리 기법을 적용한다.

2. 개발 및 구현

2.1 센서 데이터 처리 및 퓨전

2.1.1 데이터 수집 및 전처리: 카메라와 라이다 센서로부터 데이터를 수집하고, 이를 전처리하여 일관된 형식으로 변환한다.

2.1.2 센서 퓨전 알고리즘 개발: OpenCV와 PCL 라이브러리를 사용하여 카메라와 라이다 데이터를 결합한다. 객체탐지 및 차선 인식을 위한 알고리즘을 개발한다.

2.2. 강화 학습

2.2.2 데이터 전처리: 강화 학습 모델에 필요한 데이터를 전처리한다. 이미지 데이터를 Grayscale로 변환하고, Bird-eye View 변환을 통해 도로 이미지를 정규화한다.

2.2.3 모델 학습: Amazon SageMaker를 사용하여 강화 학습 모델을 학습시킨다.
SAC 등의 알고리즘을 적용하여 주행 전략을 최적화한다.

2.2.4 모델 배포: 학습된 모델을 Amazon S3에 저장하고, 이를 Aplication에 AWS RoboMaker를 통해 시뮬레이션 및 실제 차량에 배포한다.

2.3 데이터 통신

2.3.1 통신 프로토콜 구현: Adaptive AUTOSAR의 SOME/IP, DDS 등을 사용하여 모듈 간 데이터를 송수신한다.

2.3.2 이벤트 기반 통신: 서비스 제공자(Skeleton)와 소비자(Proxy) 간의 동적 연결을 통해 이벤트 기반 데이터 교환을 구현한다.

2.4 차량 제어

2.4.1 주행 경로 선택: 강화 학습된 모델을 기반으로 최적의 주행 경로를 선택한다.

2.4.2 조향 및 속도 제어: Navigation AA와 Servo AA를 통해 차량의 조향각과 속도를 제어한다.

3. 시뮬레이션 및 검증

3.1 시뮬레이션 테스트

3.1.1 시나리오 테스트: ROS 기반 시뮬레이터에서 여러 주행 시나리오를 생성하고 테스트를 수행한다. 장애물 위치 변경 등 다양한 조건에서의 성능을 평가한다.

3.1.2 데이터 분석: 시뮬레이션 결과 데이터를 수집하고 분석하여 성능을 평가해 문제점을 파악하고 파라미터 수정 등을 통해 문제를 해결한다.

3.2 실제 환경에서 테스트

3.2.1 초기 테스트: 시스템이 예상대로 작동하는지 확인한다. 실제 트랙에서 초기 주행 테스트를 수행하여 시스템의 기본 기능을 확인한다.

3.2.2 반복 테스트: 반복적으로 주행 테스트를 수행하여 발생하는 문제를 수정한다. 장애물 위치를 변경하며 다양한 조건에서의 성능을 평가한다.

3.3 성능 평가 및 최적화

3.3.1 주행 성능 평가: 주행 속도, 안정성, 트랙 유지 능력 등을 평가한다. 성능 개선을 위한 피드백을 반영한다.

3.3.3 최적화: 알고리즘, 코드, 하드웨어, 데이터 처리 과정 등을 최적화해 성능을 향상시킨다. 병목현상을 제거하기 위해 병렬 처리 기법 등을 적용한다.

○ 예상되는 장애요인 및 해결방안

1. 고속 주행 중 다양한 센서로부터 들어오는 대량의 데이터를 실시간으로 처리해 처리 속도가 느리면 주행 성능에 문제가 생길수 있음.

1.1 해결방안: 데이터를 병렬로 처리하여 처리 속도를 향상시킨다. 예를 들어, OpenMP 또는 CUDA와 같은 병렬 프로그래밍 기법을 사용하여 연산을 분산시킨다.
데이터 구조와 알고리즘을 최적화하여 처리 시간을 단축한다.

2. 센서에서 수집한 데이터가 왜곡되거나 노이즈가 포함될 수 있어 판단이 어려울 수 있다.

2.1 해결방안: 수집된 데이터를 전처리하여 노이즈를 제거하고, 필터링 기법을 통해 데이터를 정제한다.

3. 통신 문제 - 네트워크 환경에서 데이터 전송 지연이나 손실이 발생할 수 있다.

3.1 해결방안: SOME/IP 및 Service Discovery의 최적화 과정을 거친다. 패킷 우선 순위 설정, QoS(Quality of Service) 적용 등을 통해 데이터 전송의 안정성을 높인다.

○ 기타 추가 내용

- 프로젝트 관리: Agile 방법론을 활용한 주기적인 검토 및 피드백 반영

1. 주기적인 스프린트 계획: JIRA를 활용해 2주 간격으로 스프린트를 계획하고, 각 스프린트가 끝날 때마다 검토 회의를 통해 진행 상황을 평가하고 피드백을 반영한다.

2. 정기적인 회의 및 문서화를 통한 정보 공유, 교육 및 학습 최신 기술 동향 파악 예정

□ 영상처리 및 센서 기술 활용 방안 및 공부 내용

○ 영상처리 기술 활용 방안 및 공부 내용

1. OpenCV를 활용한 차선 인식
   1.1 Canny Edge Detection과 Hough 변환을 사용한 차선 검출
      1.1.1 흑백 변환: 입력된 이미지를 흑백으로 변환하여 처리한다.
      1.1.2 에지 추출: Canny Edge Detection 기법을 이용하여 에지를 정밀하게 추출하며, 잡음을 제거한다.
      1.1.3 관심 영역 설정: 차선을 식별하기 위해 관심 영역을 설정한다.
      1.1.4 직선 성분 추출: Hough 변환을 통해 에지에서 직선 성분을 추출한다.
      1.1.5 차선 분리 및 추정: 추출한 직선 성분 중에서 좌우 차선에 해당할 가능성이 높은 직선들을 분리하고, 선형 회귀 분석을 통해 가장 정확한 차선을 추정한다.
      1.1.6 진행 방향 예측: 이를 토대로 자동차의 진행 방향을 예측한다.
2. Camera Calibration과 Bird-eye View, 슬라이딩 윈도우를 활용한 곡선 차선 검출
   2.1 Camera Calibration
      2.1.1 목적: 카메라 렌즈로 인한 이미지 왜곡을 상쇄한다.
      2.1.2 방법: 체스보드 패턴을 사용하여 여러 각도에서 이미지를 캡처한 후, 카메라 행렬과 왜곡 계수를 계산한다. 이러한 보정 파라미터를 이용해 원래의 왜곡된 이미지를 보정한다.
   2.2 Bird-eye View
      2.2.1 목적: 도로 이미지를 하향식 보기로 변환해 차선의 곡률, 도로 형상을 분석한다.
      2.2.2 방법: 원근 변환(perspective transform)을 사용해 변환할 영역의 4개의 점을 지정하고, 변환 후의 직사각형 형태의 4개의 점을 지정하여 변환 행렬을 계산한 후 이미지 변환한다.
   2.3. 이미지 임계값, 이진 이미지 조절
      2.3.1 목적: 차선이 두드러지도록 이미지를 처리하여 검출을 용이하게 한다.
      2.3.2 방법: Grayscale 이미지를 사용해 가우시안 블러와 엣지 검출을 거친 후 적절한 임계값을 설정하여 이진화를 수행한다.
   2.4 슬라이딩 윈도우
      2.4.1 목적: 차선의 위치를 찾아내기 위해 한 윈도우 안에서 차선 픽셀을 추적한다.
      2.4.2 방법: 이진 이미지에서 히스토그램을 사용하여 차선의 기초 위치를 찾은 뒤 슬라이딩 윈도우를 사용해 이미지의 아래에서 위로 차선 픽셀을 추적한다. 검출된 픽셀을 기반으로 차선을 폴리피팅하여 곡선 차선을 생성한다.

○ 센서 기술 활용 방안 및 공부 내용

1. 카메라 데이터 수집

1.1 전방 시야 확보: DeepRacer의 전방 카메라를 활용하여 주행 환경을 모니터링해 전방의 차선, 장애물 등을 실시간으로 감지할 수 있다.

1.2 실시간 환경 인식: 카메라는 초당 여러 프레임을 캡처하여 실시간으로 주변 환경을 인식한다. 이를 통해 자율 주행 차량이 주행 중 발생하는 다양한 상황에 신속하게 반응할 수 있다.

1.3 데이터 처리: OpenCV 라이브러리를 사용하여 카메라 데이터를 수집하고 전처리한다. 전처리 단계에서 이미지의 해상도를 조정하고, 색상 변환 및 노이즈 제거 등의 작업을 수행한다.

2. 라이다 데이터 수집

2.1 환경 스캔: DeepRacer의 라이다 센서를 사용하여 360도 주변 환경을 스캔한다. 라이다 센서는 레이저 펄스를 방출하고 반사된 신호를 수집하여 주변 물체의 위치와 거리를 측정한다.

2.2 포인트 클라우드 데이터 수집: 라이다 센서로부터 실시간으로 포인트 클라우드 데이터를 수집하여, 차량 주변의 거리와 장애물 정보를 획득한다. 포인트 클라우드 데이터는 3D 공간에서 물체의 위치를 정밀하게 파악할 수 있는 중요한 데이터이다.

2.3 데이터 처리: PCL(Point Cloud Library)을 사용하여 라이다 데이터를 처리한다. PCL을 통해 포인트 클라우드 데이터를 필터링 및 클러스터링해 유의미한 정보로 변환한다.

3. 객체 인식

3.1 딥러닝 모델 사용: YOLO(You Only Look Once) 또는 SSD(Single Shot MultiBox Detector)와 같은 딥러닝 모델을 사용하여 이미지에서 객체를 실시간으로 인식한다. 이러한 모델은 높은 정확도와 빠른 처리 속도로 자율 주행 차량에 적합하다.

3.2 OpenCV 처리: 딥러닝 모델의 추론 결과를 OpenCV를 통해 후처리하여 객체의 위치와 크기를 파악해 객체 인식 결과를 시각화하고 추가적인 분석을 수행한다.

4. 거리 측정

4.1 위치 매칭: 인식된 객체의 위치를 라이다 데이터와 매칭시켜 거리 측정을 수행한다. 이를 통해 객체와의 정확한 거리를 파악할 수 있다.

4.2 중심점 추출: 포인트 클라우드 데이터에서 객체의 중심점을 추출하여 카메라 이미지와 연관시킨다. 이 과정은 객체의 위치 정보를 더욱 정확하게 제공한다.

4.3 삼각 측량: 삼각 측량 기법을 사용하여 객체와의 정확한 거리를 계산한다. 카메라와 라이다 센서의 데이터를 결합하여 더 정밀한 거리 정보를 얻는다.

5. 카메라-라이다 캘리브레이션

목적: 카메라와 라이다 센서의 데이터를 정확히 결합하여 객체의 위치와 거리를 정밀하게 측정하기 위함이다.

5.1 캘리브레이션 절차: 카메라와 라이다 센서의 위치와 방향을 맞추기 위해 여러 가지 측정과 조정을 수행한다. 체스보드 패턴 등을 사용하여 카메라의 내재적 매개변수를 보정하고, 라이다 센서와의 외재적 매개변수를 조정한다.

5.2 매칭 알고리즘: 각 센서에서 수집한 데이터를 동일한 좌표계로 변환하여 매칭해 두 센서의 데이터를 통합하여 하나의 일관된 3D 모델을 생성한다.

□ Adaptive AUTOSAR 표준 기반 Adaptive Application(AA) 개발 방안

○ 개발 방향 및 전략

1. 목표 설정

1.1 AWS DeepRacer의 주행 성능을 최적화하여 경주 트랙에서 안정적이고 빠르게 주행할 수 있는 자율주행 애플리케이션을 개발한다.

1.2 Adaptive AUTOSAR 표준을 준수해 확장성과 유연성을 갖춘 자율주행 시스템을 구축한다.

2. 단계별 개발 전략

2.1 1단계: 설계

2.1.2 Adaptive AUTOSAR 기반의 소프트웨어 아키텍처를 설계한다.

2.1.2 AWS DeepRacer의 하드웨어 및 소프트웨어 스펙을 분석한다.

2.1.3 강화 학습 알고리즘을 활용한 주행 프로세스 개발 요구 사항을 도출한다.

2.2 2단계: 애플리케이션 개발 및 통합

2.2.1 알고리즘을 바탕으로 주행 프로세스를 설계한다.

2.2.2 Adaptive AUTOSAR 플랫폼에 맞춘 애플리케이션을 개발 및 통합한다.

a. SensorFusionAA (Camera, Lidar AA): 카메라 및 라이다에서 수집된 데이터를 읽어 각각 'Image'와 'Lidar Value'를 생성하고, 융합하여 'Sensor Fusion Value'를 생성한다.

b. Inference AA: 융합된 센서 데이터를 입력으로 받아 AI 모델을 통해 추론을 실행하고 'Inference Result'를 생성한다.

c. Navigation AA: 추론 결과를 바탕으로 주행 경로 및 속도 제어를 위한 스로틀 각을 계산한다.

d. Servo AA: 주행 명령을 받아 실제 차량의 스티어링, 스로틀 제어를 수행한다.

2.3 3단계: 시뮬레이션 및 테스트

2.3.1 AWS RoboMaker를 활용한 시뮬레이션 환경에서 알고리즘을 테스트해 개선한다.

2.3.2 실제 주행 테스트를 통해 성능을 검증하고 피드백을 반영한다.

2.4 4단계: 최종 검증

2.4.1 최종 자율주행 애플리케이션의 통합 테스트를 수행한다.

2.4.2 Adaptive AUTOSAR 표준 준수 여부를 확인하고 최종 점검한다.

○ 관련 기술 공부 내용 및 적용 방안

1. Adaptive AUTOSAR 모델링

1.1 Adaptive AUTOSAR는 차량용 제어 애플리케이션에 대한 표준을 제공한다.

1.2 ARXML: AUTOSAR의 설계 모델을 XML 형식으로 표현한 것으로, 소프트웨어 구성 요소, 통신 인터페이스, 데이터 타입 등을 포함한다. Adaptive AUTOSAR에서 ARXML을 기반으로 시스템 설계를 진행하며, 이를 바탕으로 소스 코드를 생성한다.

1.2.1 AUTOSAR BlockSet을 사용해 ara API를 생성한다.

1.2.2 ARXML 설계 시 주의 사항:

a. standard C++ Data type을 제외한 CppImplementationDataType의 namespace를 작성한다.

b. ServiceInterface의 namespace를 작성한다.

c. 1개 AdaptiveApplicationSwComponentType의 1개 Port(Pport, Rport)에

reference된 ServiceInterface의 namespace는 중복 사용을 금지한다.

d. CppImplementationDataType에서 구조체를 사용할 경우, 구조체 내부 변수명과 TypeReference 이름의 중복 사용을 금지한다.

1.3 ROM/RAM 사용 최소화: 1개 Event마다 1개 ServiceInterface를 만들면 AA의 ROM/RAM 사이즈가 증가하므로, 최대한 1개 ServiceInterface에 Event를 통합하여 자동 생성되는 소스 코드 양을 줄인다.

1.4 이더넷 지연 최적화: ServiceInterface에 일반 Data를 각 Event로 설계하는 것보다, 구조체로 1개 Event에 통합하는 것이 SOME/IP 통신 횟수를 줄일 수 있다.

2. Adaptive Application의 통신 프로토콜 및 적용 방안

2.1 서비스 지향 통신 (Service-Oriented Communication, SOC): Adaptive AUTOSAR는 SOC를 통해 서비스 제공자(Skeleton)와 소비자(Proxy) 간의 연결을 관리한다.

2.1.1 Skeleton: 특정 기능을 구현하고, 네트워크를 통해 이를 클라이언트에 전달한다.

2.1.2 Proxy: 네트워크에서 필요한 서비스를 찾아 호출하여 데이터를 이용한다.

2.2 SOME/IP: 자동차 산업에서 사용되는 서비스 지향 통신 프로토콜로, 차량 내부의 다양한 ECU 간에 효율적인 통신을 제공한다.

2.2.1 서비스 구성 요소:

a. Event: 데이터 변경 시 또는 주기적으로 데이터를 전달하고, 전달받는다.

b. Method: 서버에 Request하고 해당 Method의 결과를 Response 받는다.

c. Field: Client가 Getter나 Setter를 전송하면, 그에 따른 결과를 Server가 전송하고, Notifier를 통해 Client에 이벤트에 따른 데이터를 전송한다.

2.3 Service Discovery: 네트워크의 각 AA가 자신의 서비스를 필요로 하거나, 자신이 필요한 서비스를 가지고 있는 AA를 동적으로 탐색하고 연결하는 프로토콜.

2.3.1 UDP 멀티캐스트 방식으로 Find Service, Offer Service를 호출한다. 이 과정을 통해 필요한 Service를 가지고 있는 Server와 Client가 단독으로 통신할 수 있는 Channel을 생성하고, 유니캐스트 방식으로 1:1 통신을 진행한다.

2.3.2 Client가 Subscribe로 Server에 등록을 요청한다. 이후 Server가 Event 등록이 가능한지 SubscribeAck로 응답한다.

2.4 프로젝트 적용 방안: AWS DeepRacer는 다양한 데이터의 송수신이 필요하고, 그에 따른 명령 값을 차량에 전달해야 한다.

2.4.1 이러한 데이터 송수신과 명령 전달 과정을 SOME/IP 프로토콜 기반 서비스로 구성하여 이를 통한 AA 간의 데이터 송수신 및 상호작용을 개발하고 활용할 수 있다.

○ 기타 추가 내용

1.각 모듈의 적용 방안

Camera AA: OpenCV와 같은 이미지 처리 라이브러리를 사용하여 이미지 데이터 전처리 및 전달.

Lidar AA: PCL(Point Cloud Library)을 사용하여 라이다 데이터 전처리 및 전달.

Sensor Fusion AA: 칼만 필터 등의 센서 융합 알고리즘을 사용하여 데이터 융합.

Inference AA: TensorFlow, PyTorch 등의 딥러닝 프레임워크를 사용하여 학습된 AI 모델을 로드하고 실시간 추론 수행.

Navigation AA: 주행 경로 생성 알고리즘 및 PID 제어기를 사용해 차량 제어 명령 생성.

Servo AA: 하드웨어 제어 라이브러리를 사용하여 실제 차량의 스티어링 및 스로틀 제어.

□ 개발 일정

No	내용	2024年				
		7月	8月	9月	10月	11月
1	프로젝트 초기 설정 및 요구사항 분석					
2	아키텍처 설계, 하드웨어 및 소프트웨어 분석					
3	카메라 및 라이다 데이터 처리 개발					
4	모듈 개발 및 통합					
5	Servo AA 개발					
6	Inference AA 개발					
7	Naviation AA 개발					
8	통합 테스트 및 디버깅					
9	시뮬레이션 및 실제 주행 테스트					
10	최종 검증 및 마무리					