제22회 임베디드SW경진대회 개발완료보고서

[자율주행 레이싱]

□ 개발 개요

○ 요약 설명

팀 명	장덕동왕족발보쌈
목 표	- Adaptive AUTOSAR의 구조와 작동 원리, 플랫폼 설정과 통신 스택 학습 - AWS Deepracer의 카메라와 라이다 센서를 활용해 강화
개발 내용 (요약)	- Adaptive AUTOSAR 솔루션(AUTOSAR.IO)를 활용한 ARXML 설계 - Adaptive Application 개발 - 강화학습 모델을 활용한 DeepRacer의 자율주행 구현
소스코드	https://github.com/jangduckdong/esw-contest-automobility-2024

○ 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 통한 AWS DeepRacer 에 성공적으로 적용시키는 것을 목표로 한다

□ 개발 방향 및 전략

○ 기술적 요구 사항

본 프로젝트는 Adaptive AUTOSAR 표준 기반의 자율주행 차량 애플리케이션 개발을 목표로 하며, 카메라와 라이다 센서 데이터를 실시간으로 처리하고, 강화 학습을 통해 주행 성능을 최적화하는 것이다. 이를 위해 다음과 같은 개발 환경을 활용했다.

1. 개발 환경

1.1 설계 환경

Adaptive AUTOSAR 표준 기반의 어플리케이션 개발 위해, PopconSAR의 설계 솔루션인 AutoSAR.io를 활용해 ARXML 파일을 설계했다. 이 도구를 활용해 각 AA(Adative Appication)의 인터페이스와 통신 구조를 정의하고, 서비스 지향 아키텍처를 기반으로 모듈 간의 데이터 흐름을 설계했다. 카메라, 라이다, 센서 융합, 추론, 내비게이션 AA의 역할과 상호 작용을 명확히 규정해 시스템의 일관성과 확장성을 확보했다.
AutoSAR.io의 활용으로 ARXML 파일을 효율적으로 생성하고 관리할 수 있었으며, 이를 통해 Adaptive AUTOSAR 표준에 부합하는 시스템 아키텍처를 구축했다. 또한 웹 기반의 인터페이스를 통해 협업과 버전 관리를 용이하게 해 개발 과정의 효율성을 높였다.

1.2 개발 환경

개발 과정에서는 VSCode를 주요 통합 개발 환경으로 사용했다. Git과 Github를 활용해 형상 관리를 체계적으로 진행했다. 또한 Popconsar의 PARA를 통해 EM 과 SM을 제공받아 플랫폼 레벨에서 필요한 애플리케이션을 구동시키는데 활용했다.

1.3 테스트 환경

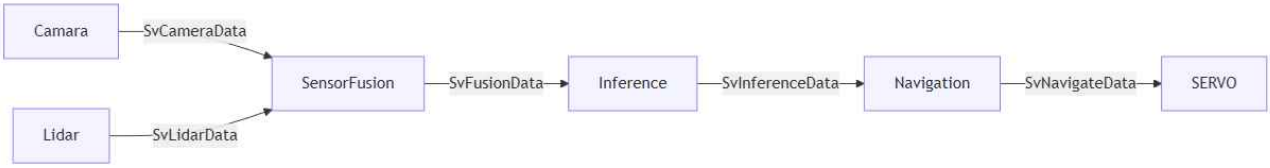
테스트 단계에서 ROS 기반 시뮬레이터를 활용해 모듈간 통신과 강화학습 모델의 주행 성능을 검증했다. 다양한 주행 시나리오를 통해 장애물 인식, 경로 추적 등 성능을 확인했으며, 데이터 송수신의 지연 및 오차를 최소화하기 위해 노력했다.

1.4 실차 환경

개발한 AA를 AWS DeepRacer와 통합해 자율주행 성능을 테스트했다. 카메라와 라이다 센서 데이터를 기반으로 주행 성능을 측정하고, 실제 주행 트랙에 방문해 강화 학습 모델의 성능을 정성적으로 평가하여 장애물 회피와 경로 유지 능력을 확인했다. 실제 주행 중 발생할 수 있는 데이터 처리 속도 문제를 사전에 분석하고, 각 모듈이 신속하게 반응할 수 있도록 최적화하여 자율주행 성능을 극대화했다.

○ 개발 방법

1.설계



Adaptive Application 설계는 각 AA 간의 효율적인 통신과 데이터 처리를 중심으로 구성했다. 각 AA는 특정 기능을 담당하며, 서비스 지향 아키텍처 개발 방식을 채택해 시스템의 확장성과 유지보수성을 높일 수 있게 설계했다.

1.1. Adaptive Application 상세 설계

1.1.1 Camera Application

기능: 차량 주변의 이미지를 실시간으로 캡처하여 객체 인식 및 주행 상황판단에 필요한 데이터를 제공한다.

통신 방식: SOME/IP 프로토콜을 사용해 SensorFusion 모듈로 이벤트 기반의 SvCameraData를 전송한다.

1.1.2 Lidar Application

기능: 라이다 센서를 활용해 주변 물체와의 거리를 측정 및 위치 데이터를 제공한다.

통신 방식: SOME/IP 프로토콜을 사용해 Sensor Fusion 모듈로 이벤트 기반의 SvLidarData를 전송한다.

1.1.3 SensorFusion Application

기능: 카메라와 라이다에서 수신한 데이터를 통합해 추론에 필요한 데이터 형식을 제공한다.

통신 방식: SOME/IP 프로토콜을 사용하여 INFERENCE 모듈과 요청/응답 방식으로 SvFusionData를 주고받는다.

1.1.4 Inference Application

기능: 센서 데이터를 기반으로 강화 학습 모델에 적용해 객체 인식, 주행 경로 예측 등의 기능을 제공한다.

통신 방식: SOME/IP 프로토콜을 통해 NAVIGATION 모듈로 이벤트 기반의 SvInferenceData를 전송한다.

1.1.5 Navigate Application

기능: Inference Aplication에서 제공한 정보를 활용해 최적의 주행 경로를 생성한다.

통신 방식: SOME/IP 프로토콜을 사용하여 SERVO 모듈로 이벤트 기반의 SvNavigateData를 전송한다.

1.1.6 Servo Application

기능: 추론 모듈에서 제공한 정보를 활용해 최적의 주행 경로를 생성하고, 실제 Deepracer에서 모터를 작동하게 해 주행을 실행한다.

2.1 Adaptive Application 서비스 인터페이스

각 모듈간의 통신은 SOME/IP 프로토콜을 활요해 모듈 간의 통신을 구현했다. SOME/IP는 서비스 지향 통신을 위한 전송 프로토콜로, 서비스 디스커버리와 결합되어 모듈 간의 동적 연결을 지원한다.

2.1.1 SvCameraData

기능: 실시간으로 캡처한 이미지 데이터를 이벤트 형태로 SensorFusion 모듈에 전송한다. 이벤트 기반 통신을 통해 새로운 이미지가 생성될 때마다 자동으로 데이터가 전달된다.

전송 데이터: CameraDataNode(Struct)

2.1.1 SvLidarData

기능: 주변 환경의 정보를 이벤트 형태로 SensorFusion 모듈에 전송한다. 이벤트 기반 통신을 통해 새로운 라이다 데이터가 생성될 때마다 자동으로 데이터가 전달된다.

전송 데이터: LidarDataNode(Struct)

2.1.1 SvFusionData

기능: 카메라와 라이다 데이터를 통합하여 생성된 환경 정보를 Inference 모듈에 요청/응답 방식으로 제공한다. 이 방식은 Inference 모듈이 필요할 때마다 데이터를 요청하고, SensorFusion 모듈이 해당 데이터를 응답으로 제공하는 구조로 구현했다.

전송 데이터: SensorFusionNode(Struct)

2.1.1 SvInferenceData

기능: SensorFusionNode를 기반으로 분석한 결과를 이벤트 형태로 Navigation 모듈에 전송한다. 이벤트 기반 통신을 통해 새로운 추론 결과가 생성될 때마다 자동으로 데이터가 전달된다.

전송 데이터: InferenceNode(Struct)

2.1.1 SvNavigateData

기능: 추론 모듈에서 제공한 정보를 활용하여 생성된 주행 경로 데이터를 이벤트 형태로 Servo 모듈에 전송한다. 이벤트 기반 통신을 통해 새로운 경로 정보가 생성될 때마다 자동으로 데이터가 전달된다.

전송 데이터:NavigateNode(Struct)

2.2 서비스 인스턴스와 머신과의 매핑

서비스 인스턴스 매핑은 각 모듈의 데이터 흐름을 관리하고, 각 서비스 인스턴스를 네트워크 상의 특정 포트에 매핑하여 효율적인 통신을 가능하게 한다. 이에 대한 구체적인 매핑은 다음 표로 갈음한다.

shortName	TCP/UDP	ServiceInstance
Mapping_Provider_SvLidarData_AALidar	21002	provider_SvLidarData_AALidar
Mapping_Provider_SvFusionData_AASensorFusion	21003	Provider_SvFusionData_AASensorFusion
Mapping_Provider_SvInferenceData_AAInference	21004	Provider_SvInferenceData_AAInference
Mapping_Provider_SvNavigateData_AANavigate	21005	Provider_SvNavigateData_AANavigate
Mapping_Provider_SvFusionData_AAS	21006	Provider_SvFusionData_AASimulato

imulator		r
Mapping_Consumer_SvCameraData_AASensorFusion	21007	Consumer_SvCameraData_AASensorFusion
Mapping_Consumer_SvLidarData_AASensorFusion	21008	Consumer_SvCameraData_AASensorFusion
Mapping_Consumer_SvFusionData_AAInference	21009	Consumer_SvCameraData_AASensorFusion
Mapping_Consumer_SvInferenceData_AANavigate	21010	Consumer_SvInferenceData_AANavigate
Mapping_Consumer_SvNavigateData_AAServo	21011	Consumer_SvNavigateData_AAServo

2.1 DataType 설계

Adaptive AUTOSAR 기반 자율주행 시스템의 데이터 처리와 통신을 위해 각 모듈에서 사용되는 데이터 타입을 구조체로 정의하였다. 데이터의 일관성과 형식을 유지하며, 각 모듈 간의 원활한 데이터 전송을 지원한다. 다음은 각 데이터 타입과 그 구조체에 대한 설명이다.

생성한 구조체 타입	멤버변수	멤버변수 타입
CameraDataNode	camera_data0, camera_data1, timestamp	UcharVector;Vector, UcharVector;Vector, int64_t;Value
LidarDataNode	timestamp, lidar_data	int64_t;Value, LidarDataVector;Vector
SensorFusionNode	lidar_data, timestamp, camera_data0, camera_data1	LidarDataVector;Vector, int64_t;Value, UcharVector;Vector, UcharVector;Vector
InferenceData	class_label, class_prob, x_min, y_min, x_max, y_max	int32_t;Value, float;Value, float;Value, float;Value, float;Value, float;Value
InferenceDataNode	timestamp, inference_data	int64_t;Value, InferenceDataVector;Vector
NavigateDataNode	angle, throttle, timestamp	float;Value, float;Value, int64_t;Value

2. 개발

각 AA의 개발은 Perception, Decision, Execution 세단계로 나누어 개발했으며, 서로 독립적이면서 유기적으로 작동하여 차량이 실시간으로 데이터를 수집하고 이를 기반으로 주행 결정을 내리도록 개발했다.

2.1 Perception 모듈 (Camera, Lidar, SensorFusion)

Perception 모듈은 Deepracer의 인식 성능을 강화하기 위해 Camera, Lidar, SensorFusion 세 가지 Adaptive Application으로 구성했다. 차량이 주행 중 실시간으로 주변 환경을 인식하고 안전하게 주행할 수 있도록, Camera, Lidar 센서로부터 수집한 데이터를 통합해 종합적인 인식 정보를 제공한다.

2.1.1 Perception 모듈 주요 개발 전략

a. 비동기 데이터 처리 및 멀티스레드 최적화

- Perception 모듈은 높은 성능을 유지하기 위해 비동기 작업과 멀티스레딩을 적극 활용해 각 센서의 데이터를 병렬로 처리한다. 이를 통해 데이터 수집과 전송 과정의 지연을 최소화하고, 주행 중 빠르게 변하는 환경 정보를 실시간으로 수집하여 정확한 인식 데이터를 제공한다.
- m_workers 객체를 사용해 각 AA의 주요 기능이 비동기로 실행되며, 이를 통해 센서 융합 과정에서 발생할 수 있는 병목 현상을 줄이고, 데이터 처리 효율성을 극대화했다.

b. 타임스탬프 기반 데이터 동기화

Camera와 Lidar에서 제공하는 데이터는 각 프레임에 타임스탬프가 포함되어 SensorFusion 모듈에서 동기화된다. 이러한 동기화는 데이터의 일관성을 보장하며, 주행 중 정확한 인식 결과를 제공한다. 이를 통해 차량이 실시간으로 상황을 파악해 안정적인 주행을 이어갈 수 있게 했다.

2.1.2 Camera AA

차량 전방의 이미지 데이터를 수집해 객체 인식과 차선 검출에 필요한 정보를 제공한다.
camera.cpp 및 cameradata.cpp에서 구현했으며, OpenCV의 라이브러리를 활용해 카메라의 실시간 프레임을 캡쳐하고, JPEG 형식으로 인코딩한 후, SensorFusion 모듈에 전달한다.

a. 주요 메서드

- ScanCameraIndex()

  메서드를 활용해 다중 카메라를 지원했으며, 각 카메라의 인덱스를 스캔해 연결 가능한 카메라를 식별한다. 각 프레임에 타임스탬프를 추가해 하나의 Node Struct로 구성해 센서 융합시 정확한 데이터를 제공할 수 있도록 구성했다.
- 주기적으로 데이터를 전송하는 SendEventCEventCyclic() 메서드를 통해 정보를 전달하며, 이벤트 방식으로 구성해 지연을 최소화하도록 구성했다.

2.1.3 Lidar AA

Lidar AA는 차량 주변의 장애물을 감지하고, 거리 기반의 3D 공간 인식 데이터를 제공한다. 이 기능은 lidar.cpp와 lidardata.cpp 파일에 구현되어 있으며, Lidar 장치의 모터를 제어하여 특정 각도 범위 내에서 장애물의 거리 데이터를 측정하고, 이를 SensorFusion 모듈에 주기적으로 전송한다.

a. 주요 메서드

- produceScanning()
  각도를 8개의 존(zone)으로 나누고, 각 존에서 측정된 최소 거리를 기록하여 장애물의 위치를 파악한다. 이를 통해 차량이 주행 중 주변 장애물과의 거리를 정확하게 인식하고, 안전한 주행 경로를 설정할 수 있도록 지원한다.
- SendEventLEventCyclic()
  라이다 데이터를 주기적으로 전송한다. SensorFusion 모듈에서 실시간으로 필요한 위치 및 거리 정보를 제공하여, 지연을 최소화하고 빠른 반응을 가능하게 구현했다.

2.1.4 SensorFusion AA

SensorFusion AA는 Camera와 Lidar에서 수집한 데이터를 통합하여 환경을 종합적으로 인식할 수 있는 융합 데이터를 생성한다. 이 기능은 sensorfusion.cpp와 fusiondata.cpp 파일에 구현되어 있으며, 각 센서로부터 제공되는 데이터를 PPort 및 RPort를 통해 제공받아 데이터의 일관성과 정확성을 유지한다.

a. 주요 메서드

- TaskReceiveCEventCyclic()과 TaskReceiveLEventCyclic()
  Camera와 Lidar 데이터를 비동기적으로 수신하며, 각 데이터는 타임스탬프를 통해 시간적으로 동기화된다. 이를 통해 주행 중 발생할 수 있는 다양한 장애물과 도로 상황을 정확하게 파악하여 차량의 주행 안전성을 확보한다.

- FusionData 포트를 통해 FusionDataNode를 주기적으로 Inferece AA에 제공한다. 이를 통해 실시간으로 활용할 수 있는 통합 데이터를 생성하여 차량의 주행 결정에 필요한 정확하고 신뢰성 높은 정보를 지원한다.

2.2 Decision 모듈 (Inference)

Decision 모듈의 Inference는 차량의 주행 결정을 담당하는 핵심 요소로, AWS DeepRacer와 Amazon SageMaker를 활용해 강화학습 모델을 개발한다. AWS DeepRacer는 자율주행 차량의 강화학습을 위한 플랫폼으로, 가상 환경에서 모델을 훈련하고 최적의 모델을 추출하여 Adaptive Application(AA)로 이식하는 과정을 통해 실차 적용을 가능하게 한다.

2.2.1 모델 학습 과정 개요

a. 환경 설정 및 강화학습 정책

AWS DeepRacer 콘솔을 통해 가상의 자율주행 트랙 환경을 설정하고, 차량의 상태 정보와 행동을 정의한다. 차량의 상태 정보는 차량의 위치, 속도, 각도 등으로 구성되며, 가능한 행동은 주행 중 조향 각도와 가속/감속 정도로 정의된다.

정책 목표는 차량이 트랙을 빠르고 안정적으로 주행하면서도 장애물과의 충돌을 피하는 것이다. 강화학습 정책은 차량이 주어진 목표를 최대한 효과적으로 달성하도록 학습하며, 반복 학습을 통해 최적의 경로를 따르는 방향으로 조향과 속도를 조정한다.

b. 보상 함수 설계

차량이 트랙 중앙을 따라 주행하도록 유도하는 보상 함수를 설계하여 학습의 주요 기준으로 설정한다. 차량이 트랙 중앙에 가까울수록 높은 보상을 주며, 트랙을 이탈하거나 경계에 지나치게 접근할 경우 페널티를 부여하여 중앙 주행을 유도한다.

추가적으로 장애물을 회피하는 것이 학습 목표에 포함되었기 때문에, 장애물과의 거리 및 상대적 위치를 고려한 보상 함수를 설계했다. 차량이 장애물과 안전거리를 유지할 경우 보상을 높이고, 가까워질수록 보상을 줄이는 방식으로 설정하여 차량이 트랙 중앙을 유지하면서도 장애물을 회피하도록 한다.

c. 모델 훈련

설계된 보상 함수를 기반으로 AWS DeepRacer 시뮬레이터에서 모델을 훈련한다. 학습 과정은 다수의 에피소드로 구성되며, 각 에피소드에서 차량은 주행 경로를 학습하고 보상을 최대화하는 방향으로 정책을 업데이트한다.

각 에피소드가 진행됨에 따라 차량의 경로 유지 능력과 장애물 회피 성능이 점차 향상되도록 훈련한다. 강화학습 알고리즘이 차량이 최적의 주행 경로를 학습하도록 학습률, 보상 값, 그리고 주행 속도 같은 다양한 파라미터를 조정한다.

d. 모델 평가 및 튜닝

학습된 모델은 AWS DeepRacer 시뮬레이터에서 평가된다. 모델이 높은 주행 점수를 얻을 수 있도록 주행 성능이 만족스럽지 않을 경우 보상 함수나 하이퍼파라미터를 조정하여 성능을 개선한다.

반복적인 평가와 튜닝을 통해 주행 경로 유지, 속도 조정, 장애물 회피 측면에서 모델의 성능을 최적화한다. 평가 과정에서 모델의 학습 곡선을 분석하여 보상 값이 목표 수준에 도달하도록 지속적으로 파라미터를 조정한다.

e. 모델 배포

최종적으로 훈련된 강화학습 모델을 Inference AA로 이식한다. AWS SageMaker와 AWS IoT Greengrass를 활용해 모델을 실차 환경에 배포하고, Inference AA가 주행 중 실시간으로 주행 경로를 결정하고 장애물을 회피할 수 있게 한다.

Inference AA는 실시간으로 차량의 위치와 상태 정보를 수신하고, 훈련된 모델에 기반해 최적의 경로와 조향, 속도 조정 결정을 내리게 한다. 차량은 이러한 실시간 판단에 따라 주행 환경에 적응하며 안전한 주행을 유지할 수 있게 한다.

2.2.2 Decision 모듈 주요 개발 전략

a. 실시간 추론 및 최적화

Inference AA은 차량의 주행 결정을 빠르고 정확하게 내리기 위해 실시간 추론을 최적화한다. 이를 위해 경량화된 추론 엔진을 사용하여 모델이 최소한의 리소스로 동작하면서도 신속한 응답을 제공하도록 설계했다.

모델 추론 과정에서 발생할 수 있는 지연을 최소화하기 위해 멀티스레딩과 비동기 처리를 적용하여 데이터를 병렬로 처리한다. 이를 통해 빠르게 변화하는 주행 환경에서도 차량이 최적의 경로를 유지할 수 있도록 한다.

b. 타임스탬프 기반 데이터 동기화

주행 중에 수집된 SensorFusion 데이터와 추론 데이터를 시간순으로 동기화하여 정확한 주행 결정을 내릴 수 있도록 한다. 각 데이터는 타임스탬프를 기반으로 정렬되며, 이를 통해 차량이 환경의 변화를 빠르게 감지하고 대응할 수 있게 한다.

TimeManager 객체를 활용해 데이터가 추론 과정에서 일관성을 유지하도록 설계하여, 주행 중에 발생할 수 있는 지연과 오류를 줄인다.

2.2.3 Inference AA

Inference AA는 차량의 주행 결정을 위한 추론 작업을 수행하는 핵심 모듈로, 주행 중 수신된 SensorFusion 데이터를 기반으로 최적의 주행 경로를 결정한다. 이 모듈은 inference.cpp 파일에 구현되어 있으며, 강화학습 모델을 기반으로 실시간 데이터를 처리하고 추론을 수행한다. 다음은 주요 메서드의 기능과 역할에 대한 설명이다

a. 주요 메서드

- Initialize()

Inference 모듈의 초기화 작업을 수행한다. FusionData와 InferenceData 포트를 생성하여 데이터 통신의 일관성을 유지하고, 추론 작업을 위한 필수 설정을 완료한다.

- TaskRequestFMethod()

Fusion 모듈로부터 데이터를 요청하는 메서드로, FMethod 핸들러를 등록하여 SensorFusion 데이터를 수신하고 OnReceiveFMethod() 함수로 전달한다. 이 과정을 통해 실시간으로 센서 데이터를 받아 추론 작업을 수행할 수 있도록 한다.

- LoadModel()

Inference 모델을 로드하는 메서드로, artifactPath로부터 모델을 읽어 inferRequest_ 객체에 저장한다.
각 입력 데이터의 속성을 설정하여 추론에 필요한 매개변수를 준비한다. 추론의 정확성을 위해 모델의 정확한 아티팩트와 매개변수를 설정한다.

- sensorCB()

Fusion 데이터가 수신되었을 때 호출되는 콜백 함수로, 수신된 데이터를 추론 엔진에서 사용할 수 있도록 적절히 전처리한다.
입력 이미지 데이터를 다양한 방식으로 전처리하여 모델에 적합한 형식으로 변환하며, 이 과정에서 필요한 이미지 채널 수와 크기를 설정한다.

- OnReceiveFMethod()

Fusion 모듈에서 데이터를 수신했을 때 호출되는 메서드로, Fusion 데이터의 전처리와 추론 작업을 수행한다. 수신된 데이터는 전처리된 후, 추론 모델에 입력되어 필요한 예측 결과를 생성한다. 이를 통해 Inference AA가 실시간으로 주행 결정을 내릴 수 있다.

- StartInference()

모델 추론을 시작하는 메서드로, SensorFusion 데이터를 기반으로 추론 작업을 수행하여 최적의 주행 경로 및 차량 제어 결정을 내린다. 이 메서드는 주기적으로 Fusion 데이터를 기반으로 모델 추론을 수행하여, 차량의 주행 환경에 맞는 결정을 실시간으로 내리도록 한다.

2.3 Execution 모듈 (Navigate, Servo AA)

Execution 모듈은 차량이 주어진 경로를 따라 안전하고 정확하게 주행할 수 있도록 Navigate AA와 Servo AA로 구성되어 있으며, 차량의 방향 ,속도, 제동을 실시간으로 제어하는 역할을 한다. 이를 위해 다음과 같은 개발 전략을 적용했다.

2.3.1 Decision 모듈 주요 개발 전략

a. 실시간 주행 명령 생성 및 데이터 동기화

Navigate AA는 Inference 모듈로부터 전달받은 주행 경로 데이터를 바탕으로 차량의 조향 각도와 속도를 실시간으로 결정한다. 이를 통해 차량이 변화하는 주행 환경에 적합하게 반응할 수 있도록 한다.

데이터를 비동기적으로 수신 및 처리하여 주행 명령 생성 과정에서의 지연을 최소화하였으며, 차량이 정확한 시간 순서에 따라 주행 경로를 따를 수 있도록 타임스탬프 기반 동기화를 적용하였다. 이를 통해 주행 경로의 일관성과 신뢰성을 보장한다.

b. 멀티스레딩 및 비동기 처리 적용

Navigate와 Servo AA 모듈 간 데이터 전송과 주행 명령 처리 속도를 높이기 위해 멀티스레딩과 비동기 처리를 적극 활용하였다. 이를 통해 각 모듈이 독립적으로 데이터를 처리하면서도 높은 성능을 유지하며, 다양한 주행 환경 변화에도 신속히 대응할 수 있도록 한다.

이를 위해 m_workers 객체를 통해 주요 주행 명령을 비동기로 처리하여 데이터 처리 병목 현상을 줄이고, 차량 제어의 효율성을 극대화하였다.

c. 안전한 제어 및 장애물 회피를 위한 주행 명령 최적화

Servo AA는 주행 안전성을 높이기 위해 Navigate 모듈로부터 수신된 주행 명령을 최적화하여, 차량의 조향, 가속, 감속, 제동이 원활하게 이루어지도록 설계하였다.

장애물 감지 및 회피 상황에서 차량이 신속히 조향과 속도 조절을 수행할 수 있도록 하였으며, 긴급 제동 시에는 즉각적으로 제어 명령을 실행하여 차량의 안전을 보장한다.

d. 지속적인 주행 상태 모니터링 및 피드백 적용

차량의 주행 상태를 지속적으로 모니터링하고, 주행 경로에 따라 조향과 속도를 조정하는 피드백 루프를 적용하였다. 이를 통해 주행 중 발생할 수 있는 변동 상황에 즉각적으로 대응할 수 있으며, 차량이 안정적으로 경로를 따르도록 한다.

Navigate AA와 Servo AA 간의 지속적인 피드백 루프를 통해 각 주행 명령이 차량에 올바르게 반영되는지 확인하고, 주행 경로를 실시간으로 조정하여 차량의 안정성을 높였다.

2.2.3 Navigate AA

Navigate AA는 차량이 주어진 경로를 따라 정확하게 이동할 수 있도록 주행 명령을 생성하는 역할을 담당하는 모듈로, Inference 모듈로부터 전달받은 주행 경로와 관련 데이터를 기반으로 최적의 주행 행동을 결정하여 Servo AA로 전달한다. 이 모듈은 navigate.cpp 파일에 구현되어 있으며, 차량의 방향과 속도를 제어하여 주행 안전성을 확보한다. 다음은 주요 메서드의 기능과 역할에 대한 설명이다.

a. 주요 메서드

- ProcessInferenceData()
  inferenceMsg 데이터를 기반으로 navigateMsg의 angle(조향각)과 throttle(가속도)을 설정한다.
  각 인퍼런스 결과를 사용해 actionValues에 속도와 조향각 데이터를 저장하고, 이를 ScaleContinuousValue()로 조정해 설정한다.

- GetMaxScaledValue()
  주어진 actionValue를 사전에 정의된 최대값으로 나누어 0~1 사이의 비율로 반환한다.
  최대값이 0 이하일 경우 로그에 오류를 남기고, 0을 반환하여 에러 상황을 처리합니다.

- GetNonLinearlyMappedSpeed()
  주어진 속도 값을 비선형 맵핑 식을 통해 변환한다.
  최종 결과는 0~1 사이의 값으로 제한되며, 비선형 속도 매핑을 적용하여 더 자연스러운 속도 조정을 가능하게 한다.

- ScaleContinuousValue()
  연속적인 범위의 값을 스케일링하는 함수.
  기존의 값 범위(minOld, maxOld)에서 새로운 범위(minNew, maxNew)로 변환한다.

- SetActionSpaceScales()
  m_actionSpaceType에 따라 action space의 최대값을 설정하는 함수
  m_actionSpace의 연속형 값(CONTINUOUS_HIGH)을 최대값으로 설정한다.

2.2.4 Servo AA

Servo AA는 Navigate 모듈로부터 전달받은 주행 명령을 바탕으로 실제 차량의 물리적 제어를 수행하는 모듈로, 차량의 조향, 가속, 제동을 실시간으로 조절하여 주행 명령을 실행한다. 이 모듈은 차량의 하드웨어와 직접 상호작용하여 주행 명령을 물리적 동작으로 변환하며, 차량의 안정성과 주행 효율성을 보장한다.

a. 주요 메서드

- Initialize()

  Servo AA를 초기화하고, calibration.json 파일에서 캘리브레이션 데이터를 불러와 calibrationMap_에 저장하여 차량 제어에 필요한 값을 준비한다. 이 초기화 과정에서는 PWM, GPIO 핀 설정을 통해 차량과의 물리적 연결을 설정한다.

- Start()

  Servo AA를 시작하여 주행 제어를 활성화한다. PWM 설정을 시작하며, 전달받은 주행 명령에 따라 실시간으로 조향과 스로틀을 제어할 수 있도록 준비한다.

- ServoSubscriber()

Navigate 모듈에서 전달된 throttle과 angle 값을 사용해 DC 모터와 서보 모터의 제어 신호를 계산하고, PWM 신호를 설정한다. calibrationMap_의 값을 이용해 입력값을 듀티 사이클로 변환하여 PWM을 통해 차량에 전달한다.

- setCalibrationValue()

  차량의 캘리브레이션 값을 설정하는 메서드로, 서보와 모터의 최소, 중간, 최대값 및 극성을 설정한다. 이를 통해 차량의 제어 신호가 정확하게 전달될 수 있도록 조정하여 차량의 주행 안정성을 높인다.

- TaskReceiveNEventCyclic()

  Navigate 모듈로부터 이벤트를 주기적으로 수신하여, 조향 및 속도 값을 실시간으로 업데이트한다. 이 메서드는 NavigateData의 이벤트 핸들러를 통해 주행 중 전달받는 명령을 적용하여 차량 제어의 민첩성을 유지한다

- setPWM()

  최종적으로 설정된 스티어링과 스로틀 값을 PWM 신호로 변환하여 실제 차량 제어 시스템에 전달한다. 차량의 물리적 동작을 수행하며, 주행 환경의 변화에 따라 실시간으로 명령을 업데이트하여 주행의 안정성을 유지한다.

이와 같은 메서드들로 구성된 Servo AA는 PWM 신호를 활용하여 차량을 직접 제어하며, Navigate 모듈에서 전달된 명령을 정확하게 실행하여 차량의 안정적 주행을 지원한다.

3. 테스트 및 실차 주행

본 프로젝트에서는 각 Adaptive Application(AA)에 대한 개별 테스트와 통합 테스트를 통해 실제 주행 환경에서의 안정성과 정확성을 검증하였다.

3.1 환경 설정 및 시나리오 구축

AWS DeepRacer 시뮬레이터에서 다양한 도로 및 장애물 환경을 설정하여 여러 주행 조건에서 성능을 검증하였다. 직선, 곡선, 장애물 회피와 같은 다양한 상황에서 시스템이 일관된 성능을 보이는지 확인하였다.

시뮬레이션을 통해 Camera와 Lidar 센서 데이터의 동기화 성능과 SensorFusion의 데이터 통합 정확도를 확인하였으며, 센서 간 데이터 오차가 줄어들고, 데이터 통합 과정에서 정확도가 높아졌다.

3.1.1 강화학습 모델 평가 및 튜닝

시뮬레이터에서 강화학습 모델을 반복적으로 평가하여 차량이 최적의 주행 경로와 장애물 회피 경로를 안정적으로 따를 수 있도록 했다. 초기 테스트에서는 장애물 회피 성공률이 낮았으나, 보상 함수를 조정한 후 회피 성능이 약 20% 향상되었다.

모델 튜닝을 통해 학습된 경로 유지 및 장애물 회피 성능이 개선되었으며, 이를 통해 주행 궤적의 정확성이 증가하고 전체 주행 시간이 감소하였다.

3.1.2 주행 명령의 실시간 응답성 테스트

시뮬레이터에서 Servo 모듈을 통한 PWM 신호 전달이 실시간으로 반영되는지 검증하였다. 조향과 속도 명령이 즉각적으로 반응하여 차량이 주행 중 지연 없이 명령을 수행할 수 있음을 확인하였다.

3.2 실차 주행 테스트

3.2.1 테스트 환경 준비

대회 측에서 제공한 트랙를 기반으로 환경을 구성해 테스트를 진행하였다. 트랙은 직선 구간과 곡선 구간, 장애물 구간으로 구성하여 실제 주행 환경과 유사한 조건을 제공하였다.

3.2.2 AA별 개별 테스트

각 AA 모듈을 개별적으로 테스트하여, Camera와 Lidar의 동기화, SensorFusion의 데이터 통합, Inference AA의 주행 경로 예측이 정확하게 이루어지는지 확인하였다. 특히, SensorFusion의 융합 데이터가 일관성 있게 생성되고, Inference AA가 장애물 회피 경로를 정확하게 계산함을 확인하였다.

3.2.2 주행 경로 및 장애물 회피 성능 테스트

Inference 모듈을 통해 생성된 주행 경로와 장애물 회피 경로를 따라 차량이 정확히 주행할 수 있음을 확인하였다. Navigate와 Servo AA가 연동되어 조향 및 속도 제어가 안정적으로 이루어졌으며, 장애물을 회피하면서도 경로를 유지할 수 있었다.

테스트 결과, 주행 경로 유지 성능이 개선되었고, 장애물 회피 시 차량의 응답 지연이 약 15% 감소하여 전체 주행 안정성이 높아졌다.

3.2.3 실시간 응답성 및 제어 정확도 검증

실차 테스트에서 Servo AA가 Navigate 모듈의 명령에 빠르게 반응하는지 검증하였다. 급격한 회전이나 장애물 인식 상황에서도 지연 없이 조향과 속도 제어가 이루어졌고, PWM 신호 전달 지연이 크게 줄어들어 주행 중 발생할 수 있는 위험 요소가 최소화되었다.

3.3 데이터 분석 및 최종 피드백

실차 주행 테스트 데이터를 분석하여 각 구간의 주행 성능과 안정성을 평가하였다. 주행 궤적의 일관성과 장애물 회피 성공률이 목표 수준에 도달하였고, 주행 속도와 조향 응답성이 개선됨에 따라 보상 함수와 제어 알고리즘의 최종 튜닝을 완료하였다.

테스트 결과를 반영하여 Inference 및 Servo 모듈의 알고리즘을 추가 개선함으로써 최종 시스템 성능이 목표 성능 기준을 충족하였다.

□ 개발 중 발생한 장애요인과 해결방안

1. 센서 데이터 동기화 문제

1.1 문제 배경

Camera와 Lidar 센서에서 수집된 데이터가 수신 타이밍에서 미세한 차이를 보이며 들어오는 문제가 발생했다.

SensorFusion 모듈에서 일관성 없는 데이터를 생성하는 원인이 되었고, 차량의 주행 중 경로 유지와 장애물 감지 등에 영향을 미쳤다. 특히, Lidar와 Camera 간의 시간차가 주행 경로와 장애물 회피 판단에 오차를 유발하여, 차량이 목표 경로를 벗어나는 상황이 발생했다.

1.2 해결 방안 및 구체적인 접근 과정

a. 타임스탬프 해상도 증가: 센서 데이터에 부여되는 타임스탬프의 해상도를 초 수준에서 밀리초 수준으로 향상하여 각 데이터의 정확한 수신 시점을 더욱 세밀하게 기록하도록 했다. 이를 통해 SensorFusion 모듈에서 데이터가 수신되는 순서가 정확하게 일치하도록 하고, 센서 간 시간차를 보다 정확하게 인지할 수 있었다.

b. 타임스탬프 보정 및 데이터 큐 관리

수신된 각 센서 데이터의 타임스탬프를 비교하여, 일정 시간차 이내에 위치한 데이터들만 유효한 데이터로 간주하는 보정 알고리즘을 추가하였다. 이 과정에서는 타임스탬프 차이가 일정 오차 범위를 초과하는 경우 해당 데이터를 보류하거나 무시하고, 동기화된 데이터를 우선적으로 선택하여 처리하도록 하였다.

c. 데이터 큐 버퍼링 기능 추가

수신된 센서 데이터를 일시적으로 보관할 버퍼를 생성하여, Camera와 Lidar 간의 데이터가 비동기적으로 수신되더라도 동기화된 상태로 유지될 수 있도록 했다. Lidar 데이터가 늦게 수신될 경우, Camera 데이터는 버퍼에 저장하고 Lidar 데이터가 도착할 때까지 대기하여 두 데이터의 시간차가 보정된 후에 처리되도록 했다.

d. 동기화 체크포인트 생성

SensorFusion 모듈 내에서 일정 시간마다 동기화 체크포인트를 생성하여, 해당 시점에 동기화된 데이터만 추출하도록 하였다. 이 체크포인트는 주행 속도에 따라 주기를 조정하여, 차량이 빠른 주행 상황에서도 정확하게 동기화된 데이터를 수집하고 처리할 수 있도록 보장했다.

1.3 적용 결과

센서 데이터 간의 미세한 시간 차이를 극복함으로써, SensorFusion 모듈이 생성하는 통합 데이터의 일관성이 크게 개선되었다. 특히, 장애물 감지 및 주행 경로 판단 시 발생하던 오차가 눈에 띄게 감소했으며, 차량이 더욱 안정적으로 주행할 수 있었다. 테스트 결과 주행 중 경로 이탈 빈도가 현저히 줄어들고, 장애물 감지의 정확성이 약 15% 향상되었다

□ 개발의 차별성

본 프로젝트에서 개발된 알고리즘과 시스템은 실시간 주행 안정성과 데이터 처리 효율성을 최적화하는 데 중점을 두어 기존 시스템과 차별화된 특징과 우수성을 보인다. 특히 센서 데이터 동기화, 강화학습 기반 주행 경로 최적화, 안정적인 차량 제어를 통해 자율주행 차량의 정확성과 안정성을 높였다.

1. 정밀한 센서 데이터 동기화 및 SensorFusion 알고리즘

정밀한 타임스탬프 기반 동기화: Camera와 Lidar 센서 데이터를 정확한 타임스탬프 기반으로 동기화하여 데이터의 일관성과 신뢰성을 확보했다. 이를 통해 주행 경로와 장애물 정보를 실시간으로 수집하고 통합하여 차량의 판단 정확성을 높였다.

1.1 데이터 버퍼링 및 동기화 보정 알고리즘:

비동기적으로 수신되는 센서 데이터를 버퍼링하고, 일정한 오차 범위 내에서 데이터를 통합하는 보정 알고리즘을 적용하여 빠르게 변화하는 주행 환경에서도 안정적으로 데이터를 수신하고 처리할 수 있도록 했다.

1.2 융합 데이터의 실시간 전송

SensorFusion을 통해 처리된 융합 데이터를 Inference 모듈로 실시간으로 전송하여 차량의 주행 경로 결정이 신속하게 이루어지도록 하여 실시간 주행 안정성을 극대화했다.

2. 강화학습 기반 주행 경로 최적화

강화학습 모델을 통한 자율 경로 결정: AWS DeepRacer와 SageMaker에서 강화학습 모델을 개발하여 최적의 주행 경로를 학습하고, 이를 기반으로 Inference 모듈이 실시간으로 차량의 주행 결정을 내리도록 했다. 이 모델은 트랙 주행 안정성을 강화하고, 장애물 회피 성능을 향상시켜 주행 효율을 극대화했다.

2.1 보상 함수 최적화를 통한 경로 유지 및 장애물 회피 성능 개선

차량이 경로를 벗어나지 않으면서 장애물을 안전하게 회피하도록 보상 함수를 최적화하였으며, 이를 통해 안전하고 효율적인 주행 경로를 확보하였다.

3. 안정적인 주행 제어 및 실시간 응답성 확보

3.1 Servo AA에서의 실시간 제어 최적화

Servo AA는 GPIO 및 PWM 신호 제어를 최적화하여, 차량이 정확하고 신속하게 주행 명령을 따를 수 있도록 구현했다. 이를 통해 차량이 주행 중 발생하는 다양한 상황에 신속하게 대응할 수 있도록 하였다.

3.2 비동기 처리 및 병렬 구조 적용

제어 명령을 비동기적으로 처리하여 조향과 속도 제어가 병렬로 실행되도록 설계함으로써, 차량이 신속하게 조향과 속도를 제어할 수 있는 반응성을 확보하였다.

3.3 캘리브레이션 맵을 통한 세밀한 조정 가능: 차량의 조향 및 속도를 세밀하게 조정할 수 있도록 캘리브레이션 맵을 활용하여 제어 신호를 정밀하게 튜닝함으로써, 차량의 주행 안정성을 높이고 불필요한 오차를 최소화하였다.

□ 개발 일정

No	내용	2024年			
		8月	9月	10月	11月
1	아키텍처 설계, 하드웨어 및 소프트웨어 분석				
2	강화학습 모델 개발				
3	AA ARXML 설계				
4	Perception AA개발				
5	Decision AA 개발				
6	Execution AA 개발				
7	시뮬레이션 및 실제 주행 테스트				
8	최종 검증 및 마무리				

□ 팀 업무 분장

No	구분	성명	참여인원의 업무 분장
1	팀장	이민규	ARXML 설계, Inference AA 담당
2	팀원	고광현	Navigate AA, Servo AA 담당
3	팀원	전규훈	강화학습 모델 구축 및 최적화, Inference AA 담당
4	팀원	정진솔	Camera, Lidar, SenserFusion AA 담당