

Robot Vacuum Cleaner

요구사항 분석 및 OOAD

3팀

202213351 김태성

202111382 최성준

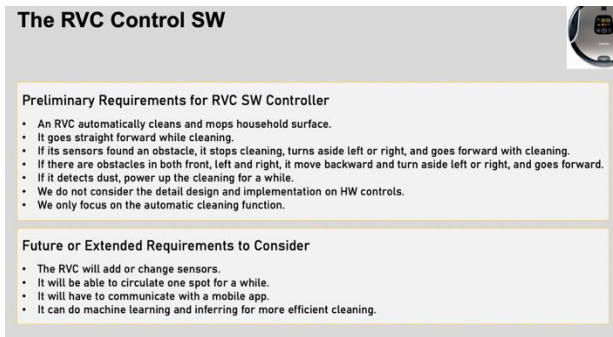
202011434 최원탁

202011380 최용근

시스템 요구사항 및 기능/비기능 요구사항 분석 프롬프트

첨부 자료

IMAGE 1



Prompt

IMAGE 1 을 보고 시스템 요구사항 및 기능/비기능 요구사항을 작성해줘.

제시되지 않은 내용은 배제하고, 추론 및 예측 없이 오로지 이미지에 적힌 내용에 기반해서 내용을 md 형식으로 작성해.

아쉬운 점

고려하지 않는 부분에 대한 요구사항도 함께 작성하여, 검토 후 직접 수정함

요구사항 개요

시스템

Robot Vacuum Cleaner (RVC) SW Controller

분석 범위

자동 청소 기능 (하드웨어 상세 구현 제외)

핵심 기능

자동 청소 / 장애물 회피 / 먼지 감지 대응

제외 범위

모바일 앱 / 머신러닝 모델 / 하드웨어 상세 제어

향후 확장 후보

선회 청소 · 모바일 앱 통신 · ML 기반 판단

기능 요구사항 (FR)

ID	요구사항	설명
FR-001	자동 청소 수행	가정용 바닥을 자동으로 청소
FR-002	청소 장치 동작	자동 청소 중 청소 장치 동작
FR-003	기본 직진 주행	장애물 없을 때 직진 청소
FR-004	전방 장애물 대응	전방 장애물 감지 시 정지
FR-005	회피 방향 전환	좌/우 방향 전환
FR-006	회피 후 청소 재개	방향 전환 후 전진 재개
FR-007	삼면 장애물 대응	전/좌/우 막힘 시 후진
FR-008	후진 후 방향 전환	후진 후 좌/우 전환
FR-009	후진 회피 후 재개	후진+전환 후 전진 재개
FR-010	먼지 감지	먼지 감지 입력 처리
FR-011	청소 출력 증가	먼지 감지 시 출력 증가
FR-012	청소 출력 복귀	시간 종료 후 일반 출력 복귀
FR-013	후진 불가 대응	사면 막힘 시 정지 유지

비기능 요구사항 (NFR)

ID	요구사항	설명
NFR-001	하드웨어 독립성	청소 로직과 하드웨어 분리
NFR-002	센서 확장성	센서 추가·변경 대응 가능 구조
NFR-003	정책 변경 용이성	회피·먼지 대응 정책 변경 쉬움
NFR-004	유지보수성	이해하기 쉽고 수정 범위 제한
NFR-005	기능 확장성	선회 청소·앱·ML 추후 추가 가능
NFR-006	설정 가능성	출력 시간·수준·회피 정책 조정 가능
NFR-007	관심사 분리	센서 해석·상태 판단·명령 생성 분리

UC 작성 프롬프트

첨부 자료

TEXT 1

기능 요구사항.md

Prompt

TEXT 1 을 보고 유스케이스를 작성해줘. 하나의 동작으로 이어져야 하는 요구사항을 분석해서 각각의 유스케이스로 묶고, 각 유스케이스의 사전 조건, 메인 시나리오, 예외/대안 시나리오를 분석해서 md 파일로 만들어줘 그리고 유스케이스 다이어그램 그려줘

아쉬운 점

UC-005 삼면 장애물 감지 후 후진 의 예외 시나리오를 명확하게 작성하지 못함
기존 : 후진 후에도 전방, 좌측, 우측이 모두 막혀 있으면 정지 상태를 유지한다
수정 : 더 이상 후진할 수 없는 경우에 좌측, 우측이 모두 막혀 있다면 정지 상태를 유지한다

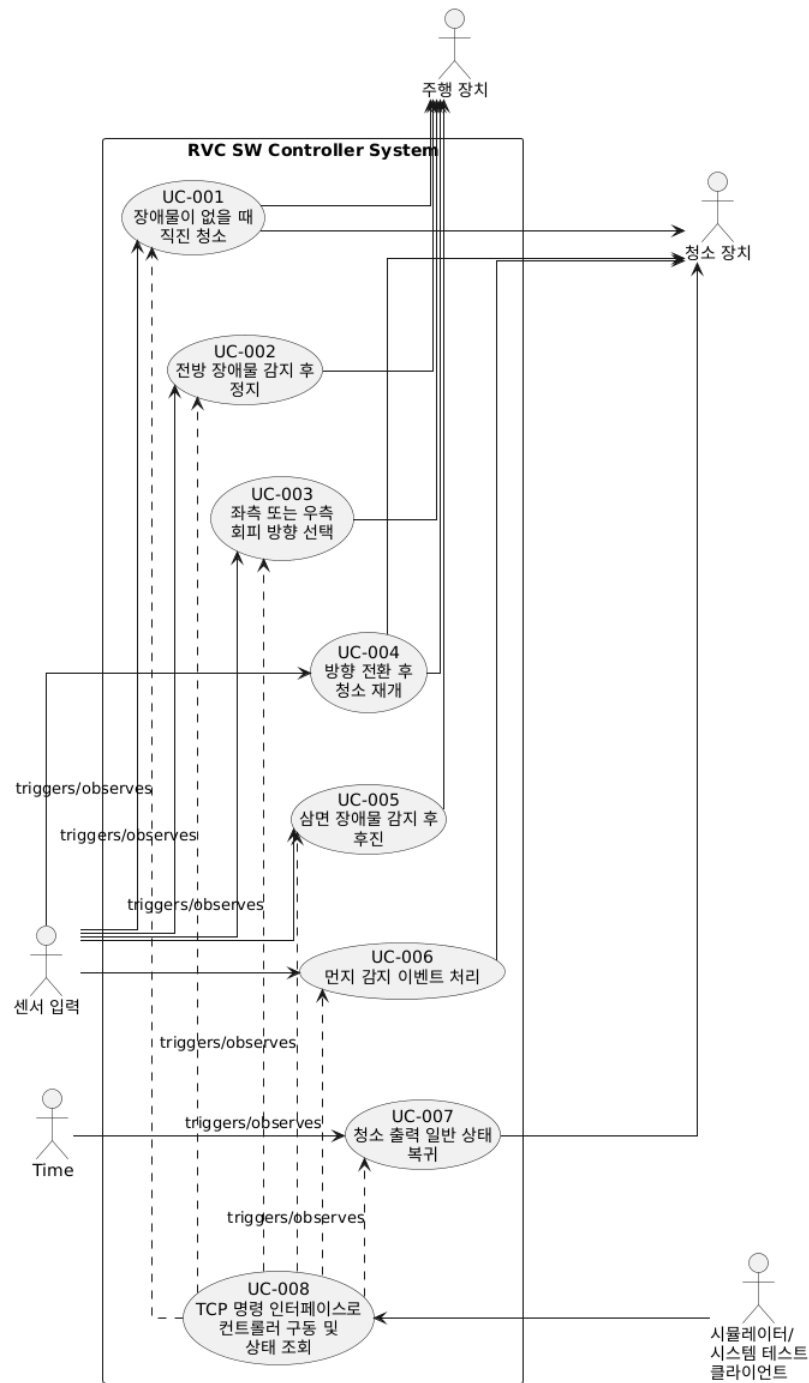
유스케이스 다이어그램의 배치가 부자연스러움

기존 : 연결 선의 형태가 곡선, 각 액터가 위 아래로 배치됨

수정 : 연결 선의 형태를 직선으로 수정 및 각 액터를 좌 우로 배치

USE CASE

Use Case Diagram



장애물이 없을 때 직진 청소

PRECONDITION

자동 청소 시작, 전방 장애물 없음

MAIN SCENARIO

1. 전방 장애물 미감지 확인
2. 청소 장치에 일반 출력 명령
3. 주행 장치에 전진 명령
4. 전진 청소 상태 유지

ALTERNATIVE

전방 장애물 → UC-002 / 먼지 감지 → UC-006

RESULT

RVC는 청소하면서 직진한다.

전방 장애물 감지 후 정지

PRECONDITION

RVC가 전진 청소 중

MAIN SCENARIO

1. 전방 장애물 감지 확인
2. 전진 명령 중단
3. 주행 장치에 정지 명령
4. 방향 전환 판단 단계로 이전

ALTERNATIVE

전/좌/우 모두 감지 → UC-005 / 정지 중 먼지 감지 → 회피 우선 처리

RESULT

RVC는 전방 장애물 앞에서 정지하고 회피 판단을 준비한다.

좌측 또는 우측 회피 방향 선택

PRECONDITION

방향 전환이 필요한 상태

MAIN SCENARIO

1. 좌/우 장애물 감지 상태 확인
2. 한쪽만 가능하면 그쪽 선택
3. 양쪽 가능하면 LEFT_FIRST 또는 RIGHT_FIRST 정책 적용
4. 선택 방향을 회전 명령 생성 단계로 전달

ALTERNATIVE

좌만 막힘 → 우 선택 / 우만 막힘 → 좌 선택 / 양쪽 막힘 → UC-005

RESULT

SW Controller는 좌회전 또는 우회전 방향을 결정한다.

방향 전환 후 청소 재개

PRECONDITION

장애물 회피를 위한 방향 전환 완료

MAIN SCENARIO

1. 전방 이동 가능 여부 확인
2. 가능하면 전진 명령
3. 청소 장치에 일반 또는 유지 출력 명령
4. 자동 청소 상태 복귀

ALTERNATIVE

전환 후 전방 재감지 → UC-003 재수행 / 먼지 대응 중이면 증가 출력 유지

RESULT

RVC는 회피 후 다시 전진하며 청소를 계속한다.

삼면 장애물 감지 후 후진

PRECONDITION

전방·좌측·우측 장애물 모두 활성화

MAIN SCENARIO

1. 삼면 장애물 및 후방 상태 확인
2. 정지 명령
3. 후방 장애물 없으면 후진 명령
4. 후진 후 방향 전환 판단으로 이전

ALTERNATIVE

후진 후 좌/우 가능 → UC-003 / 사면 모두 막힘 → 정지 유지

RESULT

RVC는 막힌 상태에서 후진하거나, 불가능하면 정지한다.

먼지 감지 이벤트 처리

PRECONDITION

RVC가 자동 청소 중

MAIN SCENARIO

1. 먼지 감지 상태 확인
2. 먼지 대응 상태로 전환
3. 설정된 증가 출력 수준 조회
4. 청소 장치에 증가 출력 명령
5. 증가 출력 유지 타이머 시작

ALTERNATIVE

장애물 회피 동시 발생 시 주행은 UC-002/UC-005 따르고 출력 증가 상태 유지

RESULT

RVC는 먼지 감지 구간에서 강화된 청소 출력을 사용한다.

청소 출력 일반 상태 복귀

PRECONDITION

청소 출력이 증가된 상태

MAIN SCENARIO

1. Time으로부터 유지 시간 종료 이벤트 수신
2. 청소 장치에 일반 출력 명령
3. 먼지 대응 상태 종료
4. 자동 청소 상태 유지

ALTERNATIVE

장애물 회피 중 시간 종료 시 주행 유지 + 청소 출력만 복귀

RESULT

RVC는 일반 청소 출력으로 복귀한다.

TCP 명령 인터페이스로 컨트롤러 구동 및 상태 조회

PRECONDITION

rvc_app 이 실행 중인 상태

MAIN SCENARIO

1. 시뮬레이터 또는 시스템 테스트 클라이언트가 rvc_app에 TCP 연결을 생성한다.
2. 클라이언트가 line-based command를 전송한다.
3. 시스템은 수신한 command line의 길이를 검증한다.
4. 시스템은 command의 문법을 검증한다.
5. 시스템은 command를 내부 처리 가능한 요청으로 변환한다.
6. 센서 입력 또는 시간 이벤트 명령이면, UC-001~UC-007 중 해당 자동 청소 흐름을 수행한다.
7. 상태 조회 명령이면, 현재 컨트롤러 상태와 최근 출력 명령을 line-based response로 반환한다.
8. 시스템은 처리 결과를 OK ... 또는 ERR ... 형식으로 반환한다.

RESULT

시스템이 명령을 수행하거나 상태를 반환한다

ssd 작성 프롬프트

첨부 자료

TEXT 1

Use Case.md

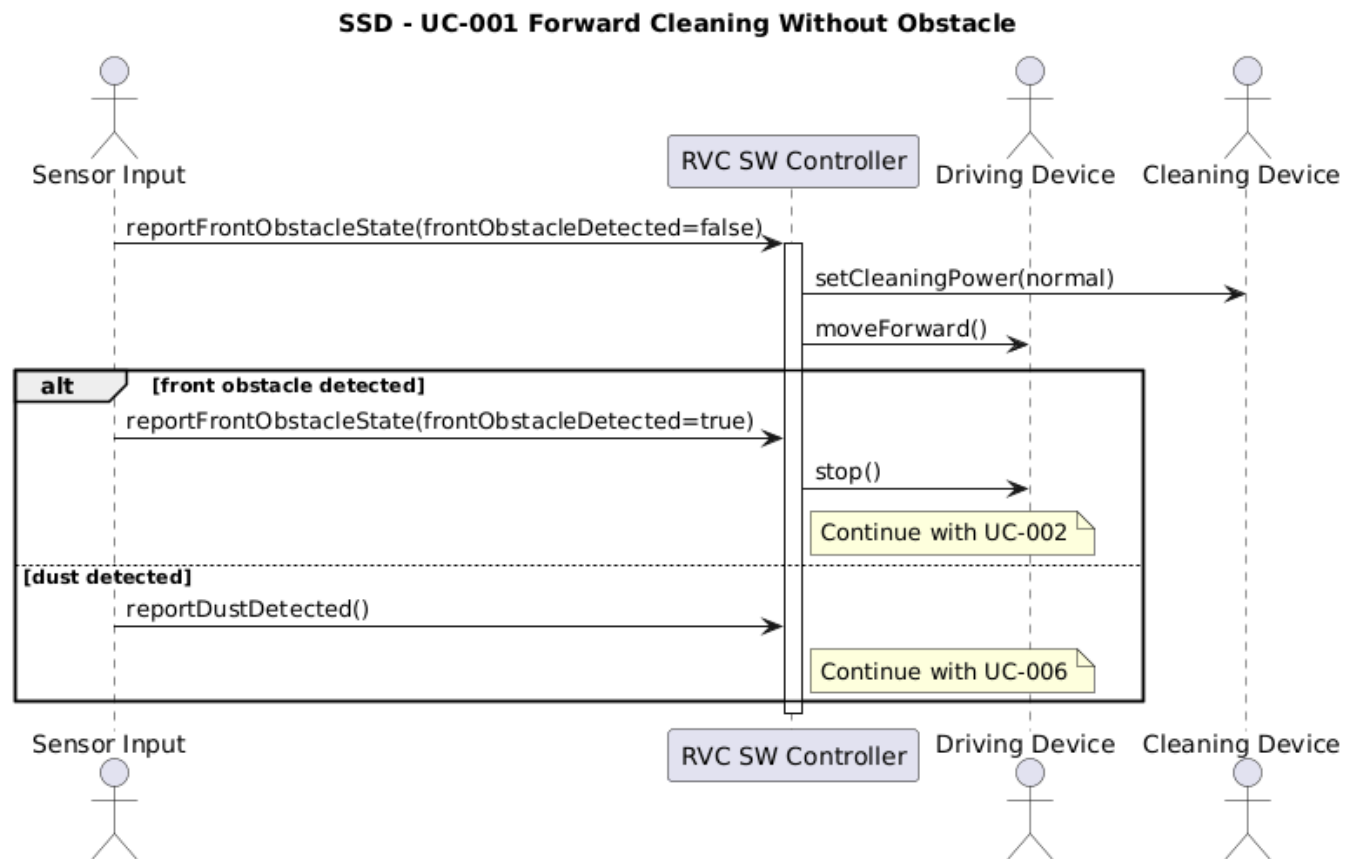
Prompt

TEXT 1 을 보고 각 유스케이스에 해당하는 시퀀스 다이어그램을 그려줘.

아쉬운 점

다수의 시퀀스 다이어그램 흐름이 작성된 유스케이스의 흐름과 일치하지 않음
재검토 (prompt : UseCase.md 와 각 시퀀스 다이어그램을 보고, 흐름이 일치하지 않는 부분을 검토해서 말해줘.) 후 수정

장애물이 없을 때 직진 청소

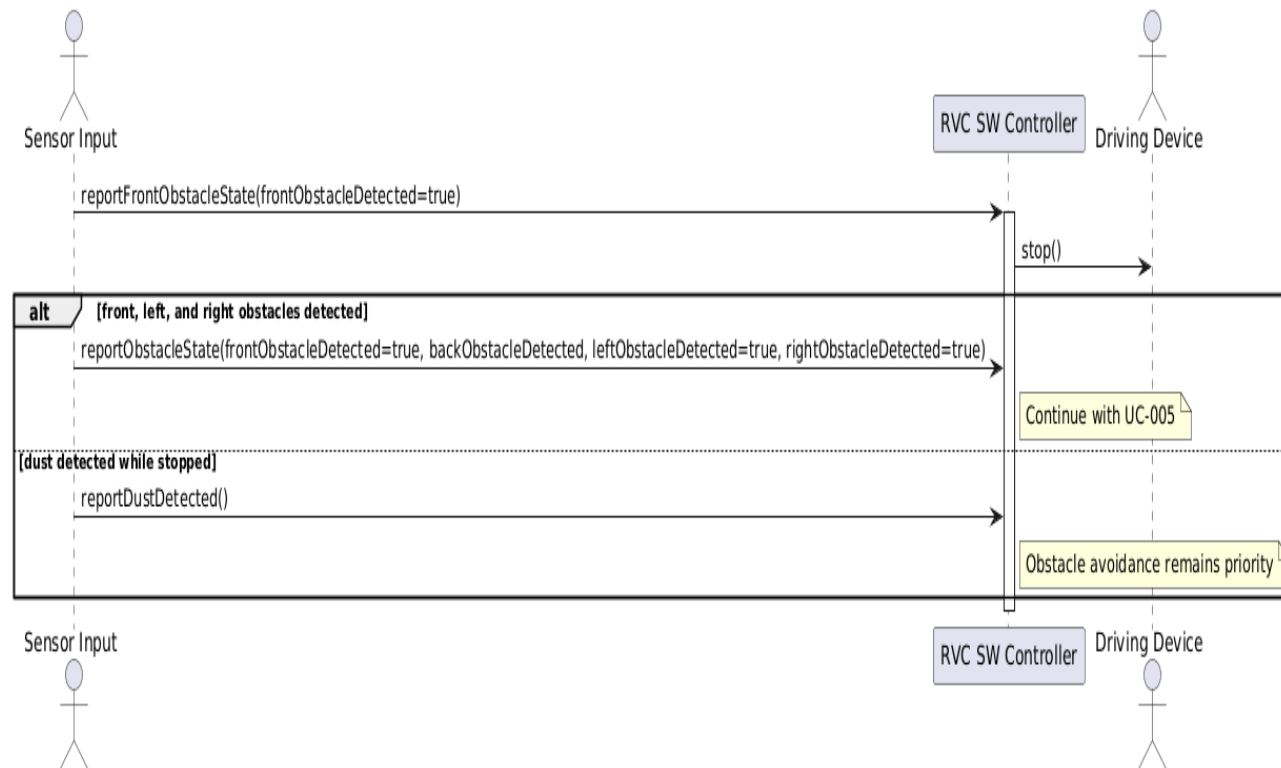


핵심

장애물 없으면 일반 출력으로 직진 청소 유지

Stop After Front Obstacle Detection

SSD - UC-002 Stop After Front Obstacle Detection

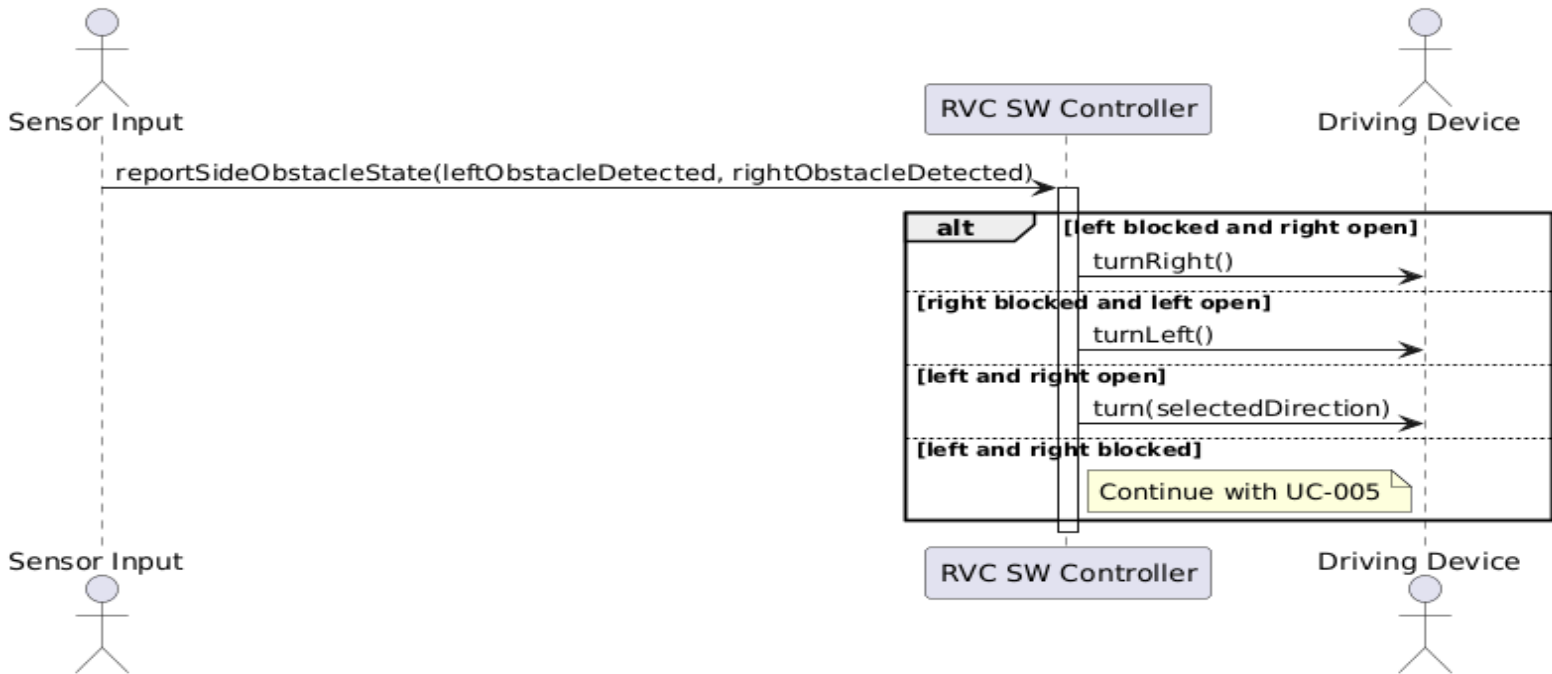


핵심

전방 장애물 감지 즉시 정지, 삼면이면 UC-005로 전환

Select Left or Right Avoidance Direction

SSD - UC-003 Select Left or Right Avoidance Direction

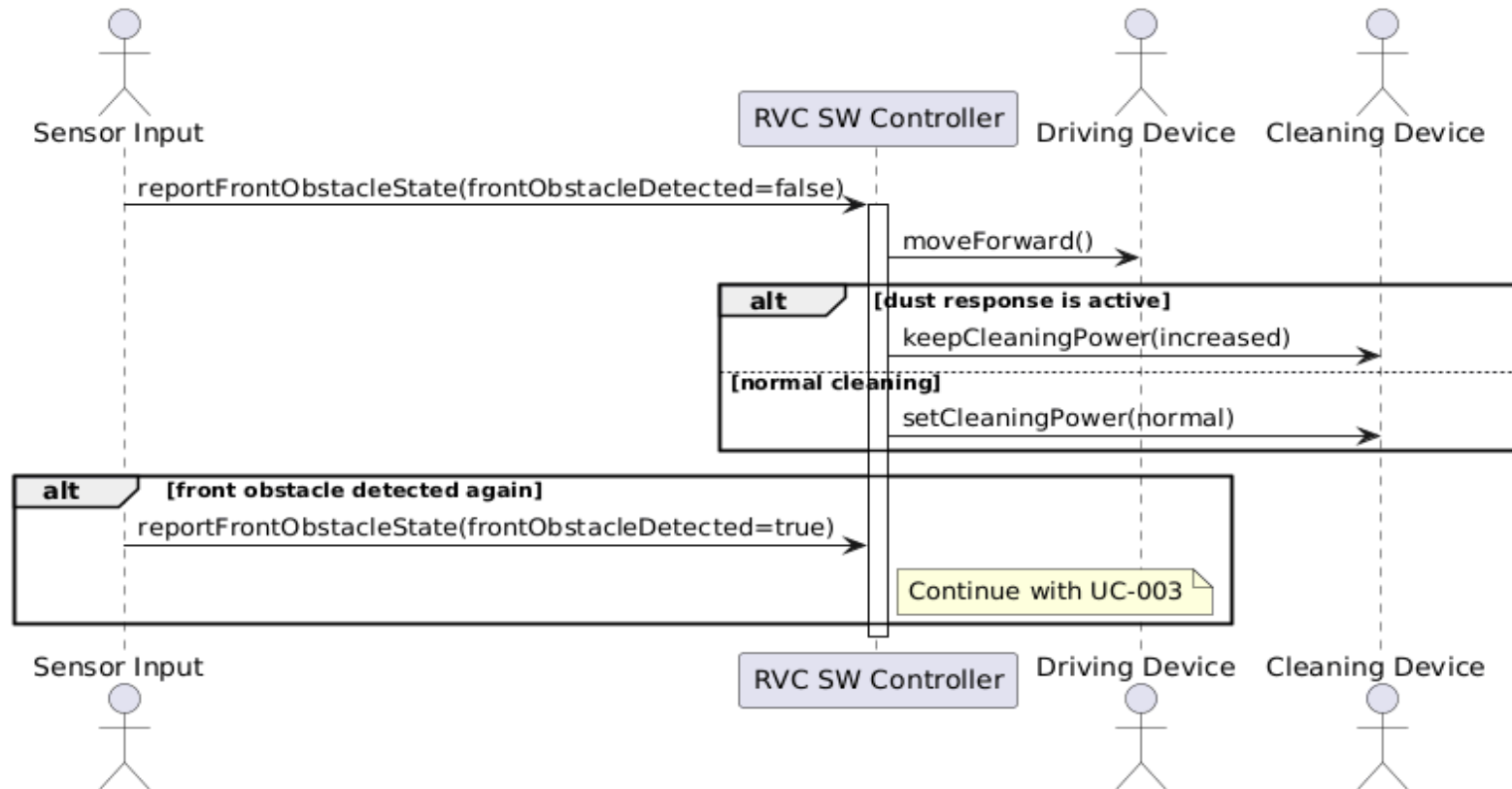


핵심

좌/우 감지 상태와 정책에 따라 회전 방향 결정

Resume Cleaning After Direction Change

SSD - UC-004 Resume Cleaning After Direction Change



핵심

전환 후 전방 열리면 전진 재개, 먼지 대응 중이면 출력 유지

Move Backward After Three-Side Obstacle Detection

SSD - UC-005 Move Backward After Three-Side Obstacle Detection

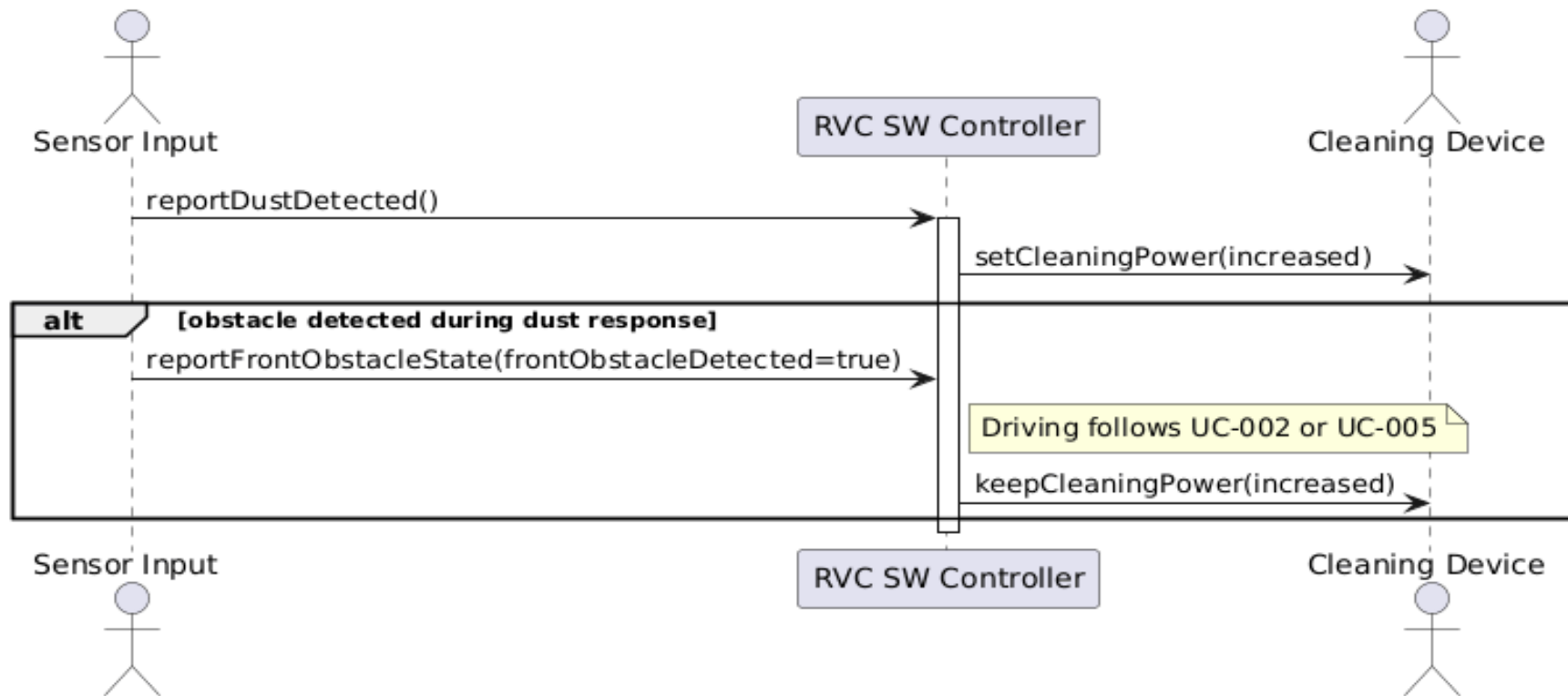


핵심

삼면 막힘 시 후진, 사면 막힘이면 정지 유지

Handle Dust Detection Event

SSD - UC-006 Handle Dust Detection Event

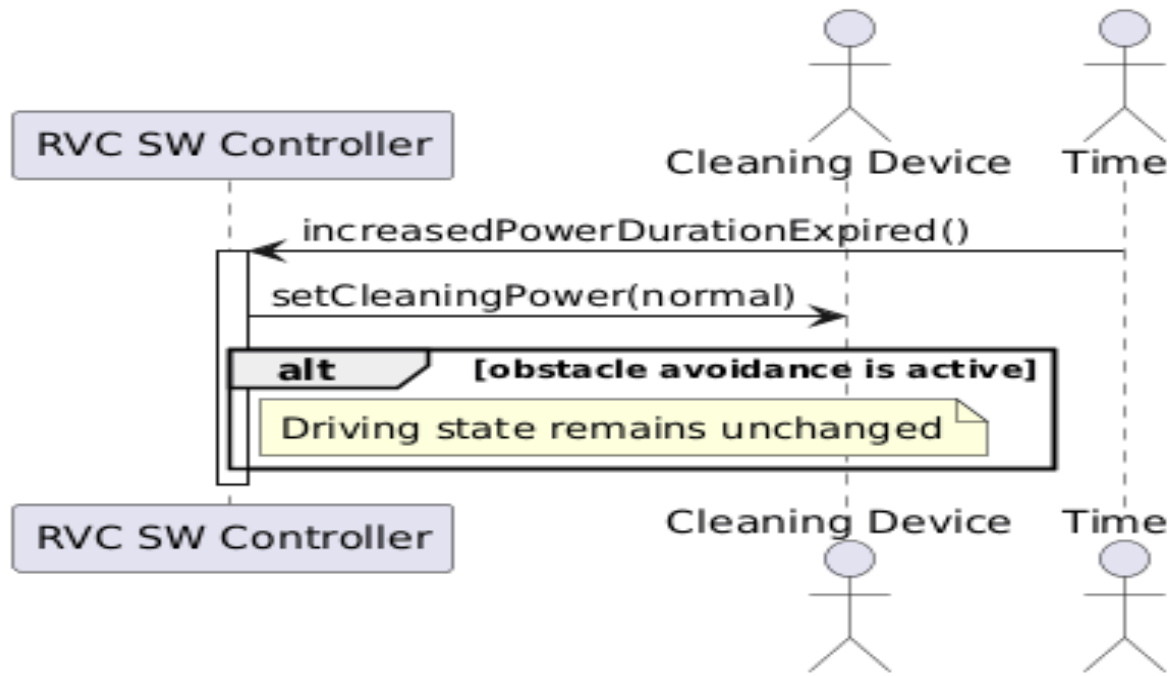


핵심

먼지 감지 시 증가 출력 전환, 장애물 회피 중에도 출력 유지

Return Cleaning Power To Normal

SSD - UC-007 Return Cleaning Power To Normal

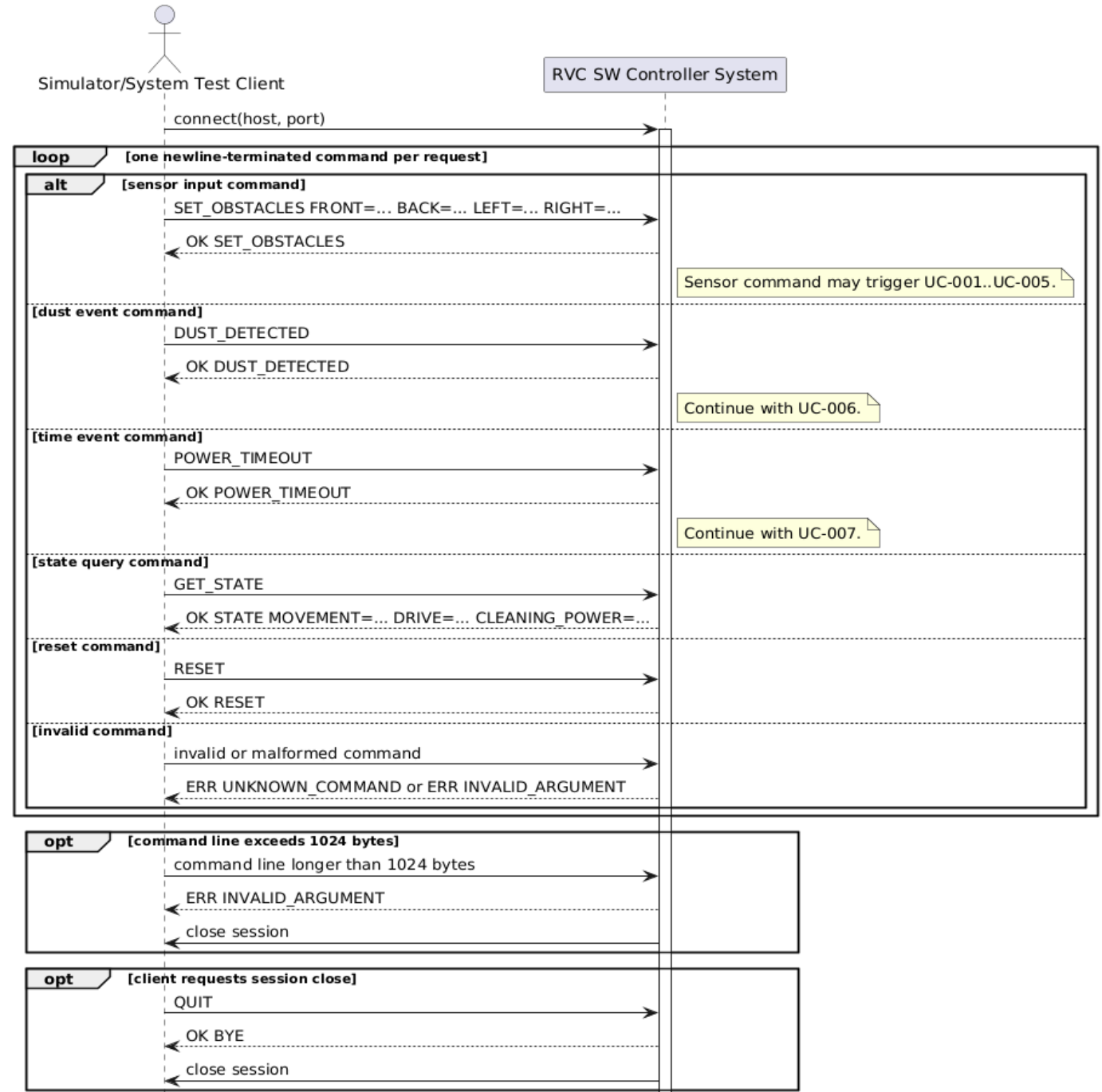


핵심

타이머 만료 시 일반 출력 복귀, 주행 상태는 그대로 유지

SSD · UC-008

Drive Controller And Query State Through TCP Commands



Domain model 작성 프롬프트

첨부 자료

TEXT 1

Use Case.md

TEXTS 2

Ssd.md (1 ~ 8)

Prompt

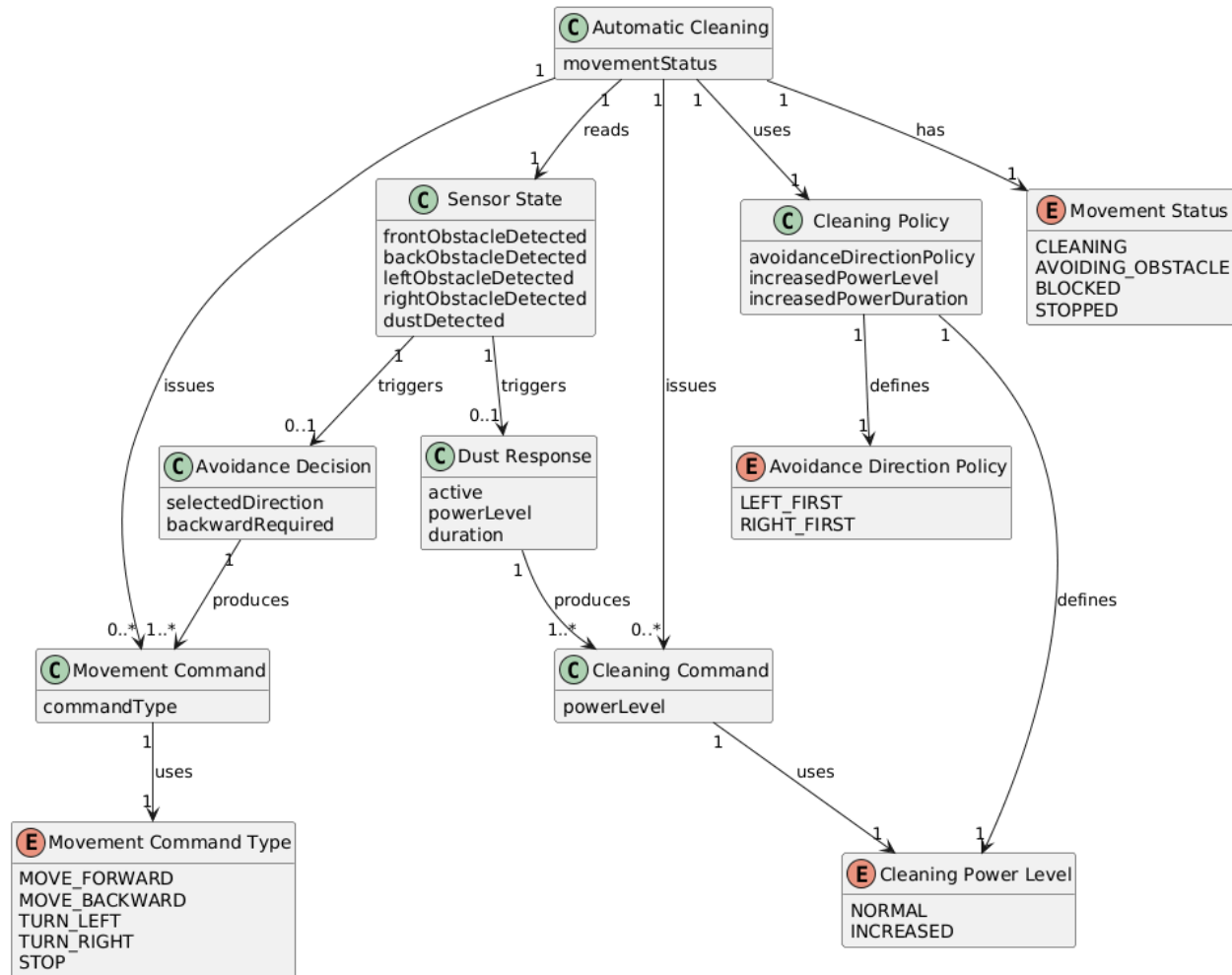
TEXT 1 과 # TEXTS 2 를 보고 도메인 모델을 puml 로 그려줘.
TEXT 1 에서 기능 범위, 핵심 도메인 개념을 추출하고, # TEXT 2 를
사용해서 클래스, 속성, 관계를 정리해줘.

아쉬운 점

도메인 모델의 내용은 괜찮으나, 가독성 있게 그리지 못함
클래스 배치를 수정함

도메인 모델

RVC SW Controller Domain Model



핵심 구조

SensorState

AvoidanceDecision / DustResponse 트리거

AvoidanceDecision

MovementCommand 생성

DustResponse

CleaningCommand 생성

CleaningPolicy

회피 방향 정책 · 출력 수준 · 유지 시간 설정값 보관

Sequence Diagram 작성 프롬프트

첨부 자료

TEXT 1

Use Case.md

TEXTS 2

SSD.md (1 ~ 8)

TEXTS 3

Domain Model

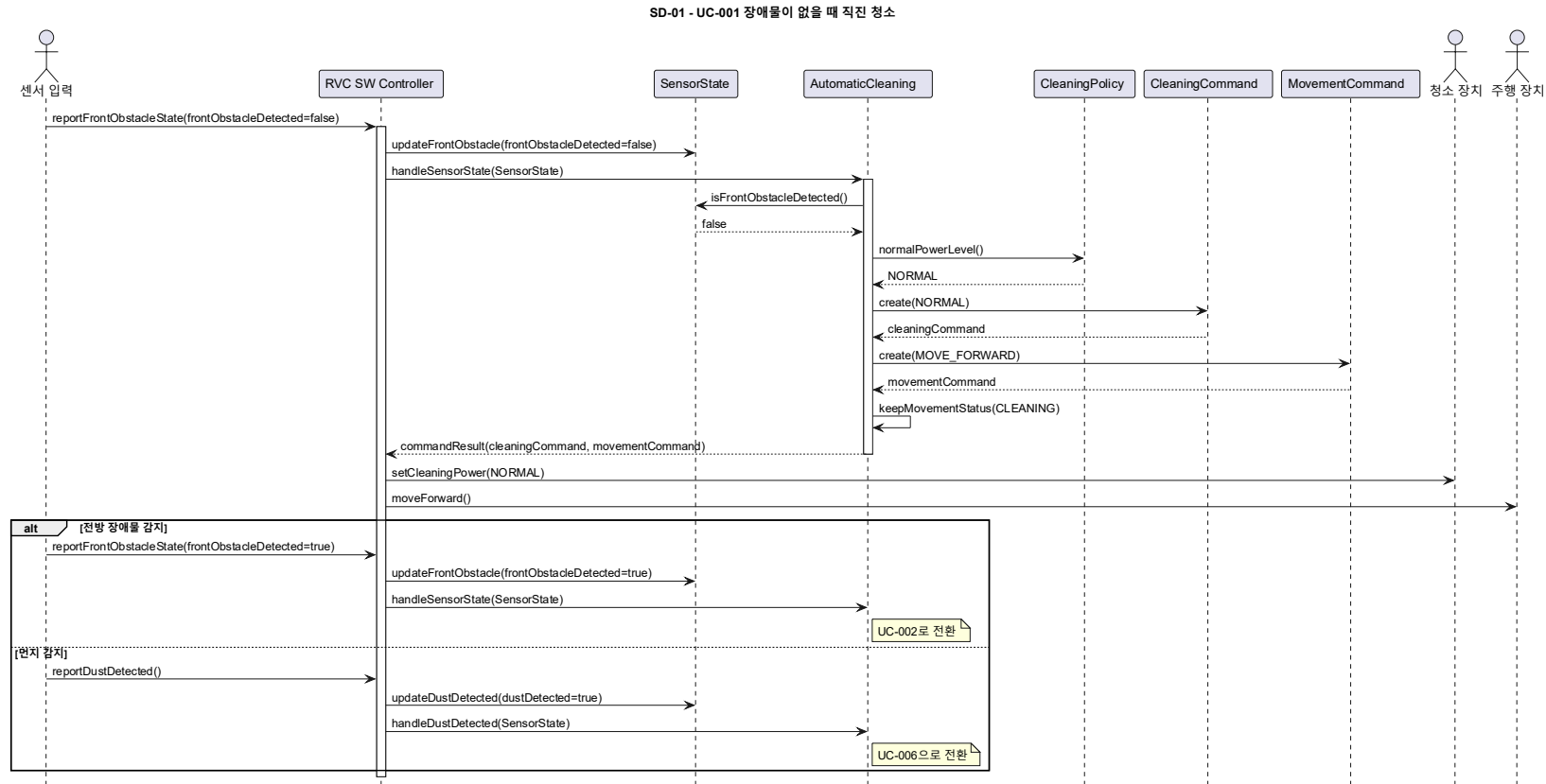
Prompt

TEXT 1과 # TEXTS 2를 보고 각 유스케이스에 해당하는 시스템 내부 객체 간 Sequence Diagram을 puml로 그려줘.
Controller, Domain Logic, Command, Device 호출 흐름이 보이게 정리해줘.

작업 내용

UC-001~UC-008을 SD-01~SD-08로 분리
Controller → SensorState → AutomaticCleaning → Device 경계 명확화
UC-008은 TCP / Protocol Adapter 흐름으로 별도 표현

장애물이 없을 때 직진 청소



핵심

Controller가 SensorState 갱신 후 AutomaticCleaning에서 NORMAL + MOVE_FORWARD 명령 생성

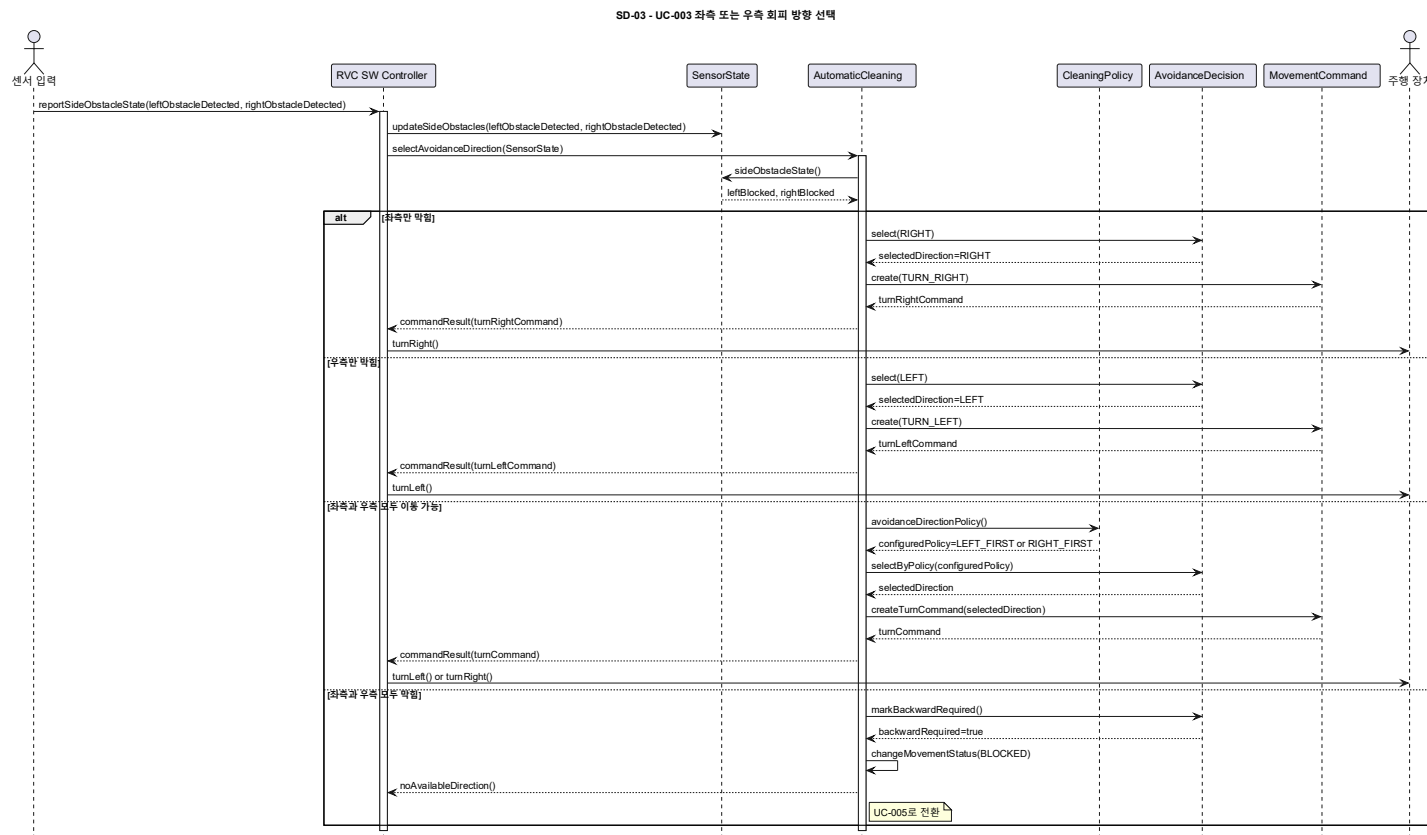
전방 장애물 감지 후 정지

SD-02 - UC-002 전방 장애물 감지 후 정지



핵심 전방 장애물 감지 시 STOP, 삼면 장애물은 UC-005 흐름으로 분기

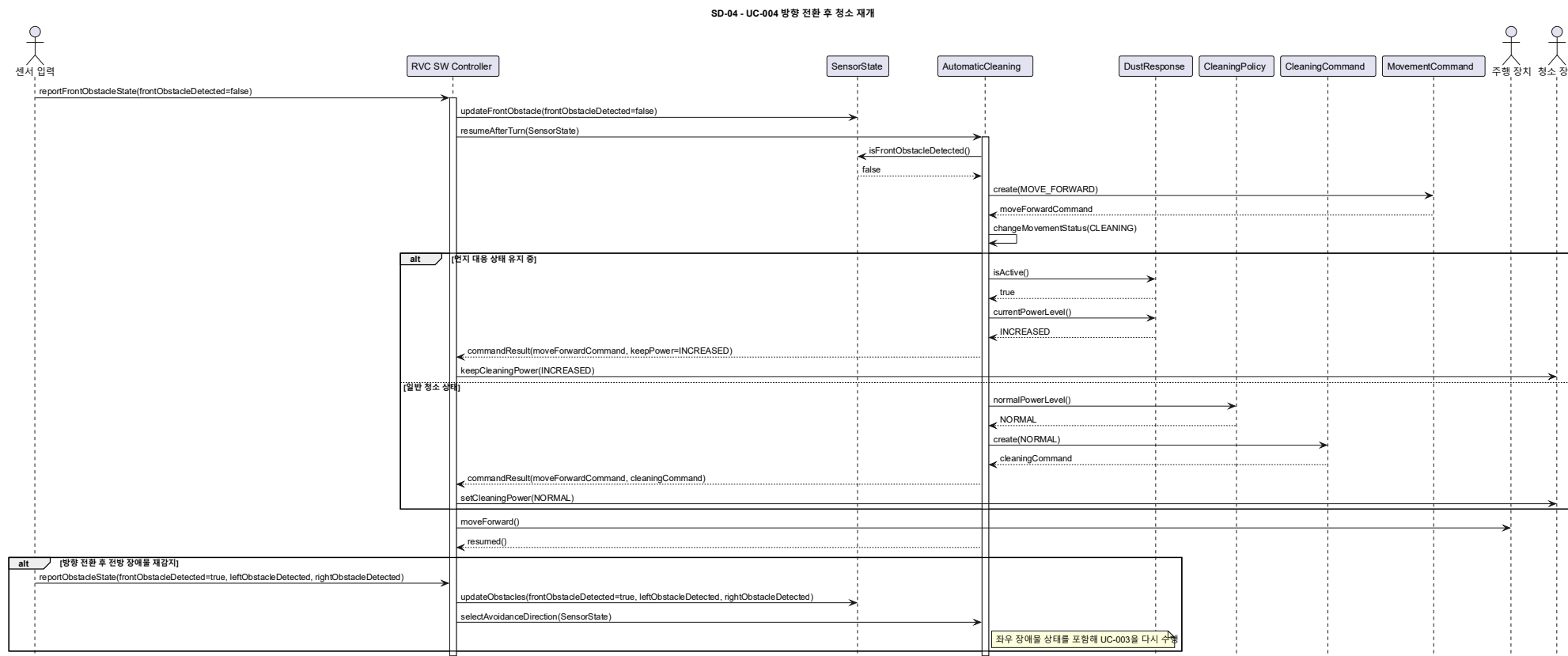
좌측 또는 우측 회피 방향 선택



핵심

좌/우 장애물 상태와 CleaningPolicy에 따라 회피 방향 결정

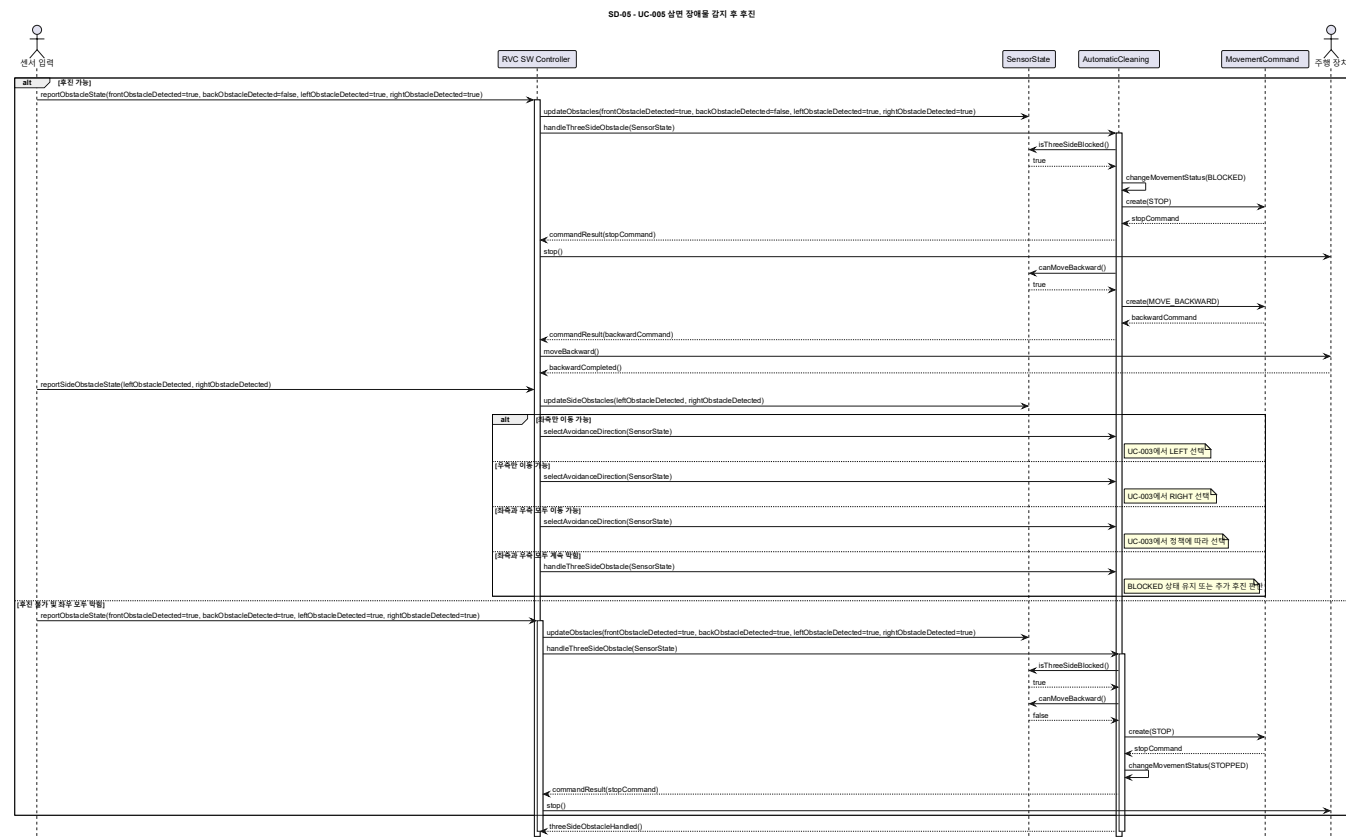
방향 전환 후 청소 재개



핵심

회전 후 전방 이동 가능하면 MOVE_FORWARD로 자동 청소 상태 복귀

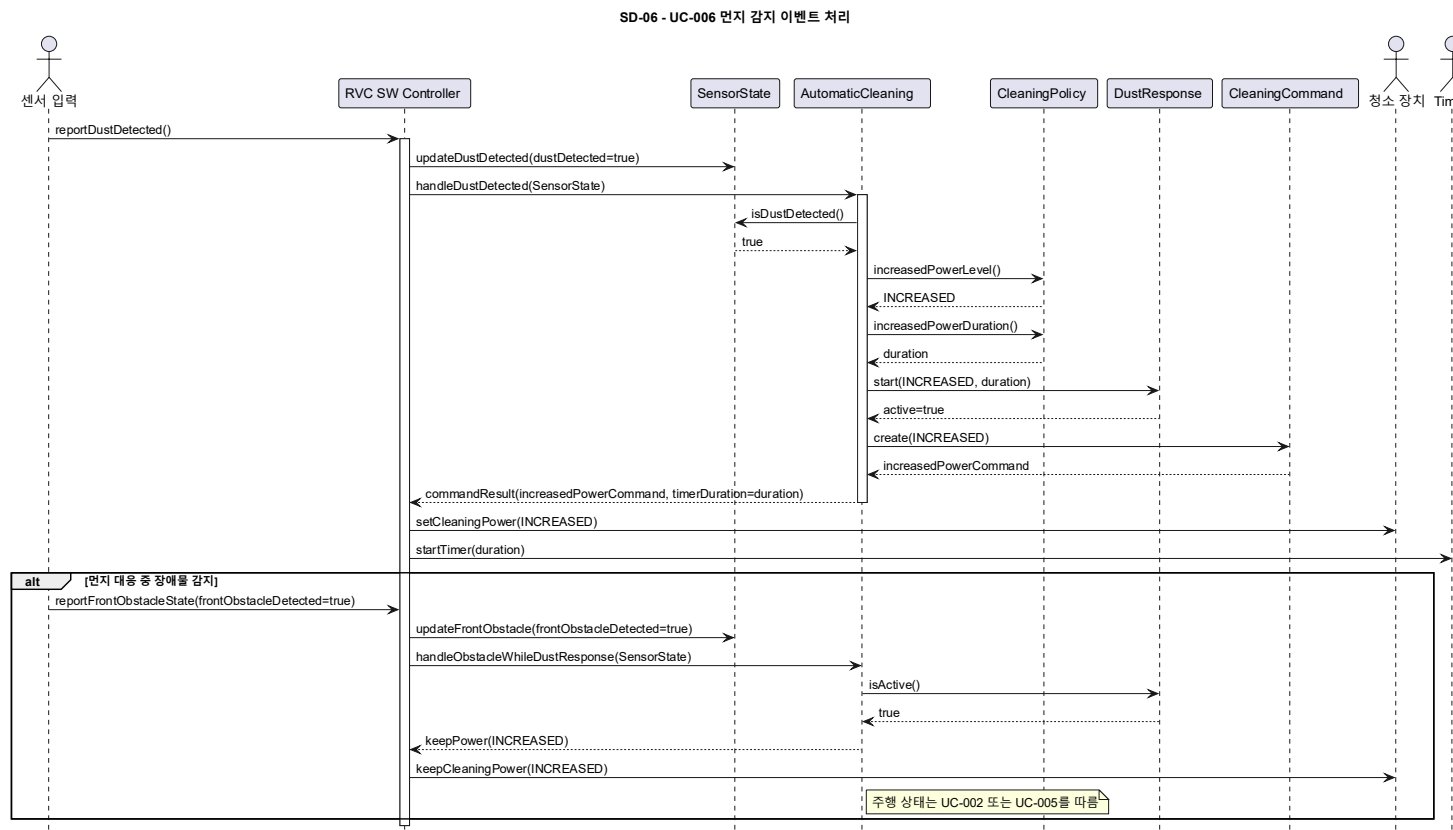
삼면 장애물 감지 후 후진



핵심

전/좌/우 막힘 시 후방 가능 여부로 MOVE_BACKWARD 또는 STOP 결정

먼지 감지 이벤트 처리

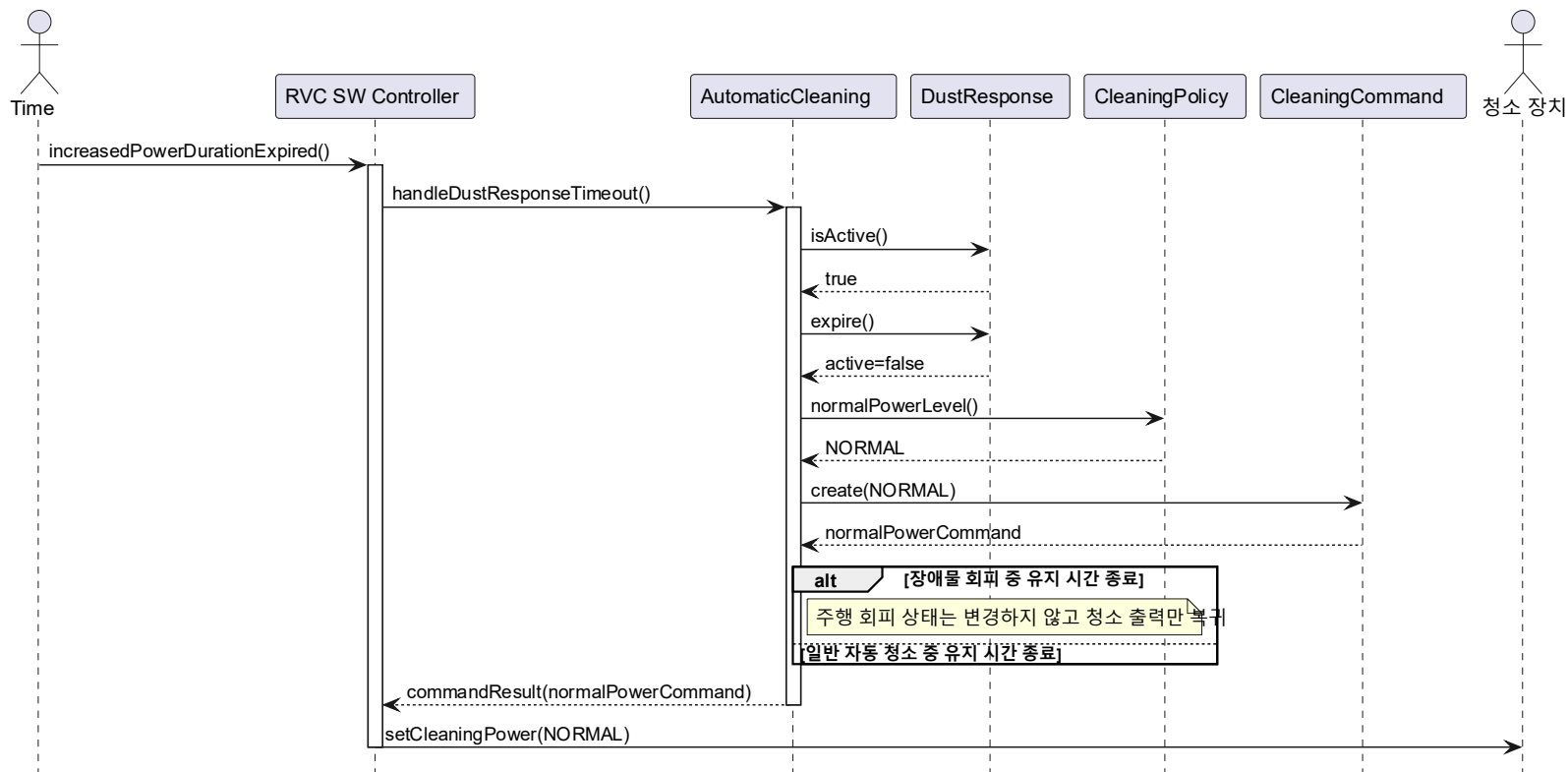


핵심

먼지 감지 시 증가 출력과 타이머를 시작하고, 회피 중이면 pending 처리

청소 출력 일반 상태 복귀

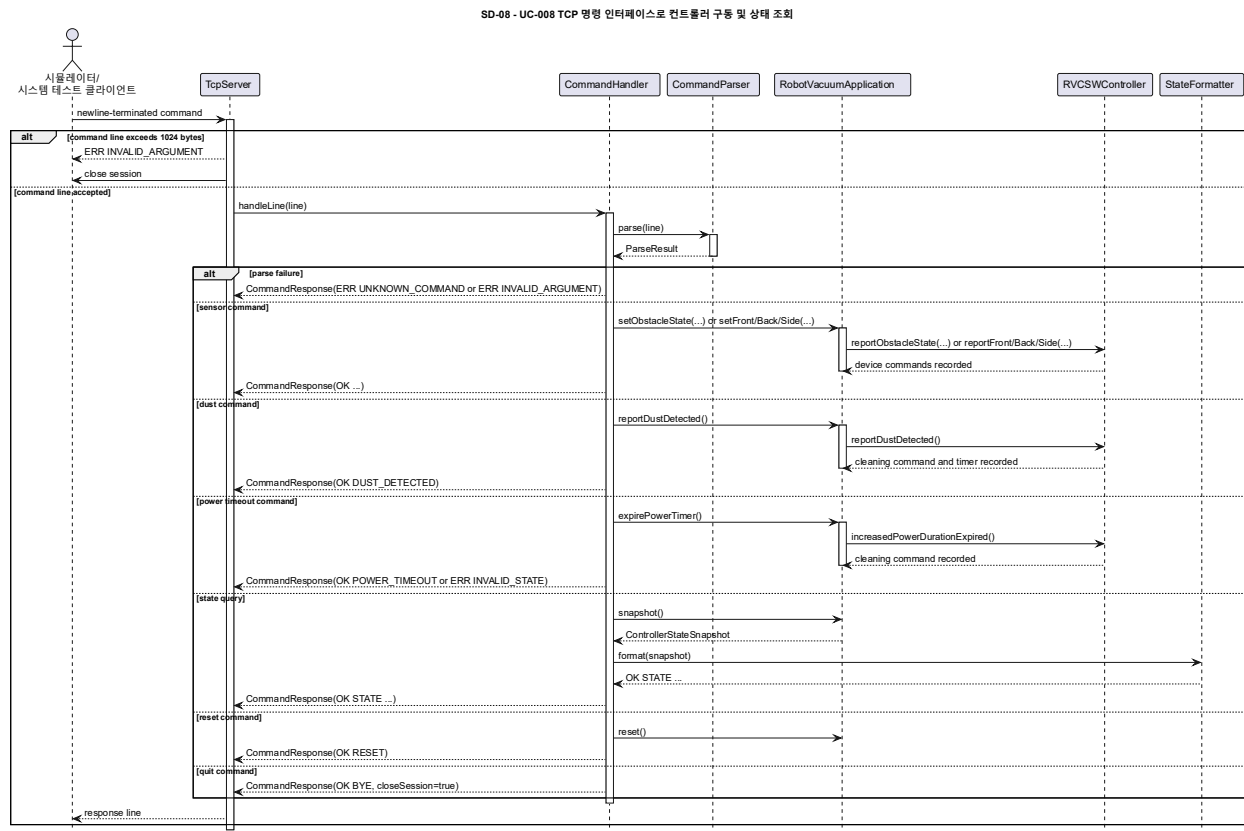
SD-07 - UC-007 청소 출력 일반 상태 복귀



핵심

타이머 만료 시 INCREASED를 NORMAL로 되돌리고 주행 상태는 유지

TCP 명령 인터페이스로 컨트롤러 구동 및 상태 조회



핵심

TCP command를 Parser / Handler / Application 경계에서 컨트롤러 입력과 상태 조회로 변환

Class Diagram 작성 프롬프트

첨부 자료

TEXTS 1

SD.md (1 ~ 8)

TEXTS 2

main branch code

TEXT 3

Command Protocol

Prompt

TEXTS 1과 main branch 구현 구조를 보고 Class Diagram을 puml로 그려줘.
Sequence Diagram에서 드러난 책임과 클래스 간 관계를 패키지 단위로 정리해줘.
Simulator 내부 구현은 외부 경계로 두고 C++ Controller 구조에 집중해줘.

작업 내용

External Interfaces / Protocol Adapter / Domain Model 패키지 분리
CommandResult, MovementCommand, CleaningCommand 생성 관계 반영
TCP command와 simulator support 경계를 클래스 구조에 반영

Robot Vacuum Cleaner

OOI 및 Unit Test

구현 프롬프트

문서대로 구현해달라고 했음에도
확인 결과 문서와 일치하지 않는
부분이 있었다.

따라서 코드가 sequence diagram을
포함한 여러 문서들에 맞게
구현됐는지 확인하라고 프롬프트로
입력했다.

그 결과는 다음과 같다.

1. UC-006 먼지 감지 시 `setCleaningPower(INCREASED)` 가
아니라 `keepCleaningPower(INCREASED)` 가 호출됨

문서상 UC-006은 먼지 감지 시 청소 출력을 증가시키는 흐름이
야.

- `docs/sd/SD-06.puml`
`Controller -> Cleaner:`
`setCleaningPower(INCREASED)`

그런데 현재 코드에서는 `reportDustDetected()` 후
`AutomaticCleaning` 이 `INCREASED` 명령을 반환하고,
`RVCSWController ::executeCleaningCommand()` 에서 먼지
대응이 active면 `keepCleaningPower()` 를 호출해.

- `src/RVCSWController.cpp`

즉, 최초 먼지 감지에도 "새로 증가 출력 설정"이 아니라 "기존 증
가 출력 유지"처럼 동작해. 이걸 수정하는 게 좋아.

3. UC-005의 "후진 불가" 예외 흐름을 코드가 표현할 수 없음

문서 UC-005에는 예외가 있어:

- 더 이상 후진할 수 없고 좌우가 막혀 있으면 정지 유지

그런데 현재 입력 모델에는 `backwardAvailable` 또는 후방 장
애물/후진 가능 여부가 없어. `handleThreeSideObstacle()` 는
삼면이 막히면 항상 `MoveBackward` 를 반환하고, controller는
`stop()` 후 `moveBackward()` 를 실행해.

즉, 현재 구현은 **삼면 장애물 = 항상 후진 가능**으로 처리하고 있어
서 UC-005의 예외 흐름을 구현하지 못해.

2. UC-004 방향 전환 후 청소 재개가 Controller API에서 직접
연결되지 않음

문서에는 UC-004가 별도 흐름으로 있어.

- 방향 전환 후 전방 이동 가능 여부 확인
- `resumeAfterTurn(SensorState)`
- 전진 재개

코드에는 `AutomaticCleaning ::resumeAfterTurn()` 이 구현
되어 있는데, `RVCSWController` 에서 이 메서드를 호출하는
public API가 없어. 현재

`reportFrontObstacleState(false)` 를 호출하면
`handleSensorState()` 로 들어가서 비슷하게 전진/청소를 하
긴 하지만, 문서의 SD-04 흐름과 직접 매핑되지는 않아.

즉, 기능은 비슷하게 되지만 SD-04에 대응되는 controller entry
point가 없음.

구현 프롬프트

다시 한 번 확인을 했음에도 문서와 일치하지 않거나 구현에 부족한 부분이 있었고 팀원들의 리뷰를 통해 수정할 수 있었다.

삼면 장애물 감지 시 후방 입력을 기다리며 정지 명령도 보내지 않습니다.

src/RVCSWController.cpp (line 61)

reportObstacleState(front, left, right)에서 isThreeSideBlocked()이면 바로 return합니다. 후진은 후방 센서 확인 후로 미루는 게 맞지만, UC-005의 안전 동작상 전/좌/우가 막힌 순간에는 최소한 stop()은 실행되어야 합니다. 지금은 후방 이벤트가 오기 전까지 주행이 계속될 수 있습니다.

후방 센서 미확인 상태가 backObstacleDetected=false와 구분되지 않아 후방 확인 없이 후진할 수 있습니다.

include/rvc/SensorState.hpp (line 41), src/AutomaticCleaning.cpp (line 63)

backObstacleDetected_ 기본값이 false라서 "후방이 비어 있음"과 "아직 모름"이 같은 상태입니다. 예를 들어 좌우가 먼저 막힌 뒤 전방 장애물이 들어오면, handleSensorState()가 canMoveBackward()를 true로 보고 바로 moveBackward()를 만들 수 있습니다. std::optional 같은 별도 unknown 상태가 필요해 보입니다.

일반 주행 중 reportSideObstacleState()만 와도 회피 회전을 실행합니다.

src/RVCSWController.cpp (line 42)

UC-003은 장애물 회피 판단 상태가 전제인데, 컨트롤러는 현재 상태를 확인하지 않고 좌우 센서 입력만으로 turnLeft/turnRight를 호출합니다. 전방 장애물이 없는 일반 청소 중 side sensor 이벤트가 들어오면 의도치 않은 회피 회전이 발생할 수 있습니다.

먼저 대응 중 후방 센서로 UC-005가 완료되는 경로에서 증가 출력 유지 명령이 빠집니다.

src/RVCSWController.cpp (line 34)

reportBackObstacleState()는 dust response 여부를 보지 않고 handleThreeSideObstacle()만 호출합니다. 그래서 먼저 대응 중 전/좌/우가 막힌 뒤 후방 센서가 들어오는 단계적 흐름에서는 keepCleaningPower(Increased)가 실행되지 않습니다.

1. RVCSWController::apply() 실행 순서

- 현재: movement 명령 실행 후 cleaning 명령 실행
- 문서: UC-001/UC-004에서 set/keepCleaningPower() 후 moveForward()
- 수정: cleaning command를 먼저 처리하고 movement command를 나중에 실행

2. reportObstacleState(front, left, right)의 회피 재선택 흐름

- 현재: frontObstacleDetected=true면 handleSensorState()로 가서 stop()만 호출
- 문서: SD-04 대안 흐름에서 좌우 상태가 같이 들어오면 selectAvoidanceDirection() 수행
- 수정: 회피 상태에서 front=true이고 좌우 정보가 함께 들어온 경우 selectAvoidanceDirection()으로 연결 필요

3. 정지/회피 중 먼저 감지 pending 처리

- 현재: reportDustDetected()가 항상 즉시 handleDustDetected() 실행, 출력 증가 및 타이머 시작
- 문서: SD-02 대안 흐름은 markDustResponsePending() 명시
- 판단 필요: 문서 의도가 "회피 중에도 즉시 출력 증가"라면 문서 수정, "회피 판단 우선 후 먼저 처리"라면 구현 수정

리뷰 남깁니다.

Findings

[HIGH] 후진 후 방향 전환을 건너뛴니다.

src/RVCSWController.cpp:167 에서 Blocked 상태는 재개/회피 판단 대상이 아니어서, UC-005로 moveBackward 한 뒤 다음 입력이 reportObstacleState(false, false, false, false) 같은 전체 clear snapshot이면 곧바로 handleSensorState() 로 내려가 moveForward 합니다. 그런데 FR-008/UC-005는 후진 뒤 반드시 좌/우 방향 전환 판단을 거치도록 되어 있습니다. 현재 테스트도 tests/RVCSWControllerTest.cpp:247 에서 moveBackward 까지만 검증해서 이 회귀를 못 잡습니다.

Fix: MovementStatus::Blocked 에서 front clear snapshot이 들어오면 selectAvoidanceDirection() 을 먼저 수행하고 turn 명령을 내도록 분기와 테스트를 추가하면 좋겠습니다.

[MEDIUM] 먼저 감지를 회피 중에는 보류하도록 바꾸면서 상위 기능 요구사항과 충돌합니다.

src/RVCSWController.cpp:185 는 cleaning 상태가 아니면 markDustResponsePending() 만 하고 출력 증가/타이머 시작을 하지 않습니다. PR 문서의 UC-006은 이 동작에 맞춰졌지만, docs/functional-and-non-functional-requirements.md:23 의 FR-010/FR-011 은 먼지를 감지하면 일정 시간 동안 청소 출력을 높여야 한다는 요구입니다. 보류가 의도라면 FR도 함께 수정해야 하고, 아니라면 회피 중에도 청소 출력 증가와 타이머는 즉시 시작되어야 합니다.

구현 개요

Controller

RVCSWController

센서 입력 수신 / 주행장치 청소장치 타이머 명령 실행

Domain Logic

AutomaticCleaning

SensorState / SideObstacleState

DustResponse / CleaningPolicy

Command / Decision

MovementCommand / CleaningCommand

CommandResult / AvoidanceDecision

External Interface (Abstract)

DrivingDevice / CleaningDevice / Time

Value Types

Duration, MovementStatus, MovementCommandType, CleaningPowerLevel,
AvoidanceDirection, AvoidanceDirectionPolicy

구현 코드

```
namespace rvc {
class RVCSWController {
private:
    void apply(CommandResult result);
    void applyInitialDustResponse(CommandResult result);
    void applyPendingDustResponseIfCleaning();
    void executeMovementCommand(const MovementCommand& command);
    void executeCleaningCommand(const CleaningCommand& command);
    void startTimerIfNeeded(const std::optional<Duration>& duration);

    DrivingDevice& drivingDevice_;
    CleaningDevice& cleaningDevice_;
    Time& time_;
    SensorState sensorState_;
    AutomaticCleaning automaticCleaning_;
};
```

- RVCSWController는 센서 이벤트를 직접 장치 제어로 바로 연결하지 않는다
- 도메인 로직이 만든 CommandResult를 apply()에서 해석
- 주행 명령, 청소 명령, 타이머 시작을 각각 private helper로 분리
- 먼지 감지 pending 처리도 별도 함수로 분리해 회피 로직과 청소 출력 로직을 분리

RVCSWControllerTest UC-001 ~ UC-005 (1/2)

UC-001 · 장애물 없을 때 직진 청소

전방 clear 시 normal 청소 전원 설정 후 moveForward 호출 검증
전방 clear 시 setNormal이 moveForward보다 먼저 호출되어 UC-001 호출 순서 검증

UC-002 · 전방 장애물 감지 후 정지

전방 blocked 입력 시 stop만 호출되고 AvoidingObstacle 상태 전환 검증
복합 센서 snapshot에서 전방 blocked 시 stop + AvoidingObstacle 경로 검증

UC-003 · 좌측 또는 우측 회피 방향 선택

좌측 blocked/우측 open 시 turnRight 호출 검증
우측 blocked/좌측 open 시 turnLeft 호출 검증
양측 open 시 기본 LeftFirst 정책에 따라 turnLeft 호출 검증
RightFirst 정책 주입 시 양측 open에서 turnRight 호출 검증
양측 blocked/후방 unknown 시 구동 호출 없이 Blocked 상태 검증
일반 주행 중 side sensor만 입력 시 회피 회전 미발생 검증
회피 중 전방 blocked+좌우 상태 snapshot 시 회피 방향 재선택 검증

UC-004 · 방향 전환 후 청소 재개

회피 상태에서 전방 clear 시 normal 설정 후 moveForward 호출 검증
회피 상태에서 전체 clear snapshot 시 normal 설정 후 moveForward 검증

UC-005 · 삼면 장애물 감지 후 후진

전/좌/우 blocked+후방 clear 시 stop 후 moveBackward 호출 순서 검증
사방 blocked 시 stop만 호출되고 Stopped 상태 검증
전/좌/우 blocked+후방 unknown 시 구동 호출 없이 대기 검증
전/좌/우 blocked 이후 후방 clear 입력 시 단계적 후진 흐름 검증
후진 후 전체 clear snapshot 시 바로 전진하지 않고 회피 방향 전환 먼저 검증
후진 후 전방 clear 단일 입력 시 회피 방향 전환 먼저 수행 검증
전/좌/우 blocked 이후 후방 blocked 시 단계적 후진 불가 흐름 검증
일반 상태에서 후방 clear만 입력 시 아무 호출도 없어 선행조건 검증

UC-001 2개 · UC-002 2개 · UC-003 7개 · UC-004 2개 · UC-005 8개 — 총 21개 테스트

RVCSWControllerTest UC-006 ~ UC-007 (2/2)

UC-006 · 먼지 감지 이벤트 처리

먼지 감지 입력 시 setIncreased 후 타이머 시작 검증
custom duration 정책에서 지정한 1234ms 타이머 전달 검증
먼지 응답 중 전방 blocked 시 stop + keepIncreased 호출 검증
먼지 응답 중 단계적 UC-005 후방 clear 시 increased 유지하며 후진 검증
먼지 응답 중 전방 clear 시 keepIncreased 후 moveForward 호출 검증
회피 중 먼지 감지 입력은 즉시 출력 증가 않고 청소 재개 시 UC-006 처리 검증

UC-007 · 청소 출력 일반 상태 복귀

먼지 응답 중 timeout 입력 시 setNormal 호출로 전원 복귀 검증
먼지 응답 없을 때 timeout 입력 시 아무 호출도 없어 비활성 guard 검증
먼지+장애물 상태에서 timeout 시 normal만 복귀하고 회피 상태 유지 검증

UC-006 6개 · UC-007 3개 — RVCSWControllerTest 총 30개 완료

AutomaticCleaningTest 도메인 로직 단위 (15개)

handleSensorState / selectAvoidanceDirection

clear SensorState 시 moveForward + normal 청소 명령 생성 검증
전방 blocked SensorState 시 stop 명령 + AvoidingObstacle 상태 검증
RightFirst 정책 + 양측 open 시 Right 방향 선택 검증
양측 blocked 시 backwardRequired + Blocked 상태 검증

resumeAfterTurn / handleThreeSideObstacle

회전 후 전방 blocked 시 다시 stop + AvoidingObstacle 재개 실패 분기 검증
전/좌/우 blocked+후방 clear 시 stop+moveBackward 도메인 명령 생성 검증
이미 Blocked 상태에서 동일 입력 반복 시 stop 중복 없이 moveBackward만 생성 검증
전방 blocked+한쪽 open 시 선행조건 미충족으로 명령 없음 검증
사방 blocked 시 stop 명령 + Stopped 상태 검증

handleDustDetected

dust=false 시 청소 명령 + 타이머 없어 먼지 감지 guard 검증
dust=true+custom duration 시 Increased 명령 + 타이머 생성 검증

handleObstacleWhileDust / Timeout / Pending

먼지 응답 활성화 중 전방 blocked 시 stop + Increased 유지 명령 생성 검증
활성 상태에서 timeout 시 normal 명령 + 비활성화 처리 검증
clear 후 timeout 입력 시 no-op 검증
pending 표시 후 clear 전까지 활성화 간주 + clear 후 비활성화 검증

도메인 로직 메서드 별 Unit Test 15개 테스트 케이스 구성

SensorStateTest / CommandTest / 결과

SensorStateTest 센서 상태 계산 (3개)

4방향 스냅샷 입력 후 isThreeSideBlocked + canMoveBackward 계산 검증
three-side 상태에서 후방 blocked 갱신 시 후진 가능 여부만 변경 검증
좌측 blocked/우측 open 입력 후 비대칭 SideObstacleState 도출 검증

CommandTest 명령 객체 단위 (5개)

Left/Right 입력 시 TurnLeft/TurnRight 타입과 방향 보존 검증
stop+moveBackward 추가 시 명령 순서와 마지막 명령 보존 검증
zero duration 입력 시 timer 값 보존 검증
three-side SensorState 시 backwardRequired + 방향 없음 상태 검증
방향 선택 후 no-available 처리 시 선택 해제 + backwardRequired=false 검증

100% tests passed, 0 tests failed out of 53

4개 테스트 그룹 · 53개 전체 케이스 · 100% 통과

Robot Vacuum Cleaner

System Test, Simulator, CI

System Test 구현 프롬프트

RVC SW Controller · 시스템 검증

첨부 자료

- docs/command-protocol.md
- docs/usecases.md
- README.md — System Tests 섹션
- rvc_app TCP command/response 구조

Prompt

*rvc_app*을 외부 프로세스로 실행하고 TCP command/response protocol을 검증하는 시스템 테스트를 작성해줘. Python 표준 라이브러리만 사용하고, .rvcst 파일로 테스트 케이스를 정의할 수 있게 해줘. positive/negative case를 분리하고, 서버 시작 실패 시 stdout/stderr가 보이도록 해줘.

작업 내용

- run_system_tests.py 작성
- .rvcst 기반 테스트 케이스 포맷 설계
- run_system_tests.sh / .ps1 / .bat 실행 래퍼
- positive 22개, negative 12개 케이스 구성

아쉬운 점

- 초기에는 성공 케이스 중심 — invalid command, empty line 등 실패 경로는 후속 요청으로 추가
- 앱 실행/종료, 포트 대기, 실패 로그 출력 같은 테스트 러너 안정성은 보완 프롬프트로 해결

TCP 프로토콜 기반 시스템 테스트

RVC SW Controller · 시스템 검증

핵심

- rvc_app을 외부 프로세스로 실행 (C++ 클래스 직접 호출 아님)
- TCP 클라이언트가 접속해 line-based command 전송
- 응답은 OK ... 또는 ERR ... 형식으로 검증
- PING / RESET / GET_STATE / SET_OBSTACLES / DUST_DETECTED / POWER_TIMEOUT / QUIT 흐름 검증

실행 흐름

- 1 rvc_app --host 127.0.0.1 --port 18765 실행
- 2 TCP port가 열릴 때까지 대기
- 3 system_tests/cases/*.rcvst 순회
- 4 각 케이스의 SEND / EXPECT / EXPECT_CONTAINS 실행
- 5 전체 결과 출력 후 앱 종료

```

python3 system_tests/run_system_tests.py --app ./build/rvc_app --host 127.0.0.1 --port 18765
PASS negative_01_unknown_command.rcvst
PASS negative_02_empty_command.rcvst
PASS negative_03_spaces_only_command.rcvst
PASS negative_04_set_front_missing_argument.rcvst
PASS negative_05_set_front_invalid_value.rcvst
PASS negative_06_set_back_invalid_value.rcvst
PASS negative_07_set_side_missing_argument.rcvst
PASS negative_08_set_side_invalid_key.rcvst
PASS negative_09_set_obstacles_missing_argument.rcvst
PASS negative_10_set_obstacles_invalid_key.rcvst
PASS negative_11_too_long_unknown_command.rcvst
PASS negative_12_power_timeout_without_timer.rcvst
PASS positive_01_ping.rcvst
PASS positive_02_get_state_initial.rcvst
PASS positive_03_set_front_clear.rcvst
PASS positive_04_set_front_blocked.rcvst
PASS positive_05_set_back_clear.rcvst
PASS positive_06_set_back_blocked.rcvst
PASS positive_07_set_back_unknown.rcvst
PASS positive_08_set_side_clear.rcvst
PASS positive_09_set_side_left_blocked_avoidance.rcvst
PASS positive_10_set_side_right_blocked_avoidance.rcvst
PASS positive_11_obstacles_all_clear.rcvst
PASS positive_12_obstacles_front_blocked.rcvst
PASS positive_13_three_side_back_available.rcvst
PASS positive_14_all_blocked.rcvst
PASS positive_15_dust_detected_response.rcvst
PASS positive_16_dust_increases_power.rcvst
PASS positive_17_power_timeout_response.rcvst
PASS positive_18_power_timeout_restores_normal.rcvst
PASS positive_19_reset_restores_initial_state.rcvst
PASS positive_20_multiple_commands_server_stays_alive.rcvst
PASS positive_21_quit.rcvst
PASS positive_22_lowercase_ping.rcvst

```

34/34 cases passed

34개 시스템 테스트 전체 통과

RVC SW Controller · 시스템 검증

34/34

cases passed

22

positive cases

12

negative cases

검증 범위

- 알 수 없는 명령 / 빈 명령 / 잘못된 인자 / timer 비활성 상태 timeout 등 예외 입력 검증

의미

- TCP command protocol이 문서와 일치하는지 자동 검증 가능
- simulator와 동일한 외부 클라이언트 관점에서 controller 상태 확인
- Google Test와 별도로 실제 실행 파일 기준 end-to-end 검증 확보

```

▶ python3 system_tests/run_system_tests.py --app ./build/rvc_app --host 127.0.0.1 --port 18765
PASS negative_01_unknown_command.rvcst
PASS negative_02_empty_command.rvcst
PASS negative_03_spaces_only_command.rvcst
PASS negative_04_set_front_missing_argument.rvcst
PASS negative_05_set_front_invalid_value.rvcst
PASS negative_06_set_back_invalid_value.rvcst
PASS negative_07_set_side_missing_argument.rvcst
PASS negative_08_set_side_invalid_key.rvcst
PASS negative_09_set_obstacles_missing_argument.rvcst
PASS negative_10_set_obstacles_invalid_key.rvcst
PASS negative_11_too_long_unknown_command.rvcst
PASS negative_12_power_timeout_without_timer.rvcst
PASS positive_01_ping.rvcst
PASS positive_02_get_state_initial.rvcst
PASS positive_03_set_front_clear.rvcst
PASS positive_04_set_front_blocked.rvcst
PASS positive_05_set_back_clear.rvcst
PASS positive_06_set_back_blocked.rvcst
PASS positive_07_set_back_unknown.rvcst
PASS positive_08_set_side_clear.rvcst
PASS positive_09_set_side_left_blocked_avoidance.rvcst
PASS positive_10_set_side_right_blocked_avoidance.rvcst
PASS positive_11_obstacles_all_clear.rvcst
PASS positive_12_obstacles_front_blocked.rvcst
PASS positive_13_three_side_back_available.rvcst
PASS positive_14_all_blocked.rvcst
PASS positive_15_dust_detected_response.rvcst
PASS positive_16_dust_increases_power.rvcst
PASS positive_17_power_timeout_response.rvcst
PASS positive_18_power_timeout_restores_normal.rvcst
PASS positive_19_reset_restores_initial_state.rvcst
PASS positive_20_multiple_commands_server_stays_alive.rvcst
PASS positive_21_quit.rvcst
PASS positive_22_lowercase_ping.rvcst

```

34/34 cases passed

Simulator 구현 프롬프트

RVC SW Controller · Python Simulator

첨부 자료

- docs/command-protocol.md
- simulator/README.md
- UC-008 TCP 명령 인터페이스 흐름

Prompt

Python Pygame으로 rvc_app에 TCP client로 연결되는 Robot Vacuum Cleaner simulator를 구현해줘. Python은 지도, 먼지, 장애물, 로봇 좌표, 렌더링, 센서 계산을 담당하고, C++ 컨트롤러는 GET_STATE 응답의 DRIVE 값으로 이동 판단만 담당하게 분리해줘. map mode와 manual mode를 모두 지원하고, 디버깅용 상태 패널을 보여줘.

작업 내용

- TCP protocol client 구현
- map / manual mode 분리
- grid map, dust, wall, cleaned cell 렌더링
- GET_STATE 응답 파싱 후 DRIVE 값에 따라 Python world 갱신
- coverage assist와 built-in preset map 추가

아쉬운 점

- 초기 버전은 벽을 따라 반복 이동 — coverage assist를 후속 요청으로 추가
- Python world와 C++ controller 책임 경계를 명확히 지시해야 구현이 깔끔해졌음

Python 시뮬레이터 구현 구조

RVC SW Controller · Python Simulator

핵심 구조

Python (World / Simulator)

- 지도, 로봇 위치/방향, 먼지 위치, 장애물, 청소 완료 셀 관리
- 주변 셀 기반으로 SET_OBSTACLES 명령 생성

C++ Controller (RVC SW Controller)

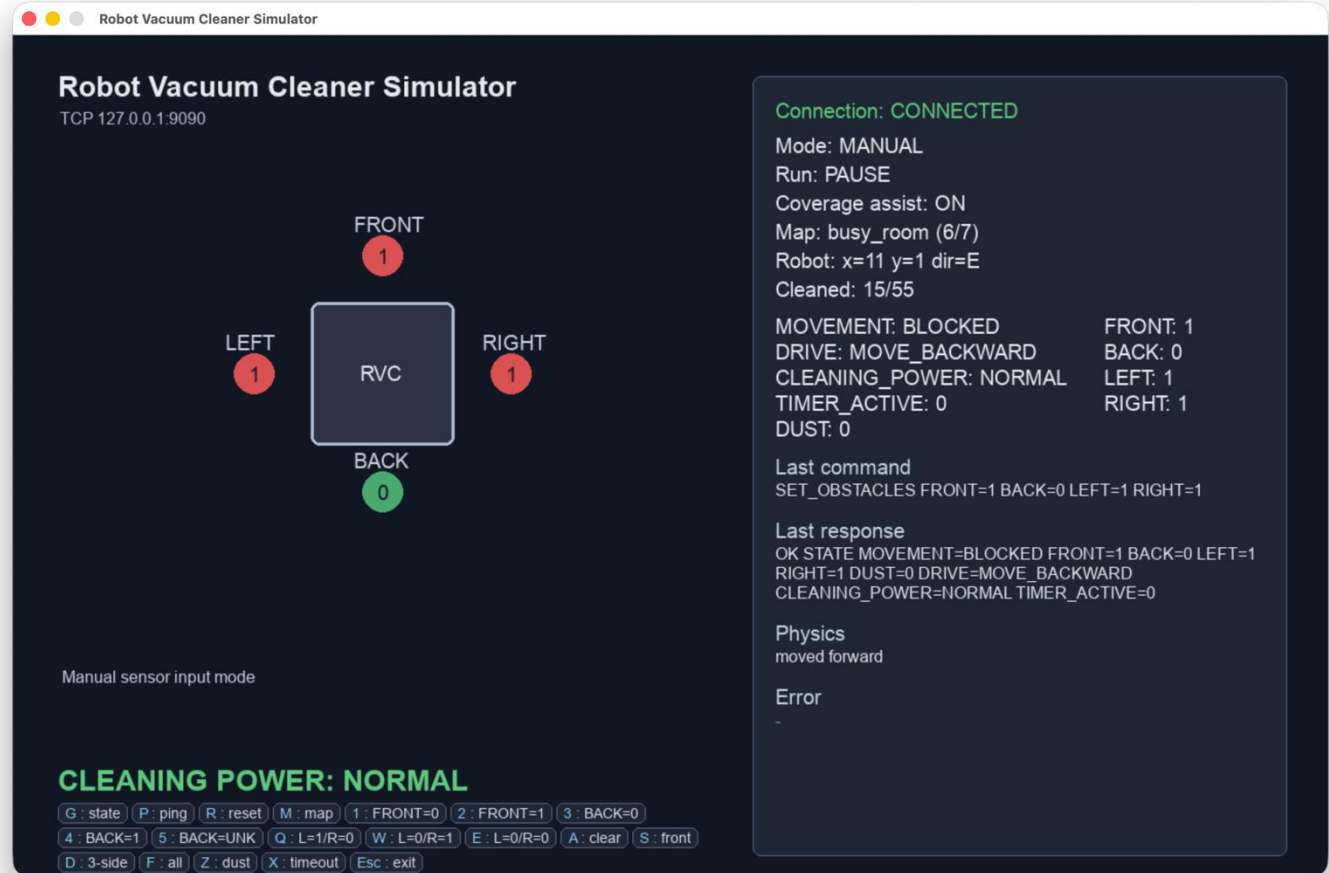
- 센서 입력을 받아 DRIVE / MOVEMENT / CLEANING_POWER 결정
- GET_STATE 응답의 DRIVE 값을 Python이 물리 이동으로 적용

Mode

- MAP: grid map 기반 자동 시뮬레이션
- MANUAL: 키보드로 센서 값을 직접 입력하는 protocol debugging mode

키보드 기능

Space: 단계 실행 | Enter: play/pause | N/V: map 전환 | B: coverage assist
| G: GET_STATE



시뮬레이터 실행 결과

RVC SW Controller · Python Simulator

Map Mode

- busy_room preset에서 robot 위치, 방향, 청소 진행률 표시
- FRONT / BACK / LEFT / RIGHT 센서 값 실시간 표시
- C++ controller 응답에 따라 MOVE_FORWARD / MOVE_BACKWARD / STOP 동작 반영
- cleaned cell과 dust가 grid에서 시각적으로 구분됨

Manual Mode

- 키 입력으로 센서 값을 직접 바꾸며 TCP command 확인 가능
- 삼면 장애물 상황에서 DRIVE=MOVE_BACKWARD 응답 확인
- protocol debugging과 시연용 화면을 동시에 제공

The screenshot displays the 'Robot Vacuum Cleaner Simulator' window. The main area shows a grid map with a robot (blue circle) at (11, 1)E. The map includes cleaned cells (green), dust (yellow diamonds), and obstacles (black). The control panel on the right shows the following status:

```

Connection: CONNECTED
Mode: MAP
Run: PAUSE
Coverage assist: ON
Map: busy_room (6/7)
Robot: x=11 y=1 dir=E
Cleaned: 15/55

MOVEMENT: CLEANING           FRONT: 0
DRIVE: MOVE_FORWARD         BACK: 1
CLEANING_POWER: NORMAL     LEFT: 1
TIMER_ACTIVE: 0            RIGHT: 1
DUST: 0

Last command
SET_OBSTACLES FRONT=0 BACK=1 LEFT=1 RIGHT=1

Last response
OK STATE MOVEMENT=CLEANING FRONT=0 BACK=1 LEFT=1
RIGHT=1 DUST=0 DRIVE=MOVE_FORWARD
CLEANING_POWER=NORMAL TIMER_ACTIVE=0

Physics
moved forward

Error
-
  
```

At the bottom, there are buttons for 'FRONT0', 'BACK1', 'LEFT1', and 'RIGHT1'. Below these, the text 'CLEANING POWER: NORMAL' is displayed. A keyboard control section includes: Space: step, Enter: play/pause, N/V: switch map, R: reset, G: GET_STATE, P: PING, C: reset dust, B: coverage assist, M: manual mode, Esc: exit.

Jenkins Pipeline 구성

RVC SW Controller · Jenkins CI

Pipeline Stages

Checkout

GitHub push 기반 checkout

Configure

CMake Debug build, coverage flag, compile_commands.json 생성

Build

cmake --build build

Test

Google Test 실행 및 JUnit XML 발행

Clang-Tidy

clang-analyzer-* / bugprone-* / performance-* 검사

Cppcheck

warning / style / performance / portability 검사

Publish Static

Jenkins Warnings NG로 정적 분석 결과 게시

Coverage

gcovr로 SonarQube XML + HTML coverage report 생성

SonarQube

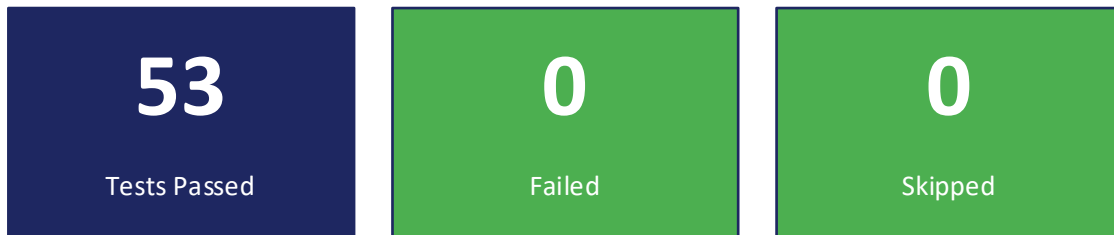
SonarQubeCloud로 품질 분석 전송

Post Actions

pipeline 성공/실패 알림 및 workspace 정리

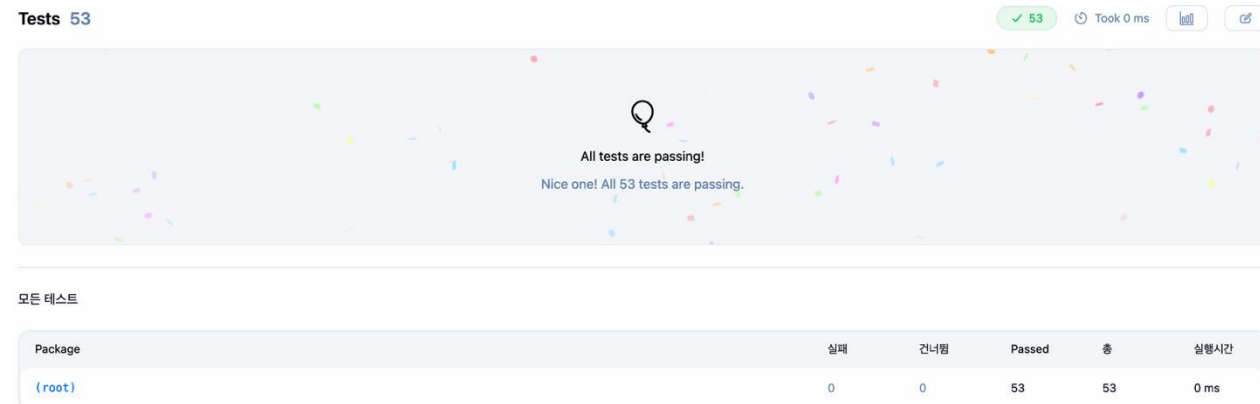
Google Test 결과

RVC SW Controller · Unit Test



의미

- 도메인 로직과 controller command 흐름을 PR마다 자동 검증
- gtest-results.xml을 JUnit report로 발행 → Jenkins UI에서 확인
- 시스템 테스트와 달리 C++ 내부 class 책임을 빠르게 검증



Clang-Tidy 정적 분석

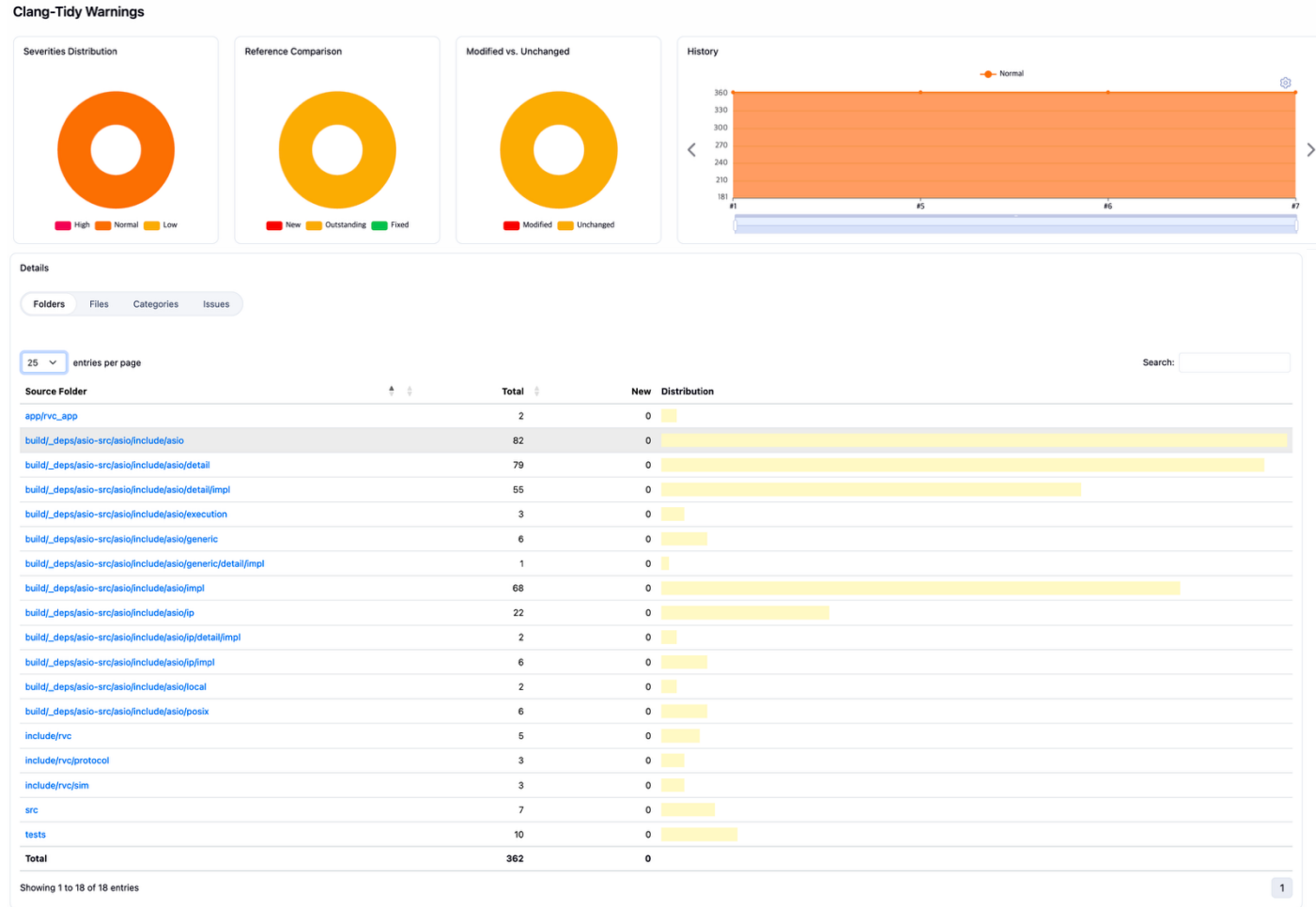
RVC SW Controller · Code Quality

Clang-Tidy

- clang-analyzer-* / bugprone-* / performance-* check 적용
- Jenkins Warnings NG에서 trend와 folder detail 확인
- Normal warning trend 약 360건 / details total 362건

의미

- 컴파일/테스트 통과 후에도 남을 수 있는 정적 품질 이슈를 별도 가시화
- 신규 warning과 기존 warning을 구분해 품질 추적 가능



Cppcheck 정적 분석

RVC SW Controller · Code Quality

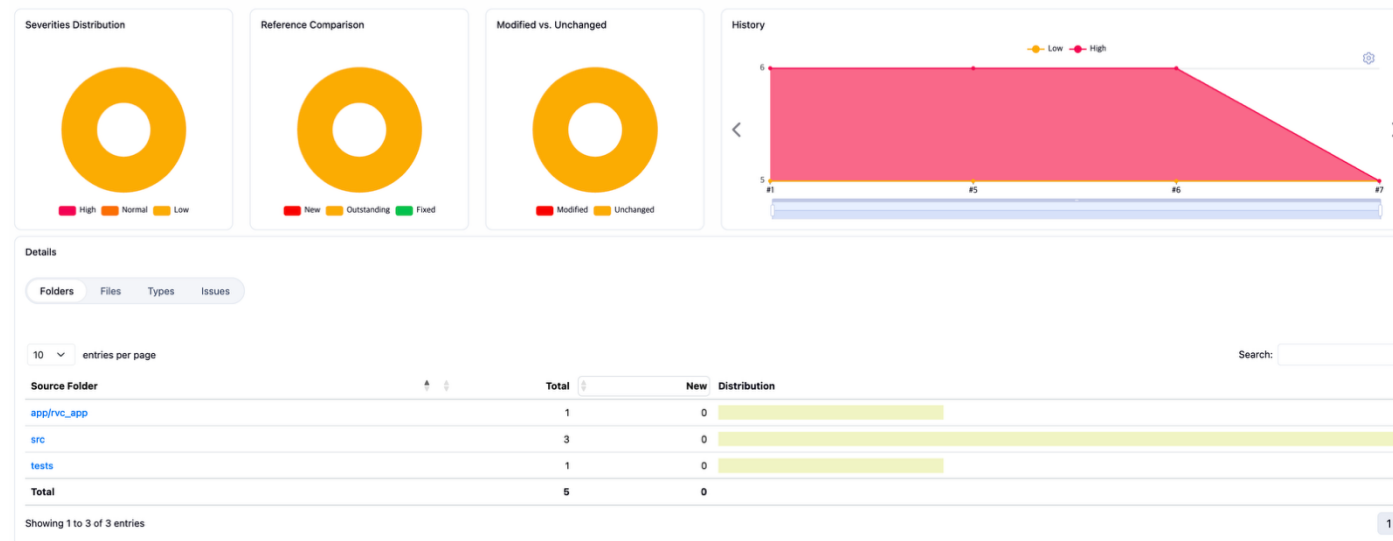
Cppcheck

- warning / style / performance / portability check 적용
- Total 5건 / New 0건 — app 1건, src 3건, tests 1건

의미

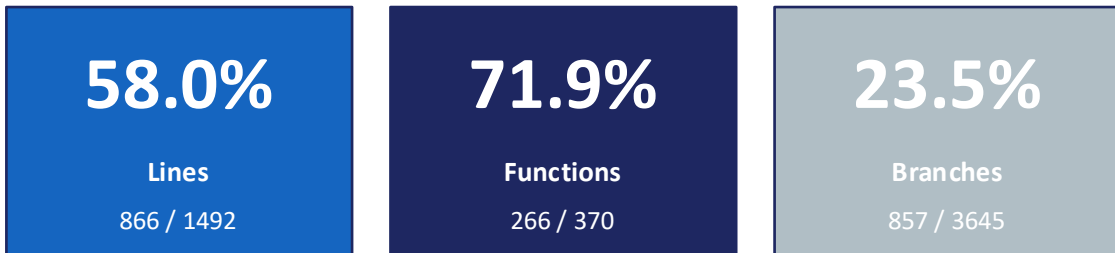
- 컴파일/테스트 통과 후에도 남을 수 있는 정적 품질 이슈를 별도 가시화
- 신규 warning과 기존 warning을 구분해 품질 추적 가능

Cppcheck Warnings



Coverage 리포트

RVC SW Controller · Test Coverage



주요 관찰

- AutomaticCleaning.cpp / Commands.cpp / SensorState.cpp / RVCSWController.cpp 등 핵심 도메인 로직은 비교적 높은 line coverage 확보

GCC Code Coverage Report

Directory: ./

Date: 2026-05-19 11:06:46

Coverage: low: ≥ 0% medium: ≥ 75.0% high: ≥ 90.0%

Exec Total Coverage

Lines: 866 1492 58.0%

Functions: 266 370 71.9%

Branches: 857 3645 23.5%

List of functions

File	Lines	Functions	Branches
app/rvc_app/main.cpp	0.0% 0 / 64	0.0% 0 / 7	0.0% 0 / 122
include/rvc/CleaningPolicy.hpp	100.0% 1 / 1	100.0% 1 / 1	-% 0 / 0
include/rvc/Devices.hpp	100.0% 3 / 3	100.0% 3 / 3	-% 0 / 0
include/rvc/Types.hpp	100.0% 6 / 6	100.0% 6 / 6	-% 0 / 0
src/AutomaticCleaning.cpp	92.7% 101 / 109	84.2% 16 / 19	57.0% 98 / 172
src/CleaningPolicy.cpp	100.0% 13 / 13	100.0% 5 / 5	-% 0 / 0
src/Commands.cpp	98.6% 72 / 73	100.0% 26 / 26	83.3% 5 / 6
src/DustResponse.cpp	87.5% 14 / 16	80.0% 4 / 5	-% 0 / 0
src/net/TcpServer.cpp	0.0% 0 / 67	0.0% 0 / 9	0.0% 0 / 86
src/protocol/CommandHandler.cpp	0.0% 0 / 63	0.0% 0 / 8	0.0% 0 / 105
src/protocol/CommandParser.cpp	0.0% 0 / 169	0.0% 0 / 28	0.0% 0 / 368
src/protocol/StateFormatter.cpp	0.0% 0 / 51	0.0% 0 / 6	0.0% 0 / 88
src/RVCSWController.cpp	74.7% 133 / 178	94.1% 16 / 17	42.2% 141 / 334
src/SensorState.cpp	94.2% 49 / 52	94.7% 18 / 19	77.8% 14 / 18
src/sim/ControllerStateSnapshot.cpp	0.0% 0 / 29	0.0% 0 / 10	-% 0 / 0
src/sim/RecordingDevices.cpp	0.0% 0 / 52	0.0% 0 / 18	0.0% 0 / 4
src/sim/RobotVacuumApplication.cpp	0.0% 0 / 68	0.0% 0 / 11	0.0% 0 / 14
tests/RVCSWControllerTest.cpp	99.2% 474 / 478	99.4% 171 / 172	25.7% 599 / 2328

Generated by: [GCOVR \(Version 7.0\)](#)

SonarQube Quality Gate

RVC SW Controller · Quality Analysis

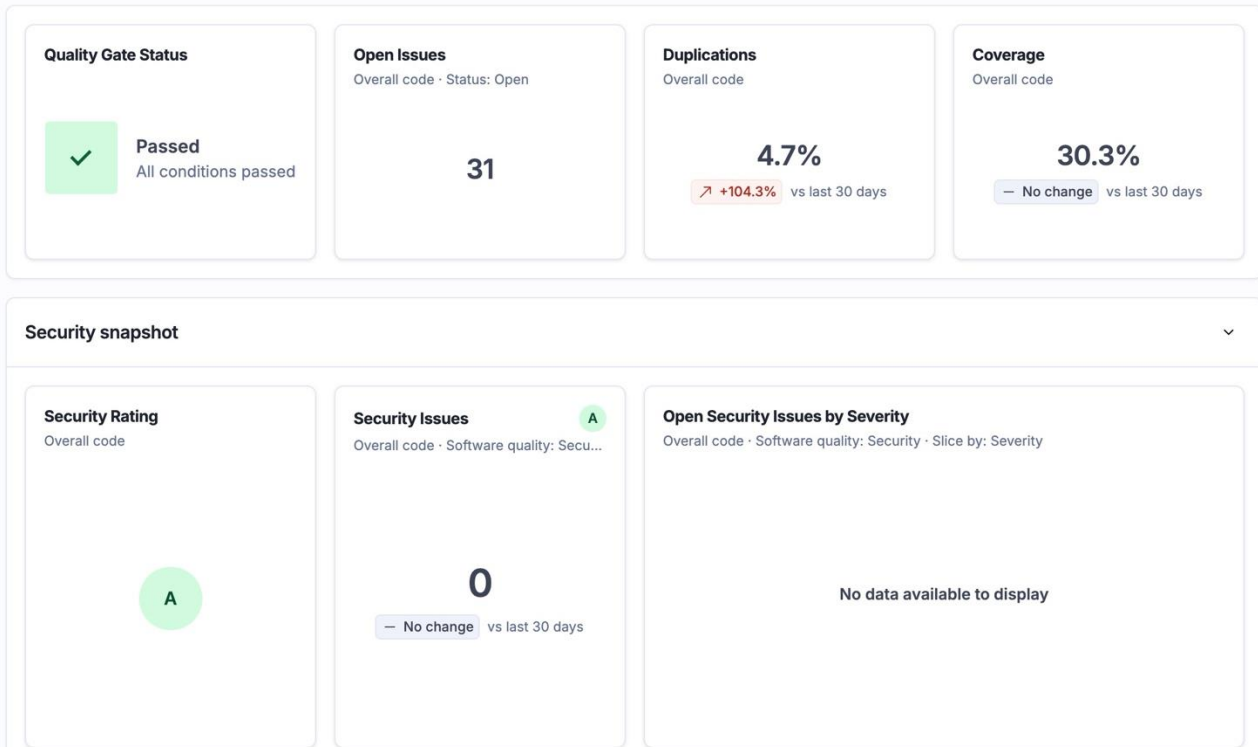


의미

Jenkins pipeline 결과와 별도로 SonarQubeCloud에서 프로젝트 품질 추적. Quality Gate 통과 여부로 merge 이후 품질 상태를 빠르게 확인 가능.

Project health dashboard

See your project's branch health at a glance by exploring trends and risk breakdowns.



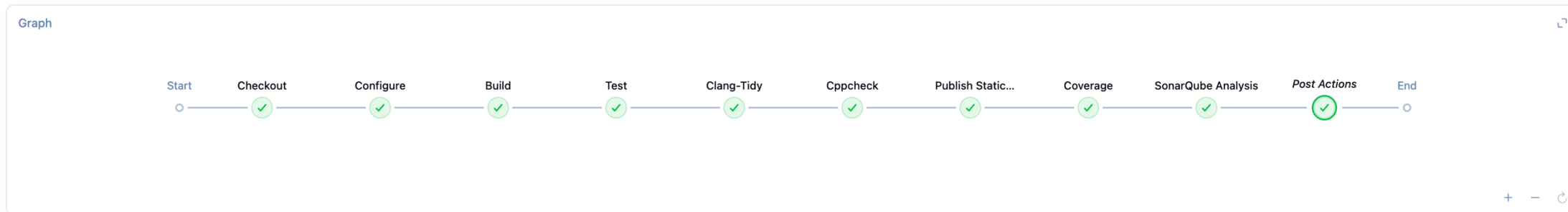
전체 파이프라인 성공

RVC SW Controller · Jenkins CI

#8 Jenkins Build	4m 42s 소요 시간	10/10 Stages 통과
----------------------------	------------------------	---------------------------

성공 Stage

- ✓ Checkout
- ✓ Configure
- ✓ Build
- ✓ Test
- ✓ Clang-Tidy
- ✓ Cppcheck
- ✓ Publish Static Analysis
- ✓ Coverage
- ✓ SonarQube Analysis
- ✓ Post Actions



Codex 사용 방식

프롬프트 기반 구현 보조

사용 방식

문서 먼저 읽기

README.md, AGENT.md, requirements.md

구현 요청

OOAD+OOI/unit, systemtest/simulator를 기능단위로 분리하여 요청

검증 요청

테스트 결과, Jenkins 결과, 정적 분석 결과 기반 보완 요청

리뷰 요청

문서와 구현 일치 여부, 빠진 예외 케이스 점검

핵심

큰 작업을 한 번에 맡기기보다 기능 단위로 쪼개고, 결과를 다시 검증하는 방식이 효과적이었음

대표 프롬프트

"*svc_app*을 외부 프로세스로 실행하는 시스템 테스트 runner를 작성해줘."

"*Python simulator*와 *C++ controller* 책임을 분리해서 TCP 기반으로 연결해줘."

"현재 문서와 구현 흐름이 일치하지 않는 부분을 찾아줘."

AGENT.md

- 개발 전 `docs/` 문서를 먼저 확인한다.
- 문서 목록: [README - Folder Structure](README.md#folder-structure)
- 유스케이스 맵: [README - Current Use Case Map](README.md#current-use-case-map)

Codex 사용 장점

프롬프트 기반 구현 보조

장점

- 문서 기반 구현 속도가 빨라짐
- 반복적인 테스트 케이스와 스크립트 작성 부담 감소
- CMake, Python runner, Jenkinsfile처럼 여러 기술 스택을 연결하는 작업에 유용
- 에러 로그를 바탕으로 원인 분석과 수정 방향을 빠르게 제시
- 구현 후 README, system test 문서, 실행 명령까지 함께 정리 가능
- 요구사항, 유스케이스, 프로토콜 문서 기반으로 구현 흐름을 맞춰갈 수 있었음
- 테스트 실패나 CI 설정 문제를 단계별로 좁혀가며 수정하는 데 도움이 됨

Codex 사용 단점 및 주의점

프롬프트 기반 구현 보조

단점

- 처음 생성된 결과가 문서와 완전히 일치하지 않는 경우가 있었음
- 예외 케이스나 경계 조건은 직접 리뷰하고 추가 요청해야 했음
- Jenkins, SonarQube처럼 외부 환경 의존성이 있는 작업은 실제 실행 전까지 확신하기 어려움
- 너무 큰 프롬프트를 한 번에 주면 일부 요구사항이 누락될 수 있음

주의점

- 산출물을 그대로 신뢰하지 않고 테스트와 코드 리뷰로 검증해야 함
- 어떤 파일/명령/검증 기준을 사용할지 명확히 지시해야 함
- 문서와 구현 차이를 발견하면 후속 프롬프트로 재검토 요청 과정이 필요함

Codex 사용 시 체크리스트

- ✓ 생성된 코드를 그대로 사용하지 않고 리뷰
- ✓ 예외 케이스를 직접 식별하고 추가 요청
- ✓ 프롬프트를 기능 단위로 작게 분리
- ✓ 외부 환경(CI, 배포)은 실제 실행으로 검증
- ✓ 문서와 구현 일치 여부 주기적 점검

소감

System Test · Simulator · CI

소감

- Codex는 구현 초안을 빠르게 만들고 테스트 자동화 구조를 잡는 데 큰 도움이 되었음
- 최종 품질은 프롬프트보다 검증 과정에서 결정된다고 느꼈음
- 시스템 테스트, 시뮬레이터, ci를 연결하면서 "문서화된 요구사항 → 실제 실행 가능한 검증" 과정을 경험

마무리

이번 작업을 통해 RVC SW Controller는 unit test · system test · simulator · CI pipeline을 모두 갖춘 검증 가능한 구조로 확장되었다.

53

Unit Tests

34

System Tests

10

CI Stages