

# Traditional vs Agent Coding #6 Maintenance

## 10팀

202112349 조용국

202211342 이윤원

202211359 장상욱

202211269 김민수

# Table of Contents

Traditional Coding 03

---

UP 수정사항

Vibe Coding 13

---

Vibe Coding 수정사항

Difference and Analysis 38

---

결과 비교

# Traditional Coding

# UP Inception Phase

## Use-Cases [수정]

Use Case Number	Functional Requirement	Use Case Name	Actor	Category
UC-01	RVC 전원 켜기	Power ON	User	Evident
UC-02	RVC 전원 끄기	Power OFF	User	Evident
UC-03	장애물 회피	Avoid Obstacle	Obstacle Sensor (Front/Left)	Evident
UC-04	흡입 파워 조절	Adjust Suction Power	Dust Sensor	Evident
UC-05	RVC 청소 수행	Perform Cleaning	User	Evident

# UP Inception Phase

## Use-Cases in Brief [수정]

<b>Use Case</b>	UC-01. Power ON
<b>Actor</b>	User
<b>Description</b>	<ul style="list-style-type: none"><li>- 사용자가 RVC의 전원 버튼을 누르면, RVC가 전원을 켜고 센서(Front/Left/Dust) 및 모터(이동 모터, 흡입 모터)를 초기화한다.</li><li>- 초기화가 정상적으로 완료되면 자동 청소(UC-05 Perform Cleaning)를 시작한다.</li><li>- 초기화에 실패하면 에러 상태로 진입하고 동작하지 않는다.</li></ul>

<b>Use Case</b>	UC-02. Power OFF
<b>Actor</b>	User
<b>Description</b>	<ul style="list-style-type: none"><li>- 사용자가 RVC의 전원 버튼을 누르면, RVC가 현재 수행 중인 이동 동작(전진, 후진, 회전)을 정지하고, 흡입 모터를 정지한 뒤 전원을 끈다.</li><li>- RVC가 어떤 상태(청소 중, 회피 중, 파워 업 중)에 있더라도 즉시 모든 동작을 안전하게 종료한다.</li></ul>

# UP Inception Phase

## Use-Cases in Brief [수정]

<b>Use Case</b>	UC-03. Avoid Obstacle
<b>Actor</b>	Obstacle Sensor (Front Sensor, Left Sensor)
<b>Description</b>	<ul style="list-style-type: none"><li>- RVC가 청소 중 전방에 장애물을 감지하면, 전진과 흡입을 멈추고 좌측의 장애물 유무를 확인한다.</li><li>- 전방에만 장애물이 있으면 좌측으로 회전한다.</li><li>- 전방과 좌측에 장애물이 있으면 우측으로 회전한다.</li><li>- 전방과 좌측에 장애물이 있어 우측으로 회전 후 전방에 장애물이 있으면 좌측으로 회전 후 후진한 뒤 좌측으로 회전한다(기본 우선순위: 좌측). 회피가 완료되면 전진과 흡입을 재개하여 청소를 계속한다.</li></ul>

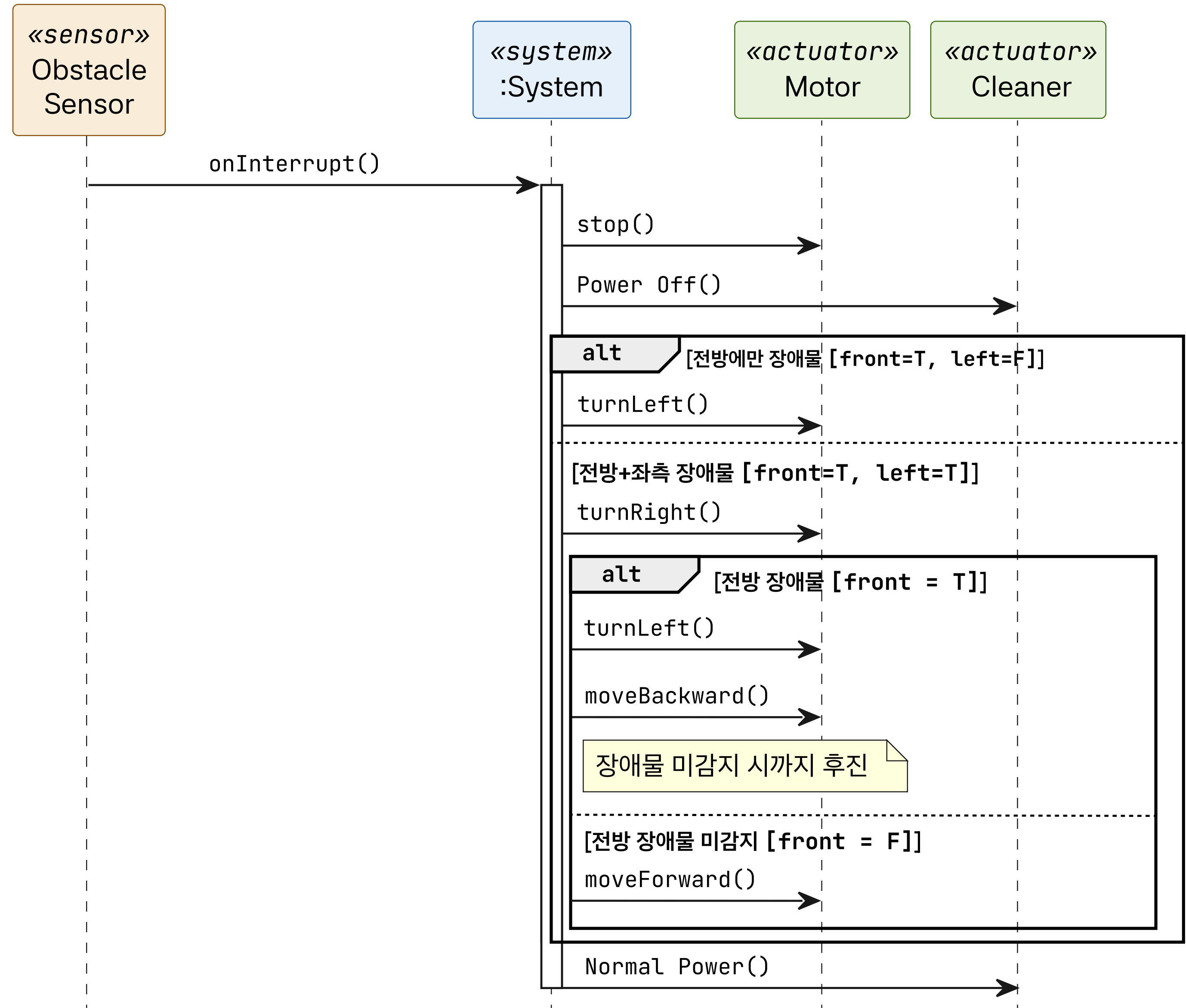
# Use-Case in Casual [수정]

Use-Case	UC-03. Avoid Obstacle
Primary Actor	Obstacle Sensor (Front Sensor, Left Sensor)
Secondary Actor	Motor, Cleaner
Description	RVC가 청소 중 센서에 의해 전방 또는 측면에 장애물이 감지되었을 때, 회피 동작을 수행한 뒤 청소를 재개합니다.
Pre-Requisites	RVC가 청소(전진 및 흡입)을 진행 중이어야 합니다.
Typical Course of Events	<p>(A): Actor, (S): System</p> <ol style="list-style-type: none"> <li>1.(A) Obstacle Sensor가 전방에 장애물을 감지합니다.</li> <li>2.(S) RVC가 전진 주행 및 흡입 정지를 명령합니다.</li> <li>3.(A) Motor가 전진 주행을 멈추고, Cleaner가 흡입을 정지합니다.</li> <li>4.(S) RVC가 장애물 방향을 판단하고, 회피 방향을 Motor에 명령합니다.</li> <li>5.(A) Motor가 지시에 따라 회피 동작을 수행합니다.</li> <li>6.(S) RVC가 회피 완료후, 전진 주행 및 흡입 재개를 명령합니다.</li> <li>7.(A) Motor가 전진 주행을 재개하고, Cleaner가 흡입을 재개합니다.</li> </ol>
Alternative Course of Events	<p>Line 4: (S) 전방에만 장애물이 존재하는 경우, 좌측 회전을 명령합니다.</p> <p>Line 4: (S) 전방 및 좌측에 장애물이 존재하는 경우, 우측 회전을 명령하고, 회전 후 전방에 장애물이 있을 시 좌측 회전을 명령하고 후진을 명령합니다.</p> <p>좌측에 장애물이 존재하는 경우, 우측 회전을 명령하고, 전방의 장애물을 감지합니다.</p> <p>우측 회전 후 전방에 장애물이 존재하는 경우, 다시 좌측 회전을 명령하고 후진을 진행합니다.</p>
Exceptional Courses of Events	Line 1: (S) Sensor와의 연결이 끊기거나 비정상적인 데이터가 입력될 경우, 에러 메시지를 표시합니다.

SSD-03

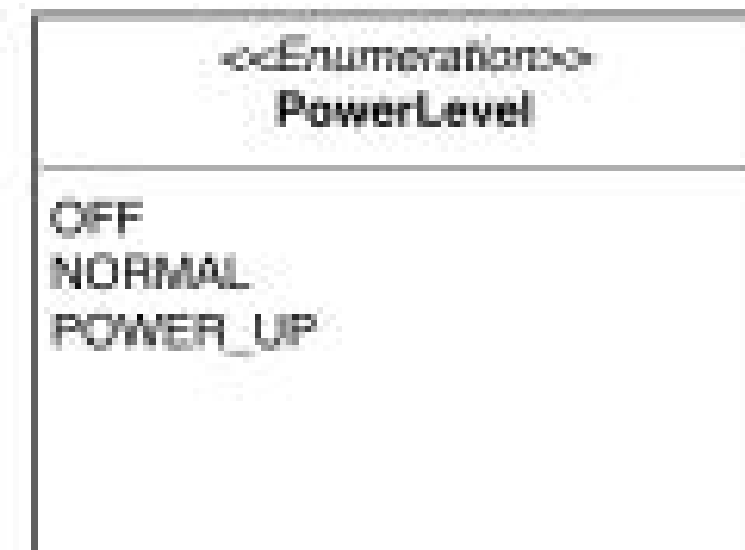
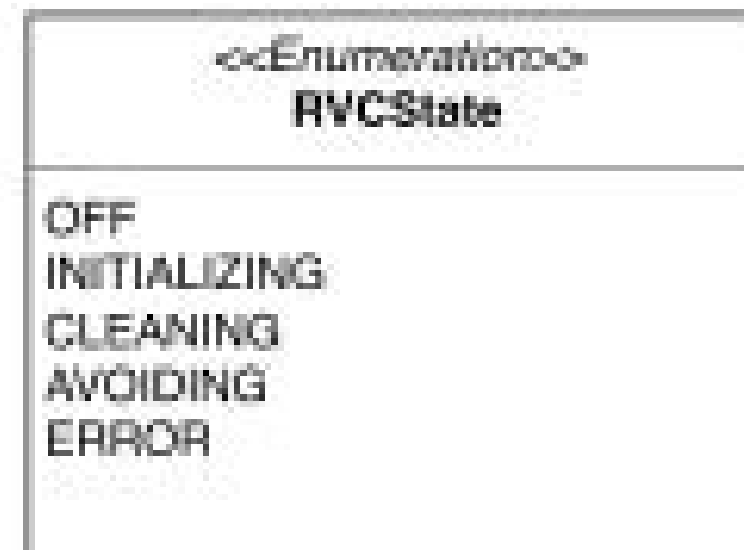
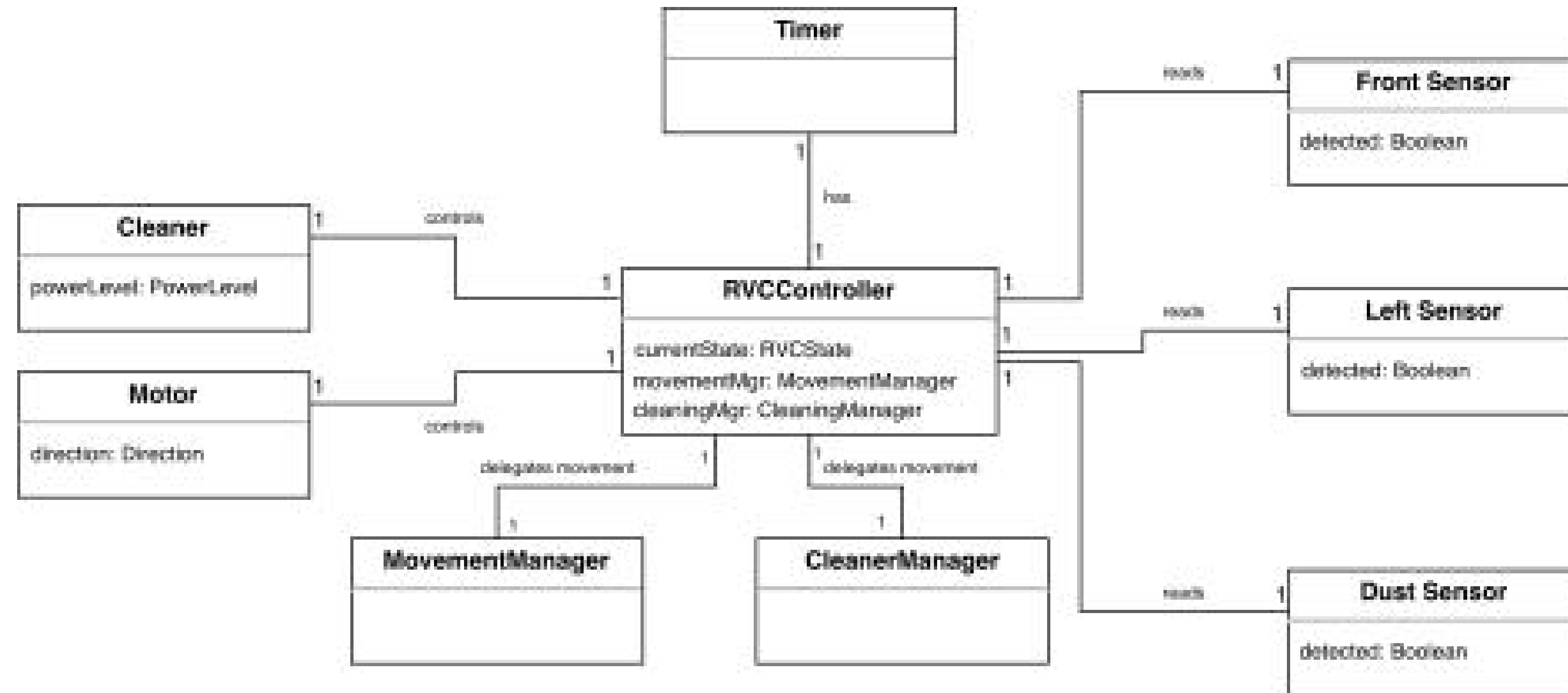
# Avoid Obstacles

[수정]



# Domain Model Diagram

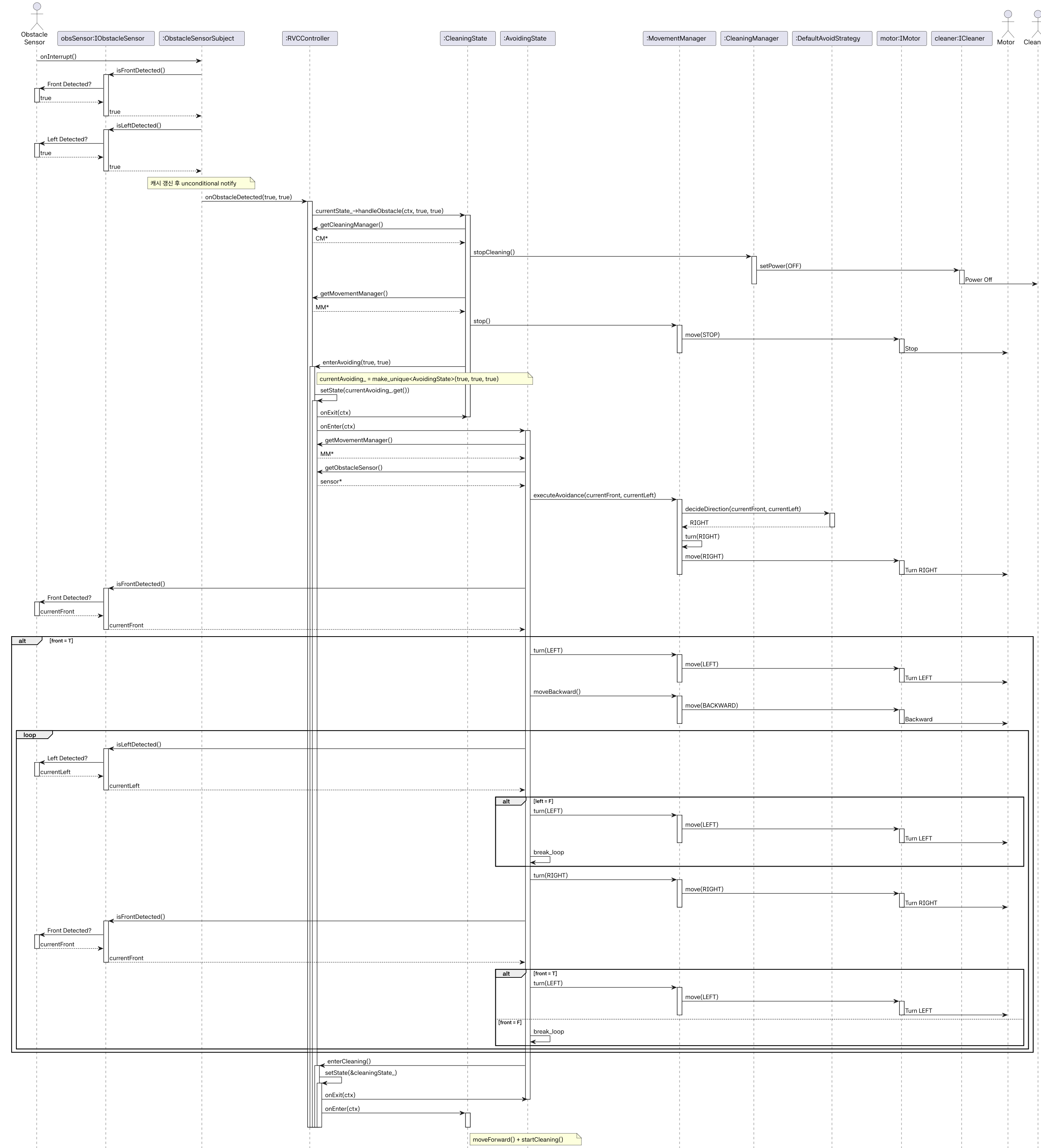
[수정]



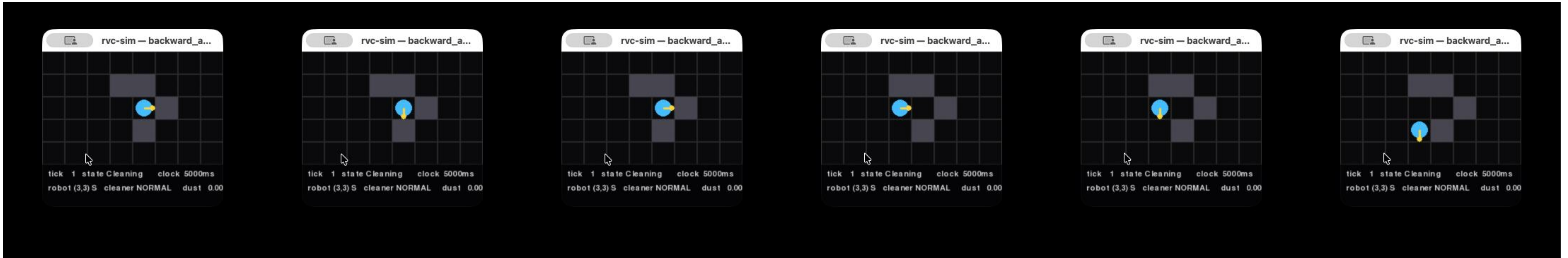


# SD-03

## [수정]



# System Test Result



# Vibe Coding

# Initial Prompts

**CRITICAL NOTE: THIS IS A REVISION PROJECT.** This project is **not** a greenfield (built from scratch) project. It is a revision and refactoring of an existing RVC (Robot Vacuum Cleaner) Controller codebase.

The primary goal is to modify the existing architecture, code, and documentation to satisfy newly changed requirements while preserving the integrity of untouched features. The project emphasizes the active application of software engineering processes, impact analysis, and safe object-oriented refactoring methodologies.

The controller focuses only on automatic cleaning behavior. Detailed hardware control implementation is outside the scope of this project unless explicitly required.

**Hardware Constraint Update:** The RVC hardware configuration has been modified. **The right sensor has been removed**, leaving only the front and left sensors. The existing controller logic and design must be revised to compensate for the lack of right-side environmental detection by utilizing new rotation and movement strategies.

---

## Development Process

---

Because this is a revision project, the focus must be on understanding the legacy system, identifying the impact of new changes, and updating existing artifacts.

The project must proceed through the following stages:

1. Requirements (Revision)
2. Analysis (Revision)
3. Design (Refactoring & Update)
4. Implementation (Modification & Extension)
5. Verification (Regression & New Testing)

Each stage has a corresponding skill file. Refer to the appropriate skill file before performing work related to that stage.

Do not skip stages. Do not modify the existing codebase before the Design stage is approved.

The user determines whether the current stage is complete. Do not proceed to the next stage unless the user explicitly approves it.

# Initial Prompts

## Purpose

Use this skill when analyzing, refining, and updating project requirements from the user's perspective.

As this is a **Revision Project**, you are NOT writing requirements from scratch. Your primary objective is to **reuse the existing requirements artifacts (Use Cases, SRS) produced in the previous project** and apply necessary modifications to reflect the new hardware constraint (the removal of the right sensor).

You must preserve existing requirements that are unaffected by this change.

## Preliminary Requirements (Original Baseline)

The legacy RVC system was originally built upon the following preliminary requirements. **Read these carefully to understand the baseline, and identify which legacy requirements must be modified due to the missing right sensor:**

- An RVC automatically cleans household surface.
- It goes straight forward while cleaning.
- If its sensors found an obstacle, it stops cleaning, turns aside left or right, and goes forward with cleaning.
- If there are obstacles in both front, left and right, it move backward and turn aside left or right, and goes forward.
- If it detects dust, power up the cleaning for a while.
- We do not consider the detail design and implementation on HW controls.
- We only focus on the automatic cleaning function.

## Responsibilities

During this stage:

- Review the existing requirements (Use Cases, SRS) from the previous project.
- **Identify and modify** only the specific Use Cases and requirements affected by the removal of the right sensor.
- **Actively detect ambiguity or inconsistency** in modifying the legacy behavior (e.g., How should the RVC handle the legacy "turn right" or "detect right obstacle" logic without a right sensor?) and ask the user for clarification.
- Reuse and preserve existing requirements that are unaffected by the hardware change.
- Update the Use Case Descriptions and Use Case Diagram to reflect the changes.
- Produce the updated Software Requirements Specification (SRS) as a revision of the legacy SRS.
- Update requirement traceability.

Implementation or structural design must not begin during this stage.

## Purpose

Use this skill when analyzing the problem domain and system interactions based on the approved Requirements (Use Cases and SRS).

**CRITICAL FOCUS: Establishing the System Boundary** A core objective of this stage is to precisely recognize and maintain the boundary of the system under development. For this revision project, **the system boundary defined in the legacy project must be reused exactly as-is without any modifications**. Your analysis must focus on integrating the altered hardware logic (the missing right sensor) strictly within this established boundary, ensuring that the system's scope of responsibility remains identical to the previous project. You must reuse the existing Analysis artifacts (Domain Model, System Sequence Diagrams) and adapt them to the new requirements while adhering to this fixed boundary.

## Input Prerequisites

Before beginning this stage, ensure you have reviewed the approved outputs from the Requirements stage:

- Revised Software Requirements Specification (SRS)
- Updated Use Case Descriptions
- Updated Use Case Diagram

## Responsibilities

During this stage:

- **Enforce Legacy System Boundary:** Strictly adhere to the system boundary established in the previous project. Do not expand, shrink, or alter the system's core responsibilities or its interfaces with external actors.
- **Review Legacy Artifacts:** Analyze the legacy Domain Model and System Sequence Diagrams (SSDs) to understand how the system boundary was previously modeled.
- **Conduct Impact Analysis:** Identify exactly which domain concepts (e.g., RightSensor entity) and system events (e.g., `detectRightObstacle()`) inside or crossing the system boundary are invalidated by the new requirements.
- **Update Domain Model:** Modify the conceptual model to remove the right sensor and introduce workaround concepts (e.g., rotation maneuvers), keeping all changes within the existing system boundary.
- **Update System Sequence Diagrams (SSD):** Redraw the SSDs for the modified Use Cases. Accurately capture the system events crossing the inherited system boundary between the Actor (Environment/Hardware) and the System (acting as a Black Box).
- **Maintain Traceability:** Ensure all conceptual updates directly trace back to the revised requirement IDs.

Implementation or structural software design (Class Diagrams, Object-oriented interfaces) must not begin during this stage.

# Inception Phase

> 이번 과제는 이전에 개발한 rvc(robot vacuum cleaner) controller의 요구사항 변경을 반영하는 거야. 지금 폴더도 이전 플젝 폴더이고, 다른건 다 동일한데 AGENT.md와 skill문서들만 새로 작성하였고, 이전 플젝에서 쓰인 agent.md와 skill은 이름 뒤에 \_old를 붙여 구분해두었어. 이제 새로운 AGENT.md와 skills/requirements/SKILL.md 를 읽고 작업을 시작하자.

- 새 요구사항 기준으로 작업하기 위해 먼저 저장소 지침과 requirements skill을 확인하겠습니다. 이후 기존 구조와 변경 범위를 파악한 뒤 필요한 수정까지 진행하겠습니다.
- 작업 디렉터리는 Git 저장소로 초기화되어 있지 않아 git status는 사용할 수 없습니다. 지금은 새 AGENT.md와 skills/Requirements/SKILL.md를 우선 읽고, 이어서 연관 skill 지침과 코드 구조를 확인하겠습니다.
- 현재 단계는 Requirements (Revision)입니다. 이 단계에서는 코드나 설계 파일을 수정하지 않고, 기존 요구사항 산출물을 재사용해 오른쪽 센서 제거 영향만 식별하겠습니다.
- 기존 요구사항은 오른쪽 센서가 존재한다는 전제에 강하게 묶여 있습니다. 특히 RVC-FR-007, 009, 010, 011, 013, 014와 UC-003, UC-004가 직접 영향 범위입니다.
- 요구사항 단계 산출물은 오른쪽 센서 제거가 확정된 부분과 아직 사용자 결정이 필요한 부분을 분리해 갱신하겠습니다. 변경 문서는 요구사항/유스케이스/유스케이스 다이어그램에 한정하고, 분석·설계·구현 산출물은 승인 전에는 건드리지 않겠습니다.

- 요구사항 산출물 3개를 갱신했습니다. 핵심은 "확정 변경"과 "사용자 결정 필요"를 분리해, 오른쪽 센서 의존 요구사항은 수정/폐기 표시하고 아직 정할 수 없는 회피 정책은 명시적으로 Open 상태로 둔 것입니다.

## • 현재 단계 : Requirements (Revision)

새 AGENT.md와 skills/Requirements/SKILL.md 기준으로 요구사항 단계만 진행했습니다. 코드/설계/분석 산출물은 아직 수정하지 않았습니다.

변경한 문서 :

- docs/requirements/srs.md
- docs/requirements/usecase-description.md
- docs/requirements/usecase-diagram.md

반영 내용 :

- 기존 오른쪽 센서 전제를 left sensor only 구조로 수정했습니다.
- 오른쪽 센서 직접 감지 요구사항은 [Modified] 또는 [Deprecated]로 표시했습니다.
- 먼지 감지, 청소 시작/종료, 기본 전진, 흡입 제어 요구사항은 영향 없음으로 보존했습니다.
- 유스케이스 다이어그램은 요구사항 skill 지시에 맞춰 PlantUML로 갱신했습니다.
- 요구사항 영향 분석과 Clarification Q&A를 SRS에 추가했습니다.

확인 필요한 요구사항 질문 :

1. 전방+좌측 장애물이 감지될 때, 오른쪽 센서 없이 바로 우회전해도 되나요? 아니면 회전/이동으로 오른쪽 경로를 간접 확인해야 하나요?
2. 기존 "전방+좌측+우측 모두 막힘" 조건은 무엇으로 대체할까요? 예 : "전방+좌측 막힘 + 오른쪽 확인 maneuver 실패"
3. 후진 탈출 중 오른쪽 센서 없이 우회전 탈출은 어떤 조건에서 선택해야 하나요?
4. 구현 인터페이스에서 오른쪽 센서 개념을 완전히 제거해야 하나요, 아니면 simulator adapter 호환을 위해 내부적으로만 유지하고 controller 로직에서는 무시해도 되나요?

이 질문들이 확정되어야 Requirements 단계를 완료하고 Analysis 단계로 넘어갈 수 있습니다.

Prompt & Answers

# Inception Phase - Use Cases [수정]

## RVC Use Case Description

### 1. 현재 단계

현재 단계는 Requirements (Revision)이다. 본 문서는 `rvc-controller`의 자동 진공 청소 제어 로직 유스케이스 변경을 정의한다.

### 2. 액터

액터	설명
사용자	청소 시작 및 종료 버튼을 누르는 외부 액터
전방 센서 하드웨어	전방 장애물 interrupt 입력을 제공하는 외부 actor
측면 센서 하드웨어	[수정] 좌측 장애물 polling 입력을 제공하는 외부 actor이다. (수정 전: 좌측 및 우측 장애물 polling 입력을 제공하는 외부 actor)
먼지 센서 하드웨어	[수정] 먼지 감지 polling 입력을 제공하는 외부 actor (수정 전: 먼지 감지 입력을 제공하는 외부 actor)
Movement Motor	<code>rvc-controller</code> 의 이동 명령을 수신하는 외부 actuator
Cleaning Motor	<code>rvc-controller</code> 의 흡입 명령을 수신하는 외부 actuator

### 3. 유스케이스 목록

ID	유스케이스	주요 관련 요구사항
UC-001	자동 진공 청소 시작	RVC-FR-001, RVC-FR-021, RVC-FR-022, RVC-FR-030, RVC-FR-031, RVC-FR-032
UC-002	자동 진공 청소 종료	RVC-FR-002, RVC-FR-020, RVC-FR-030, RVC-FR-031, RVC-FR-033
UC-003	전방 장애물 회피	RVC-FR-004 ~ RVC-FR-010, RVC-FR-015, RVC-FR-016, RVC-FR-030, [추가] RVC-FR-034 ~ RVC-FR-038
UC-004	삼방향 장애물 처리	RVC-FR-011 ~ RVC-FR-016, RVC-FR-030, [추가] RVC-FR-036 ~ RVC-FR-038
UC-005	먼지 감지에 따른 흡입 강화	RVC-FR-017 ~ RVC-FR-029, RVC-FR-031, [추가] RVC-FR-039

Use Cases (Scenario 는 #5와 동일)

# Inception Phase - Use Cases [수정]

## 4. UC-001 자동 진공 청소 시작

항목	내용
주요 액터	사용자
목적	사용자가 버튼 입력으로 RVC의 자동 진공 청소를 시작한다.
사전조건	RVC가 OFF 상태이다.
사후조건	RVC는 전원 ON 상태가 되고 NORMAL 흡입 상태로 전진을 시작한다.
관련 요구사항	RVC-FR-001, RVC-FR-021, RVC-FR-022, RVC-FR-003, RVC-FR-030, RVC-FR-031, RVC-FR-032

### 기본 흐름

1. 사용자가 청소 시작 버튼을 입력한다.
2. RVC는 OFF 상태에서 전원 ON 상태로 전환한다.
3. [수정] RVC는 전방 센서 하드웨어, 좌측 센서 하드웨어, 먼지 센서 하드웨어와의 센서 인터페이스를 초기화한다. (수정 전: RVC는 전방 센서 하드웨어, 측면 센서 하드웨어, 먼지 센서 하드웨어와의 센서 인터페이스를 초기화한다.)
4. RVC는 cleaning motor에 NORMAL 흡입 명령을 제공한다.
5. RVC는 movement motor에 전진 명령을 제공한다.

### 예외 흐름

- 현재 요구사항에서는 별도 예외 흐름을 정의하지 않는다.

## 5. UC-002 자동 진공 청소 종료

항목	내용
주요 액터	사용자
목적	사용자가 버튼 입력으로 RVC의 자동 진공 청소를 종료한다.
사전조건	RVC가 전원 ON 상태이다.
사후조건	RVC는 OFF 상태가 되고 자동 청소 제어 로직과 진공 흡입 동작을 수행하지 않는다.
관련 요구사항	RVC-FR-002, RVC-FR-020, RVC-FR-030, RVC-FR-031, RVC-FR-033

### 기본 흐름

1. 사용자가 청소 종료 버튼을 입력한다.
2. [수정] RVC는 현재 RVC state에 정의된 종료 절차를 시작한다. (수정 전: RVC는 청소 동작을 종료한다.)
3. [수정] RVC는 현재 RVC state에 정의된 movement motor 종료 명령을 제공한다. (수정 전: RVC는 movement motor에 정지 명령을 제공한다.)
4. [수정] RVC는 현재 RVC state에 정의된 cleaning motor 종료 명령을 제공한다. (수정 전: RVC는 cleaning motor에 OFF 흡입 명령을 제공한다.)
5. [수정] RVC는 현재 RVC state에 정의된 종료 절차에 따라 전방 센서 하드웨어, 좌측 센서 하드웨어, 먼지 센서 하드웨어와의 센서 인터페이스를 종료한다. (수정 전: RVC는 전방 센서 하드웨어, 측면 센서 하드웨어, 먼지 센서 하드웨어와의 센서 인터페이스를 종료한다.)
6. RVC는 OFF 상태로 전환한다.

### 예외 흐름

- [수정] 청소 종료 버튼 입력이 발생하면 현재 RVC state에 정의된 종료 절차를 수행한 후 OFF로 전환한다. (수정 전: 청소 종료 버튼 입력이 회전, 후진, POWER\_UP 중 발생했을 때의 처리 우선순위는 아직 확정되지 않았다.)

# Inception Phase - Use Cases [수정]

6. UC-003 전방 장애물 회피	
항목	내용
주요 액터	전방 센서 하드웨어, 측면 센서 하드웨어, Movement Motor
목적	[수정] 전진 중 전방 장애물이 감지되었을 때 좌측 센서 입력과 90도 우회전 후 전방 센싱을 통해 회피 방향을 결정한다. [수정 전: 전진 중 전방 장애물이 감지되었을 때 좌우 장애물 상태에 따라 회피 방향을 결정한다.]
사전조건	RVC가 전진 중이다.
사후조건	RVC는 회전 완료 후 다시 전진한다.
관련 요구사항	RVC-FR-004 ~ RVC-FR-010, RVC-FR-015, RVC-FR-016, RVC-FR-030, [추가] RVC-FR-034 ~ RVC-FR-038
<b>기본 흐름</b>	
<ol style="list-style-type: none"> <li>1. 전방 센서 하드웨어가 전방 장애물 interrupt를 제공한다.</li> <li>2. RVC는 movement motor에 정지 명령을 제공한다.</li> <li>3. [추가] RVC는 cleaning motor에 OFF 출력을 제공한다.</li> <li>4. [수정] RVC는 측면 센서 하드웨어에서 좌측 장애물 감을 확인한다. (수정 전: RVC는 측면 센서 하드웨어에서 좌측 및 우측 장애물 감을 확인한다.)</li> <li>5. 전방 장애물만 감지된 경우 RVC는 movement motor에 좌회전 명령을 제공한다.</li> <li>6. [수정] 전방 및 좌측 장애물이 감지된 경우 RVC는 우측 경로 확인을 위해 90도 우회전한다. (수정 전: 전방 및 좌측 장애물이 감지된 경우 RVC는 movement motor에 우회전 명령을 제공한다.)</li> <li>7. [삭제] 전방 및 우측 장애물이 감지된 경우 RVC는 movement motor에 좌회전 명령을 제공한다.</li> <li>8. [추가] RVC는 90도 우회전 완료 후 전방 센서 하드웨어에서 장애물 감을 확인한다.</li> <li>9. [추가] 우회전 후 전방 장애물이 감지되지 않으면 RVC는 기존 전방 및 좌측 장애물 조건의 회피 결과로 보고 cleaning motor에 NORMAL 출력을 제공한 후 우회전한 방향으로 전진한다.</li> <li>10. [추가] 우회전 후 전방 장애물이 감지되면 RVC는 기존 전방, 좌측, 우측 장애물이 모두 존재하는 상황으로 보고 90도 좌회전하여 원래 방향으로 복귀한다.</li> <li>11. [추가] 원래 방향으로 복귀한 후 UC-004 심방향 장애물 처리로 이어진다.</li> <li>12. [수정] 회전 모드 진입 후 4초가 지나면 RVC는 회전 완료로 간주한다. (수정 전: 회전 모드 진입 후 10초가 지나면 RVC는 회전 완료로 간주한다.)</li> <li>13. [수정] RVC는 cleaning motor에 NORMAL 출력을 제공한 후 다시 전진한다. (수정 전: RVC는 다시 전진한다.)</li> </ol>	
<b>대안 흐름</b>	
<ul style="list-style-type: none"> <li>* [수정] 90도 우회전 완료 후 전방 장애물이 감지되어 90도 좌회전으로 원래 방향에 복귀하면 UC-004 심방향 장애물 처리로 이어진다. (수정 전: 전방, 좌측, 우측 장애물이 모두 감지되면 UC-004 심방향 장애물 탈출로 이어진다.)</li> </ul>	

# Inception Phase - Use Cases [수정]

## 7. UC-004 삼방향 장애물 처리

항목	내용
주요 액터	전방 센서 하드웨어, 측면 센서 하드웨어, Movement Motor
목적	[수정] 기존 전방, 좌측, 우측 장애물이 모두 존재하는 상황을 우측 센서 입력 없이 90도 우회전 후 전방 센싱과 1 tick 후진 반복으로 처리한다. (수정 전: 전방, 좌측, 우측 장애물이 모두 감지된 상황에서 후진 후 장애물이 해제된 방향으로 탈출한다.)
사전조건	[수정] 전방 및 좌측 장애물이 감지된 상태에서 90도 우회전 완료 후 전방 장애물이 감지되었고, RVC가 90도 좌회전으로 원래 방향에 복귀한 상태이다. controller는 직접적인 우측 센서 입력을 판단에 사용하지 않는다. (수정 전: 전방, 좌측, 우측 장애물이 모두 감지되었다.)
사후조건	RVC는 회전 완료 후 다시 전진한다.
관련 요구사항	RVC-FR-011 ~ RVC-FR-016, RVC-FR-030, [추가] RVC-FR-036 ~ RVC-FR-038

### 기본 흐름

- [수정] RVC는 90도 우회전 완료 후 전방 장애물이 감지되었고 원래 방향으로 복귀한 상태임을 확인한다. (수정 전: RVC는 전방, 좌측, 우측 장애물이 모두 감지된 것을 확인한다.)
- RVC는 movement motor에 정지 명령을 제공한다.
- [추가] RVC는 cleaning motor에 OFF 흡입 명령을 제공한다.
- [수정] RVC는 원래 방향 복귀 상태에서 3방향 장애물 처리 루프를 시작한다. (수정 전: RVC는 movement motor에 후진 명령을 제공한다.)
- [수정] RVC는 원래 방향에서 movement motor에 1 tick 후진 명령을 제공한다. (수정 전: RVC는 후진 중 측면 센서 하드웨어에서 좌측 및 우측 장애물 값을 확인한다.)
- [수정] RVC는 1 tick 후진 후 측면 센서 하드웨어에서 좌측 장애물 값을 확인한다. (수정 전: 좌측 장애물이 먼저 감지되지 않게 되면 RVC는 movement motor에 좌회전 명령을 제공한다.)
- [수정] 좌측 장애물이 감지되지 않으면 RVC는 90도 좌회전을 완료한 후 cleaning motor에 NORMAL 흡입 명령을 제공하고 전진한다. (수정 전: 우측 장애물이 먼저 감지되지 않게 되면 RVC는 movement motor에 우회전 명령을 제공한다.)
- [수정] 좌측 장애물이 계속 감지되면 RVC는 우측 경로 확인을 위해 90도 우회전을 완료한 후 전방 센서 하드웨어에서 장애물 값을 확인한다. 우측 경로 확인 전 좌측 센서 확인은 1 tick마다 수행한다. (수정 전: 좌측 및 우측 장애물이 동시에 감지되지 않게 되면 RVC는 movement motor에 좌회전 명령을 제공한다.)
- [추가] 우회전 후 전방 장애물이 감지되지 않으면 RVC는 cleaning motor에 NORMAL 흡입 명령을 제공하고 우회전한 방향으로 전진한다.
- [추가] 우회전 후 전방 장애물이 감지되면 RVC는 90도 좌회전하여 원래 방향으로 복귀한 뒤 1 tick 후진, 좌측 장애물 확인, 필요한 경우 우측 경로 확인을 반복한다.
- [수정] 회전 모드 진입 후 4초가 지나면 RVC는 회전 완료로 간주한다. (수정 전: 회전 모드 진입 후 10초가 지나면 RVC는 회전 완료로 간주한다.)
- [수정] RVC는 탈출 방향이 결정되면 cleaning motor에 NORMAL 흡입 명령을 제공한 후 다시 전진한다. (수정 전: RVC는 다시 전진한다.)

### 예외 흐름

- 후방 장애물은 현재 요구사항 범위에서 감지하지 않는다.
- 후진 최대 허용 시간은 현재 요구사항 범위에서 정의하지 않는다.
- [추가] simulator가 제공하는 우측 장애물 정보는 controller 판단 입력으로 사용하지 않으며, 검증 결과 분석을 위한 관찰 데이터로만 사용한다.

## 8. UC-005 먼지 감지에 따른 흡입 강화

항목	내용
주요 액터	먼지 센서 하드웨어, Cleaning Motor
목적	먼지가 감지되면 진공 흡입 상태를 POWER_UP으로 전환하고 3초 단위로 유지 여부를 판단한다.
사전조건	RVC가 전환 ON 상태이다.
사후조건	전진 중 먼지가 계속 감지되면 POWER_UP이 유지되고, 먼지가 감지되지 않으면 NORMAL로 복귀한다.
관련 요구사항	RVC-FR-017 ~ RVC-FR-029, RVC-FR-031, [추가] RVC-FR-039

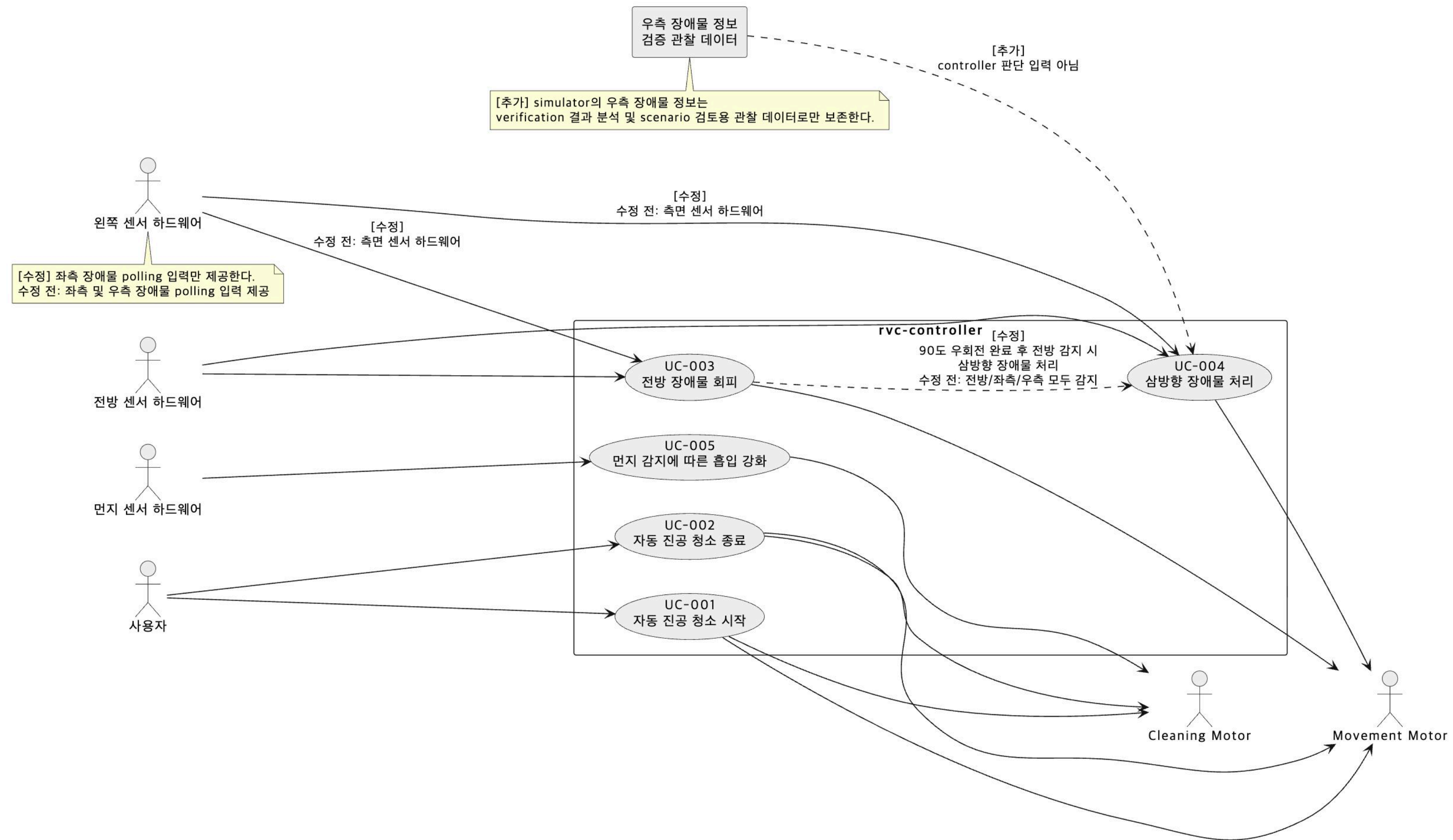
### 기본 흐름

- [수정] RVC는 먼지 센서 하드웨어에서 먼지 감지 값을 1 tick마다 polling한다. (수정 전: 먼지 센서 하드웨어가 먼지 감지 입력을 제공한다.)
- RVC는 cleaning motor에 POWER\_UP 흡입 명령을 제공한다.
- RVC는 POWER\_UP 상태를 3초 동안 유지한다.
- 3초 경과 후 RVC는 먼지 감지 값을 다시 확인한다.
- 먼지가 다시 감지되면 RVC는 POWER\_UP 상태를 유지하고 3초 타이머를 다시 시작한다.
- 먼지가 감지되지 않으면 RVC는 cleaning motor에 NORMAL 흡입 명령을 제공한다.

### 대안 흐름

- [수정] RVC가 정지, 회전, 후진 중이다. (수정 전: 정지, 회전, 후진 중 먼지가 감지된다.)
- [수정] RVC는 먼지 감지 값으로 즉시 POWER\_UP에 진입하지 않고 cleaning motor OFF 상태를 유지한다. (수정 전: RVC는 즉시 POWER\_UP으로 전환하지 않고 NORMAL 상태를 유지한다.)
- [수정] RVC는 정지, 회전, 후진 중 먼지 감지 값을 이후 POWER\_UP 진입 조건으로 저장하지 않는다. 직전 후행을 시작하는 tick의 dust polling 결과로 흡입 상태를 판단한다. (수정 전: 이후 전진을 시작할 때 POWER\_UP으로 진입한다.)

# Inception Phase - Use Cases [수정]



Use Cases (Scenario 는 #5와 동일)

# Inception Phase - SRS [수정]

## RVC Software Requirements Specification

### 1. 현재 단계

현재 단계는 Requirements (Revision)이다. 이 문서는 RVC(Robot Vacuum Cleaner)의 rvc-controller에 대한 자동 진공 청소 제어 로직 요구사항 변경을 정리한다.

### 2. 목적

본 프로젝트의 목적은 하드웨어 세부 구현에 의존하지 않는 rvc-controller의 자동 진공 청소 제어 로직 요구사항을 정의하는 것이다.

### 3. 범위

#### 3.1 시스템 바운더리

본 프로젝트에서 개발할 시스템의 바운더리는 rvc-controller이다. 실제 센서 하드웨어와 실제 actuator는 rvc-controller 외부 actor로 정의한다.

rvc-controller는 다음 외부 actor와 인터페이스를 가진다.

외부 actor	rvc-controller 관점의 역할
사용자	청소 시작 및 종료 버튼 입력을 제공한다.
전방 센서 하드웨어	전방 장애물 감지 interrupt 입력을 제공한다.
측면 센서 하드웨어	[수정] 좌측 장애물 감지 polling 입력을 제공한다. (수정 전: 좌측 및 우측 장애물 감지 polling 입력을 제공한다.)
민지 센서 하드웨어	[수정] 민지 감지 polling 입력을 제공한다. (수정 전: 민지 감지 입력을 제공한다.)
Movement Motor	전진, 정지, 후진, 좌회전, 우회전 명령을 수신하는 외부 actuator이다.
Cleaning Motor	OFF, NORMAL, POWER_UP 흡입 명령을 수신하는 외부 actuator이다.

#### 3.2 포함 범위

- 사용자 버튼 입력에 의한 청소 시작 및 종료
- 자동 진공 청소 중 기본 전진 동작
- [수정] 전방, 좌측 장애물 감지에 따른 회피 동작 (수정 전: 전방, 좌측, 우측 장애물 감지에 따른 회피 동작)
- [추가] 오른쪽 센서 제거로 인해 우측 장애물을 직접 감지할 수 없는 상황을 보충하기 위한 회진/이동 전략
- 민지 감지에 따른 진공 흡입 상태 제어
- [수정] 전방/좌측 센서, movement motor, cleaning motor와의 추상 인터페이스 요구사항 (수정 전: 전방/측면 센서, movement motor, cleaning motor와의 추상 인터페이스 요구사항)
- 하드웨어 독립적인 제어 로직 요구사항

#### 3.3 제외 범위

- 실제 센서 하드웨어 내부 동작
- 실제 actuator 내부 동작
- 실제 모터 구동 회로 및 물리 제어 세부사항
- 지도 작성, 경로 최적화, 공간 커버리지 알고리즘
- 배터리 관리, 충전, 도킹
- 앱 연동 및 사용자 인터페이스
- 후방 장애물 감지
- 후진 최대 허용 시간 처리
- 360도 회전 탈출 기능
- [추가] 오른쪽 센서 제거 문제를 해결하기 위한 simulator redesign 또는 simulator 구현 변경

# Inception Phase - SRS [수정]

## 4. 정의

용어	의미
RVC	Robot Vacuum Cleaner
rvc-controller	본 프로젝트에서 개발할 제어 시스템
외부 actor	rvc-controller 바운더리 밖에 존재하며 입력을 제공하거나 명령을 수신하는 외부 요소
actuator	rvc-controller의 명령을 받아 물리 동작을 수행하는 외부 장치
OFF	전체 시스템 전원이 꺼진 상태
NORMAL	전원 ON 상태에서의 기본 진공 흡입 상태
POWER_UP	먼지 감지 시 3초 동안 유지되는 감속 진공 흡입 상태
전방 장애물	전방 센서로 감지되는 장애물
좌측 장애물	좌측 센서로 감지되는 장애물
우측 장애물	[수정] 오른쪽에 존재하는 장애물 (수정 전 : 우측 센서로 감지되는 장애물)
[추가] 새로운 회피 전략	기존 우측 센서 감지를 90도 우회전 후 전방 센서로 대체하여 우측 방향의 장애물 존재 여부를 판단하는 회피 절차
[수정] 우측 경로 확인	RVC가 90도 우회전한 뒤 전방 센서로 기존 우측 방향의 장애물 존재 여부를 확인하는 행위 (수정 전 : 우측 센서 없이 회전(회전) 전략을 통해 우측 방향 이동 가능성을 간접적으로 판단하는 행위)
[추가] tick	RVC controller의 1회 주기 처리 단위로, 센서 polling과 상태 및 청소 로직 경선의 한 번 수행되는 control loop

## 5. 가정

- 청소 시작 및 종료는 사용자 버튼 입력으로 발생한다.
- rvc-controller는 실제 센서 하드웨어와 actuator 자체를 구현하지 않는다.
- 전방 센서 하드웨어는 interrupt 방식으로 입력을 제공한다.
- [수정] 측면 센서 하드웨어는 좌측 장애물 값을 polling 방식으로 제공한다. (수정 전 : 측면 센서 하드웨어는 좌측 및 우측 장애물 값을 polling 방식으로 제공한다.)
- [수정] 먼지 센서 하드웨어는 먼지 감지 값을 polling 방식으로 제공한다. (수정 전 : 먼지 센서 하드웨어는 먼지 감지 입력을 제공한다.)
- 후진 중 후방 장애물은 발생하지 않는다고 가정한다.
- [수정] 회전 모드 진입 후 4초가 지나면 회전이 성공적으로 완료된 것으로 간주한다. (수정 전 : 회전 모드 진입 후 10초가 지나면 회전이 성공적으로 완료된 것으로 간주한다.)
- [수정] 후진 중 우측 경로 확인 전에 수행하는 좌측 센서 polling은 1 tick마다 수행한다. (수정 전 : 후진 중 좌우 센서 polling 주기는 추후 정의한다.)
- 좌회전은 기본 우선 회전 방향이다.
- [추가] 기존 simulator와 verification infrastructure는 외부 검증 인프라로 재사용하며, 오른쪽 센서 제거 문제 해결을 위해 simulator 자체를 수정하지 않는다.

SRS

# Elaboration - OOA [수정]

현재 단계: **\*\*Analysis (Revision)\*\*** 로 전환했습니다.

요청대로 Requirements 문서도 정리했습니다.

- `RVC-TBD-002`는 `RVC-DEC-007`로 옮겨 “현재 요구사항 범위에서 고려하지 않는다”로 확정했습니다.
- `RVC-TBD-008`은 `RVC-DEC-008`로 옮겨 “Analysis 단계 진입 전 점검 완료”로 변경했습니다.
- 미정 사항은 “없음”으로 정리했습니다.
- `git diff --check` 통과했습니다.

읽은 Analysis skill 경로:

```
```text
/Users/whdydrnr/Documents/GitHub/rvc-revision-
test/skills/analysis/SKILL.md
```

Analysis 단계의 핵심 산출물은 다음입니다.

- Impact Analysis & Boundary Confirmation
- Revised Domain Model
- Revised System Sequence Diagrams, SSDs

다음 작업에서는 legacy system boundary를 유지하면서 `docs/requirements/domain-model.md`와 `docs/requirements/system-sequence-diagram.md` 를 새 요구사항에 맞게 갱신하면 됩니다.

반영 내용:

- legacy system boundary가 `rvc-controller`로 유지됨을 `analysis-impact.md`에 명시했습니다.
- 우측 센서 직접 입력을 제거하고, `90도 우회전 완료 후 전방 센싱`을 `RightPathCheck` 분석 개념으로 추가했습니다.
- domain model을 Mermaid에서 PlantUML로 갱신했습니다.
- SSD도 PlantUML로 전환하고, `UC-003`, `UC-004` 흐름을 새 요구사항에 맞게 수정했습니다.
- simulator 우측 장애물 정보는 controller 입력 actor가 아니라 검증 관찰 데이터로만 설명했습니다.
- 코드, simulator, 구현 구조는 변경하지 않았습니다.



# Elaboration - OOD [수정]

현재 단계: **\*\*Design (Refactoring & Update)\*\***

`docs/design/design-specification.md`를 새 요구사항 기준으로 갱신했습니다.

변경 파일:

[docs/design/design-specification.md] (/Users/whdydrnr/Documents/GitHub/rvc-revision-test/docs/design/design-specification.md)

반영한 핵심 내용:

- 내부 설계에서 우측 센서 입력 제거
- `RightPathCheckState`, `ReturnFromRightCheckState`, revised `BackwardState` 설계 추가
- 90도 우회전 후 전방 센싱으로 우측 경로 확인
- 3방향 장애물 처리 루프: 원래 방향 복귀 후 1 tick 후진, 좌측 polling, 필요 시 우측 경로 확인 반복
- 회전 완료 시간 4초
- movement가 멈추거나 회전/후진 중이면 cleaning OFF
- 직진 시작 시 cleaning 최소 NORMAL
- dust sensor는 1 tick polling, pending POWER\_UP 제거
- PlantUML class/state/sequence 설계 추가
- 다음 Implementation 단계에서 수정할 대상과 순서 정리

검증:

- `git diff --check` 통과
- `plantuml` 명령은 로컬에 없어 렌더링 검증은 하지 못했습니다.

Design 산출물을 검토하고 승인해주면 다음은 **\*\*Implementation (Modification & Extension)\*\*** 단계로 넘어가겠습니다.

# Elaboration - OOD [수정]

나눈 이유는 **\*\*우회전 후 센싱\*\***과 **\*\*막혔을 때 원래 방향 복귀\*\***가 서로 다른 목적의 상태라서야.

``RightPathCheckState``의 책임:

- 90도 우회전
- 회전 완료 대기
- 전방 센서 polling
- 전방이 비었으면 그 방향으로 전진
- 전방이 막혔으면 복귀 상태로 넘김

``ReturnFromRightCheckState``의 책임:

- 우측 경로 확인 실패 후 90도 좌회전
- 회전 완료 대기
- “이미 원래 방향으로 복귀한 상태”를 만든 뒤 ``BackwardState``로 넘김

이걸 하나로 합치면 한 상태가 “우회전 중인지, 센싱 직후인지, 좌회전 복귀 중인지”를 내부 플래그로 구분해야 해서 상태 내부가 복잡해져. 지금 요구사항에서 특히 중요한 부분이 “우측 확인 실패 후 원래 방향 복귀 → 1 tick 후진”이기 때문에, 복귀 완료 시점을 별도 상태로 드러내는 쪽이 추적성과 테스트가 좋다고 봤어.

다만 구현을 더 단순하게 하고 싶으면 ``RightPathCheckState`` 하나에 내부 phase를 두는 방식도 가능해. 설계 명확성은 현재 방식이 낫고, 클래스 수 최소화는 단일 상태 방식이 낫다.

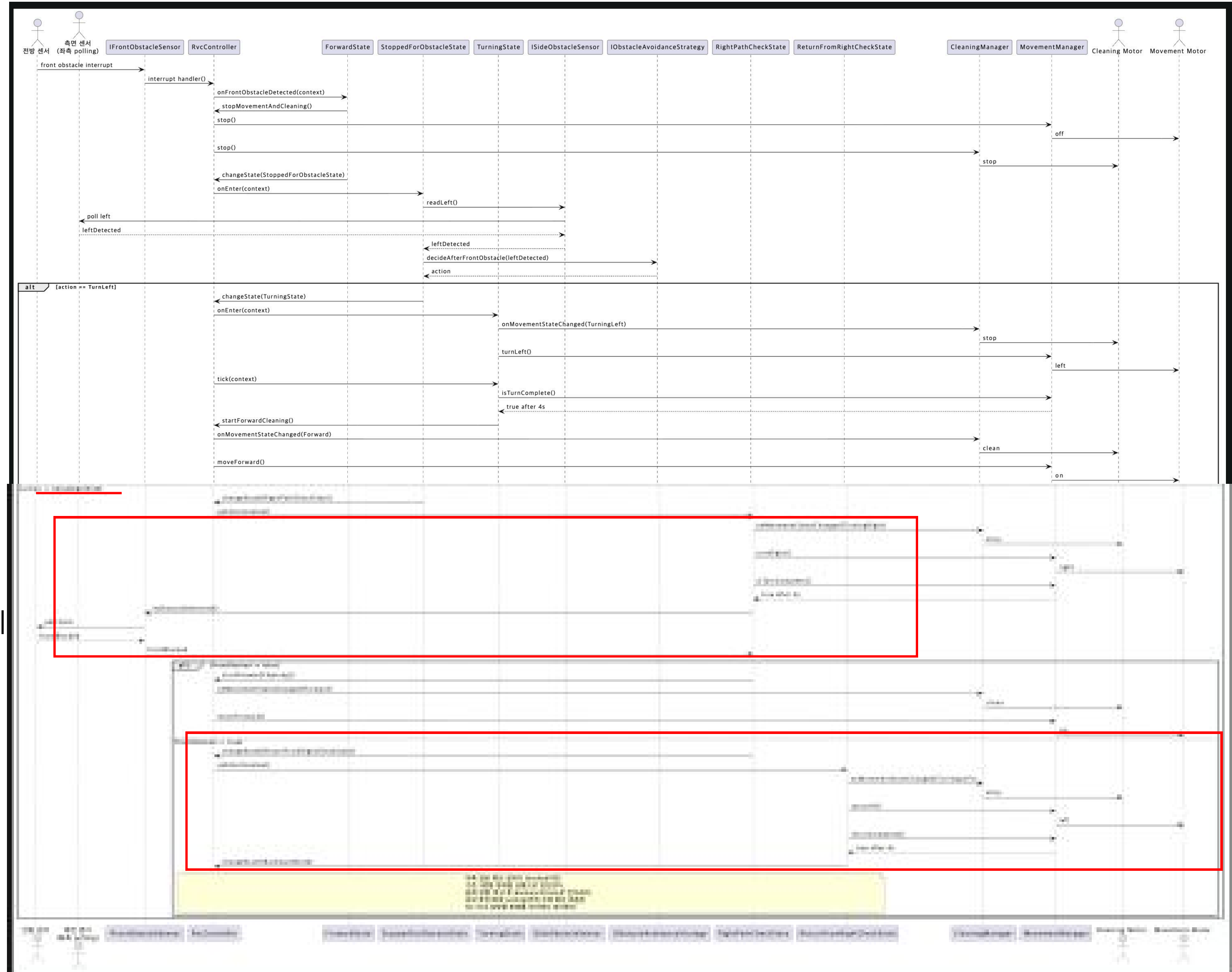
기존 AvoidingState 내에서 장애물 회피가 완료되던 구조 → 세분화된 State들로 분리

# Elaboration - OOD [수정]

## SD-003 : 전방 장애물 회피

우회전 후 전방 센서값 읽어오기

우측 확인 결과 blocked이면  
3방향 장애물 상황으로 판단.  
원래 방향 복귀 후 BackwardState로 전이,  
상세 후진/우측 확인 흐름은  
SD-004에서 정의한다.

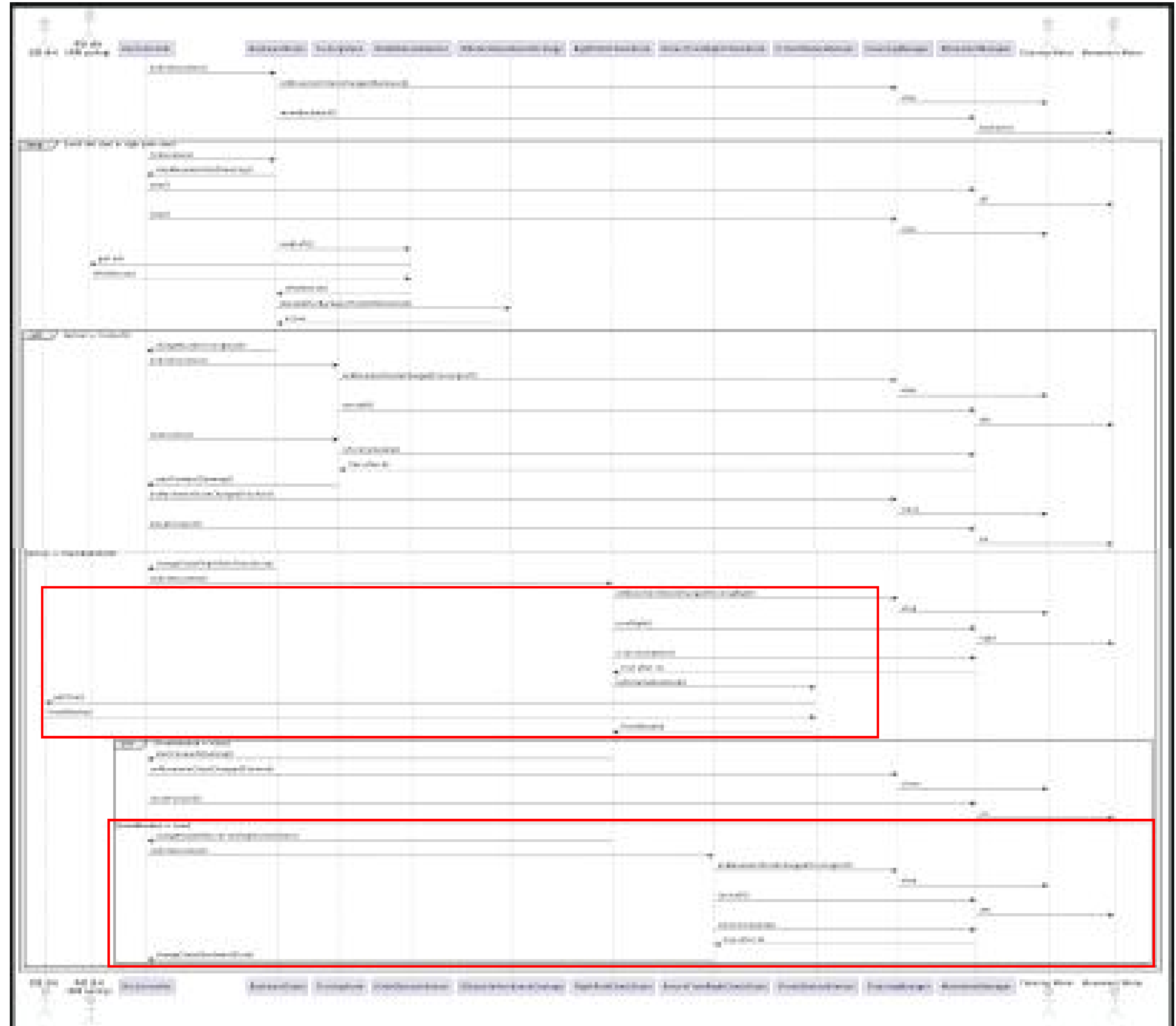


# Elaboration - OOD [수정]

## SD-004 : 3방향 장애물 회피

우회전 후 전방 센서값 읽어오기

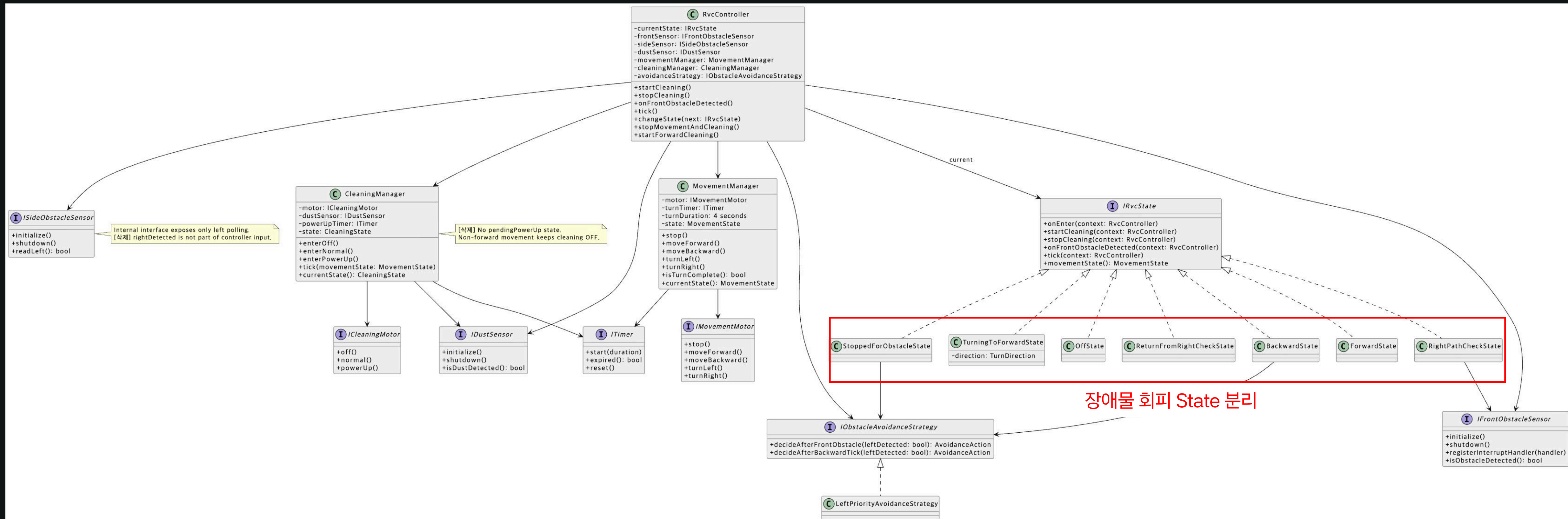
오른쪽이 막혀있을 시  
좌회전(재정렬) 후 후진 시작.  
후진 중 1틱마다 left polling, right check



# Elaboration - OOD [수정]

## 7. Revised Class Diagram [수정]

(수정 전: Mermaid class diagram으로 StoppedForObstacleState 와 BackwardState 가 side snapshot 기반으로 전략을 호출하고, RightPathCheckState / ReturnFromRightCheckState 는 존재하지 않았다.)



# Elaboration - OOD [수정]

## RVC Controller Design Specification

본 문서는 docs/design-specification\_before\_revision.md를 기준으로 오른쪽 센서 삭제 revision에 맞게 수정한 설계 산출물이다. 기존 설계 대비 변경된 부분은 [수정], [추가], [삭제] 표시와 (수정 전: ...) 형식으로 보존한다.

### 1. 현재 단계 [수정]

현재 단계는 Design (Refactoring & Update)이다. 본 문서는 승인된 Requirements 및 Analysis 산출물을 바탕으로 legacy `rvc-controller` 설계를 수정한다. (수정 전: 현재 단계는 Design이며, 요구사항 분석 산출물을 바탕으로 `rvc-controller`의 객체지향 설계를 정의한다.)

본 단계에서는 코드 구현을 수행하지 않는다. 구현은 본 설계 산출물 승인 이후 Implementation 단계에서 진행한다.

### 2. 설계 범위와 바운더리 [수정]

설계 대상은 `rvc-controller` 내부 구조이다. 사용자, 전방 센서 하드웨어, 측면 센서 하드웨어, 먼지 센서 하드웨어, Movement Motor, Cleaning Motor는 기존과 동일하게 외부 actor로 유지한다. (수정 전: 설계 대상은 `rvc-controller` 내부 소프트웨어 구조이며, 실제 센서 하드웨어와 motor는 외부 actor로 두고 추상 인터페이스를 통해 상호작용한다.)

항목	설계 결정
시스템 바운더리	legacy와 동일하게 <code>rvc-controller</code> 내부만 설계 대상으로 둔다.
simulator	[추가] 기존 simulator 및 검증 인프라는 수정하지 않는다. simulator의 우측 장애물 정보는 검증 관찰 데이터로만 사용할 수 있다.
우측 센서	[추가] controller 내부 판단 입력과 내부 인터페이스에서 제거한다.
우측 경로 확인	[추가] 90도 우회전 완료 후 전방 센싱으로 대체한다.
청소 흡입	[수정] movement가 정지/회전/후진 상태이면 cleaning motor는 OFF이다. 직진 시작 시 최소 상태는 NORMAL이다. (수정 전: 이동 상태와 cleaning motor 제어는 분리하되 함께 조정한다.)

### 3. Legacy 설계 유지 항목 [수정]

[수정 전: 설계 목표 섹션에서 센서 입력/제어 판단/jactuator 명령 분리, 상태 머신 표현, manager 분리, 인터페이스 정리, 요구사항 추적성 유지를 목표로 정리했다.]

Legacy 요소	유지 여부	이유
RvcController	유지	사용자 입력, 센서 입력, 상태 전이, manager 조정을 담당하는 최상위 조정 로직 역할이 중요하다.
State 머신	유지	이동 상태별 이벤트 처리와 전이를 분리하는 구조가 변경 요구사항에도 적합하다.
MovementManager	유지	movement motor 명령과 회전 완료 시간 관리를 분리하는 책임이 중요하다.
CleaningManager	[수정] 유지하되 수정	cleaning motor 상태와 POWER_UP 타이머 관리는 유지하되, 비전진 중 NORMAL/pending 동작은 제거한다. (수정 전: 3초 POWER_UP 유지, 재감지 시 타이머 재시작, 횡파 중 먼지 감지 지연 처리 담당)
Sensor/Motor 인터페이스	[수정] 유지하되 수정	외부 하드웨어 추상화 경계는 유지하되, 우측 센서 입력은 내부 인터페이스에서 제거한다. (수정 전: 좌측/우측 장애물 polling 입력 제공)
ITimer	[수정] 유지	4초 회전 완료와 3초 POWER_UP 유지 시간을 테스트 가능하게 한다. (수정 전: 회전 10초, POWER_UP 3초 규칙을 테스트 가능하게 함)

SDD

# Elaboration - OOD [수정]

## 4. Design Modification Rationale [수정]

(수정 전: 주요 설계 결정은 RvcController, manager 분리, Strategy 패턴, CleaningManager, Timer, State 패턴, MovementManager 채택을 정리했다.)

ID	수정 대상	수정 이유	설계 방향
DD-001	[수정] 측면 센서 인터페이스	실제 우측 센서가 제거되어 rightDetected를 controller 판단 입력으로 사용할 수 없다.	ISideObstacleSensor는 좌측 polling만 제공한다. (수정 전: 좌우 장애물 감 polling)
DD-002	[추가] 장애물 회피 상태 흐름	legacy의 front+left+right 직접 감지가 불가능하다.	RightPathCheckState를 추가하여 우회전 후 전방 센서로 우측 경로를 확인한다.
DD-003	[수정] 삼방향 장애물 처리	삼방향 조건은 우측 센서가 아니라 우측 경로 확인 실패로 판단되어야 한다.	우측 경로가 막히면 반대 방향 회전 후 BackwardState 후퇴에 진입한다. (수정 전: 직접 좌/우 센서 snapshot과 전각으로 회전/후진 결정)
DD-004	[수정] 후진 루프	180도 회전 대체가 불가능하며 1 tick 후진이 필요하다.	BackwardState는 1 tick 후진 후 마비 좌측 polling을 수행한다. (수정 전: 후진 중 좌우 snapshot으로 해제 방향 선택)
DD-005	[수정] 청소 중 입 상태	움직이지 않을 때 cleaning이 꺼져 있으면 안 된다.	movement stop/turn/backward 시 cleaning OFF, forward 시작 시 NORMAL을 보장한다. (수정 전: 회피 중 먼지 감지 지연 처리 지연)
DD-006	[수정] 먼지 감지	먼지 센서는 이벤트가 아니라 1 tick polling 입력이다.	CleaningManager::tick(movementState)에서 전진 중 dust polling을 수행한다. (수정 전: 먼지 감지 값을 제공하고 회피 중 지연 처리 가능)
DD-007	[삭제] POWER_UP pending	비전진 중 먼지 값을 나중에 POWER_UP 조건으로 저장하지 않는다.	pendingPowerUp 개념과 API를 제거한다. (수정 전: CleaningManager에 pendingPowerUp 상태 존재)
DD-008	[수정] 회전 시간	요구사항에서 회전 완료 시간이 4초로 변경되었다.	MovementManager 기본 회전 시간은 4초로 변경한다. (수정 전: 회전 완료 시간 10초)

## 5. Refactoring Targets [수정]

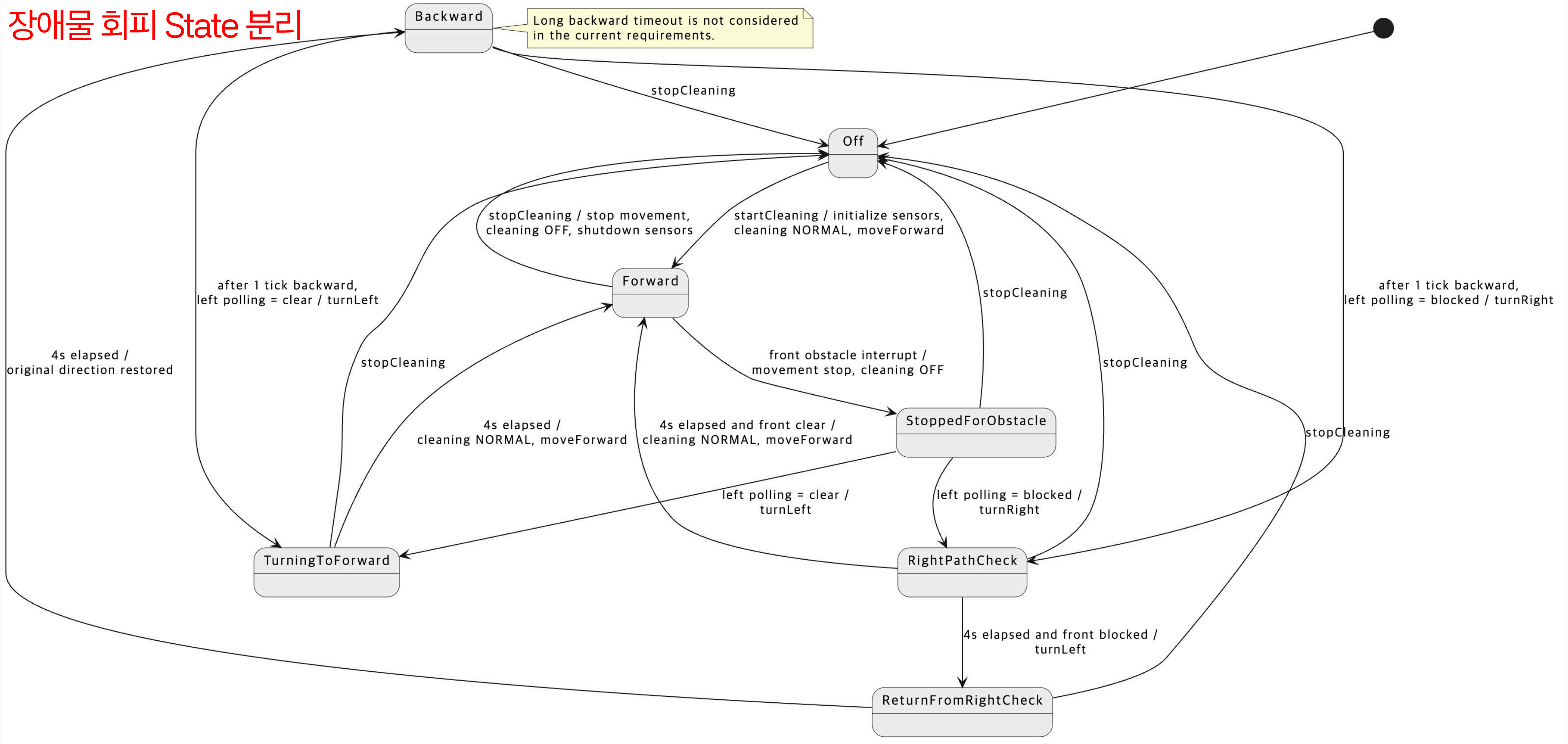
영역	Legacy 설계	Revised 설계
SideObstacleSnapshot	leftDetected, rightDetected 포함	[삭제] 제거하거나 LeftObstacleSnapshot (leftDetected)로 축소한다. controller 내부에는 rightDetected 필드가 남지 않는다.
ISideObstacleSensor	좌우 장애물 감을 polling	[수정] 좌측 장애물 감만 polling한다. 예: readLeft() 또는 read() returning LeftObstacleSnapshot.
IObstacleSensorCompatibility	legacy simulator 입력에 isRightDetected() 존재	[추가] controller 판단 흐름에서는 사용하지 않는다. 필요 시 adapter 내부 검증 권장값으로만 격리한다.
IObstacleAvoidanceStrategy	좌우 snapshot으로 좌회전/우회전/후진 결정	[수정] 좌측 감만으로 TurnLeft 또는 CheckRightPath를 결정한다. 우측 경로 결과는 state machine이 처리한다.
StoppedForObstacleState	좌우 snapshot을 읽고 즉시 방향 결정	[수정] 좌측 polling 후 좌측이 비어 있으면 좌회전, 막혀 있으면 RightPathCheckState로 전이한다.
TurningState	좌우 회전 완료 후 항상 전진	[수정] 일반 회전 완료 후에는 전진한다. 우측 경로 확인을 우회전 중 별도 상태가 전방 센싱 결과를 판단한다.
BackwardState	후진 중 좌우 snapshot으로 해제 방향 결정	[수정] 1 tick 후진 후 좌측 polling을 수행한다. 좌측이 비면 좌회전, 계속 막히면 우측 경로 확인으로 전이한다.
CleaningManager	비전진 중 NORMAL 유지 및 pending POWER_UP 지연	[수정][삭제] 비전진 중 OFF 유지, pending 제거, 전진 tick에서 dust polling으로 POWER_UP 판단.

SDD

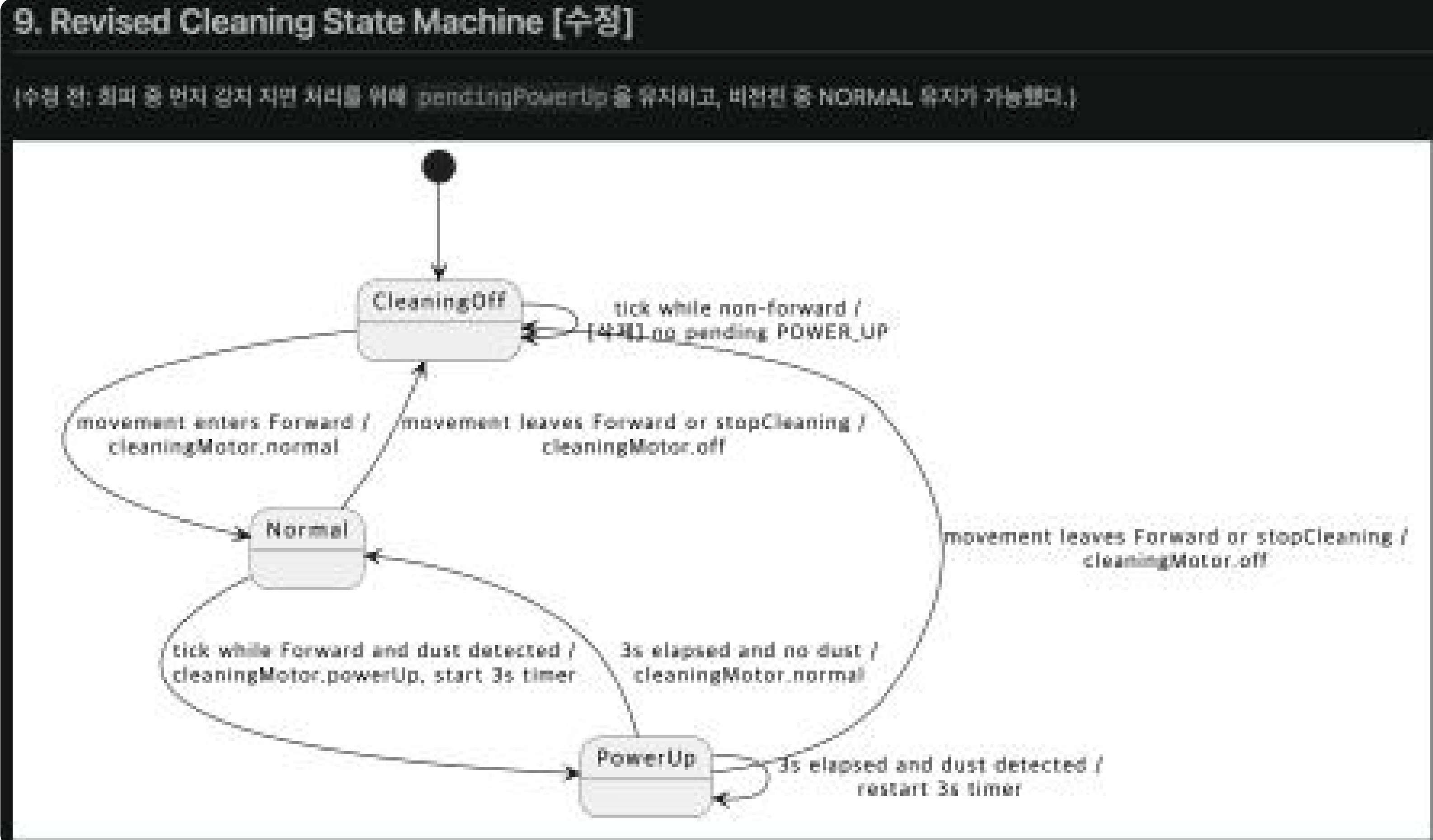
# Elaboration - OOD [수정]

8.2 Movement State Diagram [수정]

장애물 회피 State 분리



# Elaboration - OOD [수정]



# Implementation [수정]

```
155. Codex Answer 110

• timestamp: 2026-05-30T07:41:52.508Z
• json line: 1077 | phase: final_answer

이제 할 일입니다.

**Design 승인**
- 지금 'docs/design/design-specification.md' 내용이 편집되면 Design 단계를 승인하면 됩니다.
- 그러면 다음 단계인 **Implementation (Modification & Extension)** 으로 넘어가서 실제 C++ 코드를 수정합니다.
- 다음에 알을 skill 문서:
  '/Users/whdydrnr/Documents/GitHub/rvc-revision-test/skills/implementation/SKILL.md'

현재 내 기준으로는 Design 필수 산출물은 완료했고, 남은 건 네 승인입니다.

156. Prompt 046

• timestamp: 2026-05-30T07:44:04.975Z
• json line: 1082

승인, implemetation 들어가자
```

```
Implementation 단계 작업을 진행했습니다. 새 'skills/implementation/SKILL.md'는 있어서, 'AGENT.md'와 승인된 design 문서를 기준으로 구현했습니다.

핵심 변경:
- 후속 센서 내부 임박 제거: 'SideObstacleSnapshot', 'rightDetected' 기반 strategy 제거
- 'ISideObstacleSensor'를 좌측 polling 전용 'readLeft()'로 변경
- 후속 경로 확인 흐름 구현: 'RightPathCheckState', 'ReturnFromRightCheckState', revised 'BackwardState'
- 회전 기본 시간 '18s -> 4s'
- movement가 non-forward이면 cleaning 'OFF', forward 진입 시 최소 'NORMAL'
- dust는 tick 기반 polling으로 처리, 'pendingPowerUp' 제거
- 관련 unit test를 새 요구사항 기준으로 갱신

주요 수정 파일:
- [Types.hpp] (/Users/whdydrnr/Documents/GitHub/rvc-revision-test/include/rvc/Types.hpp)
- [Interfaces.hpp] (/Users/whdydrnr/Documents/GitHub/rvc-revision-test/include/rvc/Interfaces.hpp)
- [RvcStates.cpp] (/Users/whdydrnr/Documents/GitHub/rvc-revision-test/src/RvcStates.cpp)
- [CleaningManager.cpp] (/Users/whdydrnr/Documents/GitHub/rvc-revision-test/src/CleaningManager.cpp)
- [rvc_unit_tests.cpp] (/Users/whdydrnr/Documents/GitHub/rvc-revision-test/tests/rvc_unit_tests.cpp)

검증 결과:
- './commands/build.sh' 통과
- './commands/run-unit-tests.sh' 통과: 47/47
- './commands/static-analysis.sh' 통과
- 'git diff --check' 통과

주의할 점:
- './commands/run-tests.sh'는 system test까지 포함해서 현재 green이 아닙니다.
- 실패는 'simulator/scenarios/05_backward_avoidance.yaml' 1건입니다. 이 시나리오는 legacy truth table처럼 'front+left+right'에서 즉시 'BACKWARD'를 기대하지만, 새 요구사항은 'RIGHT -> front sensing -> LEFT 복귀 -> BACKWARD' 흐름이라 기대값이 충돌합니다.
- 직접 실행한 system test 결과는 '28 passed, 1 failed, 1 skipped'입니다.

다음 단계는 **Verification (Regression & New Testing)** 입니다. 여기서 legacy simulator scenario 중 새 요구사항과 충돌하는 기대값을 검토하고, 수정/제거/신규 시나리오 추가 여부를 정리하면 됩니다.
```

```
src
├── AdapterTimer.cpp
├── CleaningManager.cpp
├── LeftPriorityAvoidanceStrategy.cpp
├── LegacyAdapters.cpp
├── MovementManager.cpp
├── RvcController.cpp
├── RvcStates.cpp
├── SensorSubjects.cpp
└── SimulatorApi.cpp
```

Implementation 산출물

# Implementation - Unit Test [수정]

## 162. Codex Answer 116

```
+ timestamp: 2026-05-30T07:47:45.787Z  
+ jsonl line: 1218 | phase: commentary
```

코드 쪽 1차 변경은 들어갔습니다. 이제 테스트가 세 장벽을 검증하도록 바꾸었습니다. 특히 기존 pending POWER\_UP, 우측 센서 snapshot, "우측 해계 시 무회전" 테스트는 세 우측 경로 확인 흐름으로 대체합니다.

Implementation 단계에서 prompt 없이 알아서 테스트 코드를 수정

## 165. Codex Answer 119

- timestamp: 2026-05-30T07:50:16.313Z
- jsonl line: 1311 | phase: commentary

빌드와 전체 테스트 47개가 모두 통과했습니다.  
이제 포맷/전체 검증 스크립트까지 돌려서  
Implementation 단계 산출물이 프로젝트 검증  
기준도 통과하는지 확인하겠습니다.

Result

## build-and-test

succeeded 4 days ago in 44s

Search logs

### Run tests

0s

```
91 42/47 Test #42: SimulatorRVControllerAdapterTest.PowerOnInitializesDevicesAndStartsCoreController ..... Passed  
    0.00 sec  
92      Start 43: SimulatorRVControllerAdapterTest.TickPowerOffAndNonDetectedEventsRemainSafe  
93 43/47 Test #43: SimulatorRVControllerAdapterTest.TickPowerOffAndNonDetectedEventsRemainSafe ..... Passed  
    0.00 sec  
94      Start 44: SimulatorRVControllerAdapterTest.TickAndSensorEventsAreIgnoredWhileInError  
95 44/47 Test #44: SimulatorRVControllerAdapterTest.TickAndSensorEventsAreIgnoredWhileInError ..... Passed  
    0.00 sec  
96      Start 45: SimulatorRVControllerAdapterTest.PowerOnRetriesTransientInitializationFailures  
97 45/47 Test #45: SimulatorRVControllerAdapterTest.PowerOnRetriesTransientInitializationFailures ..... Passed  
    0.00 sec  
98      Start 46: SimulatorRVControllerAdapterTest.TickPropagatesDustDetectionToCoreController  
99 46/47 Test #46: SimulatorRVControllerAdapterTest.TickPropagatesDustDetectionToCoreController ..... Passed  
    0.00 sec  
100     Start 47: SimulatorRVControllerAdapterTest.EntersErrorWhenDeviceInitializationRetriesAreExhausted  
101 47/47 Test #47: SimulatorRVControllerAdapterTest.EntersErrorWhenDeviceInitializationRetriesAreExhausted ... Passed  
    0.00 sec  
102  
103 100% tests passed, 0 tests failed out of 47  
104  
105 Total Test time (real) = 0.11 sec
```

# Implementation - System Test [수정]

## 169. Codex Answer 123

- timestamp: 2026-05-30T07:54:23.306Z
- jsonl line: 1410 | phase: commentary

시스템 Python은 PEP 668 때문에 직접 설치가 막혀 있습니다. 워크스페이스 안에 가상환경을 만들어 pytest를 설치한 뒤 system test를 실행하는 방식으로 진행하겠습니다.

## 170. Codex Answer 124

- timestamp: 2026-05-30T07:55:29.765Z
- jsonl line: 1440 | phase: commentary

system test 30개 중 legacy 시나리오 하나가 실패했습니다. 실패한 `06\_backward\_avoidance`는 예전 truth table처럼 `front+left+right`에서 즉시 BACKWARD를 기대하는데, 새 구현은 설계대로 `RIGHT -> LEFT 복귀 -> BACKWARD` 흐름이라 기대값이 충돌합니다. 해당 시나리오를 수정할지는 Verification 단계에서 따로 다루는 게 맞는지 확인하기 위해 내용을 보겠습니다.

Implementation 과정에서 알아서 환경 구축 및 테스트 수행

주의할 점:

- `./commands/run-tests.sh`는 system test까지 포함해서 현재 green이 아닙니다.
- 실패는 `simulator/scenarios/06\_backward\_avoidance.yaml` 1건입니다. 이 시나리오는 legacy truth table처럼 `front+left+right`에서 즉시 `BACKWARD`를 기대하지만, 새 요구사항은 `RIGHT -> front sensing -> LEFT 복귀 -> BACKWARD` 흐름이라 기대값이 충돌합니다.
- 직접 실행한 system test 결과는 `28 passed, 1 failed, 1 skipped`입니다.

Result

## 172. Prompt 047

- timestamp: 2026-05-30T08:02:15.827Z
- jsonl line: 1510

이건 로직이 바뀌었으니까 테스트 시나리오06의 기대값을 바꿔야겠어. 즉시 direction을 backward로 기대하는 게 아니라 'turn right', 'turn left', 'move backward'가 차례로 이뤄지는지 테스트하는 케이스로 yaml파일 수정해

## 173. Codex Answer 126

- timestamp: 2026-05-30T08:02:21.210Z
- jsonl line: 1513 | phase: commentary

요구사항 변경으로 system scenario 기대값을 갱신하겠습니다. `06\_backward\_avoidance.yaml`만 새 흐름인 `RIGHT -> LEFT -> BACKWARD` 순서 검증으로 바꾸고 system test를 다시 돌리겠습니다.

test case 수정 prompt

## 174. Codex Answer 127

- timestamp: 2026-05-30T08:02:35.828Z
- jsonl line: 1524 | phase: final\_answer

`simulator/scenarios/06\_backward\_avoidance.yaml` 수정했습니다.

변경 내용:

- legacy 기대값: 즉시 `BACKWARD`
- 새 기대값: `RIGHT -> LEFT -> BACKWARD` 순서 확인
- 4초 우회전 + 4초 좌회전이 반영되도록 `max\_ticks`, `power\_off\_at\_tick`도 늘렸습니다.

검증:

- system test 통과: `29 passed, 1 skipped`
- `git diff --check` 통과

Result

# Difference and Analysis

# 시스템 구조 비교

## 핵심 차이

항목	rvc-controller, 정답지	rvc-revision, vibe-coding
장애물 센서 인터페이스	<code>isFrontDetected()</code> , <code>isLeftDetected()</code> 만 유지	legacy <code>IObstacleSensor</code> 에 <code>isRightDetected()</code> 잔존
observer 이벤트	<code>onObstacleDetected(front, left)</code>	<code>onObstacleDetected(front, left, right)</code> 잔존
회피 전략 API	<code>decideDirection(front, left)</code>	<code>decideDirection(front, left, right)</code> 와 새 <code>LeftPriorityAvoidanceStrategy</code> 가 공존
all-blocked 처리	오른쪽 센서 없이 "우회전 후 front 재확인"으로 판단	core는 비슷하게 구현했지만 simulator/legacy 경로에는 right truth table 유지
테스트 구조	변경 요구사항을 public contract까지 검증	자체 테스트가 남은 right 의존을 정상으로 받아들임
설계 일관성	기존 State/Strategy/Observer 구조를 유지하며 좁게 변경	새 core 구조와 legacy 호환 계층이 섞이며 시스템이 이중화됨

```
src > RvcStates.cpp
1  #include "rvc/RvcController.hpp"
2
3  namespace rvc {
4  namespace {
5
6  inline constexpr int kMaxSynchronousEscapeChecks{16};
7
8  class OffState final : public IRvcState {
9  > public: ...
24 };
25
26 class ForwardState final : public IRvcState {
27 > public: ...
40 };
41
42 class StoppedForObstacleState final : public IRvcState {
43 > public: ...
86 };
87
88 class TurningState final : public IRvcState {
89 > public: ...
107 };
108
109 class RightPathCheckState final : public IRvcState {
110 > public: ...
133 };
134
135 class ReturnFromRightCheckState final : public IRvcState {
136 > public: ...
153 };
154
155 class BackwardState final : public IRvcState {
156 > public: ...
179 };
```

# 시스템 구조 비교

## 가장 큰 문제

rvc-revision은 요구사항을 "오른쪽 센서를 컨트롤러 판단에 직접 쓰지 않는다" 정도로 해석. 그러나 정답지는 오른쪽 센서를 public API, observer event, strategy input, simulator adapter, test oracle에서 제거.

정답지는 i\_obstacle\_sensor.hpp:17에서 isRightDetected()가 없고, i\_sensor\_observer.hpp:15도 2-인자 이벤트임. 반면 revision은 Interfaces.hpp:40에 isRightDetected()가 남아 있고, SensorSubjects.hpp:12는 여전히 3-인자 이벤트임.

## 설계 관점

rvc-controller는 기존 legacy 구조를 유지한 채 요구사항 변경의 영향 범위를 추적해서 제거함. DefaultAvoidStrategy는 front+left -> RIGHT, front only -> LEFT, 그 외 FORWARD로 단 순화되고, all-blocked는 우회전 후 front/left 재확인으로 해결함. 반면 rvc-revision은 새 RvcController, IFrontObstacleSensor, ISideObstacleSensor, LeftPriorityAvoidanceStrategy를 도입해 새로 설계함. RvcStates는 좌측 polling 후 right path check를 수행하므로 개념적으로는 요구사항을 이해하지만 legacy adapter/simulator/API 계층을 완전히 정리하지 않아 SimulatorApi.cpp에는 front, left, right truth table이 남아 있음.

## 테스트 차이

정답지 테스트는 삭제된 right가 다시 나타나면 컴파일부터 실패하는 구조임. revision 테스트는 반대로 right가 남아 있는 상태를 검증. 예를 들어 rvc-revision의 rvc\_unit\_tests.cpp:152는 front+left+right일 때 reverse를 기대하고, rvc\_unit\_tests.cpp:540는 observer가 lastRight를 받는 것을 확인함. 이 때문에 자체 테스트 통과가 요구사항 충족을 보장하지 않음.

## 결론

vibe-coding 결과물은 동작 시나리오 일부를 맞추는 데는 성공했지만, maintenance 관점에서는 변경 요구사항의 추적성과 삭제 일관성이 부족. 특히 제거 요구사항에서 중요한 것은 "사용하지 않음"이 아니라 "더 이상 시스템 contract에 존재하지 않음"인데, rvc-revision은 이 기준을 통과하지 못함.

비교 결론: rvc-controller는 요구사항 변경을 아키텍처 경계까지 반영한 유지보수 결과물이고, rvc-revision은 새 구조를 만들어 기능을 맞추려 했지만 legacy right 모델을 병존시켜 회귀 가능성과 테스트 착시를 만든 결과물.

# System test 비교 : UP 버전 Simulator

## 340. Prompt 103

- timestamp: 2026-06-02T15:56:11.265Z
- jsonl line: 2521

루트 폴더 아래에 'Rvc-controller'라는 '정답지' 프로젝트 폴더를 넣었어. 지금 우리가 쓴 simulator도 여기서 가져온거야. 이 정답지 rvc-controller에서는 simulator 시나리오가 전부 통과했고, 우리의 codex 버전 controller를 '정답지 simulator'를 돌렸을 때 시나리오 2개가 실패가 뜨는데, 이유를 분석해야 해.

## 341. Codex Answer 238

- timestamp: 2026-06-02T15:56:20.809Z
- jsonl line: 2525 | phase: commentary

정답지 프로젝트와 현재 프로젝트의 controller/simulator 연결부를 비교해서, 실패가 “시나리오 기대값이 legacy라서”인지 “우리 controller 동작이 정답지와 다르게 구현돼서”인지 분리해보겠습니다. 파일 수정 없이 분석만 하겠습니다.

## 342. Codex Answer 239

- timestamp: 2026-06-02T15:56:33.079Z
- jsonl line: 2535 | phase: commentary

정답지 쪽에서 중요한 단서가 나왔습니다. 정답지 controller는 우리처럼 `RightPathCheckState/BackwardState`로 tick에 걸쳐 상태를 나누는 구조가 아니라, `AvoidingState::onEnter()`에서 회피 루틴을 즉시 실행하는 구조입니다. 이 차이가 08/26의 command count 실패와 직접 연결되는지 확인하겠습니다.

## 345. Codex Answer 242

- timestamp: 2026-06-02T15:57:22.816Z
- jsonl line: 2567 | phase: final\_answer

이제 보면 26번 실패 이유는 “정답지 시나리오가 이상하다”가 아니라, 우리 controller의 회피 흐름이 정답지 controller와 다르게 구현되어 서야.

정답지 controller의 핵심은 [Rvc-controller/src/avoiding\\_state.cpp](#)에 있어.

정답지 `AvoidingState::onEnter()` 흐름은 대략 이거야.

```
mm->executeAvoidance(front_, left_);

bool currentFront = sensor->isFrontDetected();

if (currentFront) {
    mm->turn(Direction::LEFT);
    mm->moveBackward();

    while (true) {
        const bool currentLeft = sensor->isLeftDetected();
        if (!currentLeft) {
            mm->turn(Direction::LEFT);
            break;
        }
        mm->turn(Direction::RIGHT);
        currentFront = sensor->isFrontDetected();
        if (currentFront) {
            mm->turn(Direction::LEFT);
        } else {
            break;
        }
    }
}

ctx.enterCleaning();
```

수치적인 비교를 위해 UP 팀의 Simulator로 테스트  
2개의 Scenario 에서 Fail

# System test 비교 : UP 버전 Simulator

즉, 정답지는 **AvoidingState** 진입 한 번 안에서 회피 루틴을 거의 동기적으로 끝까지 수행해. 반면 우리 구현은 `StoppedForObstacleState`, `TurningState`, `RightPathCheckState`, `ReturnFromRightCheckState`, `BackwardState` 로 나눠서 tick/state transition에 걸쳐 처리해.

26번에서 정답지가 6 을 기대하는 이유는 이거야.

- tick 1에 interrupt가 두 번 발생함.
  - `before_poll`
  - `after_poll`
- simulator runner는 interrupt 시 `arm_front_interrupt()` 를 해서 다음 front sensor read들이 `true` 가 되게 함.
- 정답지에서는 interrupt 1번마다 `AvoidingState::onEnter()` 가 즉시 실행됨.
- world가 비어 있어도 forced front read 때문에 `currentFront == true` 가 되어 후속 회피 블록에 들어감.
- 그래서 interrupt 1회당 대략 다음 avoidance command가 나옴:
  - `LEFT`
  - `BACKWARD`
  - `LEFT`
- 따라서 interrupt 2회면:
  - `3 * 2 = 6`

Scenario 08, 26 모두 장애물 회피를 여러 State로 나눈 구조 때문에 실패

우리 controller는 같은 상황에서 실제 history가:

```
FORWARD, STOP, LEFT, FORWARD, STOP, LEFT, FORWARD, STOP, LEFT, FORWARD, STOP
```

avoidance command만 세면:

```
LEFT, LEFT, LEFT
```

즉, 3 개야.

왜 3개밖에 안 나오냐면, 우리 구현은 front-only 상황에서:

- `readLeft() == false`
- `decideAfterFrontObstacle(false) == TurnLeft`
- `TurningState` 로 가서 좌회전
- 바로 `BACKWARD` 루틴으로 들어가지 않음

정답지처럼 "회피 중 front 재확인 결과가 true면 좌회전/후진/좌회전까지 즉시 수행"하지 않아.

결론적으로 실패 원인은:

- 정답지 simulator는 정답지 controller의 동기식 **AvoidingState** 회피 루틴을 기준으로 작성됨.
- 우리 controller는 요구사항 변경을 state machine으로 더 세분화했고, front-only interrupt를 단순 좌회전 회피로 끝냄.
- 특히 `AvoidingState::onEnter()` 내부의 `currentFront = sensor->isFrontDetected()` 후 `LEFT/BACKWARD/LEFT` 로 이어지는 정답지 흐름이 우리 구현에는 없음.

그래서 08번도 같은 이유로 실패해. interrupt 5개에 대해 정답지는 `5 * 3 = 15` 를 기대하지만, 우리 controller는 interrupt당 대체로 `LEFT` 하나라서 5가 나와.

# System test 비교 : UP 버전 Simulator

## User Prompt - line 2572

Timestamp: 2026-06-02T16:08:14.529Z

내 기억에 이번 '오른쪽 센서 삭제' revision 프로젝트를 시작할 때는 state가 그렇게 세분화되지 않았는데 이번 요구사항 변경을 적용하면서 state를 잘게 쪼개는 이유가 뭐야?

## Codex Output (final\_answer) - line 2575

Timestamp: 2026-06-02T16:08:31.520Z

이유는 내가 설계할 때 "오른쪽 센서 삭제"를 단순히 기존 `AvoidingState::onEnter()` 안의 로직만 바꾸는 문제가 아니라, 시간이 걸리는 회전/센싱/복귀/후진 절차를 명시적인 상태 전이로 표현해야 하는 변경으로 판단했기 때문이야.

당시에 정리했던 요구사항은 이런 흐름이었어.

- 전방+좌측 장애물 감지
- 90도 우회전
- 회전 완료 후 전방 센싱
- 막혀 있으면 90도 좌회전으로 원래 방향 복귀
- 1 tick 후진
- 후진 중 매 tick 좌측 polling
- 필요하면 다시 우측 경로 확인 반복

이걸 하나의 `AvoidingState::onEnter()` 안에서 즉시 처리하면, 실제 시간 개념인 "4초 회전 완료 후 센싱"과 "1 tick 후진 후 polling"을 표현하기 어렵다고 봤어. 그래서 다음처럼 쪼갬어.

- `StoppedForObstacleState` : 전방 장애물 후 좌측 polling
- `TurningState` : 일반 좌회전
- `RightPathCheckState` : 90도 우회전 후 전방 센싱
- `ReturnFromRightCheckState` : 우측 경로가 막혔을 때 원래 방향 복귀
- `BackwardState` : 1 tick 후진 후 좌측 polling

즉, 목적은:

- 오른쪽 센서 직접 판단 제거
- "우회전 후 전방 센싱"을 별도 단계로 명확화
- 4초 회전 완료와 1 tick 후진을 테스트 가능한 상태로 만들기
- cleaning motor OFF/ON 정책을 movement state와 일관되게 묶기

# System test 비교 : UP 버전 Simulator

## User Prompt - line 2582

Timestamp: 2026-06-02T18:41:20.378Z

정답지처럼 avoidingstate 하나 안에서 장애물 회피가 이뤄지게 수정하자. 너무 정답지의 코드를 그대로 사용하면 우리 시스템과 잘 맞지 않을 수 있으니 한 state 안에서 회피가 이루어진다는 것만 우선 적용해서.

## Codex Output (commentary) - line 2586

Timestamp: 2026-06-02T18:41:31.771Z

현재 구현을 정답지처럼 완전히 같아엮기보다, 기존 인터페이스와 변경 요구사항은 유지하면서 StoppedForObstacleState 진입 시 회피 루틴을 한 번에 실행하도록 좁게 수정하겠습니다. 먼저 현재 state/manager/timer 구조를 정확히 확인하겠습니다.

## Codex Output (commentary) - line 2616

Timestamp: 2026-06-02T18:42:37.836Z

수정 방향은 StoppedForObstacleState::onEnter() 안에서 회피 명령 시퀀스를 즉시 수행하고 ForwardState로 복귀시키는 것입니다. 기존 RightPathCheckState 등 factory는 당장 제거하지 않고 남겨서 변경 범위를 줄이겠습니다.

장애물 회피가 한 State 내부에서 완료되도록 교정

## Codex Output (final\_answer) - line 2746

Timestamp: 2026-06-02T18:45:39.898Z

정답지처럼 "회피가 한 state 안에서 끝나는" 방향으로 수정했고, 8/26번 포함 정답지 simulator가 통과하도록 맞췄어.

수정한 핵심:

- src/RvcStates.cpp
  - StoppedForObstacleState::onEnter()에서 좌측 polling, 우측 경로 확인, front 재확인, BACKWARD, 좌측 재확인을 한 번에 수행하도록 변경.
  - 회피 루틴 후 바로 startForwardCleaning()하고 ForwardState로 복귀.
  - 기존 RightPathCheckState, ReturnFromRightCheckState, BackwardState factory는 제거하지 않고 유지.
- src/SensorSubjects.cpp
  - onInterrupt()에서 front\_ = true로 고정하지 않고 실제 sensor\_.isFrontDetected()를 읽도록 변경.
  - 이게 중요했어. 기존 방식은 simulator의 force\_front\_reads가 하나 남아서 다음 poll에서 불필요한 추가 회피가 발생했고, 그래서 8/26번이 기대값보다 더 많이 나왔어.
- tests/rvc\_unit\_tests.cpp
  - 기존 "중간 state에 머문다"는 기대를 "동기식 회피 완료 후 Forward/Cleaning으로 복귀한다"로 갱신.

검증:

```
./build/tests/rvc_tests --gtest_brief=1  
47 passed
```

```
PYTHONPATH=build/bindings:simulator .venv/bin/python  
-m pytest simulator/tests/ -q  
28 passed, 1 skipped
```

# 교정 내용 반영 문서 수정 - SRS, SDD, Verification Summary

## RVC Software Requirements Specification

### 1. 현재 단계

현재 단계는 Requirements (Revision Correction)이다. 이 문서는 RVC(Robot Vacuum Cleaner)의 `rvc-controller`에 대한 자동 진공 청소 제어 로직 요구사항 변경과 교정 내용을 정리한다. (수정 전 : 현재 단계는 Requirements (Revision)이다. 이 문서는 RVC(Robot Vacuum Cleaner)의 `rvc-controller`에 대한 자동 진공 청소 제어 로직 요구사항 변경을 정리한다.)

[추가] 교정 반영 범위는 정답지 simulator 검토 후 확인된 장애물 회피 처리 흐름과 전방 장애물 interrupt 중복 처리 방지 요구사항이다. 교정 내용은 구현 class 또는 state 이름이 아니라 외부에서 관찰 가능한 센서 입력, movement motor command, cleaning motor command 기준으로 정리한다.

## RVC Controller Design Specification

### 1. 현재 단계

현재 단계는 Design이다. 본 문서는 요구사항 분석 산출물을 바탕으로 `rvc-controller`의 객체지향 설계를 정의한다.

#### [수정] Revision, Correction 반영 - 현재 단계 및 설계 범위

현재 단계는 Design (Refactoring & Update)이다. 본 문서는 승인된 Requirements 및 Analysis 산출물을 바탕으로 legacy `rvc-controller` 설계를 수정한다.

- [수정] 설계 대상은 legacy `rvc-controller` 내부 구조를 변경하는 범위로 한정한다. (수정 전 : 요구사항 분석 산출물을 바탕으로 `rvc-controller`의 객체지향 설계를 정의한다.)
- [추가] simulator 및 검증 인프라는 controller 설계 대상이 아니며, simulator의 우측 장애물 정보는 검증 관찰 데이터로만 사용할 수 있다.
- [수정] controller 내부 판단 입력과 내부 인터페이스는 좌측 장애물 polling 중심으로 변경한다. (수정 전: 좌측 및 우측 장애물 값을 controller 내부 판단 입력으로 사용한다.)
- [추가] legacy 우측 센서 감지는 90도 우회전 완료 후 전방 센싱으로 대체한다.

## RVC Verification Summary

### 1. 현재 단계

[수정] 현재 단계는 Verification (Regression & New Testing)이다. 본 문서는 오른쪽 센서 제거와 새로운 우측 경로 확인 전략이 적용된 구현 결과를 검증한 내용을 정리한다. (수정 전 : 현재 단계는 Verification이다. 검증 기준은 다음과 같이 정정되었다.)

[추가] 교정 이후 Verification은 정답지 simulator와 비교해 확인된 장애물 회피 command timing 차이를 검증 범위에 포함한다. 특히 우측 경로 확인을 별도 중간 state로 지연 처리하지 않고 StoppedForObstacleState 진입 흐름에서 회피 command sequence가 즉시 관찰되는지, 그리고 interrupt가 연속 또는 동일 tick 내에서 발생해도 중복 처리 오류 없이 회피 command 수가 기대값과 일치하는지 확인한다.

- Unit Test: GoogleTest 기반으로 수행한다.
- System Test: [수정] 기존 custom simulator를 사용한다. (수정 전 : 직접 개발되어 GitHub에 존재하는 simulator만 사용한다.)
- Mock/fake 기반 controller scenario test는 system test로 간주하지 않는다.
- Legacy test: [추가] 새 요구사항과 충돌하지 않는 경우 유지한다.
- Legacy 기대값: [추가] 새 요구사항과 충돌하는 경우 테스트 의도와 assertion을 새 요구사항 기준으로 갱신한다.