

Traditional Coding vs AI Agent Coding

The RVC Control SW

6팀 정상훈, 정수혁, 하재아, 한지훈

| 목차

Chapter 0. **Traditional Coding 요구사항 변경**

Chapter 1. **AI Agent Coding 요구사항 변경**

Chapter 2. **결과 비교 및 최종 수정**

Chapter 3. **느낀점**

Traditional Coding 요구사항 변경

0. 회피로직 변경 사항
1. UseCase(fully dressed) 수정
2. system operation 수정
3. system sequence diagram 수정
4. domain model 수정
5. sequence diagram 수정
6. class diagram 수정
7. OOI단계 수정

회피로직 변경 사항

US-01

기존: 전방 장애물 감지 시 바로 회피 로직 실행

요구사항 변경: 전방 장애물 감지 시 **turn right**으로 오른 편에 장애물이 있는지 인지 후 다시 **turn left** 후 기존처럼 회피로직 실행

US-02

기존: 후진 중에 좌우 장애물 감지, 회피가 가능할 경우 바로 회피 로직 실행

요구사항 변경: 후진 중에 `notifyObstacle`이 들어올 때마다 **turn right**으로 오른 편에 장애물이 있는지 인지 후 다시 **turn left** 후 기존처럼 회피로직 실행 (회피 할 수 없을 경우 계속 후진)

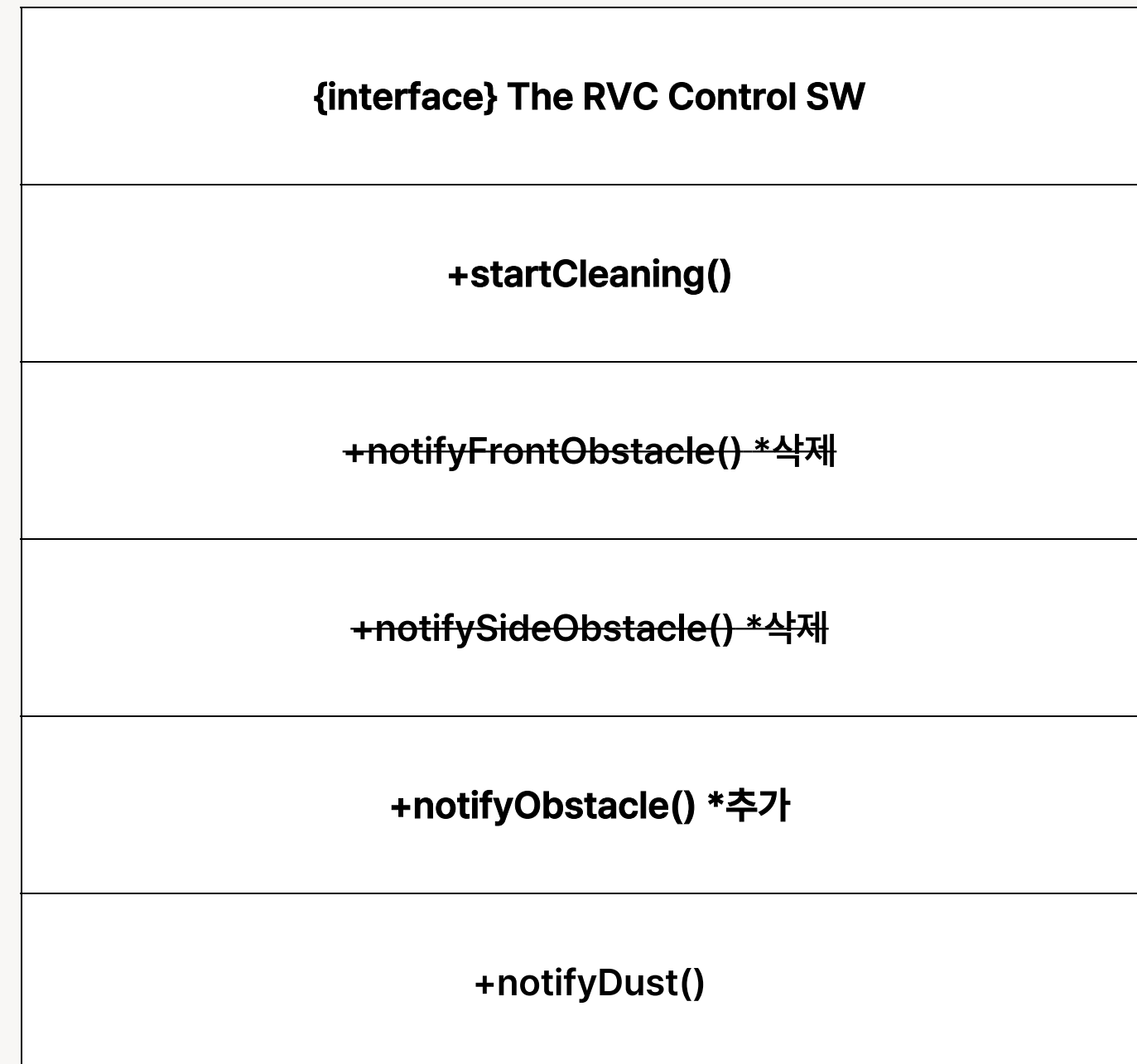
UseCase(fully dressed) 수정 UC-01

<p>Typical Courses of Events</p>	<p>(A): Actor, (S): System</p> <ol style="list-style-type: none">1. (A) Obstacle Sensor가 장애물을 감지하여 sensorData(location)을 전송한다2. (S) ObstacleSensorHandler가 ActionController의 obstacleStatus(loation)을 에 호출한다 <p><추가></p> <ol style="list-style-type: none">3. (S) ActionController가 MotorHandler의 turnRight()를 호출한다.4. (S) MotorHandler가 Motor에 turn right 명령을 전달한다.5. (A) ObstacleSensor가 재감지하여 notifyObstacle(location)을 ObstacleSensorHandler에 재전송한다.6. (S) ObstacleSensorHandler가 ActionController의 obstacleStatus(location)을 호출한다.7. (S) ActionController가 MotorHandler의 turnLeft()를 호출한다.8. (S) MotorHandler가 Motor에 turn left 명령을 전달한다. <p><₩추가></p> <ol style="list-style-type: none">9(번호 및 내용 수정). (S) 좌측이 회피 가능하면 ActionController가 MotorHandler의 turnLeft()를 호출한다10(번호 수정). (S) MotorHandler가 Motor에 turn left 명령을 전달한다11(번호 수정). (S) ActionController가 RVCController의 resumeClaening()을 호출한다
<p>overview</p>	<p>Obstacle Sensor가 넘겨준 sensorData의 값을 토대로</p> <p><추가></p> <p>좌측 및 전방의 장애물을 감지하고, turnRight()를 수행한 뒤 재감지하여, 좌측, 전방, 우측의 장애물 유무를 다 파악한다. 이어서 turnLeft()를 수행해 원래의 상태로 돌아온다.</p> <p><₩추가></p> <p>ActionController가 회피할 방향을 결정해 Motor에게 지시하고 다시 청소를 시작한다.</p> <p>좌,우 방향이 모두 막혀있으면 후진(uc-02)한다.</p>

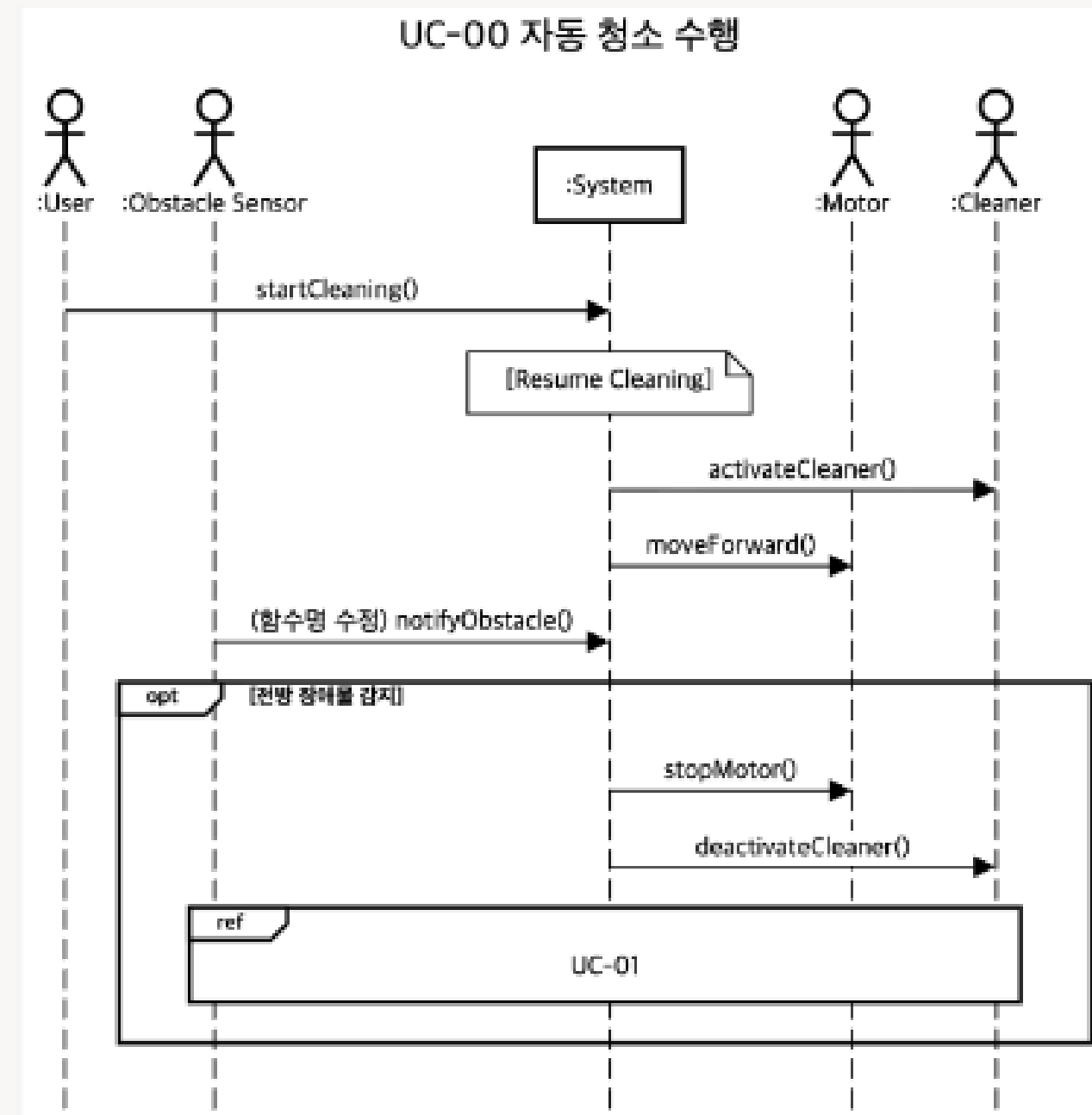
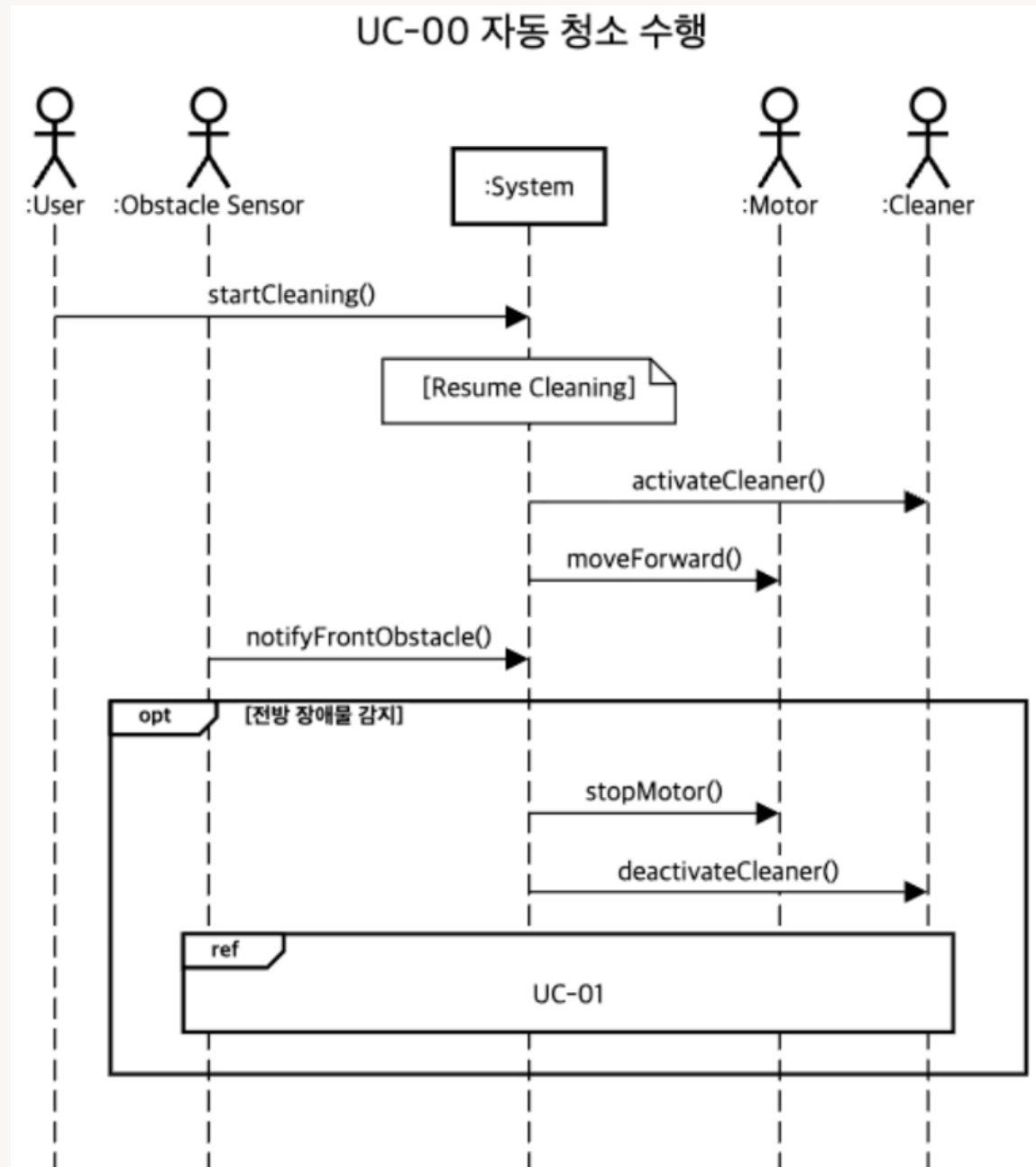
UseCase(fully dressed) 수정 UC-02

<p>Typical Courses of Events</p>	<p>(A): Actor, (S): System</p> <ol style="list-style-type: none"> 1. (S) ActionController가 MotorHandler의 moveBackward()를 호출한다. 2. (S) MotorHandler가 Motor에 move backward 명령을 전달한다. 3. (A) ObstacleSensor가 후진 중 sensorData(location)를 ObstacleSensorHandler에 전송한다. 4. (S) ObstacleSensorHandler가 ActionController의 obstacleStatus(location)를 호출한다. <p><추가></p> <ol style="list-style-type: none"> 5. (S) ActionController가 MotorHandler의 turnRight()를 호출한다. 6. (S) MotorHandler가 Motor에 turn right 명령을 전달한다. 7. (A) ObstacleSensor가 재감지하여 notifyObstacle(location)을 ObstacleSensorHandler에 재전송한다. 8. (S) ObstacleSensorHandler가 ActionController의 obstacleStatus(location)을 호출한다. 9. (S) ActionController가 MotorHandler의 turnLeft()를 호출한다. 10. (S) MotorHandler가 Motor에 turn left 명령을 전달한다. 11. (S) ActionController가 좌/우 상태를 판단한다. <p><W추가></p> <ol style="list-style-type: none"> 12 (번호 수정). (S) 좌측이 회피 가능하면 ActionController가 MotorHandler의 turnLeft()를 호출한다. 13 (번호 수정). (S) MotorHandler가 Motor에 turn left 명령을 전달한다. 14 (번호 수정). (S) ActionController가 RVCController의 resumeCleaning()을 호출한다.
<p>overview</p>	<p>ObstacleSensor가 전방 장애물과 좌·우 회피 불가 상태를 감지하면 시스템은 Motor를 후진시킨다.</p> <p><추가></p> <p>후진 후 loop 내에서 ObstacleSensor가 재감지한다. 좌측 및 전방의 장애물을 감지하고, turnRight()를 수행한 뒤 재감지하여, 좌측, 전방, 우측의 장애물 유무를 다 파악한다. 이어서 turnLeft()를 수행해 원래의 상태로 돌아온다.</p> <p><W추가></p> <p>ActionController가 방향을 결정하여 Motor를 좌회전 또는 우회전시킨다. 이후 RVCController가 resumeCleaning()을 수행하여 자동 청소로 복귀한다.</p>

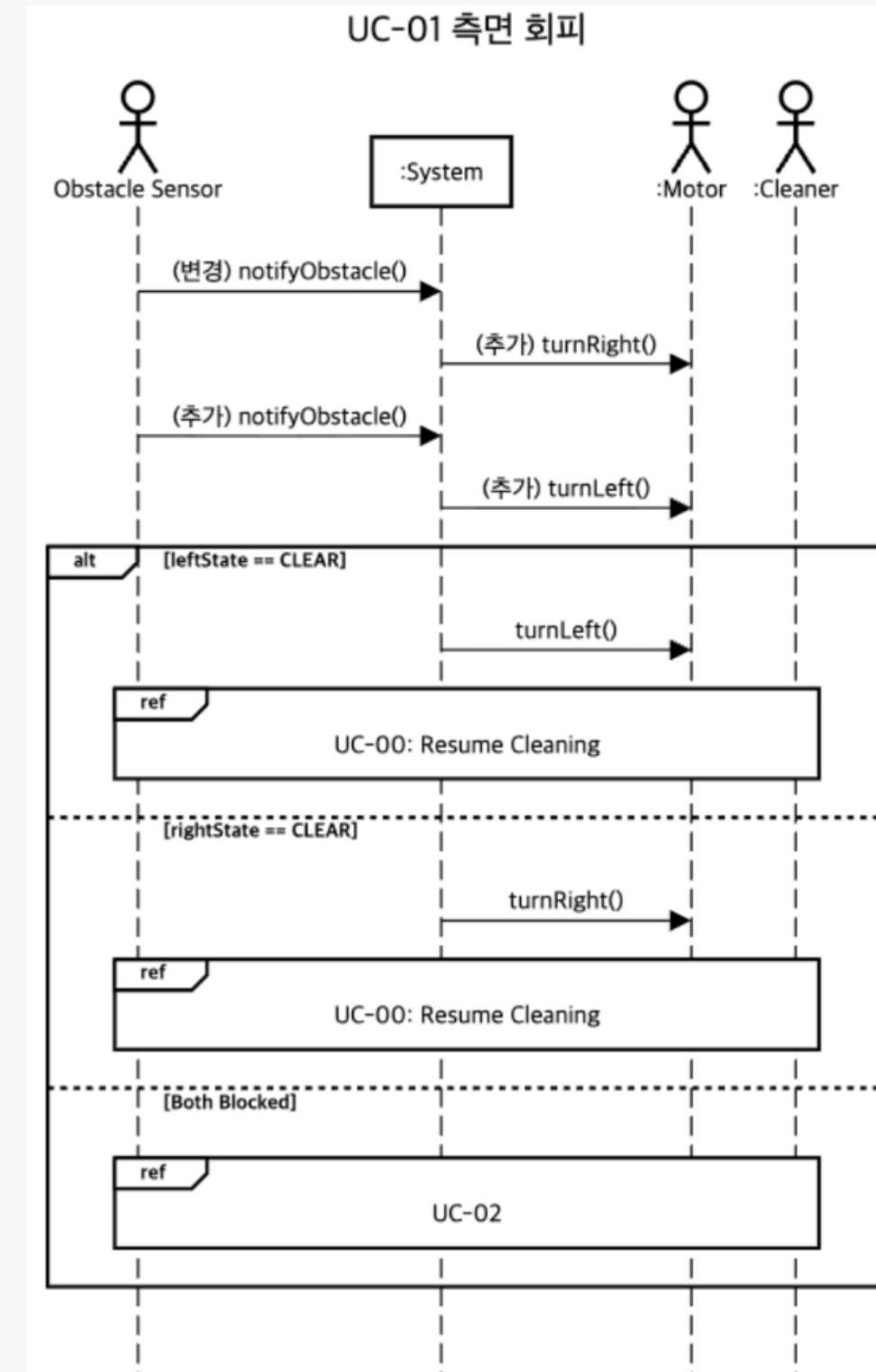
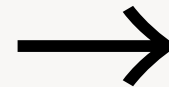
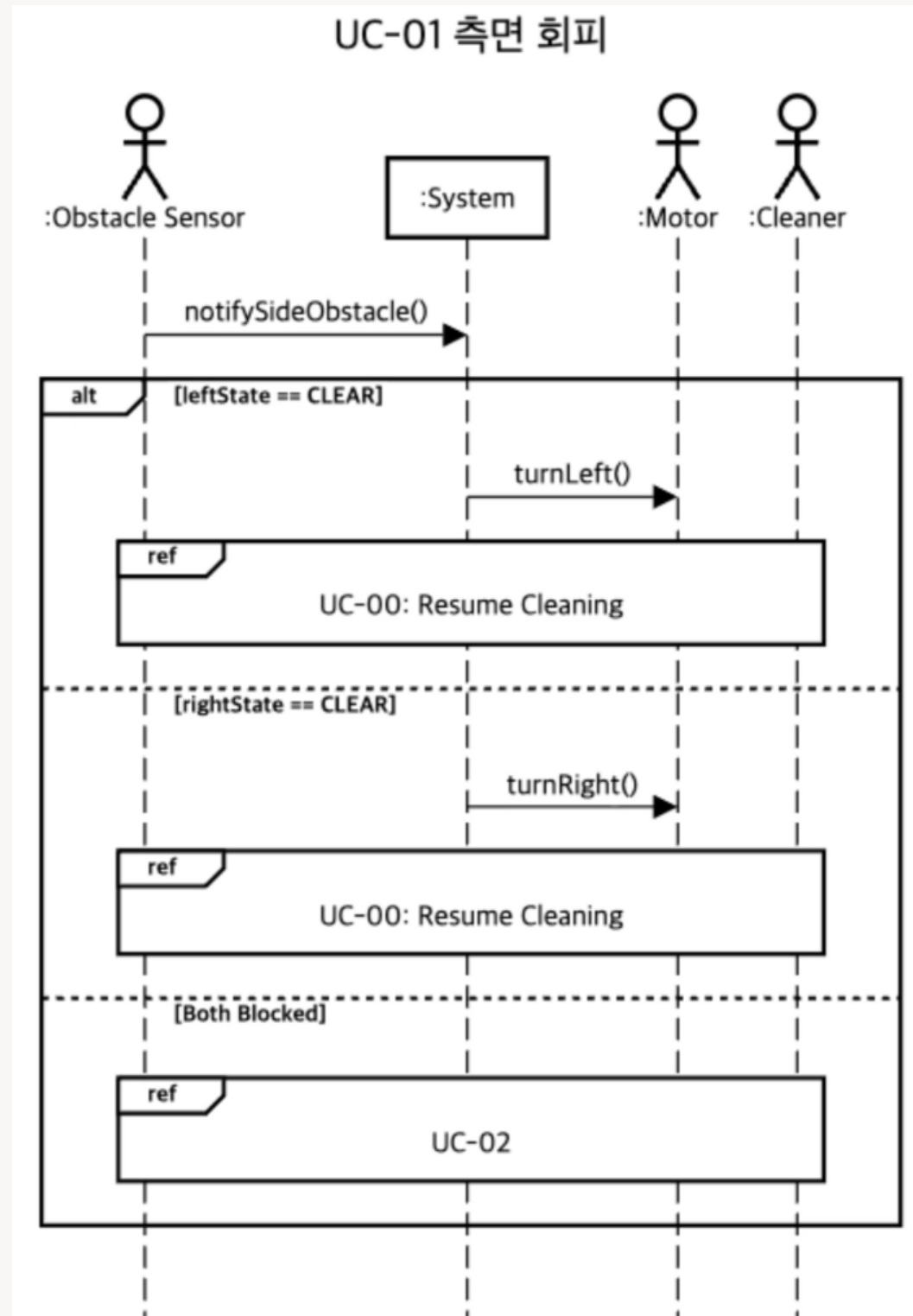
system operation 수정



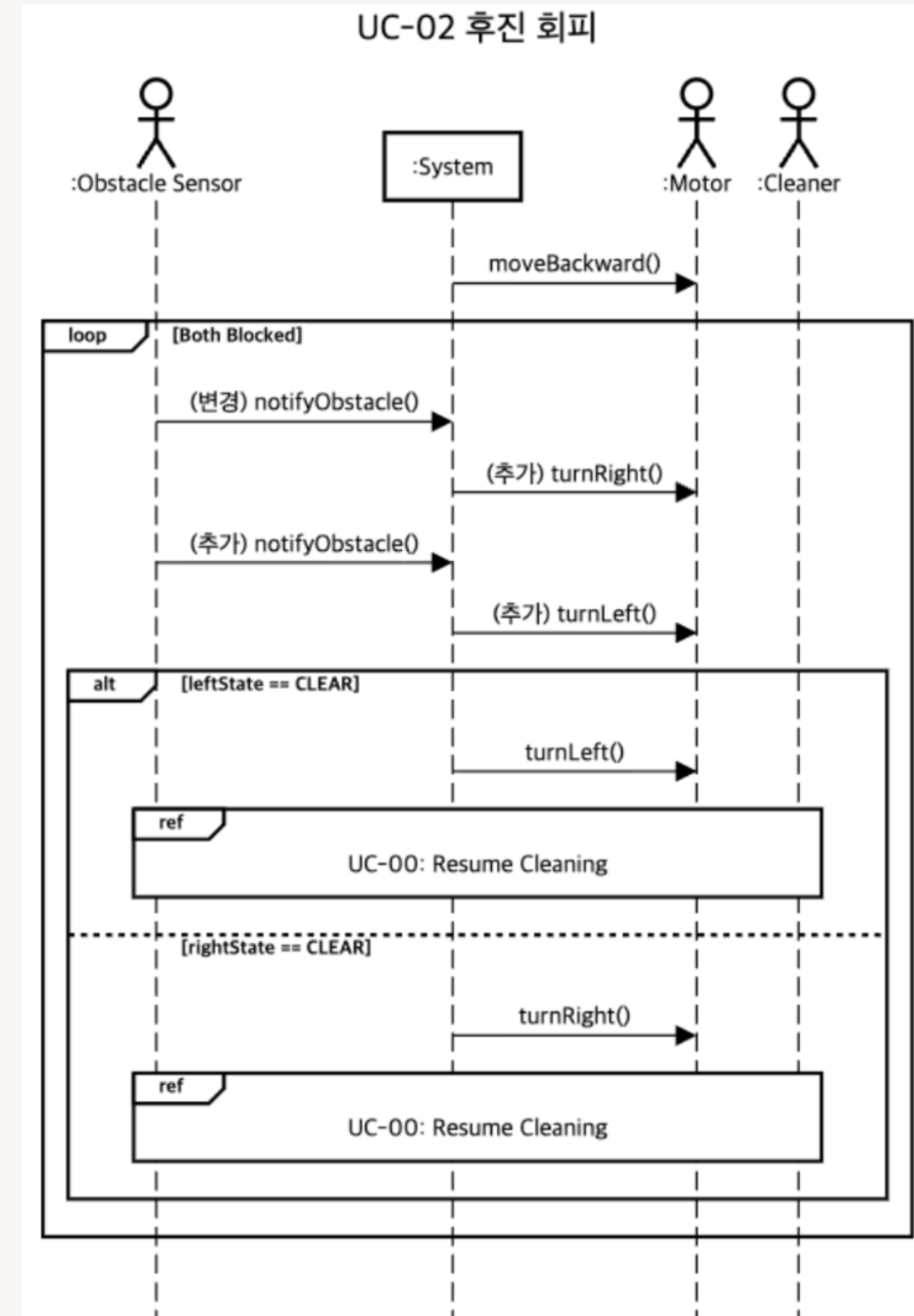
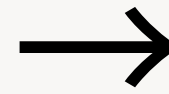
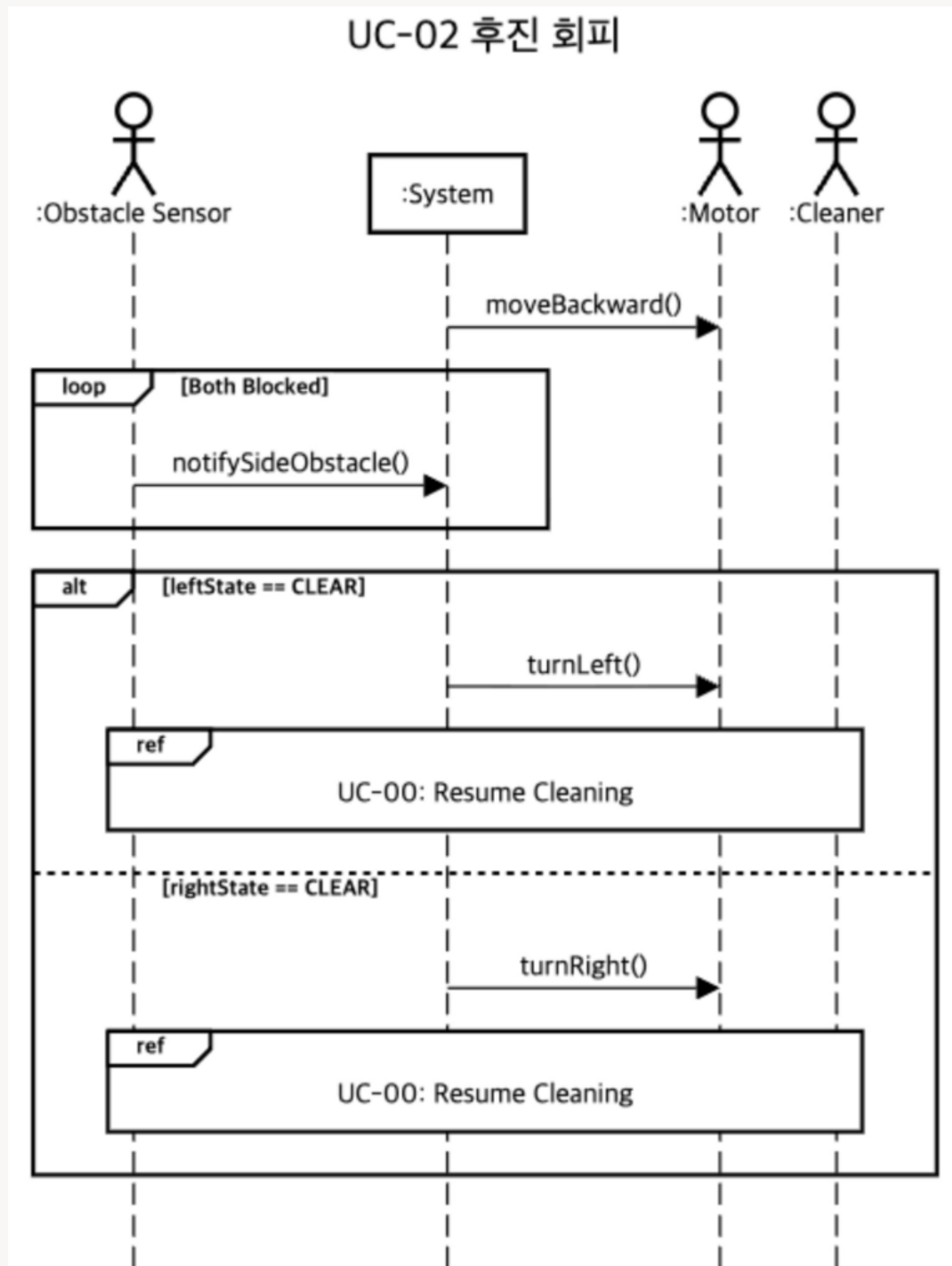
system sequence diagram 수정 - UC-00



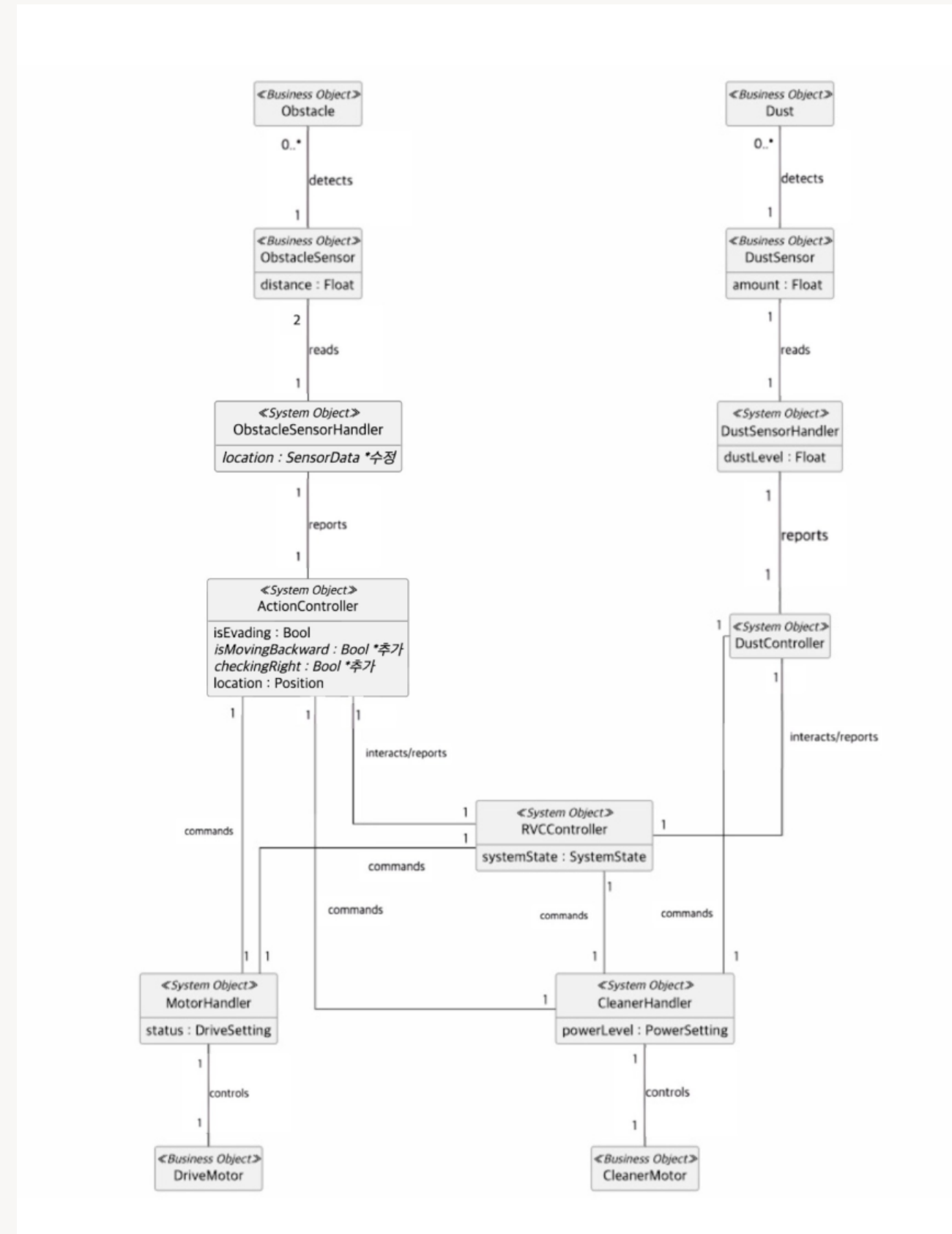
system sequence diagram 수정 - UC-01



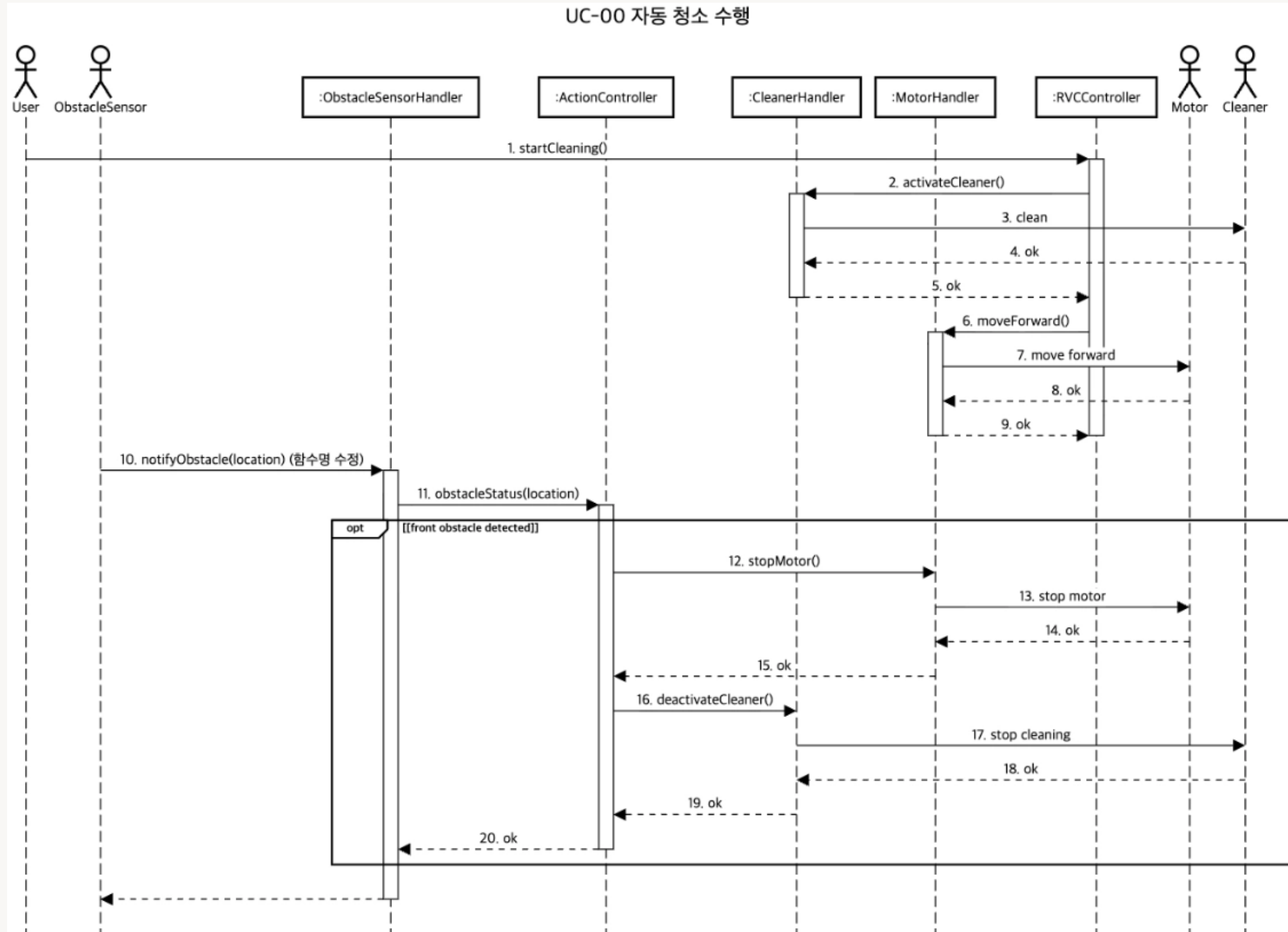
system sequence diagram 수정 - UC-02



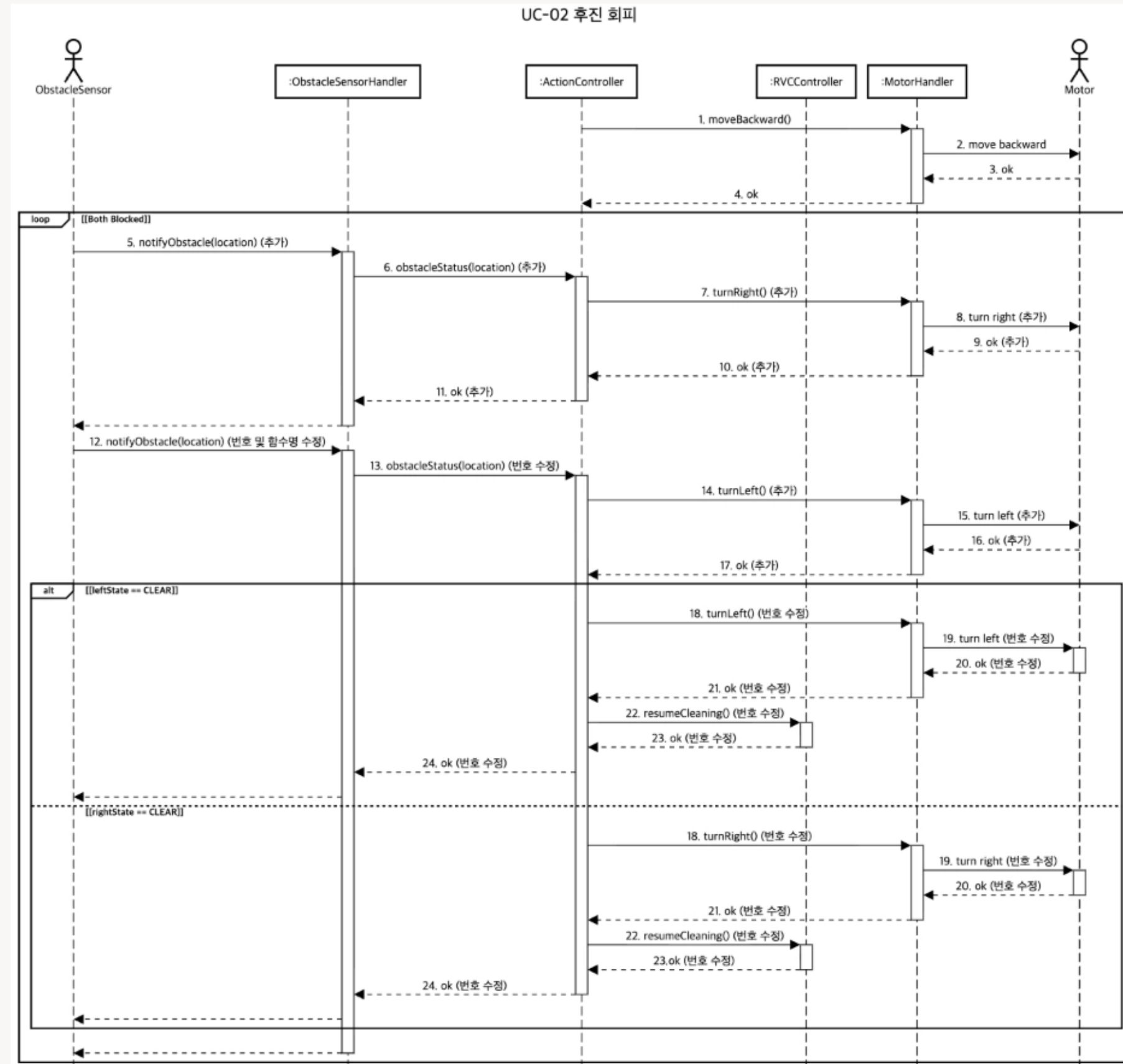
domain model 수정



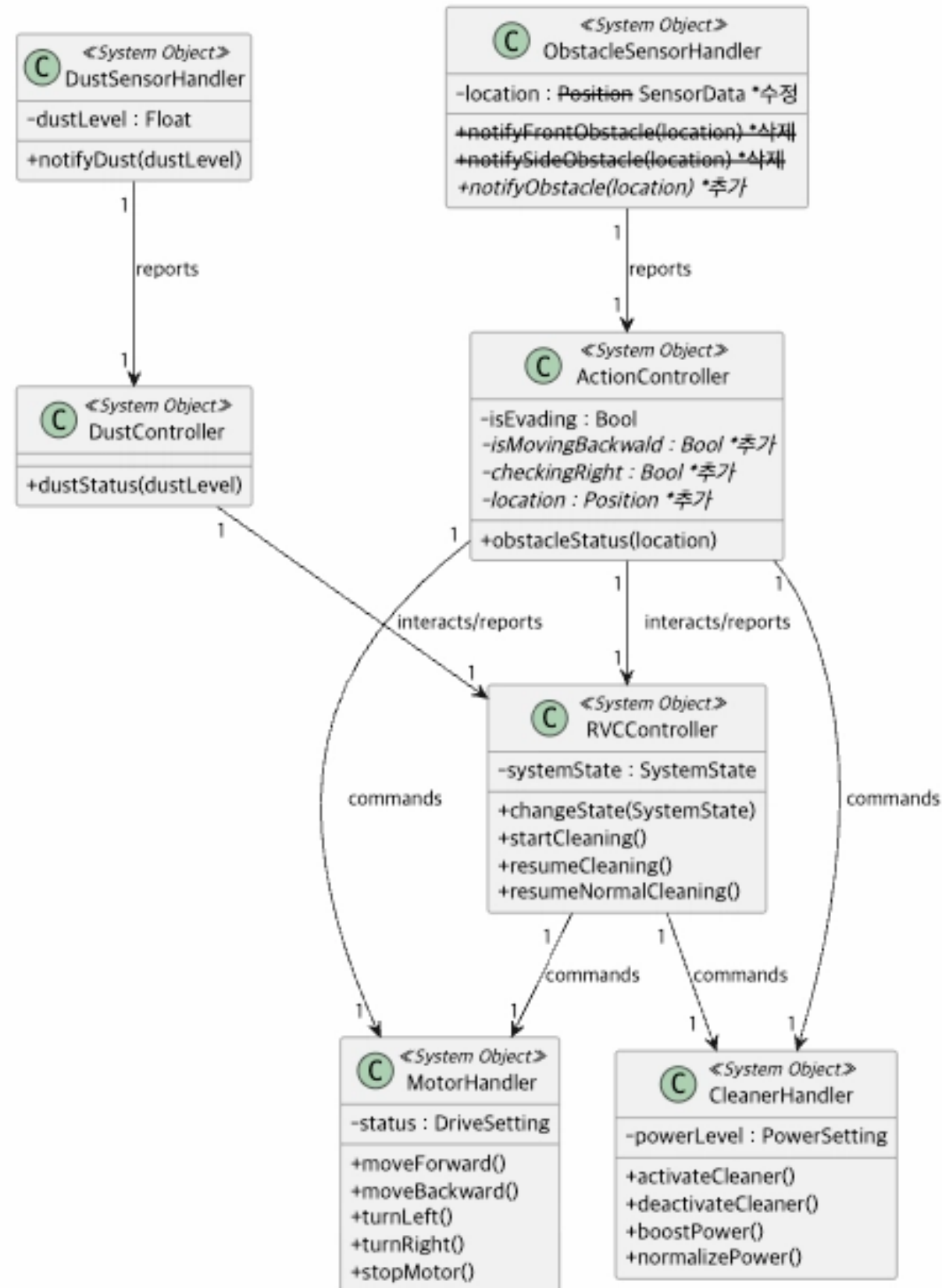
sequence diagram 수정 - UC-00



sequence diagram 수정 - UC-02



class diagram 수정



00이 수정 - Types.h

```
// 전/좌/우 방향의 장애물 감지 여부 (Position)
struct Position {
    bool isFrontBlocked = false; // 전방 장애물 유무 (true: 있음, false: 없음)
    bool isLeftBlocked = false; // 좌측 장애물 유무
    bool isRightBlocked = false; // 우측 장애물 유무
};

// 추가 코드 - ObstacleSensorHandler가 센서데이터 값을 받을 때 사용하기 위함.
struct SensorData{
    bool leftSensorData = false; // 좌측 장애물 유무 (true: 있음, false: 없음), 회피 중 turn right 상태일 때는 전방 장애물 유무 기능
    bool frontSensorData = false; // 전방 장애물 유무 (true: 있음, false: 없음), 회피 중 turn right 상태일 때는 우측 장애물 유무 판단 기능
};
```

OOI 수정 - ActionController.h

```
h ActionController.h > ...
#pragma once
#include "Types.h"

class MotorHandler;
class CleanerHandler;
class RVController;

class ActionController {
private:
    MotorHandler* motorHandler;
    CleanerHandler* cleanerHandler;
    RVController* rvcController; // 메인 시스템 상태 보고용 포인터
    bool isEvading = false; // actioncontroller 스스로 회피 중인지 기억하는 로컬 변수

    bool isMovingBackward = false; // 추가된 필드 - 현재 뒤로 후진 중인지

    bool checkingRight = false; // 추가된 필드

    // 추가 - 현재 장애물의 유무를 저장 및 회피 로직에 사용하지 위함
    Position location; // 현재까지 감지된 장애물 상태

public:
    ActionController(MotorHandler* mh, CleanerHandler* ch, RVController* rvc);

    // Position에서 SensorData로 변경
    void obstacleStatus(SensorData location); // 장애물 위치에 따른 판단 및 명령
};
```

OOI 수정 - ActionController.cpp - (1)

```

void ActionController::obstacleStatus(SensorData loc) {
    // ===== 초기 및 측면 회피 로직 =====
    // 1-1(번호 및 조건문 수정). 정상 청소 중 전방 장애물 최초 감지
    if (!isEvading && loc.frontSensorData && !isMovingBackward){
        std::cout << "[ActionController] 전방 장애물 감지! 회피 시작...\n";

        isEvading = true; // 회피 시작.
        checkingRight = true; // 추가된 로직, 오른쪽 장애물 확인을 위함.

        location.isLeftBlocked = loc.leftSensorData; // 추가된 로직, 좌측 장애물 확인을 위함
        location.isFrontBlocked = loc.frontSensorData; // 추가된 로직, 전방 장애물 확인을 위함.

        rvcController->changeState(SystemState::AVOIDING); // 메인 시스템에 보고

        motorHandler->stopMotor();
        cleanerHandler->deactivateCleaner();

        std::cout << "[ActionController] 오른쪽 장애물 유무를 위한 turn right...\n"; // 추가된 cout
        motorHandler->stopMotor();
        motorHandler->turnRight(); // 추가된 로직, 오른쪽 확인 들어감

        return;
    }
}

```

```

// 1-2. 정상 청소 중 전방 장애물 최초 감지 후 오른쪽 확인 + 회피 로직을 위한 추가 (추가된 조건문)
if (isEvading && checkingRight && !isMovingBackward) {
    checkingRight = false;

    location.isFrontBlocked=loc.leftSensorData; // 추가된 로직, 전방 장애물 확인을 위함
    location.isRightBlocked=loc.frontSensorData; // 추가된 로직, 우측 장애물 확인을 위함

    std::cout << "[ActionController] 오른쪽 장애물 유무를 확인 후 방향 복구 turn left...\n"; // 추가된 cout
    motorHandler->stopMotor();
    motorHandler->turnLeft(); // 추가된 로직, 방향 복구

    // (조건문 위치 수정) 1-3. 회피 기동 중일 때 (UC-01 회피 로직)
    if (!location.isLeftBlocked) {
        std::cout << "[ActionController] 왼쪽으로 회피...\n";
        motorHandler->turnLeft();

        isEvading = false; // 회피 끝
        rvcController->resumeCleaning(); // 메인에 청소 재개 명령
    }
    else if (!location.isRightBlocked) {
        std::cout << "[ActionController] 오른쪽으로 회피...\n";
        motorHandler->turnRight();

        isEvading = false; // 회피 끝
        rvcController->resumeCleaning();
    }
    else {
        // 양쪽이 모두 막혀있다면 (UC-02 이동)
        std::cout << "[ActionController] 양쪽 모두가 막힘. 후진 중...\n";
        isMovingBackward = true;
        motorHandler->moveBackward();
        return;
    }
}
}

```

OOI 수정 - ActionController.cpp - (2)

```
// ===== 후진 로직 =====  
// 추가된 조건문 2-1 후진 중 센서 수신  
if (isEvading && isMovingBackward && !checkingRight){  
    checkingRight = true; // 추가된 로직, 오른쪽 장애물 확인을 위함.  
  
    location.isLeftBlocked = loc.leftSensorData; // 추가된 로직, 좌측 장애물 확인을 위함  
    location.isFrontBlocked = loc.frontSensorData; // 추가된 로직, 전방 장애물 확인을 위함  
  
    std::cout << "[ActionController] 오른쪽 장애물 유무를 위한 turn right...\n"; // 추가된 cout  
    motorHandler->stopMotor();  
    motorHandler->turnRight();  
    return;  
}
```

```
// 추가된 조건문 2-2 후진 중 오른쪽 확인 후 회피 로직 수행  
if (isEvading && checkingRight && isMovingBackward){  
    checkingRight = false;  
  
    location.isFrontBlocked=loc.leftSensorData; // 추가된 로직, 전방 장애물 확인을 위함  
    location.isRightBlocked=loc.frontSensorData; // 추가된 로직, 우측 장애물 확인을 위함  
  
    std::cout << "[ActionController] 오른쪽 장애물 유무를 확인 후 방향 복구 turn left...\n"; // 추가된 cout  
    motorHandler->stopMotor();  
    motorHandler->turnLeft(); // 방향 복구  
  
    // US-01과 같은 로직으로 회피  
    if (!location.isLeftBlocked) {  
        std::cout << "[ActionController] 왼쪽으로 회피...\n";  
        motorHandler->turnLeft();  
  
        isEvading = false; // 회피 끝  
        isMovingBackward = false; // 후진 끝  
        rvcController->resumeCleaning(); // 메인에 청소 재개 명령  
    }  
    else if (!location.isRightBlocked) {  
        std::cout << "[ActionController] 오른쪽으로 회피...\n";  
        motorHandler->turnRight();  
  
        isEvading = false; // 회피 끝  
        isMovingBackward = false; // 후진 끝  
        rvcController->resumeCleaning();  
    }  
    else {  
        // 양쪽이 모두 막혀있다면 (UC-02 이동)  
        std::cout << "[ActionController] 양쪽 모두가 막힘. 후진 중...\n";  
        isMovingBackward=true;  
        motorHandler->moveBackward();  
        return;  
    }  
}
```

OOI 수정 - 커버리지 및 ST 결과

LCOV - code coverage report

Current view: top level		Coverage		Total	Hit
Test: unit_test.cpp Coverage — 100%		Lines:	100.0 %	243	243
Test Date: 2026-06-04 16:22:38		Functions:	100.0 %	38	38

Directory	Line Coverage			Function Coverage		
	Rate	Total	Hit	Rate	Total	Hit
src/	100.0 %	243	243	100.0 %	38	38

RobotCleaner System Test Report

Total	PASS	FAIL	Success Rate
30	30	0	100.0%

TC ID	Test Case	Result
TC-POS-01	UC-00 -> UC-01 좌측 회피 전체 호출	PASS
TC-POS-02	UC-00 -> UC-03 고먼지 집중 청소	PASS
TC-POS-03	UC-00 -> UC-01 -> UC-02 모터 시퀀스(왼쪽)	PASS
TC-POS-04	UC-00 -> UC-01 -> UC-02 모터 시퀀스(왼쪽 막히고 오른쪽)	PASS
TC-POS-05	UC-03 -> UC-01 집중 청소 후 회피	PASS
TC-POS-06	2회 연속 전방 장애물 회피	PASS
TC-POS-07	UC-03 2회 연속 집중 청소	PASS
TC-POS-08	UC-00 -> UC-03 미진입 (저먼지 조건) 일반 청소 유지	PASS
TC-POS-09	UC-00 -> UC-03 -> UC-01 -> UC-02 전체 순차 수행	PASS
TC-POS-10	UC-00 -> UC-01 미진입 (좌우 장애물, 전방 개방) 직진 유지	PASS
TC-NEG-01	CLEANING 중 startCleaning 중복 호출	PASS
TC-NEG-02	CLEANING 중 resumeCleaning 호출	PASS
TC-NEG-03	유효하지 않은 SystemState 변경 시도	PASS
TC-NEG-04	AVOIDING 중 startCleaning 호출	PASS
TC-NEG-05	동일한 상태로 changeState 호출	PASS
TC-NEG-06	FORWARD 중 정지 단계가 생략된 급격한 방향 전환	PASS
TC-NEG-07	회피 중 전방 장애물 재발생	PASS
TC-NEG-08	FORWARD 상태에서 moveForward 중복 호출	PASS
TC-NEG-09	BACKWARD 상태에서 moveBackward 중복 호출	PASS
TC-NEG-10	LEFT 상태에서 turnLeft 중복 호출	PASS
TC-NEG-11	RIGHT 상태에서 turnRight 중복 호출	PASS
TC-NEG-12	STOPPED 상태에서 stopMotor 중복 호출	PASS
TC-NEG-13	음수 DustLevel 입력	PASS
TC-NEG-14	매우 큰 DustLevel 입력	PASS
TC-NEG-15	OFF 상태에서 deactivateCleaner 호출	PASS
TC-NEG-16	NORMAL 상태에서 activateCleaner 중복 호출	PASS
TC-NEG-17	BOOST 상태에서 boostPower 중복 호출	PASS
TC-NEG-18	NORMAL 상태에서 normalizePower 중복 호출	PASS
TC-NEG-19	OFF 상태에서 normalizePower 호출	PASS
TC-NEG-20	OFF 상태에서 boostPower 호출	PASS

AI Agent Coding 요구사항 변경 과정

1. Claude.md 수정
2. system.md 수정
3. Inception단계 .md 수정
4. OOA단계 .md 수정
5. OOD단계 .md 수정
6. OOI단계 수정

Claude.md 수정

요구사항 변경

- 오른쪽 센서 삭제 (동작은 똑같아야 함)
이걸 반영한 새로운 버전을 출시해야 함
controller logic 수정
- SRS부터 고친다 (OOA)
- SSD 고친다 (OOD)
- Code, UT, ST 수정
2명은 UP 수정 (Oracle, 정답지)
[추가, 삭제, 변경] 이렇게 3가지 마크업을 표시하며 수정한다
2명은 Vibe Coding 수정
- 똑같이 수정하면 됨
- 새로 만들지 않고 기존의 파일을 읽고 고치게 한다(프롬프팅을) 제출할 때 md 파일들을 따로 모아서 제출하기 → 따로 후처리해서 보기 쉽게 제출해도 좋음 md 파일에도 추가, 삭제, 변경 태그가 다 표시되어 있어야 함 ci연동 잘 하면서 해야지 나중에 편할 것임. push잘 하자

프롬프트

요구사항 변경 반영 프롬프트 (Vibe Coding)

변경 내용

- 오른쪽 센서 삭제 — 하드웨어 제거이지만 시스템의 외부 동작(장애물 회피, 방향 선택, 청소 재개)은 동일하게 유지되어야 한다.

작업 규칙

1. 기존 파일을 읽고 수정한다. 새 파일을 만들지 않는다.
2. 수정된 모든 `.md` 파일에는 변경 내용을 아래 태그로 표시한다.
 - `[추가]` — 새로 추가된 내용
 - `[삭제]` — 제거된 내용 (삭제된 원문을 취소선으로 남긴다)
 - `[변경]` — 기존 내용이 수정된 경우
3. 수정 순서를 지킨다: **SRS** → **SSD** → **Code** → **UT** → **ST**
4. 각 단계 수정 후 CI가 통과하는지 확인하며 진행한다. 단계마다 `commit & push`.

수정 대상 파일

단계	파일
SRS (OOA)	<code>arch/requirements/fr-nfr.md</code> , <code>arch/system.md</code> , <code>arch/usecases/</code> , <code>arch/analysis/domain/domain.md</code> , <code>arch/analysis/ssd/</code>
SSD (OOD)	<code>arch/design/sd/</code> , <code>arch/design/class/class-diagram.md</code>
Code	<code>src/</code>
UT	<code>tests/</code>
ST	<code>arch/system-tests/</code>

제출 기준

- 수정된 `.md` 파일 전체에 `[추가]` / `[삭제]` / `[변경]` 태그가 빠짐없이 표시되어 있어야 한다.
- CI (빌드 + 단위 테스트 + 시스템 테스트 + 정적 분석) 가 최종 `push` 시 모두 통과해야 한다.

결과

Claude.md 수정

이걸 토대로 기존 Claude.md 파일을 수정해줘

프롬프트

```
# 현재 작업: v2.0 - 아래 작업 지시를 최우선으로 따른다

## v2.0 작업 지시 (오른쪽 센서 삭제)

### 변경 요약
- 오른쪽 전용 센서를 삭제한다. 우측 장애물 감지는 전방 센서 회전 스캔으로 대체한다.
- 장애물 회피 방향은 랜덤 선택에서 **좌측 우선**으로 변경한다. (좌측이 막힌 경우에만 우측)

### 작업 규칙
1. 기존 파일을 읽고 수정한다. 새 파일을 만들지 않는다.
2. 수정된 모든 `.md` 파일에는 변경 내용을 아래 태그로 빠짐없이 표시한다.
   - `[추가]` - 새로 추가된 내용
   - `[삭제]` - 제거된 내용 (삭제된 원문을 ~취소선~으로 남긴다)
   - `[변경]` - 기존 내용이 수정된 경우

---
> 이 구분선 아래는 v1.0 기준 문서다. 위 v2.0 작업 지시와 충돌 시 **v2.0을 우선 적용**한다.

---

## 설계 방향
- 구동 모터, 청소 모터는 외부 액터로 취급한다
- 시스템 내부에는 모터를 제어하는 별도 컨트롤러 객체를 둔다 (예: MotorController, CleaningController)
- [추가] 오른쪽 전용 센서는 존재하지 않는다. 우측 장애물 감지는 전방 센서를 이용한 회전 스캔으로 대체한다.

## 규칙
- 모든 다이어그램은 Mermaid 형식으로 작성한다
- 모든 코드는 객체지향 원칙을 철저히 준수한다 (simulator 제외)
- simulator를 통하여 system-tests를 테스트한다
- [변경] ~장애물 회피 시 좌/우 방향은 랜덤으로 선택한다~ -> 장애물 회피 시 좌측을 우선으로 선택한다. 좌측이 막힌 경우에만 우측으로 전환한다.
- usecases, ssd, domain-model, ssd, class-diagram 은 반드시 통일성을 가질 것
- 구현 언어: C++

## [추가] 변경 이력

| 버전 | 변경 내용 | 날짜 |
|-----|-----|-----|
| v2.0 | 오른쪽 센서 삭제 - 전방 센서 회전 스캔으로 우측 감지 대체, 장애물 회피 방향 좌측 우선으로 변경 | 2025-05-29 |
```

결과

system.md 수정

> CLAUDE.md를 먼저 읽어라. v2.0 작업 지시를 최우선으로 따른다.
arch/system.md 를 읽고 v2.0 변경사항을 반영해서 수정해라.
수정된 부분에는 [추가] / [삭제] (삭제 원문은 ~~취소선~~) / [변경] 태그를 빠짐없이 표시한다.

프롬프트

```
## 4. 주요 요구사항 요약

| ID | 요구사항 |
|-----|-----|
| SR-01 | RVC는 가정의 표면을 자동으로 청소하고 닦는다 |
| SR-02 | 청소하면서 직진한다 |
| SR-03 | [변경] ~~센서가~~ 전방 센서가 장애물을 감지하면 청소를 멈추고, 좌/우로 방향을 틀어 청소하며 전진한다 |
| SR-04 | 앞/좌/우 모두 장애물이 있으면 후진 후 좌/우로 방향을 틀어 전진한다 |
| SR-05 | 먼지를 감지하면 잠시 청소 출력을 높인다 |

## 5. 시스템 외부 인터페이스

| 인터페이스 | 방향 | 설명 |
|-----|-----|-----|
| 장애물 센서 | 입력 | [변경] ~~전방/좌/우 장애물 감지 신호~~ 전방·좌측 장애물 감지 신호 (우측 전용 센서 없음, 우측 감지는 전방 센서 회전 스캔으로 대체) |
| 먼지 센서 | 입력 | 먼지 감지 신호 |
| 구동 모터 | 출력 | 전진/후진/정지/방향 전환 제어 |
| 청소 모터 | 출력 | 청소 출력 조절 (Normal / High) |

## 6. 핵심 동작 시나리오

1. **정상 청소**: RVC가 전원을 켜면 직진하며 청소 및 닦기를 수행한다.
2. **장애물 회피 (측면 여유 있음)**: 전방 장애물 감지 → 청소 정지 → [변경] ~~좌-또는-우로-회전~~ 좌측 우선으로 회전 (좌측이 막힌 경우에만 우측) → 청소 재개 및 전진.
3. **장애물 회피 (삼면 막힘)**: 전/좌/우 장애물 모두 감지 → 후진(이때 좌 또는 우에 막히지 않는 방향이 나올 때까지 후진한다) → [변경] ~~좌-또는-우로-회전~~ 좌측 우선으로 회전 (좌측이 막힌 경우에만 우측) → 전진.
4. **고출력 청소**: 먼지 감지 → 청소 출력 High 전환 → 일정 시간 경과 후 Normal 복귀.

## 7. 개발 단계

| 단계 | 산출물 |
|-----|-----|
| 요구사항 분석 | arch/system.md, arch/requirements/fr-nfr.md |
| 00A | arch/usecases/usecases.md, arch/usecases/UC-nnn.md, arch/analysis/ssd/ssd.md, arch/analysis/domain/domain.md |
| 00D | arch/design/sd/sd.md, arch/design/class/class-diagram.md |
| 00I | src/, simulator/, arch/system-tests/system-tests.md |
```

결과

Inception 단계 .md 수정

> CLAUDE.md를 먼저 읽어라. v2.0 작업 지시를 최우선으로 따른다.
 arch/system.md를 읽고 arch/requirements/fr-nfr.md를 v2.0 변경사항을 반영해서 수정해라.
 수정된 부분에는 [추가] / [삭제] (삭제 원문은 ~취소선~) / [변경] 태그를 빠짐없이 표시한다.

프롬프트

```

## 1. 기능 요구사항 (Functional Requirements)

### FR-01: 자동 청소 및 닦기
- **설명**: RVC는 전원이 켜지면 바닥을 자동으로 청소(흡입)하고 닦으면서 직진한다.
- **출처**: SR-01, SR-02
- **입력**: 전원 ON 이벤트
- **출력**: 구동 모터 전진, 청소 모터 Normal 출력 동작
- **조건**: -

---

### FR-02: 전방 장애물 감지 시 방향 전환
- **설명**: 전방에 장애물이 감지되고 좌 또는 우 중 하나라도 여유 공간이 있으면, 청소를 멈추고 여유 있는 방향으로 회전한 뒤 청소를 재개하며 직진한다.
- **출처**: SR-03
- **입력**: [변경] ~전방-장애물-감지-신호-(장애물-센서)~ 전방 장애물 감지 신호 (전방 센서); 우측 여유 공간 감지 (전방 센서 회전 스텝으로 대체, 우측 전용 센서 없음); 좌측 여유 공간 감지 (좌측 센서)
- **출력**: 청소 모터 정지 → 구동 모터 회전(좌 또는 우) → 청소 모터 재개 및 구동 모터 전진
- **조건**: 좌/우 중 적어도 한 방향에 장애물이 없어야 한다
- **선택 정책**: [변경] ~좌/우-모두-여유-있으면-랜덤으로-방향을-선택한다~ 좌측을 우선으로 선택한다. 좌측이 막힌 경우에만 우측으로 선택한다.

---

### FR-03: 삼면 장애물 감지 시 후진 후 방향 전환
- **설명**: 전방·좌·우 모두 장애물이 감지되면 후진하고, 좌 또는 우에 막히지 않는 방향이 나올 때까지 후진한 뒤 해당 방향으로 회전하여 직진한다.
- **출처**: SR-04
- **입력**: [변경] ~전방-좌-우-장애물-감지-신호-동시-수신~ 전방·좌측 장애물 감지 신호 동시 수신 (우측 장애물 감지는 전방 센서 회전 스텝으로 대체, 우측 전용 센서 없음)
- **출력**: 구동 모터 후진 → (좌 또는 우 여유 공간 감지 시) 구동 모터 회전 → 전진
- **조건**: 후진은 좌 또는 우 중 하나의 장애물이 해소될 때까지 계속한다
- **선택 정책**: [변경] ~후진-중-좌/우-모두-동시에-여유-공간이-확보되면-랜덤으로-방향을-선택한다~ 후진 중 좌/우 여유 공간이 확보되면 좌측을 우선으로 선택한다. 좌측이 막힌 경우에만 우측으로 선택한다.

---

### FR-04: 먼지 감지 시 청소 출력 강화
- **설명**: 먼지 센서가 먼지를 감지하면 청소 모터 출력을 High로 높이고, 일정 시간이 경과하면 Normal 출력으로 복귀한다.
- **출처**: SR-05
- **입력**: 먼지 감지 신호 (먼지 센서)
- **출력**: 청소 모터 출력 Normal → High → (일정 시간 후) Normal
- **조건**: 출력 강화 지속 시간은 별도 파라미터로 관리한다.
  
```

결과

Inception 단계 .md 수정

> CLAUDE.md를 먼저 읽어라. v2.0 작업 지시를 최우선으로 따른다. arch/requirements/fr-nfr.md를 읽고 arch/usecases/ 아래 파일을 모두 v2.0 변경사항을 반영해서 수정해라. 수정된 부분에는 [추가] / [삭제] (삭제 원문은 ~~취소선~~) / [변경] 태그를 빠짐없이 표시한다.

프롬프트

UC-001: 자동 청소 수행

기본 정보

항목	내용
Use Case ID	UC-001
이름	자동 청소 수행
액터	User (Primary), DriveMotor (Secondary), CleaningMotor (Seco
관련 요구사항	FR-01, SR-01, SR-02
종류	Base Use Case

사전 조건 (Preconditions)

- User가 RVC를 시작시킨다.

사후 조건 (Postconditions)

- RVC가 CleaningMotor Normal 출력으로 직진하여 청소 및 퇴기를 수행하고 있다.

기본 흐름 (Main Flow)

단계	액티비티	행위
1	User	RVC를 시작시킨다.
2	System	CleaningMotor를 Normal 출력으로 가동한다.
3	System	DriveMotor를 전진 방향으로 구동한다.
4	System	직진하여 바닥을 청소하고 다닌다.
5	System	단계 4를 계속 반복한다.

확장 포인트 (Extension Points)

확장 포인트	조건
EP-1	[변경] frontObstacleDetected AND (FR-01-유리유입) AND frontObst
[삭제] EP-2	
EP-3	dustDetected

UC-002: 측면 장애물 회피

기본 정보

항목	내용
Use Case ID	UC-002
이름	측면 장애물 회피
액터	ObstacleSensor (Pri
관련 요구사항	FR-02, SR-03
종류	Extension of UC-001

사전 조건 (Preconditions)

- UC-001이 실행 중이다.
- 바닥에 장애물이 감지되었다.
- [변경] 좌 또는 우측 하나 이상의 방향

사후 조건 (Postconditions)

- [변경] RVC가 장애물이 없는 방향으로

기본 흐름 (Main Flow)

단계	액티비티	행위
1	ObstacleSensor	전방 장애물
2	System	CleaningM
3	System	DriveMoto
4	ObstacleSensor	[변경] 좌측
5	System	[추가] Driv
6	ObstacleSensor	[추가] side
7	System	[추가] Driv
8	System	[변경] 여유
9	System	CleaningM
10	System	DriveMoto
11		UC-001 기

대안 흐름 (Alternative Flov

[변경] A1- 좌우 모두 여유 있을 때

- [변경] 단계 9에서 좌우 모두 여유 있

예외 흐름 (Exception Flow

E1: [변경] 회피 도중 좌우 모두 여유 공간 확보

- [변경] 단계 9에서 좌우 모두 동시에 확보되면, 랜덤으로 방향을 선택하여 회전한다. 단계 3에서 sideStatus(LEFT, clear)가 수신되면 단계 4-6을 건너뛰고 즉시 좌회전한다. 좌우 모두 동시에 확보되더라도 항상 좌측을 우선으로 선택한다.

UC-003: 삼면 장애물 후진 회피

기본 정보

항목	내용
Use Case ID	UC-003
이름	삼면 장애물 후진 회피
액터	ObstacleSensor (Primary), DriveMotor (Secondary), [추가] CleaningMotor (Secondary)
관련 요구사항	FR-03, SR-04
종류	[변경] Extension of UC-001 (EP-2) Extension of UC-002 (E1)

사전 조건 (Preconditions)

- UC-001 및 UC-002가 실행 중이다.
- [변경] 전방 좌우 방향 모두에 장애물이 감지되었다. UC-002 내에서 sideStatus(LEFT, blocked) 및 sideStatus(RIGHT, blocked) 가 모두 수신되어 좌우 모두 막힘이 확인되었다 (UC-002 E1). 우측 전용 센서 없음.

사후 조건 (Postconditions)

- RVC가 후진 중 확보된 여유 방향(좌 또는 우)으로 회전한 뒤 전진하고 있다.

기본 흐름 (Main Flow)

단계	액티비티	행위
1	(UC-002 E1)	[변경] ObstacleSensor-전방-좌측 장애물 감지 신호를 전달한다. 좌우 모두 막힘이 확인된 상태로 UC-002 E1에서 진입한다. (CleaningMotor는 UC-002 단계 2에서 이미 정지됨.)
2	System	DriveMotor를 후진 방향으로 구동한다.
3	ObstacleSensor	[변경] 좌측 장애물 유무 신호를 계속 전달한다. sideStatus(LEFT, status) — 좌측 센서가 좌측 장애물 유무 신호를 전달한다.
4	System	[추가] 좌측이 여전히 막힌 경우 DriveMotor에 rotateRight()를 명령한다 (전방 센서를 우측으로 회전).
5	ObstacleSensor	[추가] sideStatus(RIGHT, status) — 전방 센서(우측 방향)가 우측 장애물 유무 신호를 전달한다.
6	System	[추가] rotateRight()를 호출한 경우 DriveMotor에 rotateLeft()를 명령한다 (행면 복귀).
7	System	좌 또는 우에 여유 공간이 확보될 때까지 단계 2-6을 반복한다.
8	System	DriveMotor를 정지한다.
9	System	[변경] 여유 공간이 확보된 방향(좌 또는 우)으로 DriveMotor를 회전시킨다. 좌측에 여유 공간이 확보된 경우 좌회전한다. 좌측이 막힌 경우에만 우측으로 DriveMotor를 회전시킨다.
10	System	DriveMotor를 전진 방향으로 구동한다.
11	System	CleaningMotor를 Normal 출력으로 재가동한다.
12		UC-001 기본 흐름 단계 4로 복귀한다.

대안 흐름 (Alternative Flow)

[변경] A1- 후진 중 좌우 동시에 여유 공간 확보 A1: 후진 중 좌측 여유 공간 우선 선택

- [변경] 단계 6에서 좌우 모두 동시에 확보되면, 랜덤으로 방향을 선택하여 회전한다. 단계 3에서 sideStatus(LEFT, clear)가 수신되면 단계 4-6을 건너뛰고 즉시 좌회전한다. 좌우 모두 동시에 확보되더라도 항상 좌측을 우선으로 선택한다.

결과

OOA 단계 .md 수정

> CLAUDE.md를 먼저 읽어라. v2.0 작업 지시를 최우선으로 따르라.
arch/usecases/ 아래 파일을 읽고 arch/analysis/domain/domain.md와 arch/analysis/ssd/ 아래 파일을 모두 v2.0 변경사항을 반영해서 수정해라.
수정된 부분에는 [추가] / [삭제] (삭제 원문은 ~취소선~) / [변경] 태그를 빠짐없이 표시한다.

최초 프롬프트

```
> allSidesBlocked(), sideStatus(), frontObstacleDetected() 가 나뉘어 있으면 외부 센서에서 이걸 구별해서 호출한다는 뜻인가?
• 맞습니다. SSD에서 외부 액터가 특정 오퍼레이션을 호출한다는 것은, 그 액터가 그 조건을 스스로 판단해서 해당 신호를 보낼 수 있다는 의미입니다.
---
이것이 v2.0에서 생기는 설계 모순입니다.
```

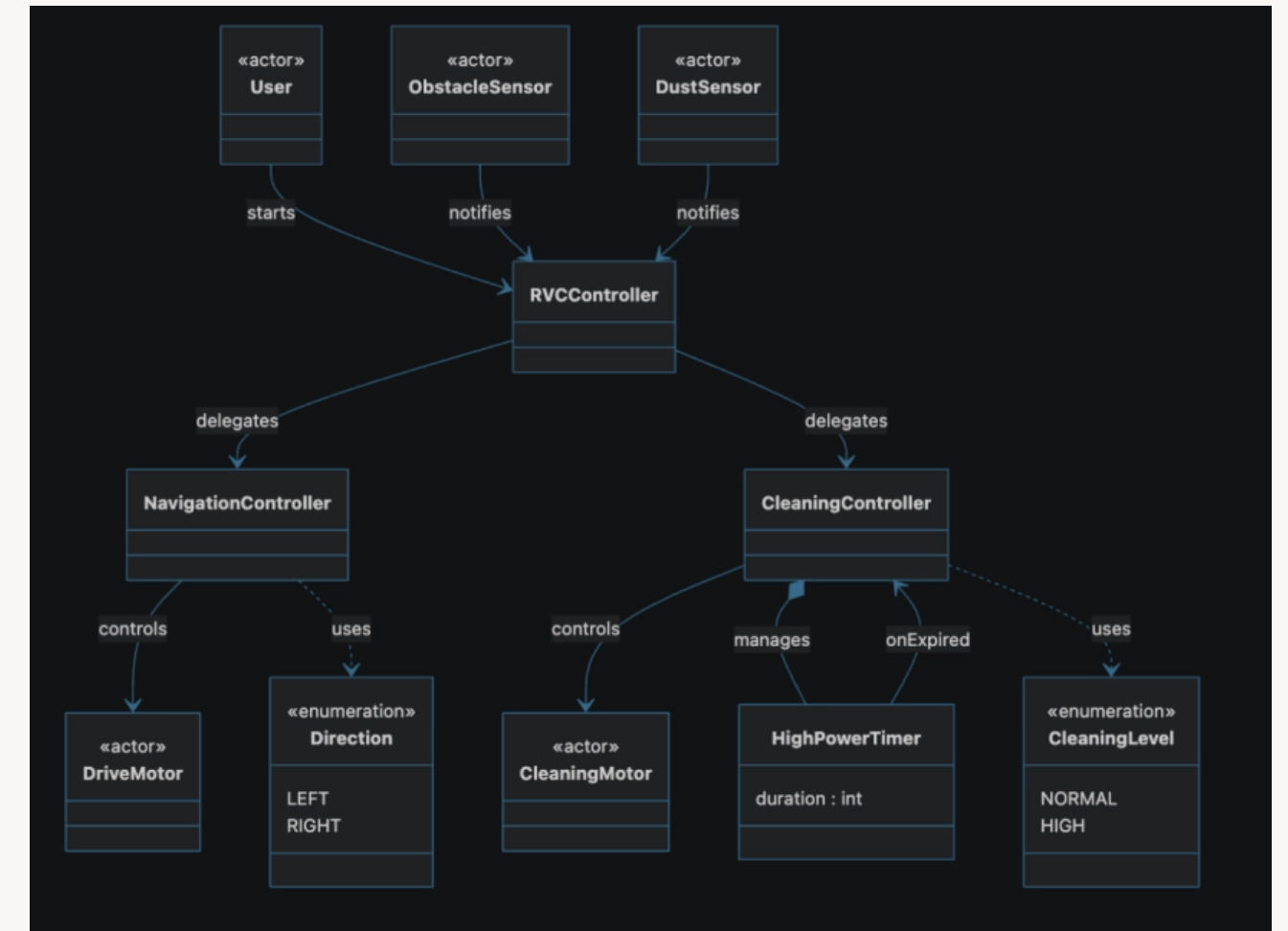
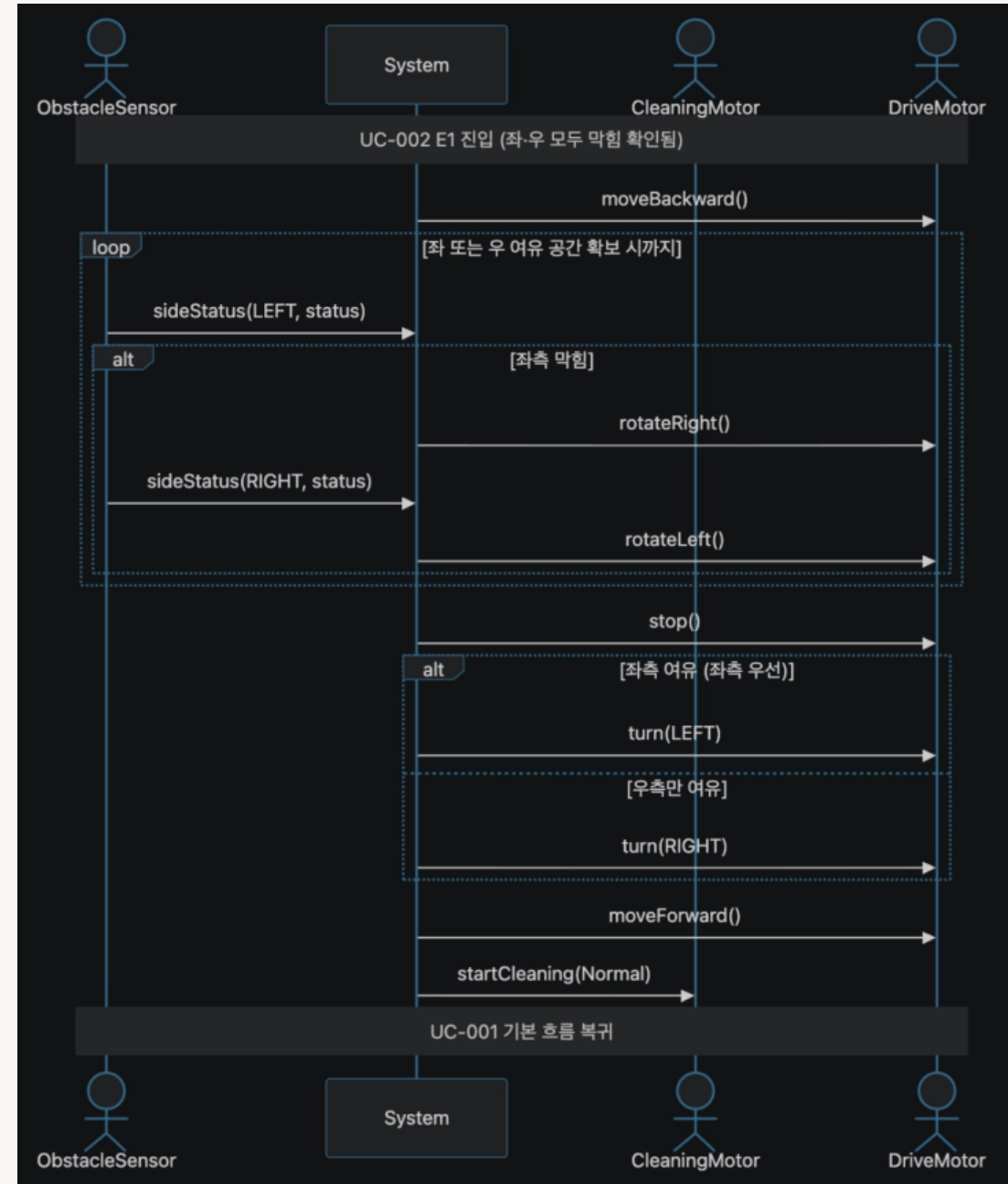
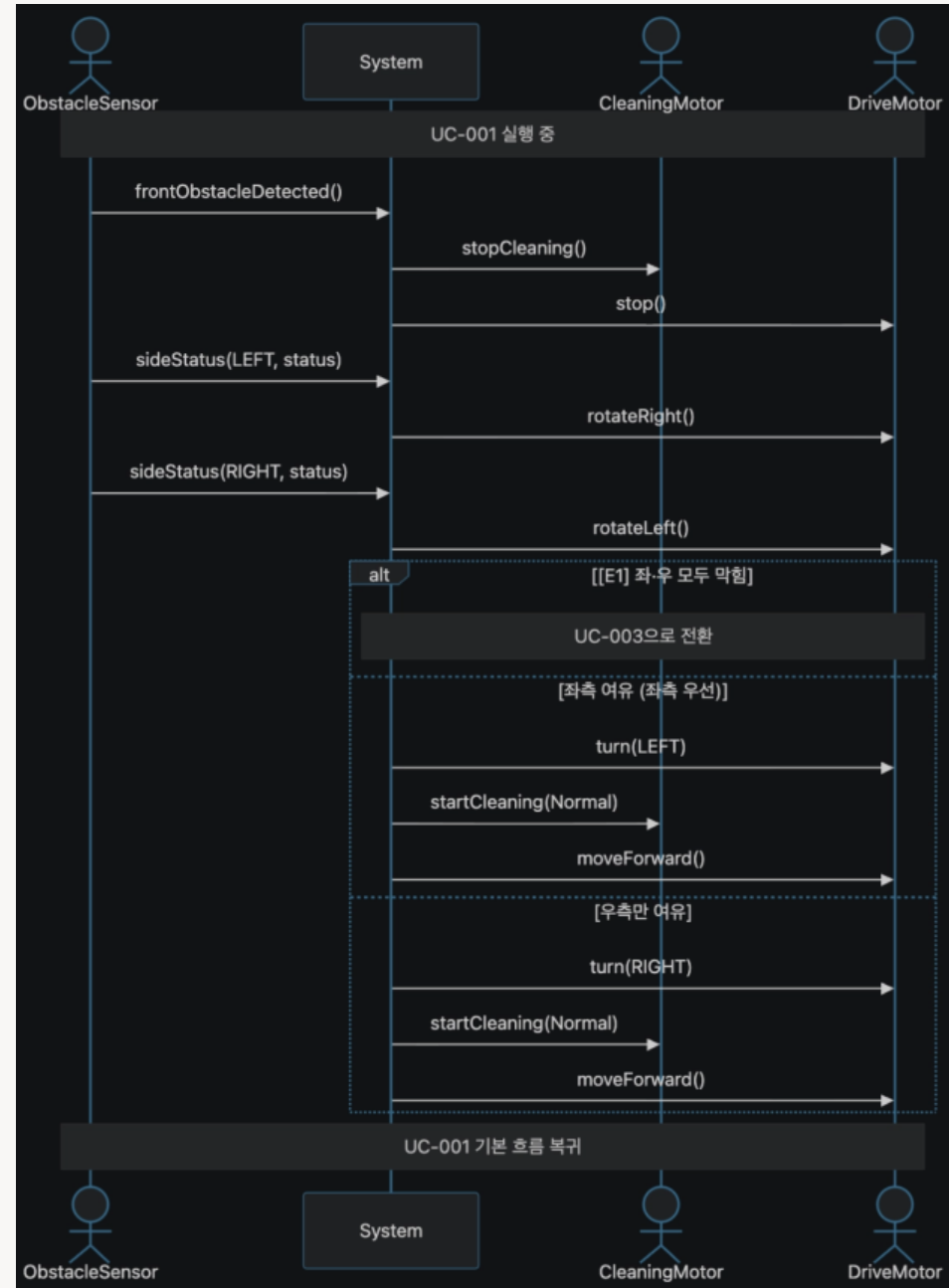
오퍼레이션	v1.0	v2.0 문제
frontObstacleDetected()	ObstacleSensor → System	유효 (전방 센서 존재)
sideStatus(available)	ObstacleSensor → System	부분 유효 (좌측만 감지 가능)
allSidesBlocked()	ObstacleSensor → System	모순 - 우측은 센서가 모름

```
allSidesBlocked()는 더 이상 ObstacleSensor가 판단해서 보낼 수 없습니다. 우측이 막혔는지는 시스템이 내부 스캔으로 알아내는 것이기 때문입니다.
---
올바른 v2.0 설계 방향
UC-003은 ObstacleSensor가 직접 트리거하는 것이 아니라, UC-002 예외 흐름에서 시스템이 내부 판단으로 진입하는 구조가 맞습니다.
ObstacleSensor → frontObstacleDetected() ← UC-002 진입
└─ 시스템이 좌측 확인 (sideStatus)
└─ 시스템이 우측 내부 스캔
    └─ 양쪽 모두 막힘 판단 → UC-003 진입 ← 시스템 내부 전환
즉 SSD-003은 OS->>SYS: allSidesBlocked()로 시작하면 안 되고, SSD-002의 예외 분기에서 이어지는 구조로 재설계해야 합니다.
수정 할까요? SSD-002와 SSD-003을 다시 설계하면 됩니다.
```

```
> 근데 frontObstacleDetected로 UC002에 진입하더라도 우측 내부를 스캔하려면 또 센서값을 받아야할텐데? 시스템이 자체적으로 우측 내부를 스캔 못하잖아
• 정확한 지적입니다.
"전방 센서 회전 스캔"이라는 말에서 회전은 DriveMotor로 로봇을 돌리는 것이고, 스캔은 여전히 ObstacleSensor가 읽어서 시스템에 전달하는 것입니다. 시스템이 하드웨어 센서를 직접 읽을 수는 없으니깐요.
---
결국 v2.0에서 실제 일어나는 일은 이렇습니다:
OS->>SYS: frontObstacleDetected()
SYS->>DM: stop()
SYS->>DM: rotateToFaceLeft() ← 좌측을 향해 회전
OS->>SYS: sideStatus(leftStatus) ← 좌측 ObstacleSensor 읽기
SYS->>DM: rotateToFaceRight() ← 우측을 향해 회전
OS->>SYS: sideStatus(rightStatus) ← 전방 센서가 우측 향해 읽기
v1.0과 다른 점은 우측 전용 센서가 없다는 것뿐이고, 여전히 sideStatus()는 ObstacleSensor → System 외부 호출입니다.
---
sideStatus(available: Direction[])
* Sautéed for 47s
> 2개의 별도 호출로 분리하자. 그리고 rotateToFaceLeft 가 왜 일어나 안 일어나지. 우리는 좌측/전방 센서는 존재해. 우측 센서만 없어진 거니까 우측으로만 돌아서 확인하면 되지.
• 맞습니다, 제가 틀렸습니다. 정리하면:
- 좌측 센서 있음 → sideStatus(LEFT, ...) 는 회전 없이 바로 수신
- 전방 센서 있음 → 우측 확인할 때만 전방 센서를 우측으로 회전
- 우측 센서 없음 → 전방 센서를 우측으로 돌려서 sideStatus(RIGHT, ...) 수신
---
따라서 올바른 흐름은:
OS->>SYS: frontObstacleDetected()
SYS->>DM: stopCleaning()
SYS->>DM: stop()
OS->>SYS: sideStatus(LEFT, status) ← 좌측 센서 직접 읽기 (회전 불필요)
SYS->>DM: rotateRight() ← 전방 센서를 우측으로 회전
OS->>SYS: sideStatus(RIGHT, status) ← 전방 센서가 우측 읽기
그리고 이 구조에서 allSidesBlocked()는 더 이상 ObstacleSensor가 보내는 외부 신호가 아니라, 위 두 sideStatus() 결과를 받은 시스템이 내부에서 판단하는 조건이 됩니다. 즉 UC-003은 UC-002 예외 흐름에서 시스템이 내부 판단으로 진입합니다.
---
이 방향으로 SSD-002, SSD-003, domain.md, usecases.md, UC-002.md, UC-003.md 전부 수정할까요?
* Churned for 36s
> 응 전부 수정해
```

추가 프롬프트

OOA 단계 .md 수정



결과

OOD단계 .md 수정

```
> CLAUDE.md를 먼저 읽어라. v2.0 작업 지시를 최우선으로 따른다.  
arch/analysis/ 아래 파일을 읽고 arch/design/sd/ 아래 파일과 arch/design/class/class-diagram.md 를 v2.0 변경사항을 반영해서 수정해라.  
수정된 부분에는 [추가] / [삭제] (삭제 원문은 취소선) / [변경] 태그를 빠짐없이 표시한다.
```

프롬프트

OOD단계 .md 수정



결과

OOI 단계 수정

```
> CLAUDE.md를 먼저 읽어라. v2.0 작업 지시를 최우선으로 따른다.  
arch/design/ 아래 파일을 읽고 src/ 아래 파일을 v2.0 변경사항을 반영해서 수정해라.  
빌드가 통과하는지 확인한다.
```

프롬프트

OOI 단계 수정

LCOV - code coverage report

Current view: top level
 Test: RVC Unit Test Coverage
 Test Date: 2026-05-30 18:11:50

Coverage Total Hit
 Lines: 100.0 % 139 139
 Functions: 93.9 % 33 31

Directory	Line Coverage			Function Coverage		
	Rate	Total	Hit	Rate	Total	Hit
src/	100.0 %	139	139	93.9 %	33	31

결과 요약:

파일	라인	함수
CleaningController.cpp	100% (24/24)	100% (6/6)
HighPowerTimer.cpp	100% (36/36)	100% (9/9)
NavigationController.cpp	100% (28/28)	100% (8/8)
RVCController.cpp	100% (58/58)	100% (6/6)
ITimerExpiredCallback.hpp	100% (1/1)	50% (2/4)
합계	100% (139/139)	93.9% (31/33)

ITimerExpiredCallback.hpp 함수 커버리지 50%는 순수 가상 인터페이스의 컴파일러가 생성하는 삭제 소멸자(DtEv) 변형이 테스트에서 delete 경로로 직접 호출되지 않아서 발생하는 정상적인 false positive입니다. 실질적인 비즈니스 로직은 라인-함수 모두 100% 커버됩니다.

RVC 시스템 테스트 리포트

Positive [UC 정상 동작 검증]

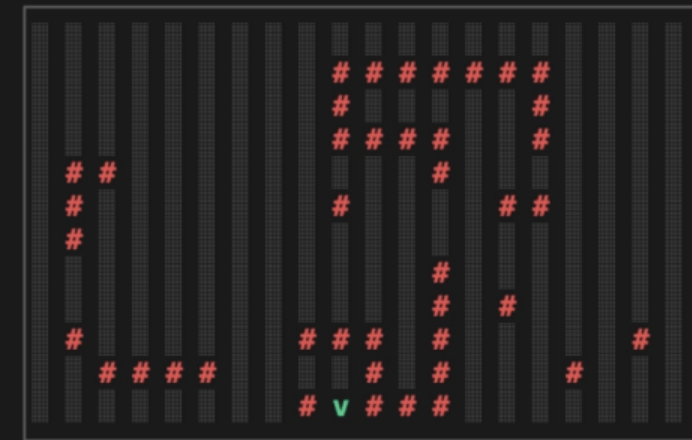
- ✓ ST-P01 자동 청소 시작 - NORMAL + forward
- ✓ ST-P02 전방 장애물 감지 - stopCleaning + stop
- ✓ ST-P03 장애물 회피 우측 - 완전 시퀀스
- ✓ ST-P04 장애물 회피 좌측 - 완전 시퀀스
- ✓ ST-P05 장애물 좌·우 모두 여유 - LEFT 우선 선택 (v2.0)
- ✓ ST-P06 삼면 장애물 후 우측 전진 - 순서 포함
- ✓ ST-P07 삼면 장애물 후 좌측 전진 - 순서 포함
- ✓ ST-P08 삼면 장애물 - 좌우 막힘 반복 후 방향 확보
- ✓ ST-P09 먼지 감지 - HIGH 전환
- ✓ ST-P10 먼지 감지 후 타이머 만료 - NORMAL 자동 복귀

Negative [상태·순서·격리 위반 검증]

- ✓ ST-N01 IDLE에서 sideStatus(LEFT) - turn 미호출
- ✓ ST-N02 IDLE에서 sideStatus(LEFT,false) - 모든 모터 미호출
- ✓ ST-N03 CLEANING에서 sideStatus(LEFT) - turn 미호출
- ✓ ST-N04 CLEANING에서 sideStatus(LEFT,false) - 모터 추가 없음
- ✓ ST-N05 REVERSING에서 좌우 막힘 - stop/turn/forward 없음
- ✓ ST-N06 REVERSING에서 좌우 막힘 3회 - forward 0회
- ✓ ST-N07 dustDetected 2회 - startCleaning(HIGH) 1회만
- ✓ ST-N08 dustDetected - DriveMotor 호출 전혀 없음
- ✓ ST-N09 frontObstacleDetected - startCleaning 미호출
- ✓ ST-N10 E1 진입 - startCleaning 미호출 (allSidesBlocked 제거)
- ✓ ST-N11 좌·우 모두 여유 - turn 1회, LEFT 선택
- ✓ ST-N12 UC-002 순서: stopCleaning → stop
- ✓ ST-N13 UC-002 순서: turn → startCleaning(NORMAL)
- ✓ ST-N14 UC-003 순서: forward → startCleaning(NORMAL)
- ✓ ST-N15 start() 후 DriveMotor.stop 미호출
- ✓ ST-N16 start() 후 moveForward 정확히 1회
- ✓ ST-N17 E1 진입 - stopCleaning 1회, moveBackward 1회
- ✓ ST-N18 REVERSING → frontObstacleDetected → sideStatus 처리
- ✓ ST-N19 장애물 2회 연속 - turn 총 2회
- ✓ ST-N20 고출력 중 장애물 - stop 1회, 마지막 호출 NORMAL

결과 전체 : 30 통과 : 30 실패 : 0

RVC 인터랙티브 시뮬레이터 - 시뮬레이션 모드



상태
 상태 : DONE 방향 : SOUTH 위치 : (11,9) 청소출력 : NORMAL
 청소 : 198/198 (100%)

P 일시정지/재개 E 편집모드 Q 종료

이벤트 로그
 [Drive] rotateRight()
 [Drive] rotateLeft()
 [Drive] turn(RIGHT)
 [Clean] startCleaning(NORMAL)
 [Drive] moveForward()
 청소 완료!

결과

Chapter 1.

OOI 단계 수정

이제 문제는 없는 거 같네. src의 수정사안 (unit test, system test제외)을 따로 정리하는 .md 파일을 src/code.md 경로에 추가해서 아까와 동일한 규칙으로 삭제, 변경, 추가 인덱스를 통해 기록해줘

프롬프트

```
src/ 코드 변경 이력 - v2.0
변경 태그: [추가] / [삭제] [취소선] / [변경] 단위 테스트( tests/ ) 시스템 테스트( system-tests/ ) 파일은 별도 관리하여 이 문서에 포함하지 않는다. 변경이 없는 파일은 목록 하단 변경 없음 줄에 나열한다.

IDriveMotor.hpp
구분 항목 내용
[추가] rotateRight() = 0 전방 센서를 우측으로 회전시키기 위한 순수 가상 메서드 추가
[추가] rotateLeft() = 0 우측 스캔 후 정면으로 복귀하기 위한 순수 가상 메서드 추가

// [추가]
virtual void rotateRight() = 0;
virtual void rotateLeft() = 0;

NavigationController.hpp
구분 항목 내용
[삭제] #include <vector> 시그니처 변경으로 vector 불필요
[변경] turn(const std::vector<Direction>&) -> turn(Direction direction) 단일 방향 인자로 단순화
[추가] rotateRight() IDriveMotor::rotateRight() 위임 메서드
[추가] rotateLeft() IDriveMotor::rotateLeft() 위임 메서드

// [변경] 이전: void turn(const std::vector<Direction>& directions);
void turn(Direction direction); // [변경]
void rotateRight(); // [추가]
void rotateLeft(); // [추가]

NavigationController.cpp
구분 항목 내용
[삭제] #include <vector>, #include <random> 랜덤 선택 로직 제거로 헤더 불필요
[변경] turn() 구현 벡터를 받아 랜덤으로 방향 선택 -> 단일 방향을 motor_.turn(direction) 에 그대로 위임
[추가] rotateRight() 구현 motor_.rotateRight() 위임
[추가] rotateLeft() 구현 motor_.rotateLeft() 위임

// [변경] 이전: 벡터에서 랜덤 선택하는 로직
void NavigationController::turn(Direction direction) {
    motor_.turn(direction);
}

// [추가]
void NavigationController::rotateRight() { motor_.rotateRight(); }
void NavigationController::rotateLeft() { motor_.rotateLeft(); }
```

```
RVCController.hpp
구분 항목 내용
[삭제] #include <vector> sideStatus() 시그니처 변경으로 불필요
[삭제] void allSidesBlocked() 전방 좌우 동시 역침 방향을 sideStatus() 내부 상태만으로 대체
[변경] sideStatus(const std::vector<Direction>&) -> sideStatus(Direction direction, bool clear) 방향별 상태를 순차 호출로 분리
[변경] State enum IDLE, CLEANING, REVERSING 3개 -> 6개 상태로 확장
[추가] State::STOPPED frontObstacleDetected() 수신 후 좌측 스캔 대기 상태
[추가] State::STOPPED_CHECKING_RIGHT sideStatus(LEFT) 수신 후 rotateRight() 호출, 우측 스캔 대기 상태
[추가] State::REVERSING_CHECKING_RIGHT 후진 중 sideStatus(LEFT) 역침 확인 후 rotateRight(), 우측 스캔 대기 상태
[추가] bool leftClear() STOPPED -> STOPPED_CHECKING_RIGHT 전이 시 좌측 결과 임시 보던

// [변경] 이전: void allSidesBlocked(); - [삭제]
// [변경] 이전: void sideStatus(const std::vector<Direction>& clearDirections);
void sideStatus(Direction direction, bool clear); // [변경]

enum class State {
    IDLE,
    CLEANING,
    STOPPED, // [추가]
    STOPPED_CHECKING_RIGHT, // [추가]
    REVERSING,
    REVERSING_CHECKING_RIGHT // [추가]
};

bool leftClear_; // [추가]

RVCController.cpp
구분 항목 내용
[삭제] allSidesBlocked() 구현 삭제, 좌우 모두 역침은 STOPPED_CHECKING_RIGHT 상태에서 내부 판명
[변경] sideStatus() 시그니처 const std::vector<Direction>& -> Direction direction, bool clear
[변경] sideStatus() 내부 로직 방향 예측 기반 단순화 -> 4개 상태에 걸친 순차 상태만으로 판명 재작성
[추가] STOPPED 분기 leftClear_ 재인 후 rotateRight() 호출, STOPPED_CHECKING_RIGHT 전이
[추가] STOPPED_CHECKING_RIGHT 분기 rotateLeft() 판명 복귀 후 LEFT 우선 방향 필요, 양역 역침 시 moveBackward() -> REVERSING
[추가] REVERSING 분기 좌측 역침 시 즉시 stop+turn(LEFT)+forward, 역침 시 rotateRight() -> REVERSING_CHECKING_RIGHT
[추가] REVERSING_CHECKING_RIGHT 분기 rotateLeft() 복귀 후 우측 역침 시 stop+turn(RIGHT)+forward, 역침 시 REVERSING 복귀(계속 후진)

상태 전이 요약

frontObstacleDetected()
CLEANING -> STOPPED

sideStatus(LEFT, clear) [STOPPED]
- rotateLeft()
- STOPPED_CHECKING_RIGHT (leftClear_ 지양)

sideStatus(RIGHT, clear) [STOPPED_CHECKING_RIGHT]
- rotateLeft()
- leftClear_==true : turn(LEFT) -> CLEANING
- clear==true : turn(RIGHT) -> CLEANING
- 둘 다 false : moveBackward() -> REVERSING - E1 전이

sideStatus(LEFT, clear) [REVERSING]
- clear==true : stop -> turn(LEFT) -> CLEANING
- clear==false : rotateRight() -> REVERSING_CHECKING_RIGHT

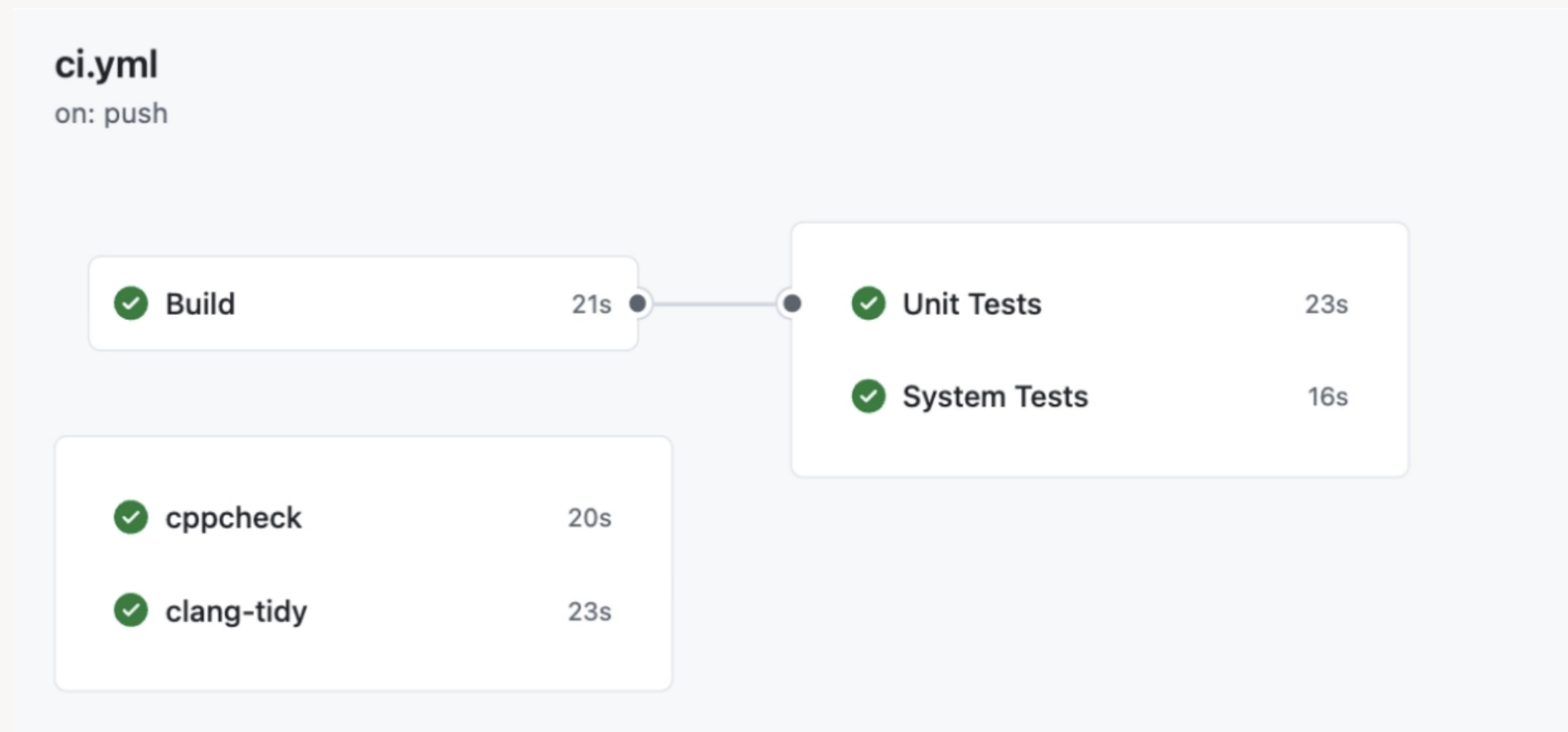
sideStatus(RIGHT, clear) [REVERSING_CHECKING_RIGHT]
- rotateLeft()
- clear==true : stop -> turn(RIGHT) -> CLEANING
- clear==false : REVERSING (계속 후진)
```

결과

OOI 단계 수정

> 일단 이대로 git에 올리자. 올리기 전에 각 static analysis 문제 없는지 먼저 파악하고 올려

프롬프트



결과

Chapter 2.

결과 비교 및 최종 수정

UP팀 시스템 테스트 시나리오 명세

개요

- 오른쪽 센서 삭제 후 변경된 요구사항 반영
- 센서 입력은 SensorData {frontSensorData, leftSensorData} 2개 필드
- 오른쪽 장애물 확인은 우회전 후 전방 센서값으로 대체
- 따라서 장애물 감지 시 2번의 push로 판단

Positive 시나리오

TC-POS-01: UC-00 → UC-01 좌측 회피 전체 흐름

항목	내용
사전 조건	청소 중 (CLEANING, FORWARD)
입력 1	{frontSensorData=false, leftSensorData=true} (전방 장애물, 좌측 돌림)
입력 2	{frontSensorData=true, leftSensorData=true} (우회전 후 전방 막힘 → 우측 돌림)
기대 동작	turnLeft → resumeCleaning
기대 상태	SystemState=CLEANING, Motor=FORWARD

TC-POS-02: UC-00 → UC-03 고먼지 집중 청소

항목	내용
사전 조건	청소 중 (CLEANING)
입력	dustLevel=80.0f
기대 동작	boostPower → normalizePower → resumeCleaning
기대 상태	SystemState=CLEANING, Cleaner=NORMAL

TC-POS-03: UC-00 → UC-01 → UC-02 후진 후 좌측 탈출

항목	내용
사전 조건	청소 중 (CLEANING)
입력 1	{true, true} (전방+좌측 막힘)
입력 2	{true, true} (우회전 후 전방도 막힘 → 양측 막힘 → 후진)
입력 3	{false, false} (후진 중 좌측 돌림)
입력 4	{false, false} (우회전 후 전방도 돌림 → 좌측 탈출)
기대 동작	moveBackward → turnLeft → resumeCleaning
기대 상태	SystemState=CLEANING, Motor=FORWARD

TC-POS-04: UC-00 → UC-01 → UC-02 후진 후 우측 탈출

항목	내용
사전 조건	청소 중 (CLEANING)
입력 1	{true, true} (전방+좌측 막힘)
입력 2	{true, false} (우회전 후 전방 돌림 → 우측 돌림 → 바로 우측 탈출)
기대 동작	turnRight → resumeCleaning
기대 상태	SystemState=CLEANING, Motor=FORWARD

TC-POS-05: UC-03 → UC-01 집중 청소 후 회피

항목	내용
사전 조건	청소 중 (CLEANING)
입력 1	dustLevel=80.0f
입력 2	{true, true} (집중 청소 복귀 후 전방 장애물)
입력 3	{true, false} (우회전 후 우측 돌림 → 우측 탈출)
기대 동작	boostPower → resumeCleaning → turnRight → resumeCleaning
기대 상태	SystemState=CLEANING, Motor=FORWARD

TC-POS-06: 2회 연속 전방 장애물 회피

항목	내용
사전 조건	청소 중 (CLEANING)
1회차 입력 1	{false, true} (전방 장애물, 좌측 돌림)
1회차 입력 2	{true, true} (우측 막힘 → 양측 막힘 아님, 좌측으로 탈출)
2회차 입력 1	{true, true} (전방+좌측 막힘)
2회차 입력 2	{true, false} (우측 돌림 → 우측 탈출)
기대 상태	SystemState=CLEANING, Motor=FORWARD

TC-POS-07: UC-03 2회 연속 집중 청소

항목	내용
사전 조건	청소 중 (CLEANING)
입력 1	dustLevel=75.0f
입력 2	dustLevel=75.0f
기대 상태 (1회)	SystemState=CLEANING, Cleaner=NORMAL
기대 상태 (2회)	SystemState=CLEANING, Cleaner=NORMAL

TC-POS-08: UC-03 미진입 (저먼지) 일반 청소 유지

항목	내용
사전 조건	청소 중 (CLEANING)
입력	dustLevel=30.0f
기대 동작	아무 동작 없음
기대 상태	SystemState=CLEANING, Motor=FORWARD, Cleaner=NORMAL

TC-POS-09: UC-00 → UC-03 → UC-01 → UC-02 전체 순차

항목	내용
사전 조건	청소 중 (CLEANING)
입력 1	dustLevel=75.0f
입력 2	{true, true} (전방+좌측 막힘)
입력 3	{true, true} (우측도 막힘 → 후진)
입력 4	{false, false} (후진 중 좌측 돌림)
입력 5	{false, false} (우회전 후 돌림 → 좌측 탈출)
기대 상태	SystemState=CLEANING, Motor=FORWARD

TC-POS-10: UC-01 미진입 (전방 개방) 직진 유지

항목	내용
사전 조건	청소 중 (CLEANING)
입력	{true, false} (좌우 장애물, 전방 개방)
기대 동작	회피 없이 직진 유지
기대 상태	SystemState=CLEANING, Motor=FORWARD, Cleaner=NORMAL

Negative 시나리오

TC-NEG-01: CLEANING 중 startCleaning 중복 호출

항목	내용
입력	startCleaning() × 2
기대 동작	두 번째 호출 무시
기대 상태	SystemState=CLEANING, Motor=FORWARD

TC-NEG-02: CLEANING 중 resumeCleaning 호출

항목	내용
입력	startCleaning() → resumeCleaning()
기대 동작	상태 변화 없음
기대 상태	SystemState=CLEANING, Motor=FORWARD

TC-NEG-03: 유효하지 않은 SystemState 변경 시도

항목	내용
입력	changeState(999)
기대 동작	요청 거부
기대 상태	SystemState=CLEANING

TC-NEG-04: AVOIDING 중 startCleaning 호출

항목	내용
입력 1	{true, true} → {true, true} (후진 상태 진입)
입력 2	startCleaning()
기대 동작	새 청소 명령 무시
기대 상태	SystemState=AVOIDING, Motor=BACKWARD

TC-NEG-05: 동일한 상태로 changeState 호출

항목	내용
입력	changeState(CLEANING) (이미 CLEANING 상태)
기대 동작	상태 전환 로직 수행 안 함
기대 상태	SystemState=CLEANING

TC-NEG-06: FORWARD 중 정지 단계 생략 급격한 방향 전환

항목	내용
입력	turnLeft() → turnRight() (FORWARD 상태에서)
기대 동작	비정상 전환 거부
기대 상태	Motor=FORWARD

TC-NEG-07: 회피 중 전방 장애물 재발생

항목	내용
입력 1	{true, true} → {true, true} (후진 진입)
입력 2	{true, true} → {true, true} (후진 중 재감지)
기대 동작	기존 회피 동작 유지
기대 상태	SystemState=AVOIDING, Motor=BACKWARD, Cleaner=OFF

TC-NEG-08 ~ TC-NEG-12: Motor 중복 명령

TC	상황	기대 동작
NEG-08	FORWARD → moveForward()	중복 무시, FORWARD 유지
NEG-09	BACKWARD → moveBackward()	중복 무시, BACKWARD 유지
NEG-10	LEFT → turnLeft()	중복 무시, LEFT 유지
NEG-11	RIGHT → turnRight()	중복 무시, RIGHT 유지
NEG-12	STOPPED → stopMotor()	중복 무시, STOPPED 유지

TC-NEG-13: 음수 DustLevel 입력

항목	내용
입력	dustLevel=-1.0f
기대 동작	유효성 검사 후 명령 차단
기대 상태	SystemState=CLEANING, Cleaner=NORMAL

TC-NEG-14: 매우 큰 DustLevel 입력

항목	내용
입력	dustLevel=FLT_MAX
기대 동작	최대값으로 처리, 크래시 없음
기대 상태	SystemState=CLEANING, Cleaner=NORMAL

TC-NEG-15 ~ TC-NEG-20: Cleaner 중복/비정상 명령

TC	상황	기대 동작
NEG-15	OFF → deactivateCleaner()	중복 무시, OFF 유지
NEG-16	NORMAL → activateCleaner()	중복 무시, NORMAL 유지
NEG-17	BOOST → boostPower()	중복 무시, BOOST 유지
NEG-18	NORMAL → normalizePower()	중복 무시, NORMAL 유지
NEG-19	OFF → normalizePower()	비활성 상태 전역 조절 차단, OFF 유지
NEG-20	OFF → boostPower()	비활성 상태 전역 조절 차단, OFF 유지

결과 비교 및 최종 수정

```
> 아래는 UP팀이 작성한 시스템 테스트 시나리오 명세야.  
system-test/ST_scenarios.md  
  
이 레포의 코드를 읽고 다음 두 가지를 해줘.  
  
1. 위 시나리오를 이 레포 인터페이스에 맞게 테스트 코드로 작성해줘.  
   기존 시스템 테스트 파일은 수정하지 말고 test_v2.cpp 로 새로 만들어줘.  
  
2. test_v2.cpp를 실행한 후 PASS/FAIL 결과를 바탕으로  
   UP팀 코드와 이 레포 코드의 차이점을 분석한 md 파일을 system-test/ 아래에 만들어줘.  
   - 어떤 시나리오가 실패했는지  
   - 왜 실패했는지 (로직 차이, 인터페이스 차이 등)  
   - 어떻게 수정하면 되는지
```

프롬프트

Chapter 2.

결과 비교 및 최종 수정

```
// POSITIVE 시나리오 (TC-POS-01 ~ TC-POS-10)
//
// TC-POS-01: UC-00 ~ UC-01 좌측 회피 전체 흐름
// UP팀 push1={false,true} 전방장애물, 좌측돌림 / push2={true,true} 무회전후 우측막힘 -> turnLeft
static bool POS01(std::string& failMsg) {
    RVCSimulator sim;
    sim.start();
    sim.frontObstacleDetected();
    sim.sideStatus(D::LEFT, true); // 좌측 돌림
    sim.sideStatus(D::RIGHT, false); // 우측 막힘 -> leftClear=true -> turn(LEFT)

    CHECK_E0(sim.cleaning().stopCount, 1, "stopCleaning 1회");
    CHECK_E0((int)sim.drive().turnCalls.size(), 1, "turn 1회");
    CHECK(sim.drive().turnCalls[0] == D::LEFT, "turnLeft");
    CHECK(sim.cleaning().startCalls.size() >= 2, "startCleaning 재호출");
    CHECK(sim.cleaning().startCalls.back() == CleaningLevel::NORMAL, "재개 레벨 NORMAL");
    CHECK(sim.drive().forwardCount >= 2, "moveForward 재호출");
    return true;
}

// TC-POS-02: UC-00 ~ UC-03 고먼지 집중 청소
// UP팀 dustLevel=80 -> boostPower -> normalizePower -> resumeCleaning
static bool POS02(std::string& failMsg) {
    RVCSimulator sim(false, 80); // 80ms 타이머 = boostPower 지속시간
    sim.start();
    sim.resetRecording();
    sim.dustDetected();
    std::this_thread::sleep_for(std::chrono::milliseconds(200)); // 타이머 만료 대기

    CHECK(sim.cleaning().startCalls.size() >= 2, "HIGH + NORMAL 최소 2회");
    CHECK(sim.cleaning().startCalls[0] == CleaningLevel::HIGH, "boostPower=HIGH");
    CHECK(sim.cleaning().startCalls.back() == CleaningLevel::NORMAL, "normalizePower=NORMAL 자동복귀");
    CHECK_E0(sim.drive().forwardCount, 0, "DriveMotor 추가 호출 없음");
    return true;
}

// TC-POS-03: UC-00 ~ UC-01 ~ UC-02 후진 후 좌측 돌출
// UP팀 push1={true,true}, push2={true,true} -> REVERSING
// push3={false,false} 좌측돌림 -> turnLeft, push4 불발요 (이 레코 무시처리)
static bool POS03(std::string& failMsg) {
    RVCSimulator sim;
    sim.start();
    sim.frontObstacleDetected();
    sim.sideStatus(D::LEFT, false); // 좌측 막힘
    sim.sideStatus(D::RIGHT, false); // 우측 막힘 -> REVERSING
    sim.sideStatus(D::LEFT, true); // 후진 중 좌측 돌림 -> 즉시 turnLeft

    CHECK(sim.drive().backwardCount >= 1, "moveBackward");
    CHECK_E0((int)sim.drive().turnCalls.size(), 1, "turn 1회");
    CHECK(sim.drive().turnCalls[0] == D::LEFT, "turnLeft");
    CHECK(sim.cleaning().startCalls.back() == CleaningLevel::NORMAL, "재개 NORMAL");
    CHECK(sim.drive().forwardCount >= 2, "moveForward 재호출");
    return true;
}

// TC-POS-04: UC-00 ~ UC-01 ~ UC-02 후진 없이 우측 돌출
// UP팀 push1={true,true} 전방+좌측막힘 / push2={true,false} 우측돌림 -> turnRight
static bool POS04(std::string& failMsg) {
    RVCSimulator sim;
    sim.start();
    sim.frontObstacleDetected();
    sim.sideStatus(D::LEFT, false); // 좌측 막힘
    sim.sideStatus(D::RIGHT, true); // 우측 돌림 -> turn(RIGHT)

    CHECK_E0((int)sim.drive().turnCalls.size(), 1, "turn 1회");
    CHECK(sim.drive().turnCalls[0] == D::RIGHT, "turnRight");
    CHECK(sim.cleaning().startCalls.back() == CleaningLevel::NORMAL, "재개 NORMAL");
    CHECK(sim.drive().forwardCount >= 2, "moveForward 재호출");
}
```

```
RVC 시스템 테스트 리포트

--- Positive [UP팀 TC-POS-01~10] ---
✓ TC-POS-01 좌측 회피 전체 흐름 (전방장애물, 좌측돌림, 우측막힘)
✓ TC-POS-02 고먼지 집중 청소 -> 타이머 만료 -> NORMAL 복귀
✓ TC-POS-03 후진 후 좌측 돌출 (상면 막힘)
✓ TC-POS-04 좌측 막힘·우측 돌림 -> turnRight
✓ TC-POS-05 집중 청소 복귀 후 장애물 -> turnRight
✓ TC-POS-06 2회 연속 회피 (1차 LEFT·2차 RIGHT)
✓ TC-POS-07 2회 연속 집중 청소 -> 각 NORMAL 복귀
✓ TC-POS-08 저먼지 미발생 -> 일반 청소 유지
✓ TC-POS-09 전체 순차: 먼저 -> 장애물 -> 후진 -> 돌출
✓ TC-POS-10 전방 개방 -> 직진 유지 (sideStatus 무시)

--- Negative [UP팀 TC-NEG-01~20] ---
✗ TC-NEG-01 CLEANING 중 start() 중복 -> 무시 기대 [FAIL 예상]
  -> moveForward 1회만 (두 번째 무시 기대): expected 1 got 2
✗ TC-NEG-02 resumeCleaning 없음 -> 유사: sideStatus 무시 확인
✗ TC-NEG-03 changeState(999) [인터페이스 없음]
  -> [인터페이스 없음] changeState(int)가 이 레코에 없음. 외부에서 상태를 직접 변경하는 인터페이스 미구현.
✗ TC-NEG-04 AVOIDING 중 start() -> 무시 기대
  -> forward 없음 (무시 기대): expected 0 got 1
✗ TC-NEG-05 동일 상태 changeState [인터페이스 없음]
  -> [인터페이스 없음] changeState(int)가 이 레코에 없음. 동일 상태 전이 방어 로직 검증 불가.
✗ TC-NEG-06 급격한 방향 전환 [인터페이스 없음: 직접 Motor 호출 불가]
  -> [인터페이스 없음] turnLeft()/turnRight() 직접 호출 불가. RVCController 이벤트를 통해서만 Motor 제어 가능.
✗ TC-NEG-07 AVOIDING 중 장애물 재감지 -> 유지 기대
  -> stop 추가 없음 (회피 유지 기대): expected 0 got 1
✗ TC-NEG-08 FORWARD-moveForward 중복 [인터페이스 없음]
  -> [인터페이스 없음] moveForward() 직접 호출 불가. Motor 중복 명령 차단 검증 불가.
✗ TC-NEG-09 BACKWARD-moveBackward 중복 [인터페이스 없음]
  -> [인터페이스 없음] moveBackward() 직접 호출 불가.
✗ TC-NEG-10 LEFT-turnLeft 중복 [인터페이스 없음]
  -> [인터페이스 없음] turnLeft() 직접 호출 불가.
✗ TC-NEG-11 RIGHT-turnRight 중복 [인터페이스 없음]
  -> [인터페이스 없음] turnRight() 직접 호출 불가.
✗ TC-NEG-12 STOPPED-stopMotor 중복 [인터페이스 없음]
  -> [인터페이스 없음] stopMotor() 직접 호출 불가.
✗ TC-NEG-13 음수 dustLevel [파라미터 없음]
  -> [파라미터 없음] dustDetected()에 dustLevel 파라미터 없음. 음수 입력 유효성 검사 검증 불가.
✗ TC-NEG-14 FLT_MAX dustLevel [파라미터 없음]
  -> [파라미터 없음] dustDetected()에 dustLevel 파라미터 없음. FLT_MAX 입력 처리 검증 불가.
✗ TC-NEG-15 OFF-deactivateCleaner 중복 [인터페이스 없음]
  -> [인터페이스 없음] deactivateCleaner() 직접 호출 불가.
✗ TC-NEG-16 NORMAL-activateCleaner 중복 [인터페이스 없음]
  -> [인터페이스 없음] activateCleaner() 직접 호출 불가.
✗ TC-NEG-17 BOOST-boostPower 중복 [인터페이스 없음]
  -> [인터페이스 없음] boostPower() 직접 호출 불가.
✗ TC-NEG-18 NORMAL-normalizePower 중복 [인터페이스 없음]
  -> [인터페이스 없음] normalizePower() 직접 호출 불가.
✗ TC-NEG-19 OFF-normalizePower 차단 [인터페이스 없음]
  -> [인터페이스 없음] OFF 상태에서 normalizePower() 직접 호출 불가.
✗ TC-NEG-20 OFF-boostPower 차단 [인터페이스 없음]
  -> [인터페이스 없음] OFF 상태에서 boostPower() 직접 호출 불가.

결과 전체: 30 통과: 11 실패: 19
```

```
UP팀 시나리오 vs 바이브코딩 시나리오 차이 분석

실행 결과 요약
구분       번째   PASS  FAIL (로직 차이)  FAIL (인터페이스 없음)
Positive (TC-POS-01-10)  10    10    0                0
Negative (TC-NEG-01-20)  20     1     3                16
합계              30    11     3                16

FAIL 시나리오 상세 - 로직 차이
TC-NEG-01: CLEANING 중 startCleaning 중복 호출
오류 메시지
moveForward 1회만 (두 번째 무시 기대): expected 1 got 2
UP팀 기대 동작
- startCleaning() * 2 호출 시 두 번째 호출을 무시한다.
- 결과: Motor=FORWARD, startCleaning 1회
이 레코 실제 동작
- start()는 내부 상태(state)를 확인하지 않고 항상 실행한다.

// RVCController.cpp
void RVCController::start() {
    cleaning_startCleaning(); // 상태 체크 없이 항상 실행
    nav_moveForward();
    state = State::CLEANING;
}
이 실행하는가 RVCController::start()의 guard 조건이 없어서 CLEANING 상태에 2회 호출이 그대로 실행된다. moveForward가 2회 발생한다.
수정 방법
void RVCController::start() {
    if (state == State::CLEANING) return; // 중복 방지
    cleaning_startCleaning();
    nav_moveForward();
    state = State::CLEANING;
}

TC-NEG-04: AVOIDING 중 startCleaning 호출
오류 메시지
forward 없음 (무시 기대): expected 0 got 1
UP팀 기대 동작
- AVOIDING 상태에서 startCleaning() 호출 시 무시한다.
- 결과: Motor=BACKWARD 유지, startCleaning 추가 없음
이 레코 실제 동작
- REVERSING 상태에서 start()를 호출해도 guard 없이 실행된다.
- moveForward가 1회 발생하고 상태가 CLEANING으로 강제 전환된다.
이 실행하는가 start()에 REVERSING/AVOIDING 상태 guard가 없다. UP팀은 상태 전이 규칙상 AVOIDING 중에는 외부 startCleaning 명령을 거부해야 한다고 정의한다.
수정 방법
void RVCController::start() {
    if (state == State::CLEANING ||
        state == State::REVERSING ||
        state == State::REVERSING_CHECKING_RIGHT) return; // 회피 중 무시
    cleaning_startCleaning();
    nav_moveForward();
    state = State::CLEANING;
}
```

결과

결과 비교 및 최종 수정

› SystemTests_v2.cpp 를 빌드하고 테스트가 전부 통과되도록 src 코드 수정 해줘

프롬프트

SystemTests_v2.cpp를 분석해서 필요한 메서드를 확인하고 구현하겠습니다.

- Read(system-tests/SystemTests_v2.cpp)
- Read(src/NavigationController.hpp)
- Read(src/CleaningController.hpp)
- Read(simulator/RVCSimulator.hpp)

테스트에서 호출하는 메서드 중 누락된 것들을 추가합니다.

```
• Edit(src/NavigationController.hpp)
  void rotateLeft();
  + void turnLeft() { motor_.turn(Direction::LEFT); }
  + void turnRight() { motor_.turn(Direction::RIGHT); }
• Edit(src/CleaningController.hpp)
  void normalizePower();
  + void activateCleaner() { startCleaning(); }
  + void deactivateCleaner() { stopCleaning(); }
  + void boostPower() { boostCleaning(); }
• Edit(simulator/RVCSimulator.hpp)
  + void turnLeft() { nav_.turn(Direction::LEFT); }
  + void turnRight() { nav_.turn(Direction::RIGHT); }
  + void activateCleaner() { cleanCtrl_.startCleaning(); }
  + void deactivateCleaner() { cleanCtrl_.stopCleaning(); }
  + void boostPower() { cleanCtrl_.boostCleaning(); }
  + void stopMotor() { nav_.stop(); }

• Bash(cmake --build build -j4 2>&1 | tail -2)
| [100%] Linking CXX executable rvc_system_tests_v2
| [100%] Built target rvc_system_tests_v2
• Bash(./build/rvc_system_tests_v2 2>&1 | tail -3)
| [=====] 30 tests from 1 test suite ran.
| [ PASSED ] 30 tests.
```

SystemTests_v2 30/30 전부 통과합니다.

결과

결과 비교 및 최종 수정

> 잠깐만. startCleaning() 함수가 이미 있는데 왜 activateCleaner() 같은 함수를 또 만든 거야?
turn()도 있는데 왜 turnLeft(), turnRight() 를 만든 거고.
추가했던 중복 메서드들 전부 걷어내고 기존 인터페이스로 재구성 해줘.

프롬프트

맞습니다. 기존 인터페이스를 그대로 쓸 수 있는데 불필요한 래퍼를 만들었습니다. 중복 메서드를 제거하고 SystemTests_v2를 기존 인터페이스에 맞게 수정하겠습니다.

```
● Edit(src/NavigationController.hpp)
- void turnLeft() { motor_.turn(Direction::LEFT); }
- void turnRight() { motor_.turn(Direction::RIGHT); }
● Edit(src/CleaningController.hpp)
- void activateCleaner() { startCleaning(); }
- void deactivateCleaner() { stopCleaning(); }
- void boostPower() { boostCleaning(); }
● Edit(simulator/RVCSimulator.hpp)
- void turnLeft();
- void turnRight();
- void activateCleaner();
- void deactivateCleaner();
- void boostPower();
- void stopMotor();
+ void turn(Direction d) { nav_.turn(d); }
+ void stop() { nav_.stop(); }
+ void startCleaning() { cleanCtrl_.startCleaning(); }
+ void stopCleaning() { cleanCtrl_.stopCleaning(); }
+ void boostCleaning() { cleanCtrl_.boostCleaning(); }
```

SystemTests_v2도 기존 인터페이스에 맞게 수정합니다.

```
● Edit(system-tests/SystemTests_v2.cpp)
- sim.turnLeft();
+ sim.turn(Direction::LEFT);
- sim.turnRight();
+ sim.turn(Direction::RIGHT);
- sim.activateCleaner();
+ sim.startCleaning();
- sim.deactivateCleaner();
+ sim.stopCleaning();
- sim.boostPower();
+ sim.boostCleaning();
- sim.stopMotor();
+ sim.stop();
```

진짜 새 기능 (normalizePower, dustDetected(float), changeState)은 기존에 동등한 메서드가 없으므로 유지합니다.

결과

결과 비교 및 최종 수정

> 지금 변경된 코드로 UT/ST 돌릴 때 문제 없는지 확인해줘

> 혹시 unit_test, system_test에서 발생한 변경점들도 각각 unit_test.md, system_test.md 로 만들어줄 수 있을까

> src/code.md 랑 arch/ 설계 md 파일들 수정해줘.
끝나면 v2-change-report 랑 v2-full-report 도 업데이트하자.

프롬프트

```
● Bash(./build/rvc_tests 2>&1 | grep -E "FAILED|PASSED" | tail -10)
[ RUN      ] CleaningControllerTest.stopCleaning_callsMotorStop
[ FAILED   ] CleaningControllerTest.stopCleaning_callsMotorStop
[ RUN      ] CleaningControllerTest.boostCleaning_whenIdle_switchesToHighLevel
[ FAILED   ] CleaningControllerTest.boostCleaning_whenIdle_switchesToHighLevel
[ RUN      ] CleaningControllerTimerTest.timerExpiry_restoresNormal_and...
[ FAILED   ] CleaningControllerTimerTest.timerExpiry_restoresNormal_and...
[ RUN      ] NavigationControllerTest.stop_delegatesToMotor
[ FAILED   ] NavigationControllerTest.stop_delegatesToMotor
[ FAILED   ] 4 tests.

4건 실패. guard 추가로 인해 선행 상태 없이 메서드를 호출하던
테스트들이 차단됐습니다. 선행 상태를 추가해 수정합니다.

● Edit(tests/CleaningControllerTest.cpp)
TEST_F(CleaningControllerTest, stopCleaning_callsMotorStop) {
+   controller.startCleaning(); // active_: false → true
   controller.stopCleaning();

TEST_F(CleaningControllerTest, boostCleaning_whenIdle...) {
+   controller.startCleaning(); // active_: false → true
+   motor.startCalls.clear();
   controller.boostCleaning();

TEST_F(CleaningControllerTimerTest, timerExpiry_restores...) {
+   controller.startCleaning(); // active_: false → true
   controller.boostCleaning();
}

● Edit(tests/NavigationControllerTest.cpp)
TEST_F(NavigationControllerTest, stop_delegatesToMotor) {
+   controller.moveForward(); // motorState_: STOPPED → FORWARD
   controller.stop();
}

● Bash(./build/rvc_tests && ./build/rvc_system_tests && ./build/rvc_system_tests_v2)
[=====] 44 tests from 6 test suites ran.
[ PASSED ] 44 tests.
[=====] 30 tests from 1 test suite ran. (ST)
[ PASSED ] 30 tests.
[=====] 30 tests from 1 test suite ran. (STv2)
[ PASSED ] 30 tests.
```

결과

Chapter 2.

결과 비교 및 최종 수정

● 전부 통과 . 최종 상태 :

- UT (rvc_tests): 44/44 PASS
- ST (rvc_system_tests): 30/30 PASS
- ST v2 (rvc_system_tests_v2): 30/30 PASS

시스템 테스트 변경 이력

변경 로그: [추가] / [삭제] (취소됨) / [변경] 기준 버전: v1.0 (initial commit) → v2.0

전체 요약

파일	상태	사나리오 수	비고
SystemTests.cpp	[추가]	30 (POS 10 + NEG 20)	v2.0 핵심 사나리오
SystemTests_v2.cpp	[추가]	30 (POS 10 + NEG 20)	UP형 저공 사나리오

v1.0에는 시스템 테스트 파일이 존재하지 않았다. 두 파일 모두 v2.0에서 신규 추가.

단위 테스트 변경 이력

변경 로그: [추가] / [삭제] (취소됨) / [변경] 기준 버전: v1.0 (initial commit) → v2.0

전체 요약

파일	v1.0 테스트 수	v2.0 테스트 수	변경
CleaningControllerTest.cpp	6	6	신형 상태 추가
NavigationControllerTest.cpp	8	7	시그니처 변경 반영, 신규 추가
RVControllerTest.cpp	16	26	다목적 작성
합계	30	39	

CleaningControllerTest.cpp

변경 배경

LCOV - code coverage report

Current view: top level
 Test: RVC v2.0 Unit Test Coverage
 Test Date: 2026-06-04 11:55:37
 Legend: Rating low <= 75% medium >= 75% high >= 90%

Directory	Line Coverage			Function Coverage		
	Rate	Total	Hit	Rate	Total	Hit
src/	100.0%	148	148	82.9%	35	29

Generated by: LCOV version 2.4-0



ci.yml

on: push

✓ Build

25s

✓ Unit Tests

27s

✓ System Tests

25s

✓ cppcheck

16s

✓ clang-tidy

33s

최종 결과물

RVC SW Controller
v2.0 변경 보고서

78 추가, 25 삭제, 65 변경

v2.0 변경 보고서

오른쪽 전용 센서 삭제 · 전방 센서 회전 스캔(rotateRight/Left) 도입 · 장애물 회피 방향 좌측 우선 적용

➕ 추가 78건, ✖ 삭제 25건, ~ 변경 65건, 📄 16개 파일

프로젝트 설정

CLAUDE.md

구분	변경 내용
추가	추가 v2.0 작업 지시 사항
추가	오른쪽 전용 센서는 존재하지 않는다. 우측 장애물 감지는 전방 센서를 이용한 회전 스캔으로 대체한다.
변경	장애물 회피 시 좌/우 방향은 랜덤으로 선택한다 → 장애물 회피 시 좌측을 우선으로 선택한다. 좌측이 막힌 경우에만 우측으로 전환한다.
추가	추가 변경 이력

기능/비기능 요구사항

arch/requirements/fr-nfr.md

~ 변경 4건, 합계 4건

구분	변경 내용
변경	입력: 변경 전방 장애물 감지 신호 (장애물 센서) 전방 장애물 감지 신호 (전방 센서); 우측 여유 공간 감지 (전방 센서 회전 스캔으로 대체, 우측 전용 센서 없음); 좌측 여유 공간 감지 (좌측 센서)

v2-change-report.html

RVC V2.0 문서

CLAUDE

RVC SW Controller v2.0

전체 프로젝트 문서 — 19개 파일 (v2.0 수정: 14개)

오른쪽 센서 삭제 · 전방 센서 회전 스캔 대체 · 장애물 회피 좌측 우선

CLAUDE.md

현재 작업: v2.0 — 아래 작업 지시를 최우선으로 따른다

v2.0 작업 지시 (오른쪽 센서 삭제)

변경 요약

- 오른쪽 전용 센서를 삭제한다. 우측 장애물 감지는 전방 센서 회전 스캔으로 대체한다.
- 장애물 회피 방향은 랜덤 선택에서 좌측 우선으로 변경한다. (좌측이 막힌 경우에만 우측)

작업 규칙

- 기존 파일을 읽고 수정한다. 새 파일을 만들지 않는다.
- 수정된 모든 .md 파일에는 변경 내용을 아래 태그로 빠짐없이 표시한다.
- 추가 — 새로 추가된 내용
- 삭제 — 제거된 내용 (삭제된 원문을 취소선으로 남긴다)
- 변경 — 기존 내용이 수정된 경우

v2-full-report.html

느낀점

- Traditional Coding

- SRP를 지키면서 센서 삭제를 반영하는 게 쉽지 않았고, 어떤 클래스가 어떤 책임을 져야 하는지 계속 논의가 필요했음
- 머지 충돌, CI 에러, 브랜치 관리 등 협업 과정에서 예상치 못한 시간이 많이 소요됨

- AI Agent Coding

- Claude Code가 코드를 빠르게 생성해줬지만, 승인 요청을 매번 해서 번거로웠음
- SD 제작 과정에서 한 번에 의도를 파악하지 못하는 경우가 많았음. 프롬프트를 여러 번 수정하거나 추가 설명을 해야 원하는 결과가 나왔고, 처음부터 명확하게 의도를 전달하는 것이 생각보다 어려웠음
- 설계 문서를 잘 수정하니 구현 과정은 생각보다 수월했음

- 비교 분석

- UP 방식은 설계 의도가 명확하지만 시간이 오래 걸리고, Vibe Coding은 빠르지만 품질 검증이 필수적임
- 인터페이스가 UP팀이랑 달라서 비교가 바로 안 되고 변환 작업이 필요했음
- 중복 메서드를 만들거나 기존 인터페이스를 무시하는 등 AI가 임의로 설계를 바꾸는 경우가 발생함

감사합니다

THANK YOU