객체지향개발방법론
Practice #6

202211287 김태인

202311252 곽수호

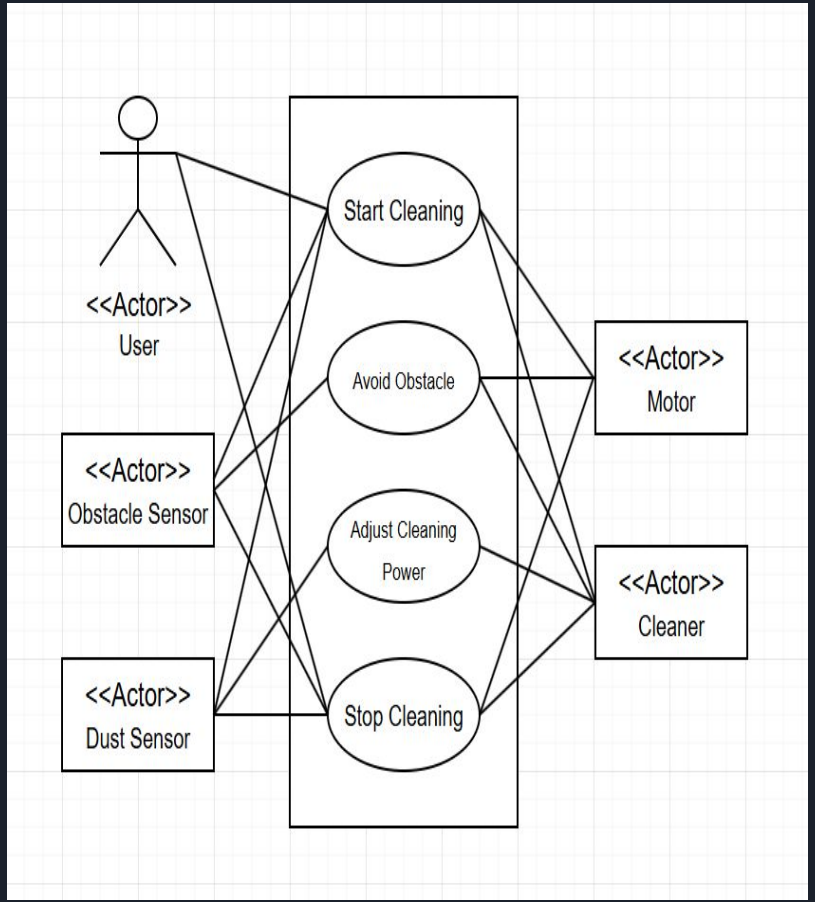
202111368 정선민

202211378 조성원

목차

1. Hand 작업으로 수정한 RVC software
2. AI를 활용한 요구사항 수정 반영
3. 차이 비교

usecase diagram



Use case

2

Use Case : 2. Avoid Obstacle

Actors : Obstacle Sensor, Motor, Cleaner

Description :

- 청소 중 Obstacle Sensor가 전방 장애물을 감지하면 이 유스케이스가 시작됨.

- System은 Motor, Cleaner의 동작을 중지시킴

- System은 장애물 위치에 따라 회피 기동 방향(좌/우 회전)을 판단하며, 사방에 장애물이 있을 경우 후진하며 회피 기동 방향 (좌/우 회전)을 판단.

- Motor가 장애물을 피하고 나서 전진 이동을 재개하며 Cleaner도 동작을 재개하면 유스케이스가 종료됨.

Use case

2

Pre-requisites :

- 로봇이 청소 동작을 수행하며 Motor가 전진중이어야 한다.
- Obstacle Sensor가 정상적으로 동작하고 있어야 한다.
- Motor가 제어 가능한 상태여야 한다.
- Cleaner가 제어 가능한 상태여야 한다.

Typical Courses of Events : (S) System (A) Actor

(A1): Obstacle Sensor (A2):Motor (A3): Cleaner

- 1. (A1) Obstacle Sensor가 전방 장애물을 감지한다.
- 2. (A1) Obstacle Sensor는 장애물 감지 신호를
- System 에 전달한다.
- 3. (S) System은 Motor와 Cleaner에게 정지명령을 전달
- 4. (A2) (A3) Motor와 Cleaner가 정지한다.
- 5. (S) System은 회피 가능한 방향을 판단한다.
- 6. (S) System은 좌측이 비어 있음을 확인한다.
- 7. (S) System은 Motor에 좌회전 명령을 전달한다.
- 8. (A2) Motor는 좌회전하여 장애물을 회피한다.
- *9. (S) 시스템은 센서 값 오류 여부를 확인한다*
- 10. (S) System은 Motor에게 전진 이동과 클리너에게 시작을 전달한다
- 11. (A2) (A3) Cleaner를 키고 Motor는 전진한다.
- 12. 유스케이스가 종료된다.

Use case

2 Alternative Courses of Events :

- A1. 오른쪽으로 회피 가능한 경우
  - 5a. (S) System은 좌측에 장애물이 있다고 판단
  - 6a. (S) System은 (A2)Motor에 시계방향 90도로 회전명령을 전달한다.
  - 7a. (S) System은 전방센서의 장애물 여부를 확인한다.
  - 8a. (S) System은 장애물이 없다고 판단하면 이후 기본 흐름 10단계로 진행한다
- A2. 양쪽 모두 장애물이 있는 경우
  - 5b. (S) System은 좌측에 장애물이 있다고 판단
  - 6b. (S) System은 (A2)Motor에 시계방향 90도로 회전명령을 전달한다.
  - 7b. (S) System은 전방센서의 장애물 여부를 확인한다.
  - 8b. (S) System은 전방 장애물이 있다고 판단
  - 9b. (S) System은 (A2)Motor에 시계방향 90도로 회전명령을 전달한다.
  - 10b. (S) System은 (A2)Motor에게 후진명령을 하고 다시 기본 흐름 4단계부터 진행

Use case

2 Exceptional Courses of Events :

- 장애물 감지 센서 오류 (전부 삭제)
  - 9. 좌회전 명령 또는 우회전 명령을 받고 로봇을 회전시켰을 때 회전시킨 방향의 반대 센서가 장애물을 없다고 인지하는 경우 장애물 감지 센서 3개중 최소 하나는 오류가 있음으로 판단하여 시스템은 로봇을 즉각 정지.

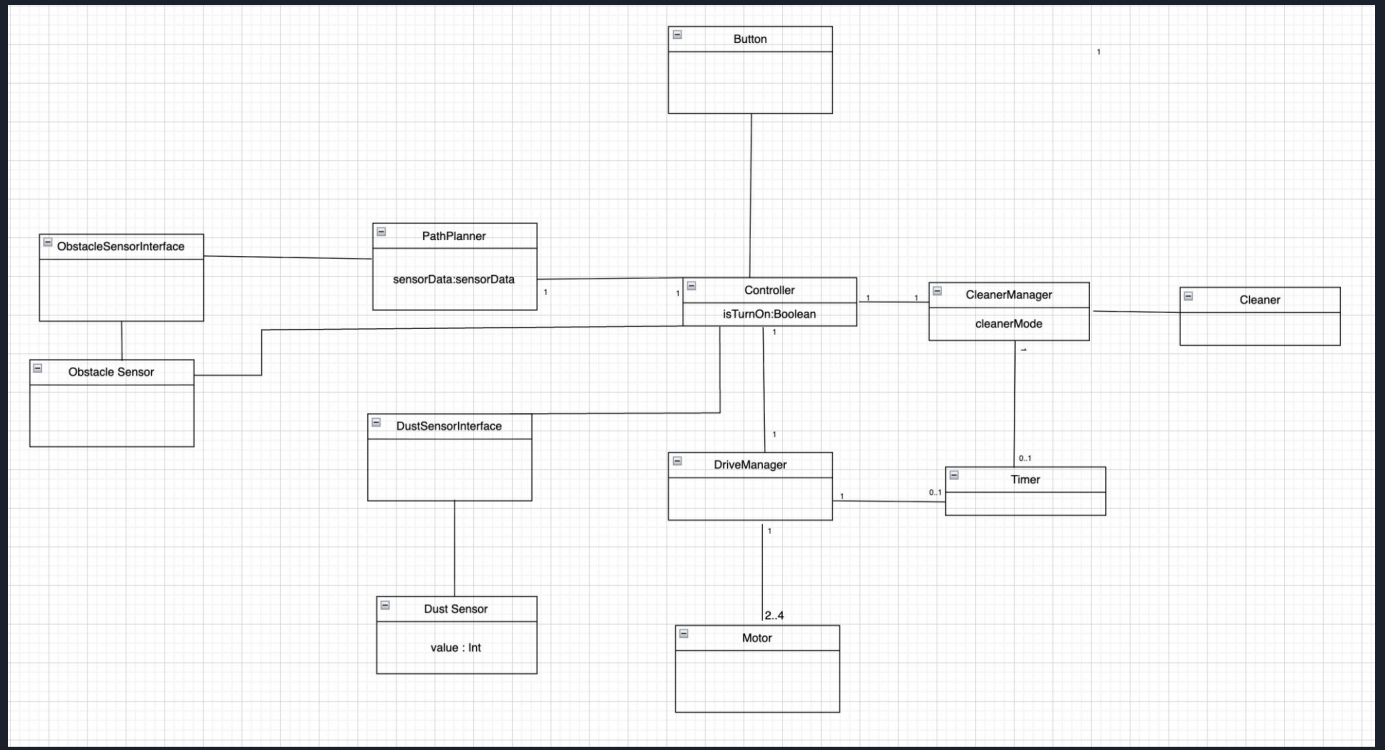
FR

Ref.#	Functional Requirements	Use-Case Number & Name	Category
R1.1	Start Process	1. Start Cleaning	**Evident**
R2.1	Go Straight Forward	1. Start Cleaning	Hidden
R3.1	Detect Obstacle	2. Avoid Obstacle	Hidden
R3.2.1.	Turn Left	2. Avoid Obstacle	Hidden
R3.2.2.1	Check Right Obstacle	2. Avoid Obstacle	Hidden
R3.2.2.2	Turn Right	2. Avoid Obstacle	Hidden
R3.2.3.1	Move Backward	2. Avoid Obstacle	Hidden
R3.2.3.2	Check Side Obstacles	2. Avoid Obstacle	Hidden
R4.1	Power Up Cleaning	3. Adjust Cleaning Power	Hidden
R5.1	Stop Process	4. Stop Cleaning	**Evident**

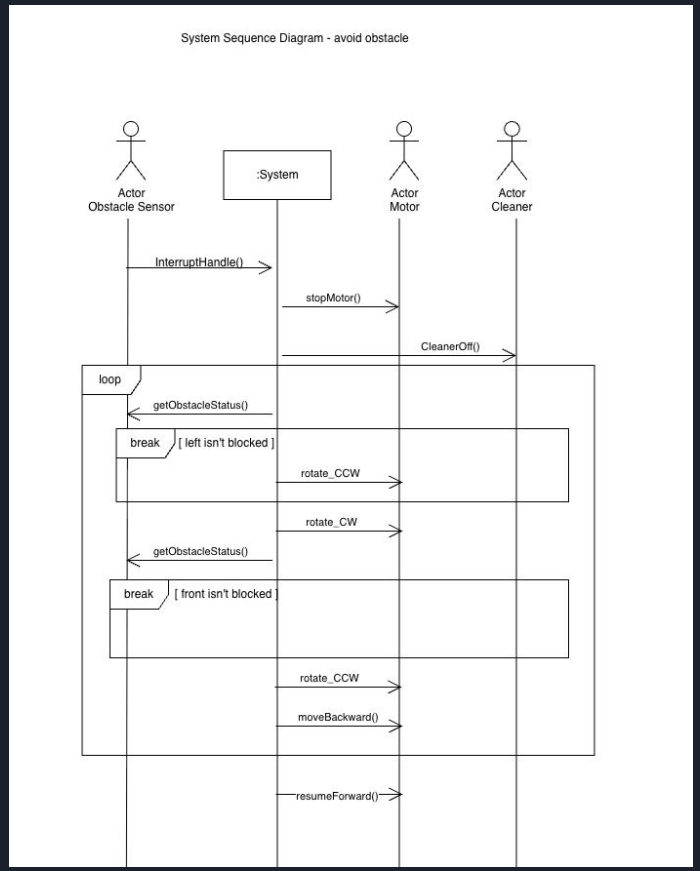
FR

R1.1 Start Process	**전원이 꺼진 상태에서 전원을 사용자가 누르면 켜진다 전원이 켜지면 정지 상태로 obstacle 여부와 dust 정보를 감지한다.**	**R5.1 Stop Process**	**켜진 상태에서 사용자가 전원을 누르면 모든 기능이 꺼진다.**
R2.1 Go Straight Forward	앞에 장애물이 없으면 전진하면서 청소를 한다.	R4.1 Power Up Cleaning	전진하면서 dust 센서가 true 면 cleaner가 3초 powerup이 되고, powerup 중 dust 센서가 true일 때마다 다시 3초씩 타이머 재카운트 되고 시간이 다 되면 cleaner가 일반 모드로 바뀐다.
R3.1 Detect Obstacle	전진 중 장애물이 전방에 있을 경우 cleaner을 끄고, motor를 정지한다.	R3.2.1 Turn Left	전진 중 장애물을 만났을 때 왼쪽에 장애물이 없으면 왼쪽으로 타이머가 끝날 때까지 회전한다.
R3.2.2.1 Check Right Obstacle	전진 중 장애물을 만났을 때 왼쪽에도 장애물이 있는 경우 오른쪽으로 타이머가 끝날 때까지 회전하여 전방 센서를 이용해 오른쪽 장애물 여부를 확인 한다.	R3.2.2.2 Turn Right	전방 센서를 이용해 오른쪽 장애물 여부를 확인후 오른쪽 장애물이 없는 경우 전진하면서 청소를 시작한다.
R3.2.3.1 Move Backward	전방 센서를 이용해 오른쪽 장애물 여부를 확인후 오른쪽 장애물이 있는 경우 다시 왼쪽으로 타이머가 끝날 때까지 회전한다. 이후 좌측, 우측 중 한쪽이라도 장애물이 없을 때까지 후진하다가 멈추고 장애물이 없는 방향으로 회전한 이후 전진하면서 청소를 시작한다.	R3.2.3.2 Check Side Obstacles	후진 하면서 타이머 시간마다 다음 행동을 반복한다. -좌측 장애물 여부 확인 -우회전 후 우측 장애물 여부 확인 -다시 좌회전

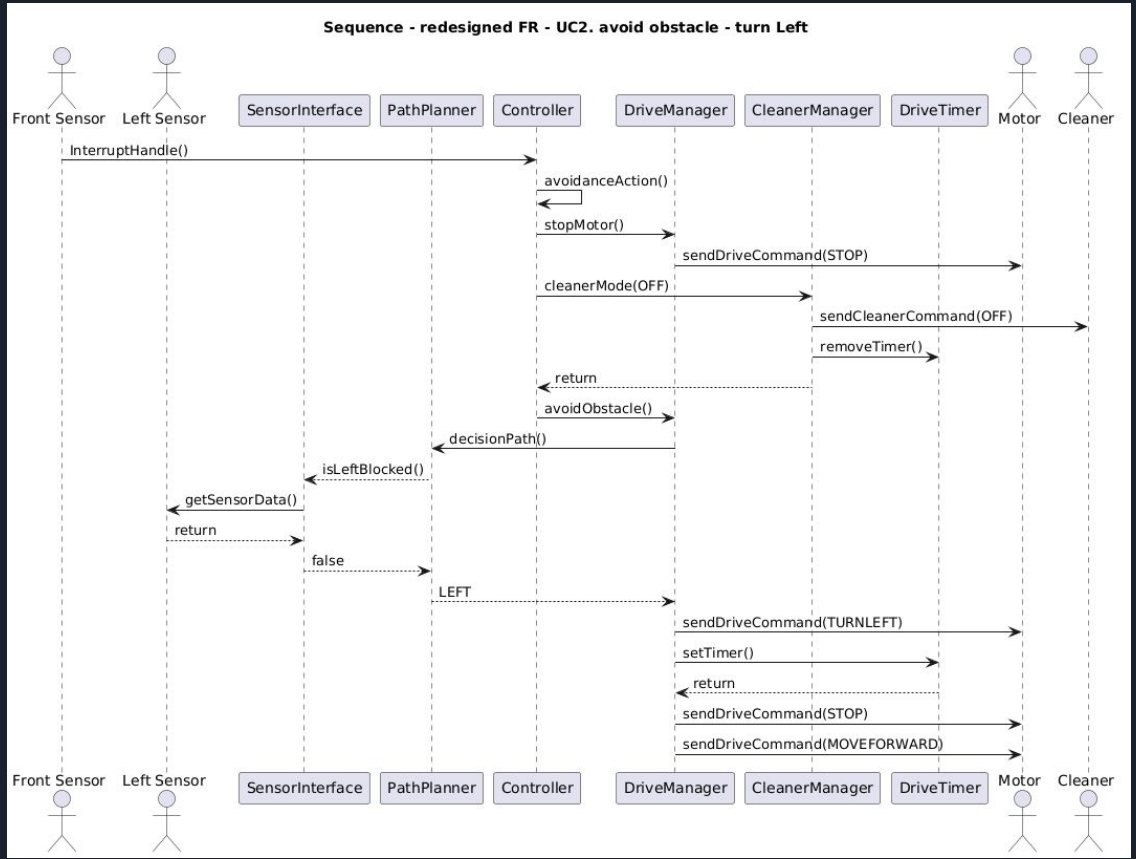
Domain(수정사항 없음)



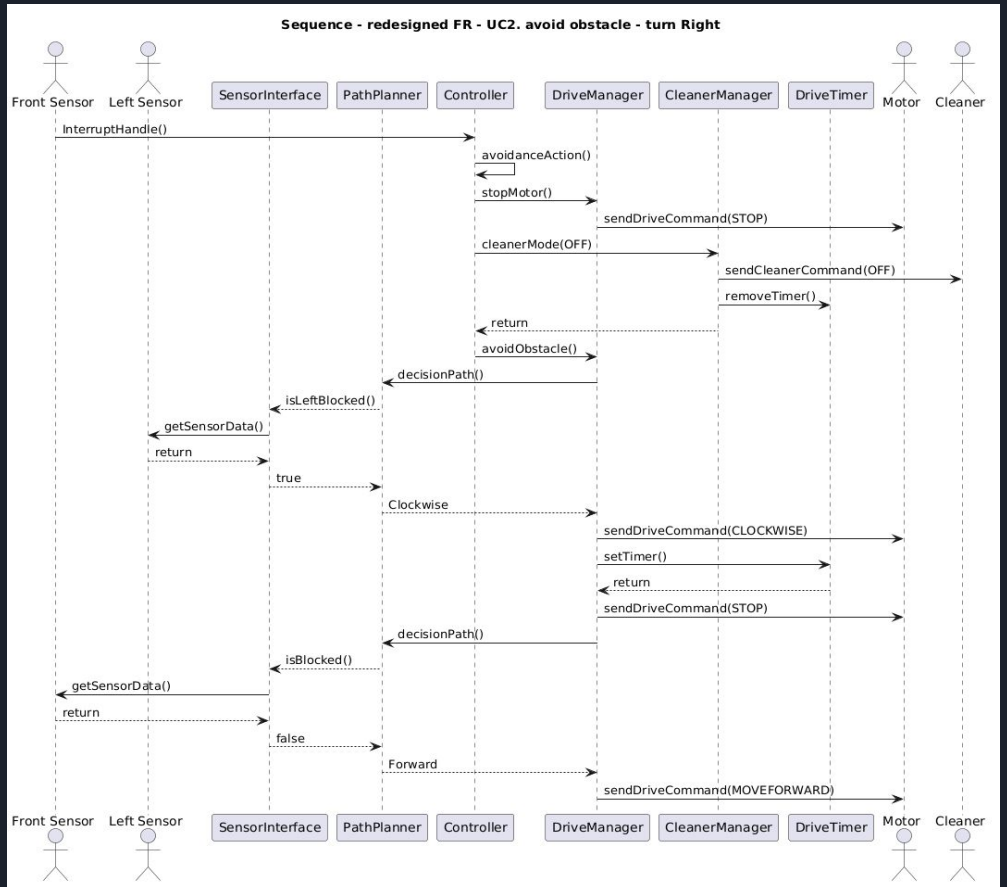
SSD (수정) - UC 02. avoid obstacle



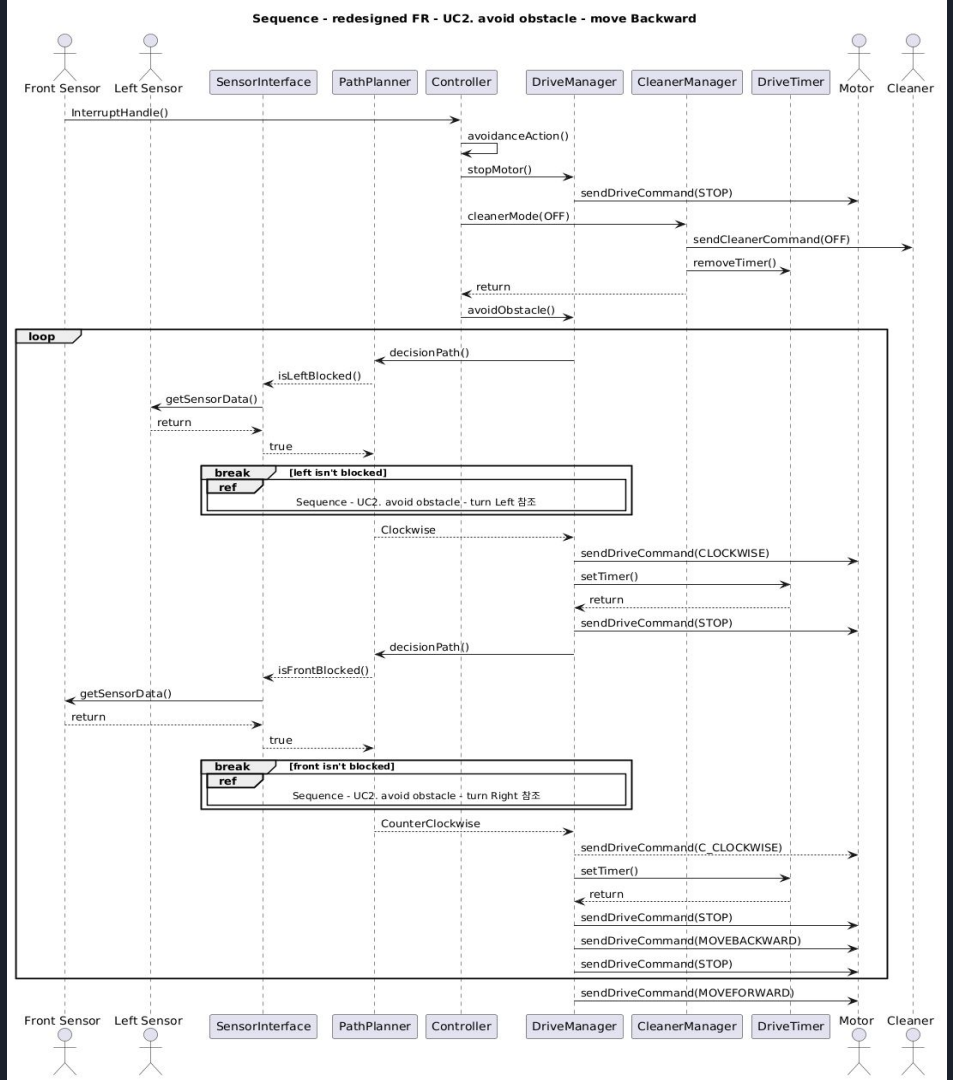
sequence diagram (수정) - 좌회전 관련 동작



sequence diagram (수정) - 우회전 관련 동작



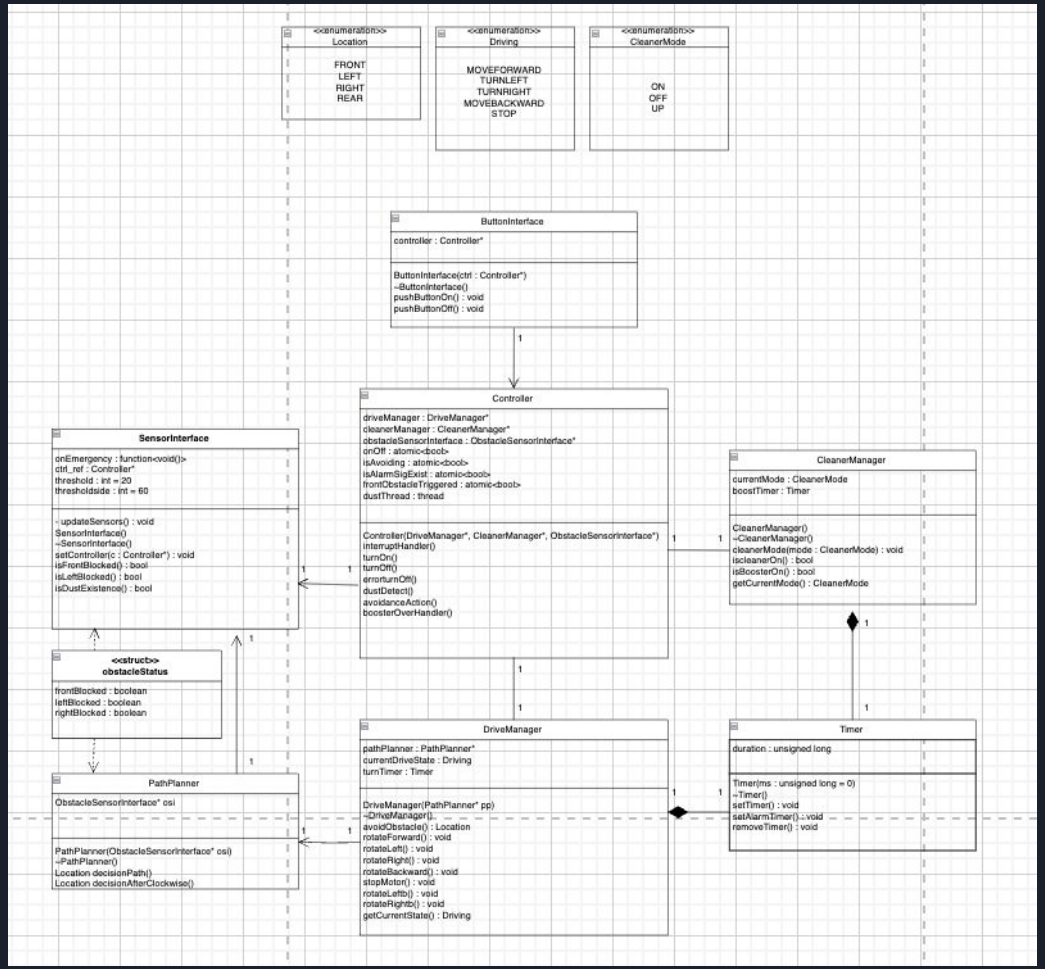
sequence diagram (수정) - 후진 관련 동작


Sequence - redesigned FR - UC2. avoid obstacle - move Backward
Front Sensor
Left Sensor
SensorInterface
PathPlanner
Controller
DriveManager
CleanerManager
DriveTimer
Motor
Cleaner
InterruptHandle()
avoidanceAction()
stopMotor()
sendDriveCommand(STOP)
cleanerMode(OFF)
sendCleanerCommand(OFF)
removeTimer()
return
avoidObstacle()
loop
decisionPath()
isLeftBlocked()
getSensorData()
return
true
break
[left isn't blocked]
ref
Sequence - UC2. avoid obstacle - turn Left 참조
Clockwise
sendDriveCommand(CLOCKWISE)
setTimer()
return
sendDriveCommand(STOP)
decisionPath()
isFrontBlocked()
getSensorData()
return
true
break
[front isn't blocked]
ref
Sequence - UC2. avoid obstacle - turn Right 참조
CounterClockwise
sendDriveCommand(C_CLOCKWISE)
setTimer()
return
sendDriveCommand(STOP)
sendDriveCommand(MOVEBACKWARD)
sendDriveCommand(STOP)
sendDriveCommand(MOVEFORWARD)

Class Diagram

기존 Class diagram에서
변경사항:

- 오른쪽 센서 관련 로직 삭제
- 변경된 로직에 대해
  Pathplanner 객체에서
  dicisionafterclockwise()
  operation 새로 생성



UT 수정사항(SensorInterface)

우측 센서값 테스트 제거

```cpp
#include <gtest/gtest.h>
#include <gmock/gmock.h>

#include "rvc_controller.h"
#include "UTest_Mock.h"

using namespace testing;

class FakeOSI : public ObstacleSensorInterface {
public:
    SensorData fake{0, 0, 0, 0};

    void setSensorData(int front, int left, int right, int dust) {
        fake = SensorData{front, left, right, dust};
    }

    // 테스트가 임계값을 그대로 참조할 수 있도록 접근자 제공 (protected 멤버 노출)
    int frontThreshold() const { return threshold; }
    int sideThreshold()  const { return thresholdside; }
    //readSensorData() 호출을 노출 (coverage 영끌용)
    SensorData callBaseReadSensorData() {
        return ObstacleSensorInterface::readSensorData();
    }

protected:
    SensorData readSensorData() override { return fake; }
};

class ObstacleSensorInterfaceTest : public ::testing::Test {
protected:
    FakeOSI osi;
};
```

```cpp
// 전방센서 임계값 = threshold =20
// 1-1-1 전방센서 장애물 여부 반환값 확인 (경계값 테스트)
TEST_F(ObstacleSensorInterfaceTest, Check_Front_Blocked) {
    ASSERT_EQ(osi.frontThreshold(), 20);

    struct Case { int front; bool expected; };
    const Case cases[] = {
        {  2, true},
        { 10, true},
        { 19, true},
        { 20, true},
        { 21, false},
        { 52, false},
        { 76, false},
    };

    for (const auto& c : cases) {
        osi.setSensorData(c.front, 100, 100, 0);
        EXPECT_EQ(osi.isFrontBlocked(), c.expected)
            << "front=" << c.front;
    }
}
```

```cpp
// 측면 센서 임계값 = thresholdside = 60
// 1-1-2 좌측센서 장애물 여부 반환값 확인 (경계값 테스트)
TEST_F(ObstacleSensorInterfaceTest, Check_Left_Blocked) {
    ASSERT_EQ(osi.sideThreshold(), 60);

    struct Case { int left; bool expected;};
    const Case cases[] = {
        { 18,  true},
        { 44,  true},
        { 59,  true},
        { 60,  true},
        { 61,  false},
        { 73,  false},
        { 115, false},
    };

    for (const auto& c : cases) {
        osi.setSensorData(100, c.left, 100, 0);
        EXPECT_EQ(osi.isLeftBlocked(), c.expected)
            << "left=" << c.left;
    }
}
```

```cpp
// 2-1 먼지 여부 반환값 확인
TEST_F(ObstacleSensorInterfaceTest, IsDustExistence_BoundaryValues) {
    struct Case { int dust; bool expected;};
    const Case cases[] = {
        {   0, false},
        {   1, true},
        {  27, true},
        { 101, true},
    };

    for (const auto& c : cases) {
        osi.setSensorData(100, 100, 100, c.dust);
        EXPECT_EQ(osi.isDustExistence(), c.expected)
            << "dust=" << c.dust;
    }
}
```

System Test 수정사항

TC3 전방 20cm, 좌측 20cm 장애물 환경에서 전방장애물 발견 시 우회전 후 전진 확인

+ TC23 전방 20cm, 좌측 20cm, 우측 20cm 장애물 환경에서 전방장애물 발견 시 우회전 후 다시 원상 복귀하여 후진 확인

+TC24 후진 중 좌측 장애물이 계속 막힌 경우 오른쪽을 확인하기 위해 우회전 후 다시 원상 복귀하는 것 확인

+TC25 후진 중 좌측 장애물이 없을 경우 좌회전하여 청소재개 확인

+TC26 후진 중 좌측 장애물이 막혀 우회전 후 전방 센서에 장애물이 없어 청소 재개 확인

~~TC22 정면 센서 오류 확인~~

AI를 활용한 요구사항 수정 반영

Rule 파일

```
---
description: 요구사항 변경 시 SRS/SDD/CODE/UT 산출물의 추가·삭제·변경을 색상으로 표시하는 규칙
alwaysApply: true
---

# Change Tracking (SRS / SDD / CODE / UT)

요구사항 변경으로 산출물을 수정할 때, **기존 내용과 수정 내용을 한눈에 구분**할 수 있도록 색상 표기를 적용한다.

## 적용 범위

| 산출물 | 대표 경로 |
| --- | --- |
| SRS | `docs/inception/requirements.md`, `docs/elaboration/use-case-details.md` |
| SDD | `docs/elaboration/diagrams/*.puml`, `docs/inception/diagrams/*.puml` |
| CODE | `src/**`, `include/**`, `apps/**` |
| UT | `tests/**` |

SRS·SDD·CODE·UT 중 **하나라도** 추가·삭제·변경이 발생하면, 연관된 다른 산출물도 같은 변경 ID로 함께 추적한다.

## 색상 규칙

| 구분 | 색 | 의미 | 표기 |
| --- | --- | --- | --- |
| **삭제** | 빨간색 | 더 이상 유효하지 않은 내용 | `<span style="color:#c62828">...</span>` 또는 `~~삭제된 텍스트~~` |
| **추가** | 초록색 | 새로 도입된 내용 | `<span style="color:#2e7d32">...</span>` |
| **변경** | 파란색 | 의미·동작·표기가 바뀐 내용 | `<span style="color:#1565c0">변경 후</span>` (필요 시 `<span style="color:#c62828">변경 전</span>` → `<span style="color:#
```

요구 사항 변경

RVC Input/Output Event Definitions

Input/ Output Event	Description	Format / Type
Front Sensor Input	Detects obstacles in front of the RVC	True / False, Interrupt
Left Sensor Input	Detects obstacles in the left side of the RVC periodically	True / False, Periodic
~~Right Sensor Input~~	~~Detects obstacles in the right side of the RVC periodically~~ (CHG-001: removed)	~~True / False, Periodic~~
Dust Sensor Input	Detects dust on the floor periodically	True / False, Periodic
Direction	Direction commands to the motor (go forward / turn left with an angle / turn right with an angle)	Forward / Backward / Left / Right
Clean	Turn off / Turn on / Power-Up	On / Off / Up

[CHG-001] 1차 요구사항 변경: Right Sensor 입력을 제거한다. 장애물 감지는 Front Sensor + Left Sensor 두 개만 사용한다. Preliminary Requirements(자동 청소, 직진, 장애물 시 회피/후진, 먼지 시 power-up)는 그대로 충족하되, 우측 장애물은 센싱하지 않고 좌측 센서와 전방 센서만으로 회피 방향을 결정한다. Motor의 `Right` (우회전) 출력 자체는 유지된다(센서 없이 수행하는 fallback 회전).

커서 1차 설계 제안

최초 제안 (1차안) — "좌측 막힘 → 즉시 후진"

Right Sensor를 없앤 뒤, 우측을 직접 확인할 수 없으니 **"전방 + 좌측 막힘"을 후진 트리거**로 삼았습니다.

```
전방 장애물 감지
  → cleaner Off, 좌측 센서 확인

  ┌ 좌측 clear ──→ 좌회전 ──→ 전진 청소 복귀
  │
  └ 좌측 blocked ──→ ★ 바로 후진 ★
                          │
                          └ 후진 완료 → 좌측 재확인
                                ├ 좌측 clear → 좌회전 → 복귀
                                └ 좌측 여전히 blocked → 우회전(fallback, 센서 없음)
```

1차 설계 수정

설계 의도

- 우측 센서가 없어 **"전·좌·우 모두 막힘"을 직접 감지할 수 없음**
- 그래서 **front + left blocked ≈ 막힌 상황**으로 보고 후진을 트리거
- 후진 후에도 좌측이 막혀 있으면, 남은 방향인 **우측으로 우회전**(센서 없이 blind turn)

이후 바뀐 이유 (참고)

이 1차안에 대해 **"우측이 실제로 열려 있는데도 불필요하게 후진한다"**고 지적하셔서, 2차안으로 바뀌었습니다.

```
좌측 blocked → 우회전 → 전방 재확인
  ├ 전방 clear → 후진 없이 복귀 (우측이 열려 있던 경우)
  └ 전방 still blocked → dead-end → (Option C) 좌회전 복원 → 후진
```

커서 2차 설계 제안

더 나은 설계 (후진을 dead-end에서만)

우측 센서가 없어도 **"우회전한 뒤 전방 센서로 다시 확인"** 하면 우측 상태를 간접적으로 알 수 있습니다.

상황	동작
전방 막힘, 좌측 clear	좌회전 → 복귀
전방 막힘, 좌측 blocked	**우회전** → 전방 재확인
└ 우회전 후 전방 clear (= 우측이 비어 있었음)	**후진 없이** 전진 청소 복귀 ✅
└ 우회전 후 전방 blocked (= 전·좌·우 모두 막힌 dead-end)	**후진** 후 회피 재시작

이렇게 하면:

- 우회전으로 회피 가능한 경우 **후진하지 않습니다** (사용자 지적 사항 해결)
- 후진은 **전·좌·우가 모두 막힌 진짜 dead-end**에서만 발생 → Preliminary의 "obstacles in front, left and right → backward" 의미와 정확히 일치
- Front + Left 센서 2개만 사용 (우측은 우회전+전방센서로 간접 확인)

이 설계로 문서를 다시 수정하겠습니다.

커서 2차 설계 제안

UC-003 Avoid Obstacle

Field	Detail
Trigger	`FrontObstacleDetected()` while powered on.
Main Flow	Controller cancels power-up timer, turns cleaner off, enters `AvoidingObstacle`, checks the **left sensor only**, and if clear turns left, then resumes forward cleaning. ~~checks left/right sensors, turns toward clear side~~
Alternative A	Left blocked: turn right (no right sensor; the opposite side is the likely opening), then re-check the front sensor. ~~Left blocked and right clear: turn right.~~
Alternative B	After turning right, front clear (right was open): resume forward cleaning without backing up. Front still blocked (front + left + right all blocked, a dead-end): move `Backward`, then restart avoidance from the left-sensor check. ~~Left and right blocked: move backward repeatedly until one side is clear, then turn toward clear side.~~
Exception	Dust detected during avoidance is ignored. Cleaner remains off.
Exception	User button during avoidance immediately stops motor, turns cleaner off, cancels timer, and powers off.

2차 설계 수정

Front Obstacle Detected
-> Left Blocked
-> Turn Right
-> Front Blocked
-> Backward

⇒ 후진시 벽에 부딪힘

UC-003 Avoid Obstacle

Field	Detail
Trigger	`FrontObstacleDetected()` while powered on.
Main Flow	Controller cancels power-up timer, turns cleaner off, enters `AvoidingObstacle`, checks the **left sensor only**, and if clear turns left, then resumes forward cleaning. ~~checks left/right sensors, turns toward clear side~~
Alternative A	Left blocked: turn right (no right sensor; the opposite side is the likely opening), then re-check the front sensor. ~~Left blocked and right clear: turn right.~~
Alternative B	After turning right, front clear (right was open): resume forward cleaning without backing up. Front still blocked (front + left + right all blocked, a dead-end): move `Backward`, then restart avoidance from the left-sensor check. ~~Left and right blocked: move backward repeatedly until one side is clear, then turn toward clear side.~~
Exception	Dust detected during avoidance is ignored. Cleaner remains off.
Exception	User button during avoidance immediately stops motor, turns cleaner off, cancels timer, and powers off.

커서 3차 설계 제안

좌측 장애물 확인
우회전
전방 장애물 확인
후진

이렇게 하게되면, 뒤에있는 장애물과 부딪힐 텐데

Opus 4.8 High

Thought for 132s

아주 정확한 지적입니다. 맞습니다 — **지금 설계의 "우회전 → (전방 막힘) → 후진" 단계는 방금 등진 좌측 장애물로 후진해 들어가 충돌**합니다. 제가 시뮬레이터의 충돌 방지(`move_robot` 가 막힌 칸이면 안 움직임)에 기대어 "제자리"라고 설명했지만, **설계 차원에서는 장애물을 향해 후진 명령을 내리는 잘못된 동작**이 맞습니다.

왜 충돌하나

```
북쪽 바라봄:  전방(N) 막힘, 좌측(W) 막힘
 → 우회전 → 이제 동쪽(E)을 바라봄
 → 전방(E)도 막힘  (= dead-end)
 → 후진?  동쪽을 보는 상태의 "뒤"는 서쪽(W)
          그런데 W는 방금 등진 좌측 장애물! → 충돌 ❌
```

우회전으로 이미 방향을 틀었기 때문에, "뒤"가 더 이상 안전한 "왔던 방향(남쪽)"이 아니라 좌측 벽이 되어버립니다.

수정안 두 가지 (요구사항 변경이라 선택 필요)

방안 A — 막힌 방향을 피해 회전으로 탈출 (후진 명령 없음)

```
좌측 막힘 → 우회전 → 전방 확인
   전방 열림  → 전진 (효율적, 후진 X)
   전방 막힘  → 우회전 한 번 더 → 전방 확인 ...
              → 열린 방향(보통 남쪽=왔던 쪽)을 정면으로 만들면 전진
```

- 장점: **충돌 없음**(전진은 항상 전방 clear일 때만), 우측 열리면 후진 안 함(효율적)
- 단점: dead-end에서 **literal "Backward" 명령을 쓰지 않음** → Preliminary의 "move backward" 문구를 *글자 그대로*는 충족 못 함(열린 방향으로 돌아 나가는 것으로 대체)

방안 B — 좌측 막히면 먼저 후진(안전), 그다음 회전

```
좌측 막힘 → 후진(왔던 쪽=항상 열림, 안전) → 좌측 재확인
   좌측 열림 → 좌회전 → 전진
   좌측 막힘 → 우회전 → 전진
```

- 장점: **literal Backward 사용**(Preliminary 충족), **충돌 없음**(왔던 방향으로만 후진)
- 단점: 우측이 열려 있어도 **일단 한 칸 후진** → 예전에 지적하셨던 "불필요한 후진" 재발

(참고) 방안 C — 우측 먼저 떠보고, 막혔으면 원래 방향으로 되돌린 뒤 안전하게 후진

```
좌측 막힘 → 우회전 → 전방 확인
   열림 → 전진
   막힘 → 좌회전(원래 북쪽 복귀) → 후진(남쪽, 안전) → 재확인
```

- 효율(우측 열리면 후진X) + literal 후진 + 충돌 없음을 모두 만족
- 단점: "우회전했다 다시 좌회전" 하는 탐색 동작이 다소 부자연스러움

커서 3차 설계

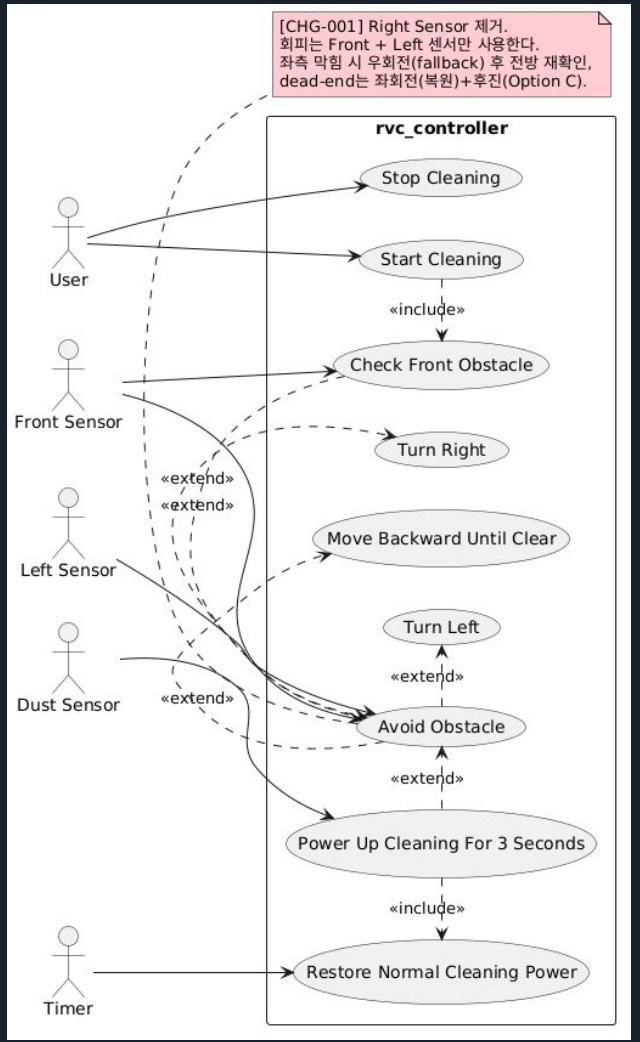
핵심 변경 (dead-end 충돌 해결)

후진 시 등진 좌측 장애물과 부딪히던 문제를, **dead-end에서 먼저 좌회전으로 원래 진행 방향을 복원한 뒤** 항상 열려 있는 뒤쪽(왔던 칸)으로 후진하도록 바꿨습니다.

```
좌측 막힘 → 우회전 → 전방 재확인
    전방 열림  → 전진 복귀 (후진 X, 효율적)
    전방 막힘  → 🔄 좌회전(원래 방향 복원) → 🔄 후진(열린 뒤쪽) → 🔄 회피 재시작
```

`right_turn_attempted_` 플래그로 **복원용 좌회전(→ 후진)** 과 **일반 좌측 회피 좌회전(→ 전진 복귀)** 을 구분합니다.

Use Case Diagram



Use Case(수정 사항)

UC-003 Avoid Obstacle

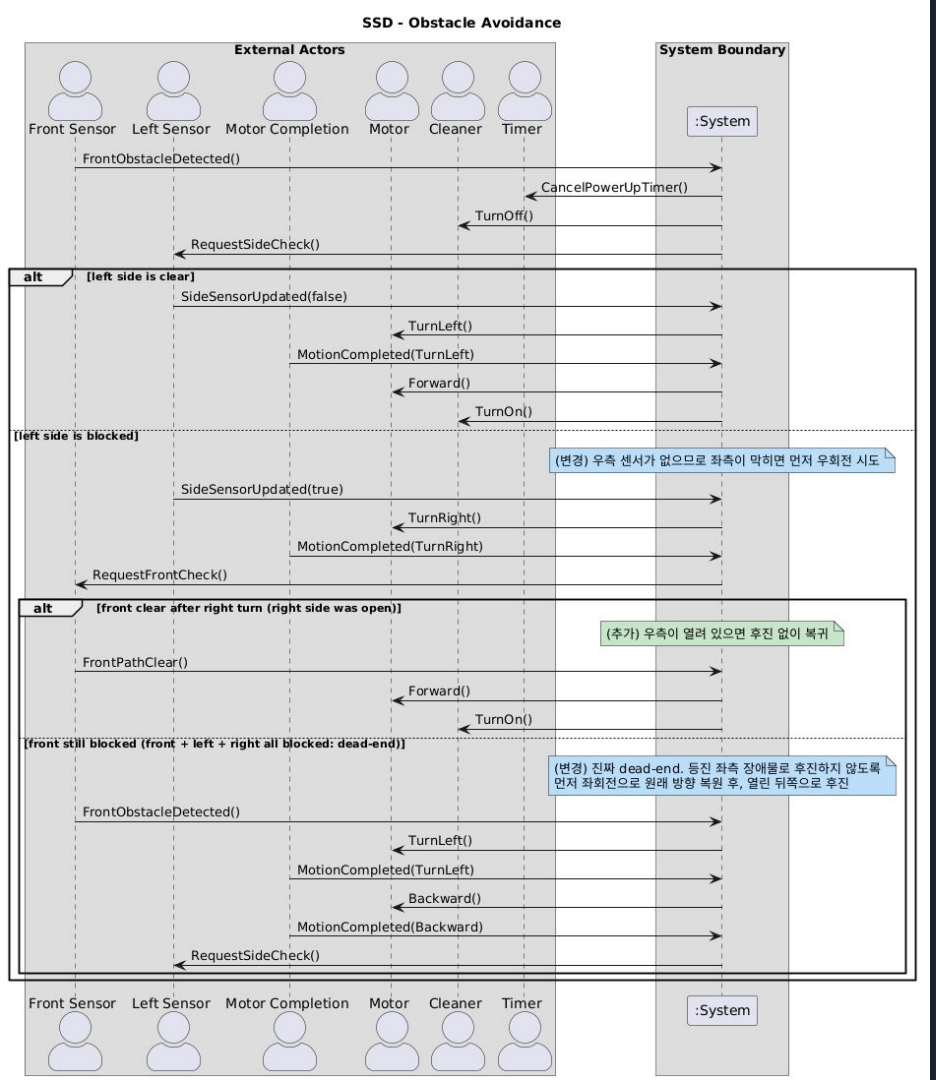
Field	Description
Primary Actor	Front Sensor
Supporting Actors	Left Sensor, ~~Right Sensor,~~ Motor, Cleaner (CHG-001: Right Sensor 제거)
Trigger	전원이 켜져 있거나 청소 중일 때 전방 장애물이 감지된다.
Preconditions	`rvc_controller` 전원이 켜져 있다.
Postconditions	장애물 회피 후 전진 청소 상태로 복귀한다.
Main Success Scenario	1. Front Sensor가 전방 장애물을 감지한다. 2. `rvc_controller`가 Cleaner에 `Off` 명령을 보낸다. 3. `rvc_controller`가 Left Sensor를 확인한다. 4. 좌측 장애물이 없다. 5. `rvc_controller`가 Motor에 `TurnLeft` 명령을 보낸다. 6. 회피가 종료된다. 7. `rvc_controller`가 Motor에 `Forward` 명령을 보낸다. 8. `rvc_controller`가 Cleaner에 `On` 명령을 보낸다.
Extensions	4a. 좌측 장애물이 있으면 `rvc_controller`가 Motor에 `TurnRight` 명령을 보낸다(우측 센서가 없으므로 좌측의 반대쪽인 우측으로 회피 시도). 4a1. 우회전이 완료되면 Front Sensor로 전방을 다시 확인한다. 4a2. 전방이 비어 있으면(우측이 비어 있었던 경우) 후진 없이 Motor `Forward` + Cleaner `On`으로 복귀한다. 4a3. 전방이 여전히 막혀 있으면(전방·좌측·우측 모두 막힌 dead-end) 먼저 Motor에 `TurnLeft` 명령을 보내 원래 진행 방향을 복원한 뒤(우회전으로 등진 좌측 장애물로 후진하지 않기 위함), 열린 뒤쪽으로 `Backward` 명령을 보내고 회피를 다시 시작한다. ~~4a. 좌측 장애물이 있으면 Right Sensor를 확인한다. 4a1. 우측이 없으면 우회전. 4a2. 둘 다 막히면 후진. 4a3. 빈 방향으로 회전.~~

FR, NFR(수정사항)

| FR-008 | 장애물 회피 프로세스는 좌측 센서로 좌측 장애물 여부를 확인해야 한다. ~~먼저~~ (CHG-001: 우측 센서가 없으므로 측면 센서는 좌측 하나뿐이다.) | | FR-009 | 좌측 장애물이 없으면 `rvc_controller`는 motor에 좌회전 명령을 내려야 한다. | | FR-010 | 좌측 장애물이 있으면 `rvc_controller`는 motor에 우회전 명령을 내려야 한다(우측 센서가 없으므로, 좌측이 막혔을 때 반대쪽인 우측으로 회피를 시도한다). ~~좌측 장애물이 있으면 우측 장애물 여부를 확인해야 한다.~~ (CHG-001) | | FR-011 | 우회전이 완료되면 `rvc_controller`는 전방 장애물 여부를 다시 확인해야 한다(이때의 전방은 회전 전의 우측 방향이며, 전방 센서로 우측 가용 여부를 간접 판단한다). ~~우측 장애물이 없으면 motor에 우회전 명령을 내려야 한다.~~ (CHG-001) | | FR-012 | 우회전 후 전방 장애물이 없으면(= 우측이 비어 있었던 경우) `rvc_controller`는 후진 없이 전진 청소로 복귀해야 한다. ~~좌측과 우측 모두 장애물이 있으면 한쪽이 비어 있을 때까지 후진 명령을 내려야 한다.~~ (CHG-001) | | FR-013 | 우회전 후에도 전방 장애물이 있으면(= 전방·좌측·우측이 모두 막힌 dead-end) `rvc_controller`는 우회전으로 이미 방향을 틀어 등진 좌측 장애물로 후진하지 않도록, 먼저 좌회전 명령으로 원래 진행 방향을 복원한 뒤 열린 뒤쪽으로 후진 명령을 내리고 회피 프로세스를 다시 시작해야 한다. ~~후진 중 좌측 또는 우측 중 하나가 비어 있으면 장애물이 없는 방향으로 회전해야 한다.~~ (CHG-001, Option C) | | FR-014 | 장애물 회피 프로세스가 종료되면 `rvc_controller`는 다시 motor를 전진시키고 cleaner를 켜야 한다. | | FR-015 | 전진 중 전방 먼지가 감지되면 `rvc_controller`는 motor의 전진 동작을 멈추거나 지연시키지 않은 채 cleaner의 흡입력만 3초 동안 증가시켜야 한다. | | FR-016 | 흡입력 증가 후 3초가 지나면 `rvc_controller`는 cleaner의 흡입력을 정상화해야 하며, 이 변경은 motor의 전진 동작에 영향을 주지 않아야 한다. | | FR-017 | 흡입력 증가 중 전방 장애물이 감지되면 `rvc_controller`는 증가된 시간을 저장하지 않고 즉시 cleaner를 끄고 장애물 회피 프로세스를 시작해야 한다. | | FR-021 | 흡입력 증가 중 다시 먼지가 감지되면 `rvc_controller`는 이전 타이머를 폐기하고 그 시점부터 다시 3초 타이머를 시작해야 한다. 이 재시작은 motor 동작에 영향을 주지 않는다. | | FR-018 | `rvc_controller`는 motor 명령을 `Forward`, `Backward`, `TurnLeft`, `TurnRight`, `Stop` 중 하나로 출력해야 한다. | | FR-019 | `rvc_controller`는 cleaner 명령을 `Off`, `On`, `PowerUp` 중 하나로 출력해야 한다. | | FR-020 | `rvc_controller`는 전방 센서, 좌측 센서, ~~우측 센서,~~ 먼지 센서, 사용자 버튼 입력을 처리해야 한다. (CHG-001: 우측 센서 입력 제거) |

NFR-007	회피 정책은 좌측 우선 정책을 기본으로 하되, 향후 정책 변경이 가능하도록 controller 내부 의사결정과 actuator 출력이 분리되어야 한다. (CHG-001/Option C: 측면 장애물 입력은 좌측 센서 하나만 사용한다. 좌측 차단 시 우회전 → 전방 재확인 순서로 회피하며, 후진은 우회전 후에도 전방이 막힌 dead-end에서만 사용하되 좌회전으로 원래 방향을 복원한 뒤 열린 뒤쪽으로 후진한다.)

SSD(수정사항)


SSD - Obstacle Avoidance
External Actors
System Boundary
Front Sensor
Left Sensor
Motor Completion
Motor
Cleaner
Timer
:System
FrontObstacleDetected()
CancelPowerUpTimer()
TurnOff()
RequestSideCheck()
alt
[left side is clear]
SideSensorUpdated(false)
TurnLeft()
MotionCompleted(TurnLeft)
Forward()
TurnOn()
[left side is blocked]
(변경) 우측 센서가 없으므로 좌측이 막히면 먼저 우회전 시도
SideSensorUpdated(true)
TurnRight()
MotionCompleted(TurnRight)
RequestFrontCheck()
alt
[front clear after right turn (right side was open)]
(추가) 우측이 열려 있으면 후진 없이 복귀
FrontPathClear()
Forward()
TurnOn()
[front still blocked (front + left + right all blocked: dead-end)]
(변경) 진짜 dead-end. 등진 좌측 장애물로 후진하지 않도록
먼저 좌회전으로 원래 방향 복원 후, 열린 뒤쪽으로 후진
FrontObstacleDetected()
TurnLeft()
MotionCompleted(TurnLeft)
Backward()
MotionCompleted(Backward)
RequestSideCheck()
Front Sensor
Left Sensor
Motor Completion
Motor
Cleaner
Timer
:System

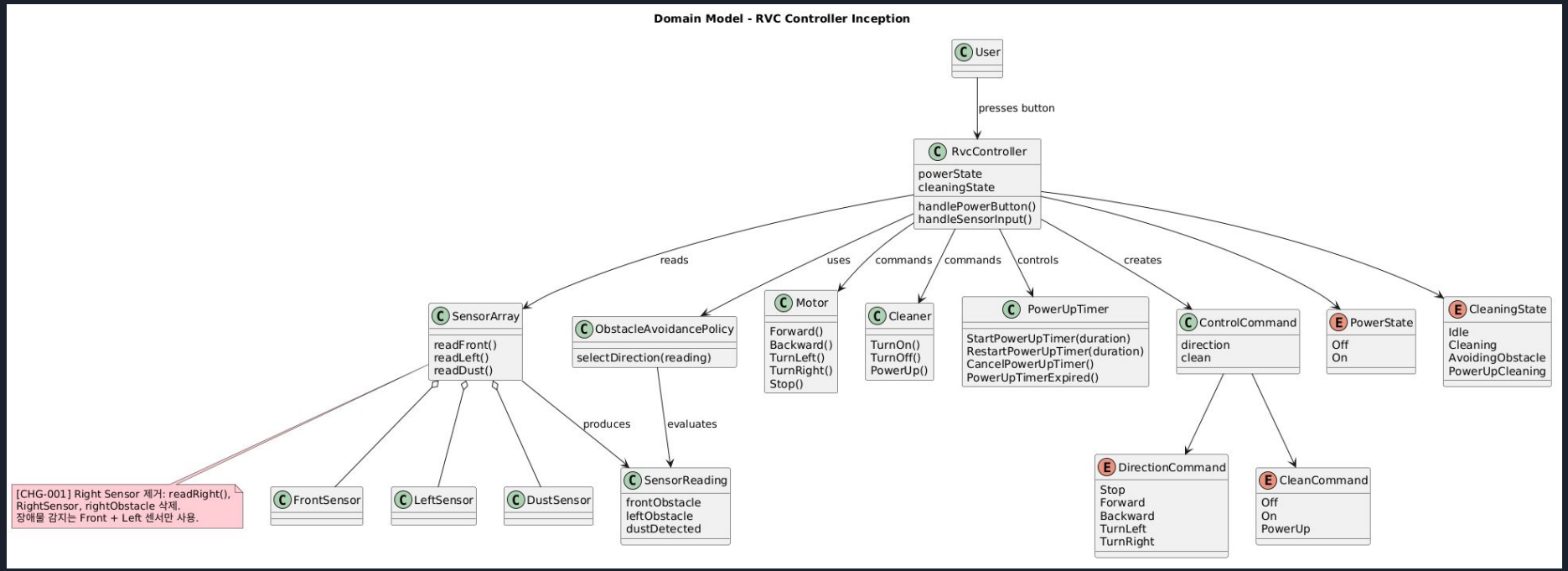
System Operation(수정사항)

System Operations

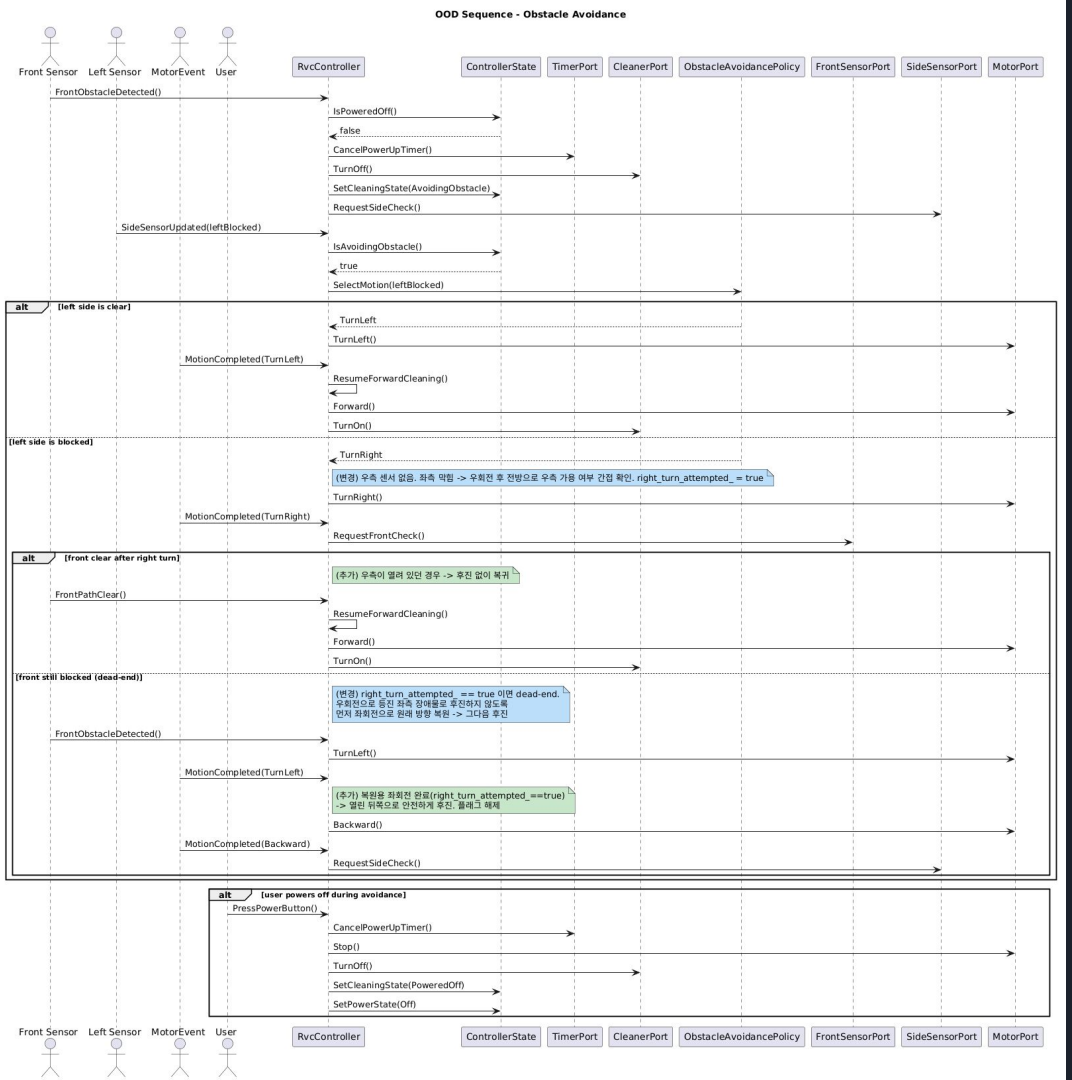
시스템 오퍼레이션은 외부 actor, sensor, timer, actuator completion event가 `rvc_controller`로 보내는 메시지다.

Operation	Source	Description	Related FR / UC
`PressPowerButton()`	User	전원 off이면 시작, 전원 on이면 즉시 종료한다.	FR-001, FR-002, FR-003, UC-001, UC-002
`FrontObstacleDetected()`	Front Sensor	전방 장애물 감지 interrupt. 즉시 cleaning/power-up을 중단하고 회피로 전이한다.	FR-006, FR-007, FR-017, UC-003
`FrontPathClear()`	Front Sensor	전원 on 또는 회피 종료 후 전방이 비어 있음을 알린다.	FR-004, FR-005, FR-014, UC-001
`SideSensorUpdated(leftBlocked)`	Left Sensor	회피 중 좌측 장애물 상태만 controller에 전달한다 (CHG-001: 우측 센서 제거, `rightBlocked` 파라미터 삭제).	FR-008, FR-009, FR-010, FR-011, FR-012, FR-013
`MotionCompleted(motion)`	Motor	회전 또는 후진 step 완료를 알린다. 하드웨어 세부 구현은 제외한다.	FR-013, FR-014
`DustDetected()`	Dust Sensor	전진 청소 중 먼지 감지를 전달한다.	FR-015, FR-017, FR-021, UC-004
`PowerUpTimerExpired()`	Timer	3초 power-up 시간이 지났음을 전달한다.	FR-016, UC-004

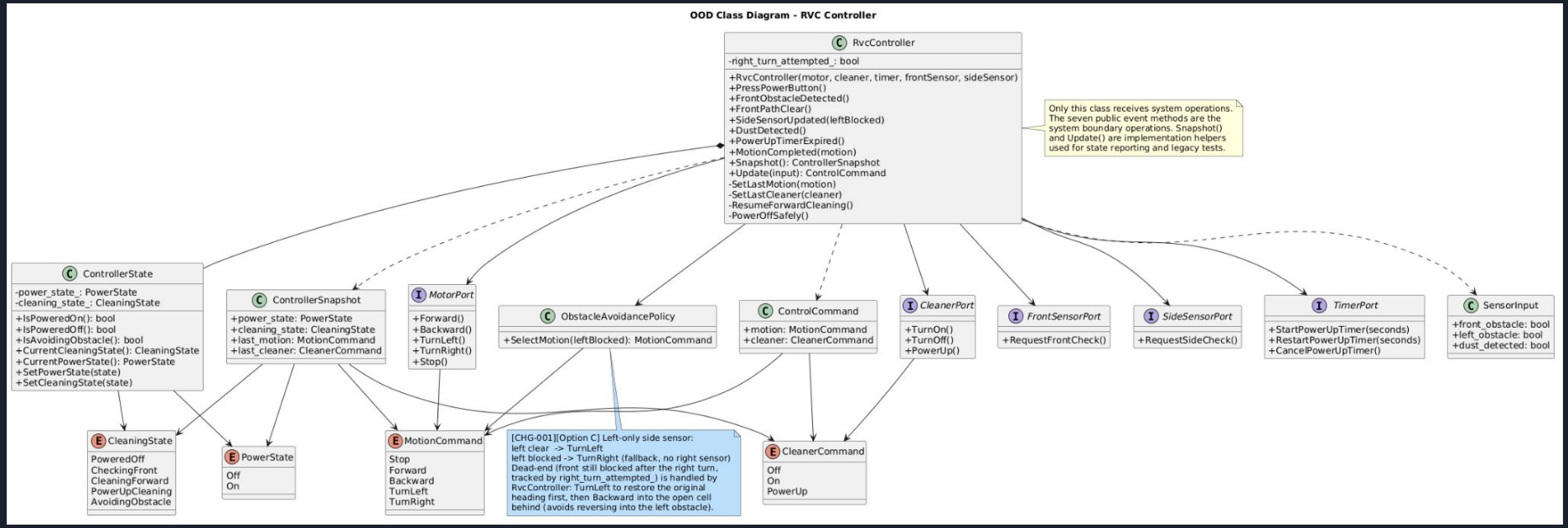
Domain Model(수정사항)



SD(수정사항)


OOD Sequence - Obstacle Avoidance
Front Sensor
Left Sensor
MotorEvent
User
RvcController
ControllerState
TimerPort
CleanerPort
ObstacleAvoidancePolicy
FrontSensorPort
SideSensorPort
MotorPort
FrontObstacleDetected()
IsPoweredOff()
false
CancelPowerUpTimer()
TurnOff()
SetCleaningState(AvoidingObstacle)
RequestSideCheck()
SideSensorUpdated(leftBlocked)
IsAvoidingObstacle()
true
SelectMotion(leftBlocked)
alt
[left side is clear]
TurnLeft
TurnLeft()
MotionCompleted(TurnLeft)
ResumeForwardCleaning()
Forward()
TurnOn()
[left side is blocked]
TurnRight
(변경) 우측 센서 없음. 좌측 막힘 -> 우회전 후 전방으로 우측 가용 여부 간접 확인. right_turn_attempted_ = true
TurnRight()
MotionCompleted(TurnRight)
RequestFrontCheck()
alt
[front clear after right turn]
(추가) 우측이 열려 있던 경우 -> 후진 없이 복귀
FrontPathClear()
ResumeForwardCleaning()
Forward()
TurnOn()
[front still blocked (dead-end)]
(변경) right_turn_attempted_ == true 이면 dead-end.
우회전으로 등진 좌측 장애물로 후진하지 않도록
먼저 좌회전으로 원래 방향 복원 -> 그다음 후진
FrontObstacleDetected()
TurnLeft()
MotionCompleted(TurnLeft)
(추가) 복원용 좌회전 완료(right_turn_attempted_==true)
-> 열린 뒤쪽으로 안전하게 후진. 플래그 해제
Backward()
MotionCompleted(Backward)
RequestSideCheck()
alt
[user powers off during avoidance]
PressPowerButton()
CancelPowerUpTimer()
Stop()
TurnOff()
SetCleaningState(PoweredOff)
SetPowerState(Off)

Class Diagram(수정사항)



OOI(수정사항)

CODE	`include/rvc/RvcController.h`, `src/RvcController.cpp`, `simulator/RvcSimulatorBridge.cpp`	✅ `SideSensorUpdated(leftBlocked)`, `right_turn_attempted_` dead-end 로직 (Option C: 좌회전 복원 → 후진)
UT	`tests/RvcControllerTest.cpp`	✅ 2센서 재작성 + dead-end 케이스 (Option C: 좌회전→후진→side.request)
System Test	`system_tests/suites/rvc_30_system_tests.json`, `system_tests/cases/*.json`	✅ 04~08·16·26·30 + cases 2개 2센서, dead-end에 TurnLeft 복원 스텝 추가
Simulator	`simulator/simulator_ui.py`	✅ 수동 버튼 2개, 좌측 단일 센싱, 06~08 시각 맵 재설계

수작업 방식 vs AI 에이전트 사용 방식 비교

수작업 방식 vs AI 에이전트 사용 방식

수정한 회피 로직 비교

우회전 케이스
- 수작업 방식 & AI 방식:
  - 좌측 막힘 → 우회전 → 전방 열림 → 전진

후진 케이스
- 수작업 방식 & AI 방식:
  - 좌측 막힘 → 우회전 → 전방 막힘 → 좌회전 → 후진 → 후진 후 재회피

⇒ 거의 동일한 회피 로직으로 작동

수작업 방식 vs AI 에이전트 사용 방식

SRS 비교

FR	수작업	AI
내용	분기 하나가 요구사항 (비슷한 FR들을 추적하기에 편리함)	행동 하나가 요구사항 (테스트가 편리함)
요구사항 변경사항 표시	X	CHG-001로 표시

NFR	수작업	AI
구조	PR와 OR로 분류	단순나열
내용	반응 시간, 상태 갱신 주기에 명확한 단위를 줌	흡입력 증가 시간에 명확한 단위를 줌
내용	사용자 제어 범위를 작성	회피 정책을 작성 (CHG-001 : 요구사항 변경과 관련된 내용을 표시)

수작업 방식 vs AI 에이전트 방식

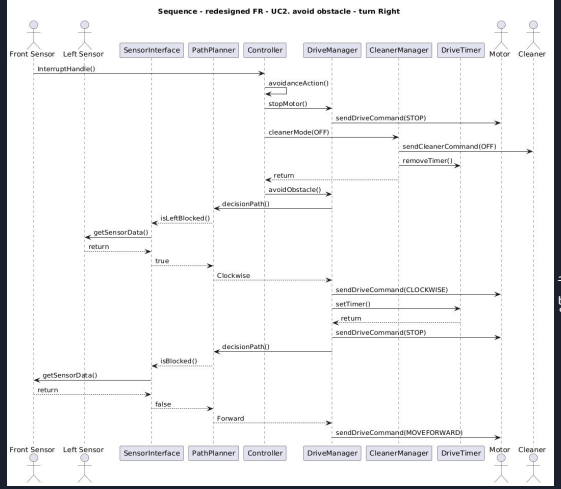
SD 비교

수작업 방식:

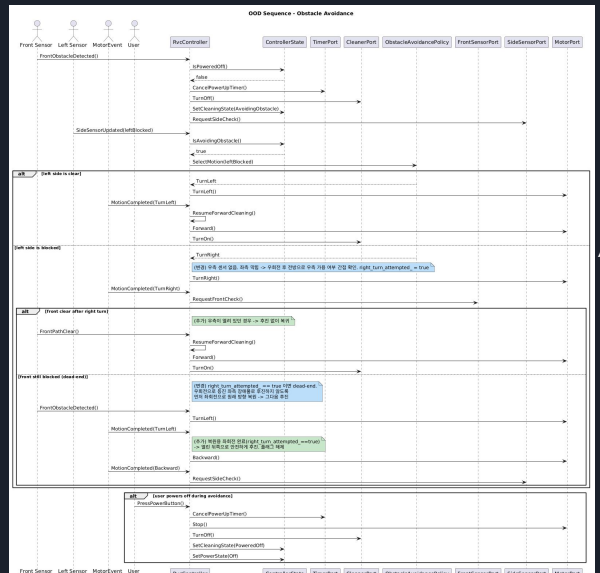
- 동기 시퀀스
- 한 use case = 긴 시퀀스 1개

AI 방식:

- 이벤트 시퀀스
- UC-003 = 짧은 조각 여러 개가 이어 붙음



수작업 방식



AI 에이전트 방식

수작업 방식 vs AI 에이전트 방식

System Operation 비교

수작업 방식:

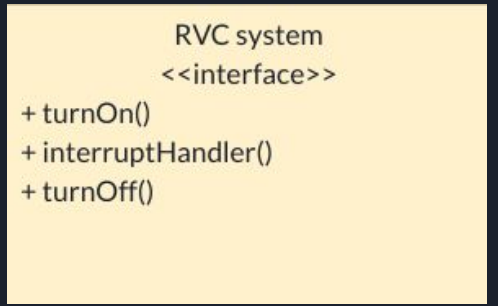
시스템 바운더리 = 외부를 User(버튼) + 하드웨어 인터럽트로
정의

회피, 먼지, 타이머는 외부 이벤트가 아닌 내부 로직으로 구현

AI 방식:

시스템 바운더리 = 외부를 User(버튼) + 센서, 타이머, 모터로 정의

외부액터 User, Front Sensor, Dust Sensor, Left Sensor, Timer, Motor(동작 완료시)에서 시스템 오퍼레이션(7개)을 호출



수작업 방식 (3개)

SO	누가/언제 호출
PressPowerButton()	사용자 전원 (on/off 토글)
FrontObstacleDetected()	전방 센서
FrontPathClear()	전방 센서
SideSensorUpdated(leftBlocked)	좌측 센서 (회피 중)
DustDetected()	머지 센서
PowerUpTimerExpired()	3초 타이머
MotionCompleted(motion)	모터/시뮬 ("회전·후진 끝")

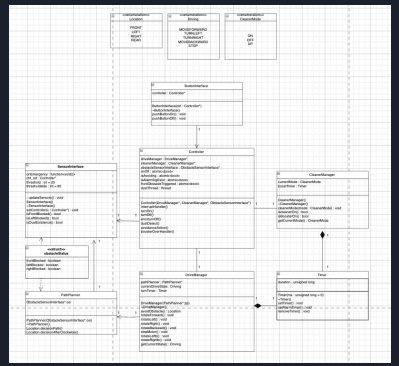
AI 에이전트 방식

수작업 방식 vs AI 에이전트 방식

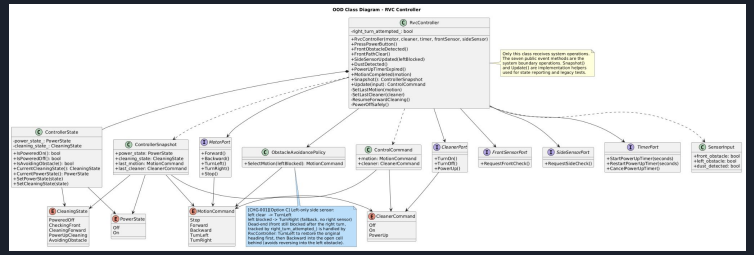
OOAD 중심에서 비교

Cursor: RvcController 클래스에 대부분의 기능이 집중되어 있음
수작업 방식: Controller, DriveManager, CleanerManager,

PathPlanner, ObstacleSensorInterface 즉 기능별 인터페이스를 분리해서 책임을 분산



수작업 방식



AI 에이전트 방식

수작업 방식 vs AI 에이전트 사용 방식

Unit Test

수작업:

- CMakeLists.txt
- UTest_ButtonInterface.cpp
- UTest_CleanerManager.cpp
- UTest_Controller.cpp
- UTest_DriveManager.cpp
- UTest_Mock.h
- UTest_ObstacleSensorInterface.cpp
- UTest_PathPlanner.cpp
- UTest_Timer.cpp

AI:

- CMakeLists.txt
- FakePorts.h
- RvcControllerTest.cpp

-수작업이 사용하는 함수들:
MOCK_METHOD(), EXPECT_EQ(), EXPECT_TRUE, EXPECT_CALL()

AI의 경우 log_.add() 활용
- 벡터가 같은지를 비교

1. 전체 동작 시퀀스 확인 가능
2. 어디서 의도가 어긋났는지 찾기 쉬움

```
TEST(RvcControllerTest, StartsForwardCleaningWhenFrontPathIsClear) {
  ControllerFixture fixture;

  fixture.controller.PressPowerButton();
  fixture.controller.FrontPathClear();

  const ControllerSnapshot snapshot = fixture.controller.Snapshot();
  EXPECT_EQ(snapshot.cleaning_state, CleaningState::CleaningForward);
  EXPECT_EQ(snapshot.last_motion, MotionCommand::Forward);
  EXPECT_EQ(snapshot.last_cleaner, CleanerCommand::On);
  EXPECT_EQ(fixture.log.calls,
            (std::vector<std::string>{"front.request", "motor.forward",
                                      "cleaner.on"}));
}
```

커서와 비교후 기존코드에서 개선할 점

기존 코드의 부족한점

하드웨어 api와 직접 통신하는 매니저 클래스를 준비함.
Pathplanner를 호출해 행동 판단 후 하드웨어 api와 직접 통신하는 방식으로 설계함.
하드웨어가 변경되었을 때 매니저 클래스만 수정하여 모듈화를 꾀함.
하지만 해당 매니저 클래스 안에 구체적인 하드웨어 시퀀스가 명시적으로 적혀 있었음. 따라서 단일책임원칙을 완벽히 구현하지 못했고, 책임이 매니저 클래스에 집중됨.

```
Location DriveManager::avoidObstacle() {
    while (true) {
        stopMotor();

        Location turn = pathPlanner->decisionPath();

        // Sequence - avoid obstacle - turn Left
        if (turn == Location::LEFT) {
            std::cout << "[System] Left side cleared! Escaping to Left." << std::endl;

            rotateLeft();
            turnTimer.setTimer();
            stopMotor();

            rotateForward();
            return Location::LEFT;
        }
```

커서의 고도화된 객체지향 설계

커서의 코드는 하드웨어 API와 통신하기 직전에 포트(Port)라는 추상화 된 인터페이스 레이어를 하나 더 추가했습니다. 단순히 명령전달에만 집중함과 동시에 하드웨어의 세부로직을 하나로 묶어 캡슐화 하여 전달함.

이로인해 ObstacleAvoidancePolicy가 센서 데이터 바탕으로 순수하게 방향만 판단한 후, 포트에 [illegible] 포트 너머의 하드웨어 객체가 전담하

```
// [CHG-001] Right Sensor 제거: 좌측 센서 하나로 회전 방향만 결정한다.
// 좌측이 비면 좌회전, 막혀 있으면 우회전(우측 센서가 없는 fallback).
MotionCommand ObstacleAvoidancePolicy::SelectMotion(
    const bool left_blocked) const {
  return left_blocked ? MotionCommand::TurnRight : MotionCommand::TurnLeft;
}
```

```
void RvcController::SideSensorUpdated(const bool left_blocked) {
  if (!state_.IsAvoidingObstacle()) {
    return;
  }

  // 1. 행동 결정 (로직에게 질의)
  const MotionCommand motion = avoidance_policy_.SelectMotion(left_blocked);

  // 2. 하드웨어 통신 (포트를 통해 API 호출)
  if (motion == MotionCommand::TurnRight) {
    motor_.TurnRight(); // <- 하드웨어와 연결된 톨게이트(인터페이스) 호출!
    right_turn_attempted_ = true;
  } else {
    motor_.TurnLeft();  // <- 하드웨어와 연결된 톨게이트(인터페이스) 호출!
    right_turn_attempted_ = false;
  }

  SetLastMotion(motion);
}
```

이와 같은 설계의 장점
- 실제 현업 임베디드 소프트웨어 관점에서

1. 하드웨어를 교체할 때의 수월함 (ex: a 회사의 모터 -> b 회사의 모터)
   - 직접 작성한 코드 방식: 실제 모터의 스펙, 통신 방식 등에 맞게 driveManager의 코드 본문을 수정해야 함.
   - 포트 형식의 경우: 모터의 종류마다 클래스를 상속하여 새로운 클래스로 관리 가능. 본문의 내용은 변하지 않고 일관된 형태를 유지, 상속된 클래스에 모터의 개별적 특성에 맞게 작성, 수정하면 된다.
2. 유닛 테스트 작성, 실행에 더 쉬워짐
   - 모킹 객체 준비가 더 쉬워 테스트 작성과 실행이 더 용이함.

기존코드의 해당 부분을 개선하려면?

기존 코드의 하드웨어 동작 시퀀스를 별도의 클래스로 분리하여 책임을 분산하고, 매니저 클래스는 행동방향 결정과 하드웨어 통신 자체만을 담당하는 클래스로 수정 하여 객체지향 달성을 고도화 할 수 있음.

수작업 방식 vs AI 에이전트 방식

느낀점

AI 코딩 장점:

압도적인 초안 작성 속도, 빠른 일괄 수정. 원하는 사항의 검색, 추적 바로 가능.
프로젝트의 구현, 산출물 정렬을 도와주며 체계적 관리 가능.
도메인 지식 등에 관해 부족한 부분들을 AI로 보충 가능.

AI 코딩 단점:

사람이 계속해서 방향을 잡아주고 수정을 요구해야함.
사전 지식, 검증 없이 진행 할 경우, 방대해진 프로젝트에 그럴듯한 내용으로만 채워짐
사용자가 어느정도의 지식이 있는지에 따라서 설계부터 구현까지 시간, 퀄리티 면에서 큰 차이가 날 것으로 생각됨

감사합니다.