Object-Oriented Analysis and Design 3227

객체지향개발방법론

1팀 6주차 발표

Traditional Coding vs. Vibe Coding

202211327 윤승모

202211261 김강민

202212353 문서인

202011362 정상현

CONTENTS

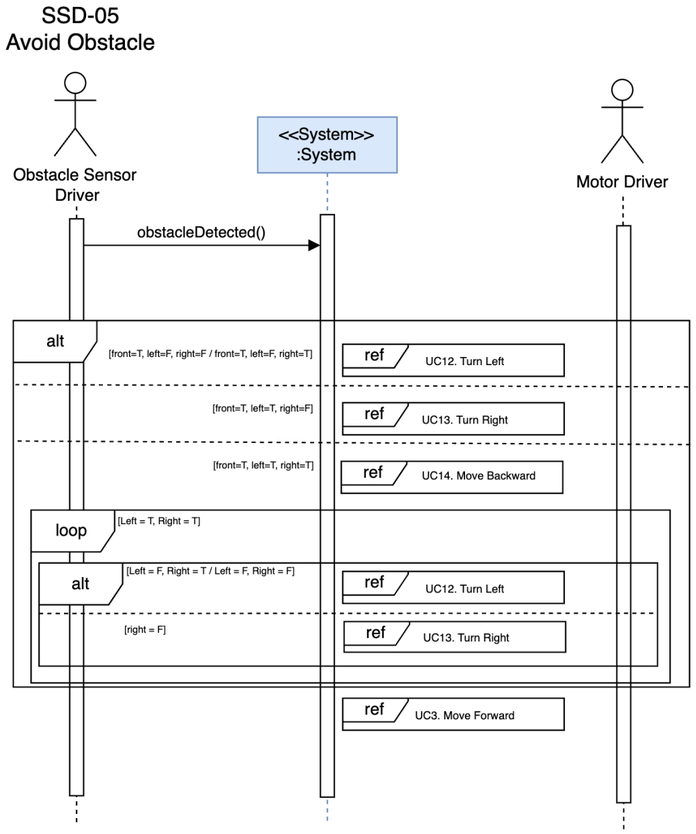
Part1 SDD

: Use Cases Descriptions(Refined) UC5

항목	내용
Use Case	UC5. Avoid Obstacle
Actor	Obstacle Sensors
Purpose	장애물 감지 시 방향을 전환하여 장애물을 회피한다
Overview	Obstacle Sensors가 장애물을 감지하면 signal을 System에 전송하고, System은 회피 방향을 결정한다.
Type	Essential
Cross Reference	FR-UC5-01, FR-UC5-02, FR-UC5-03
Pre-Requisites	시스템이 Normal 또는 Boost 모드 상태여야 한다.
Typical Courses of Events	1. Obstacle Sensors가 장애물을 감지한다. (A) 2. Obstacle Sensors가 obstacle signal을 시스템에 전송한다. (A) 3. 시스템이 장애물 회피 방향을 결정한다. (S) 4. 시스템이 결정된 방향으로 이동 명령을 전달한다. (S)
Alternative Courses of Events	감지된 장애물 방향에 따라 다음과 같이 회피한다. (좌회전 우선) front=T, left=F~~, right=F~~[삭제] → Turn Left ~~front=T, left=F, right=T → Turn Left~~[삭제] front=T, left=T~~, right=F~~[삭제] → Turn Right (이후, front=T면 Move Forward, front=F면 Turn Left 이후, Move Backward)[추가] ~~front=T, left=T, right=T → 왼쪽, 오른쪽중 장애물이 없을 때까지 Move Backward후 해당 방향으로 Turn~~[삭제]
Exceptional Courses of Events	N/A

Part1 SDD

: System Sequence Diagrams(원본) UC5



1. Obstacle Sensors가 장애물을 감지한다. (A)

2. Obstacle Sensors가 obstacle signal을 시스템에 전송한다. (A)

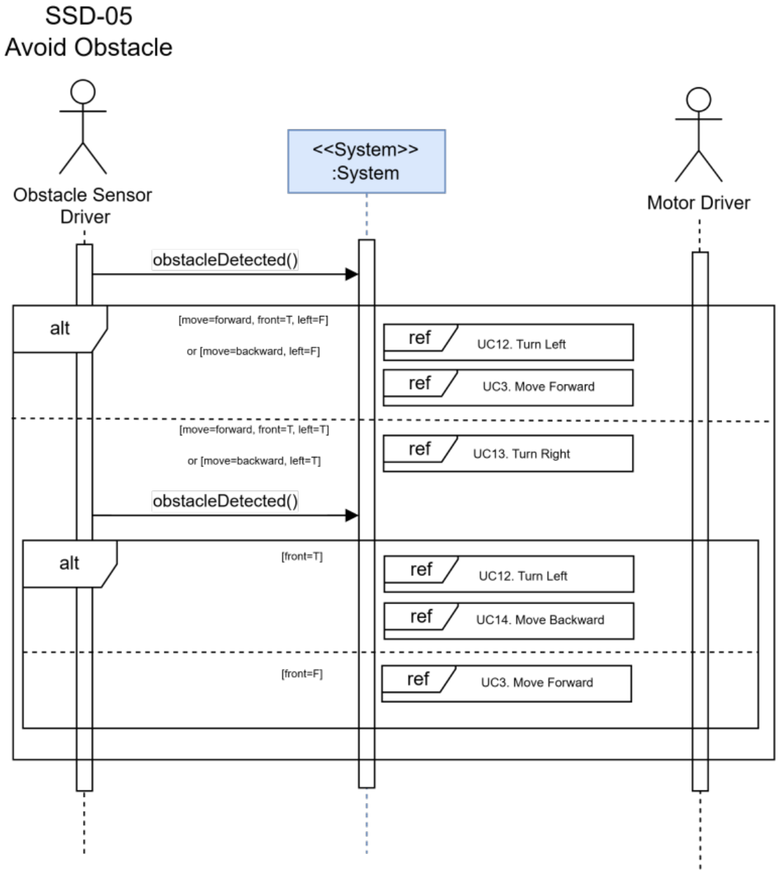
3. 시스템이 장애물 회피 방향을 결정한다. (S)

4. 시스템이 결정된 방향으로 이동 명령을 전달한다. (S)

Part1 SDD

: System Sequence Diagrams(수정본)

UC5



1. Obstacle Sensors가 장애물을 감지한다. (A)

2. Obstacle Sensors가 obstacle signal을 시스템에 전송한다. (A)

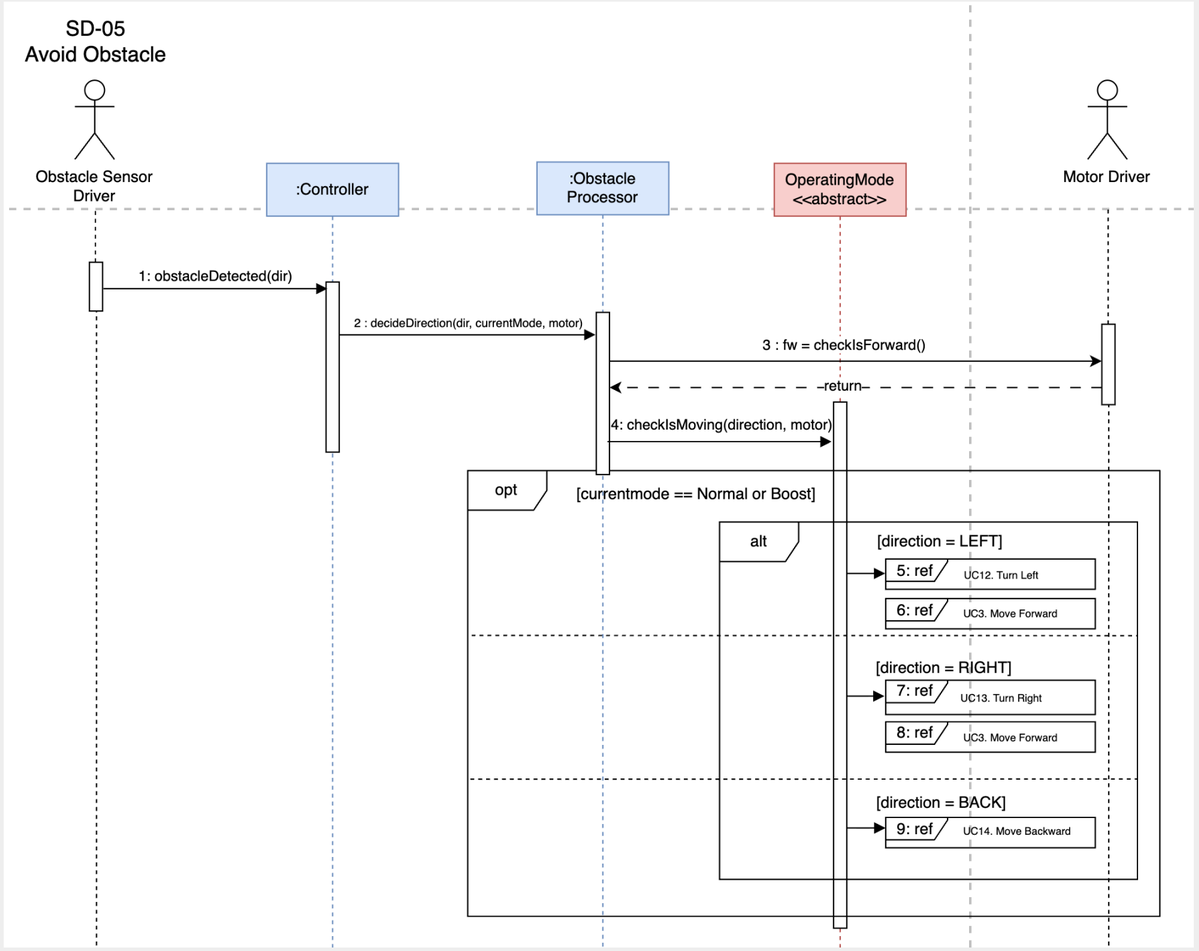
3. 시스템이 장애물 회피 방향을 결정한다. (S)

4. 시스템이 결정된 방향으로 이동 명령을 전달한다. (S)

Part1 SDD

: Sequence Diagrams(원본) UC5

SD-05 AVOID OBSTACLE



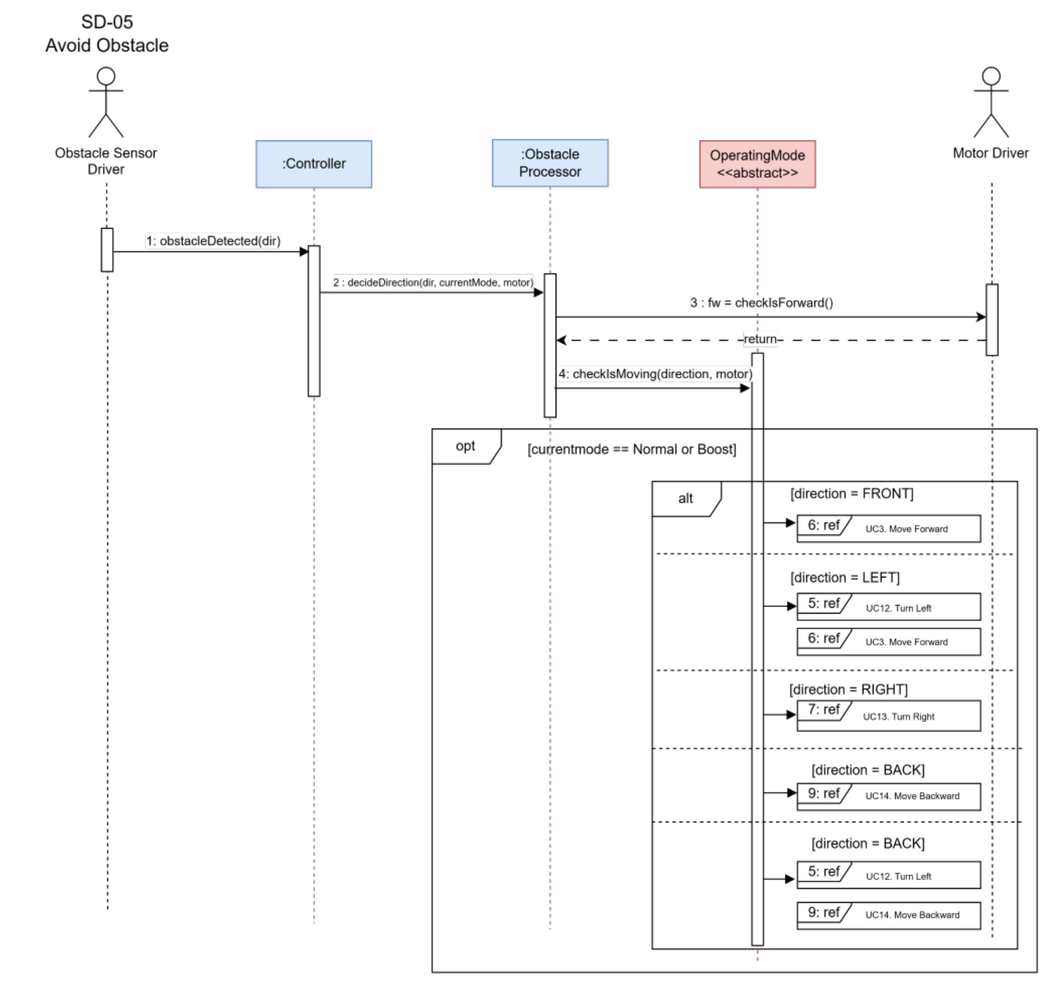
fw가 false면 obstacle processor가 front sensor는 고려하지 않는다.

Part1 SDD

: Sequence Diagrams(수정본)

UC5

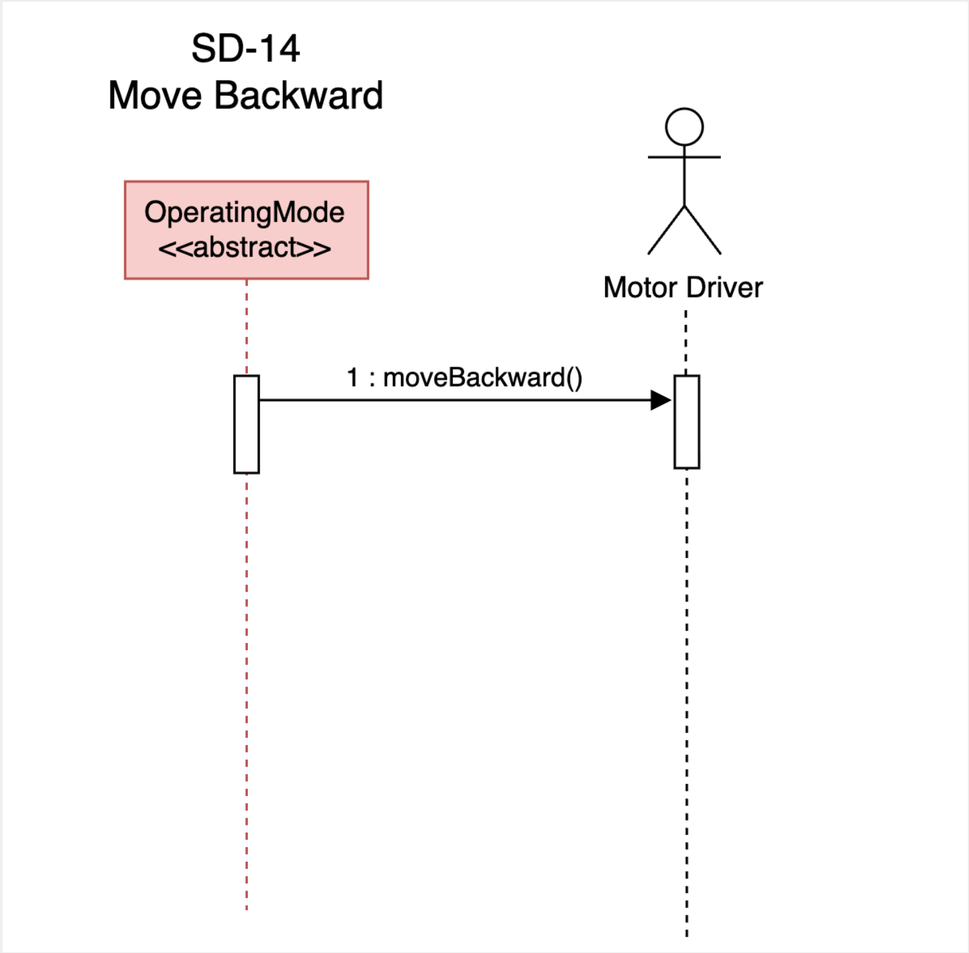
SD-05 AVOID OBSTACLE



decide함수에서 내부 플래그로 우회전 상태 확인(우회전시 플래그1, forward, turn left시 플래그0)[추가]

: Sequence Diagrams(원본) UC14

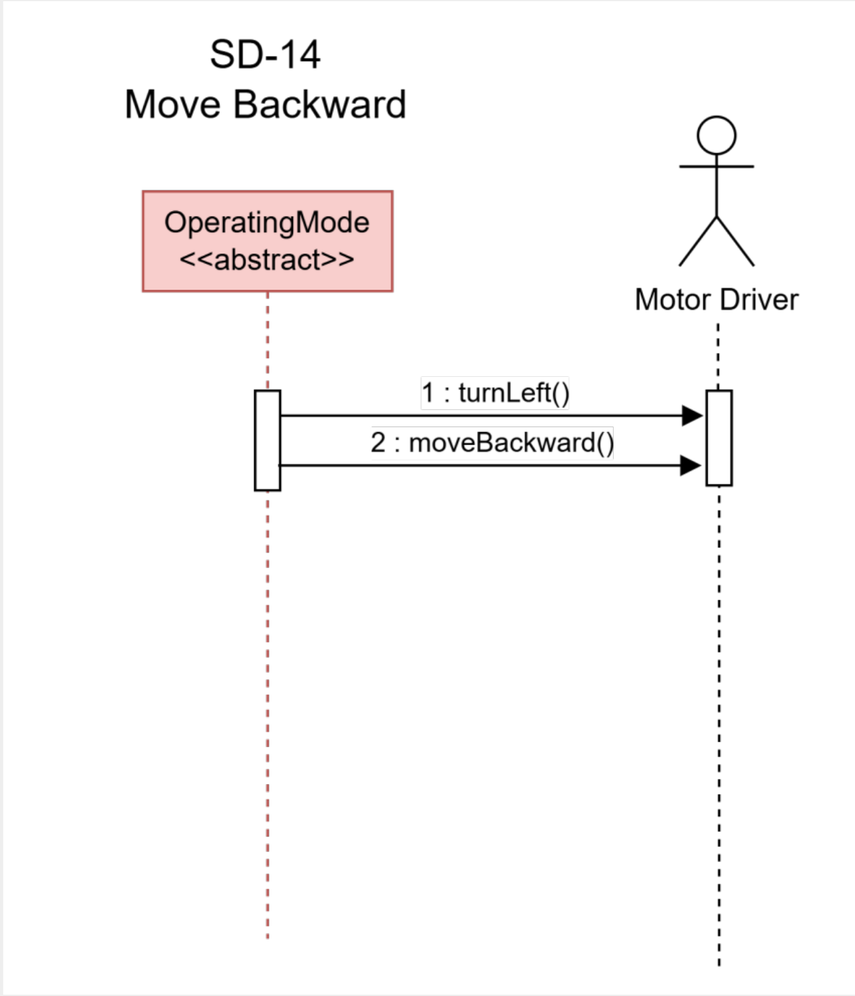
SD-14 MOVE BACKWARD



: Sequence Diagrams(수정본)

UC14

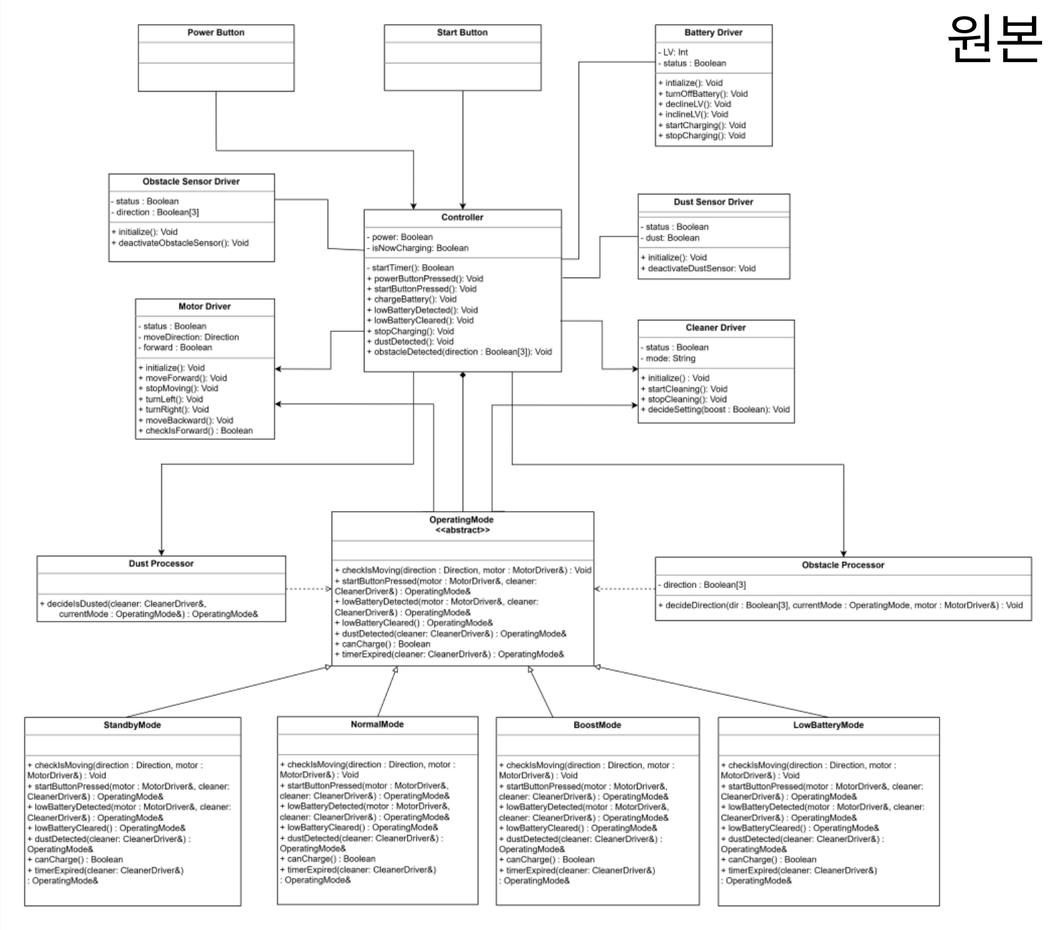
SD-14 MOVE BACKWARD



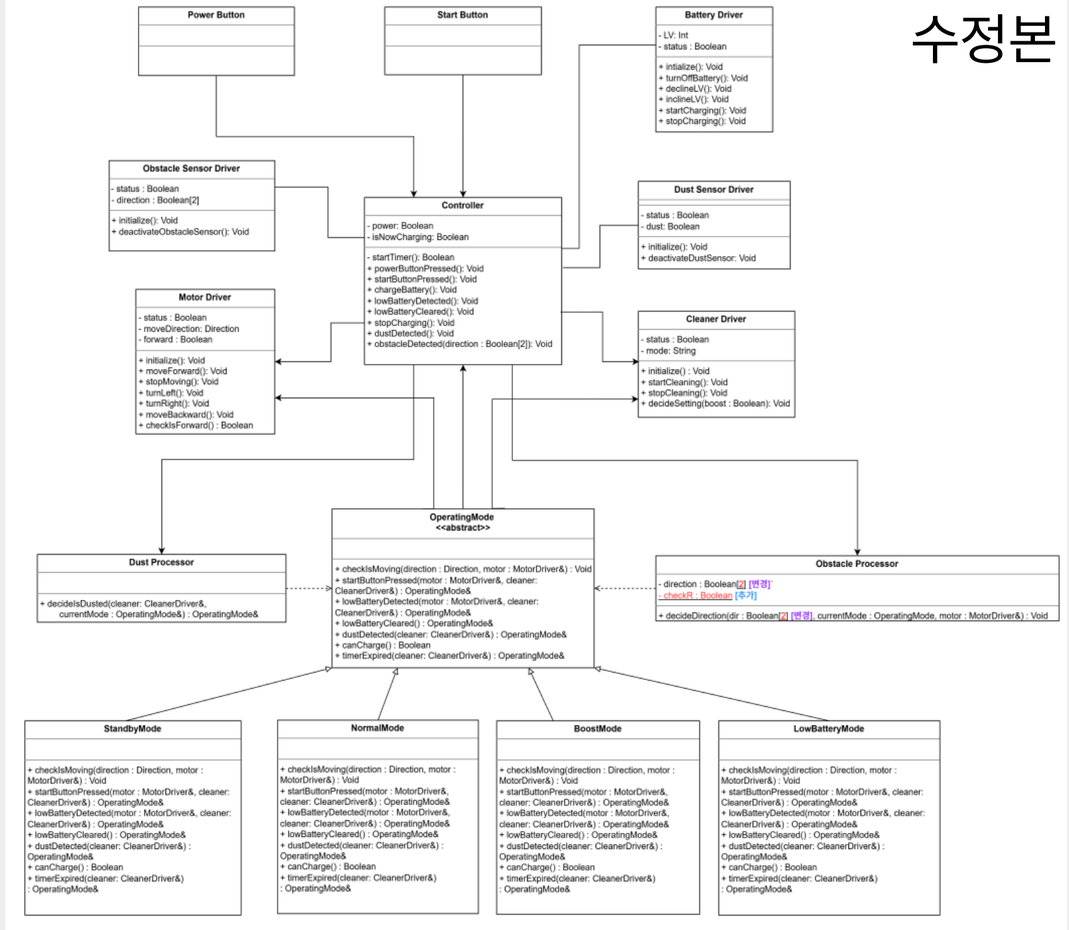
Part1 SDD

: Class Diagrams(원본&수정본)

원본



수정본



수정사항 (1)Obstacle 관련 장애물 Boolean 배열을 3→2로 줄였습니다(Controller,Obstacle Sensor Driver, Obstacle Processor 속 attribute 수정)[변경]

Part1 Program(Code)

: Program(Code) 변경사항

변경 방식

- 컨트롤러, 장애물프로세서, 모드, TC, 시뮬레이터 수정
- branch: rightsensormissT1

핵심 변경

- obstacleDetected 입력: 3개 -> 2개
- right sensor 직접 입력 제거
- RIGHT는 감지값이 아닌 회피 후보 방향으로 유지

수정 영향 파일

- include/rvc/Controller.hpp
- src/rvc/Controller.cpp
- src/rvc/ObstacleProcessor.cpp
- src/rvc/Modes.cpp

Part1 Program(Code)

: Controller API 변경

[Before]

```
void obstacleDetected(const bool direction[3]);

std::array<bool, 3> dir = {
direction[0], direction[1], direction[2]
};
```

[After]

```
void obstacleDetected(const bool direction[2]);

std::array<bool, 2> dir = {
direction[0], direction[1]
};
```

[Summary] obstacleDetected()는 RVC system boundary로 들어오는 true System Operation이다. 단, 입력 정보가 front/left로 줄어들었다.

Part1 Program(Code)

: ObstacleProcessor 판단 방식 변경

[Before]

- front clear → FRONT
- left clear → LEFT
- right clear → RIGHT
- all blocked → BACK

[After]

- front+left blocked → RIGHT 후보
- checkR=true로 다음 sensor event를 재확인
- front clear → FRONT
- front blocked → BACK

[Key logic]

```
if (checkR) {
direction = !dir[0] ? Direction::FRONT : Direction::BACK;
checkR = false; }
```

Part1 Program(Code)

: moveBackward 관련 변경 포인트

Mode mapping

- Direction::RIGHT → turnRight()
- Direction::BACK → turnLeft() 후 moveBackward()
- moveBackward는 System Operation이 아니라 output command

MotorDriver

```
void MotorDriver::moveBackward()
{
this->forward = false;
}
```

[Key point]

right sensor가 없어도 RIGHT 방향 자체는 motor direction으로 남는다.

다만 RIGHT는 sensor input이 아니라 ObstacleProcessor가 선택하는 회피 후보 방향이다.

Part1 Program(Code)

: UT / ST (수정본)

[수정사항]

- ST) ObstacleDetected 1~10 (TC 24~33)
  & ObstacleDetectedBoost 1~10(TC 34~43)
  에서 Right Sensor 관련 값 제거
- UT) ControllerObstacleDetectedTest에서 Right Sensor 관련 값 제거
- Simulator에서 Right Sensor 관련 값 제거

[코드 수정 예시]

Right sensor를 통한 장애물이 없는 Left,Front 장애물만 존재하는 상황으로 Test,Simulator 변경

Part2 vibe

: Documents (SRS / SDD)(수정본)

핵심 : "3센서 동시 판단" → "2센서(front/left) 입력 + 행동 후 재측정" 모델로 전환.
오른쪽은 회전해서 전방 센서로 다시 본다.

삭제: bool right state + 입력 시그니처에서 right 제거

변경: System Operation obstacleDetected(all) → obstacleDetected(front/left) + front recheck

추가: P12에 UC13 Turn Right use case

Part2 vibe

: Code (수정본)

[Before]

- front clear → FRONT
- left clear → LEFT
- right clear → RIGHT ← 센서값 직접 확인
- all blocked → BACK
- 한 함수가 한 번에 4방향 결정

[After]

- front+left blocked → RIGHT
  ("우회전 후 재측정" 신호)
- right 센서 분기 제거
- Backward 판단은 재측정 함수로 이동

[Key logic]

```
decideDirectionAfterRightTurn():
    return !front ? Direction::Forward : Direction::Backward;
```

Part2 vibe

: UT / ST (수정본)

[Before]

UT : 104개
- obstacle 입력 : front / left / right (3개)
- direction[3]의 right 값으로 회피방향 검증

ST : 42개
- all obstacle 상황에서 backward 이동 확인

[After]

UT : 106개

ST : 42개
- right : 우회전 후 front 재측정으로 검증

1) front clear : move forward

2) front blocked : move backward

[Key logic]

```
SendObstacleWithRecheck(
    front, left,frontAfterRightTurn,
    leftAfterBackward,frontAfterBackwardRightCheck
)
```

Part2 vibe

: Simulator (수정본)

[Before]

- map에서 3방향 obstacle 직접 계산
- 오른쪽 칸 상태를 right sensor처럼 전달
- SensorSnapshot: obstacleBlocked[3] 사용
- Controller의 motor 방향을 map 이동방향으로사용

[After]

- map sensor input : front / left 2개
- SensorSnapshot에 right-turn / backward recheck 상태추가
- RvcAdapter : heading ↔ 실제 이동방향 구분
- 후진 recovery 중 left 우선 재확인 흐름 유지

[Key logic]

```
frontAfterRightTurnBlocked = blockedInDirection(rightOf(heading))
movementDirection_ = rightOf(headingBeforeSensors)
```

Part3 비교분석

: 비교분석 동일 요구사항 변경(Right Sensor 제거)을 두 방식에 독립 적용

변경 범위 (주요 변경 파일 기준, git diff --numstat 기반)

산출물	Team A (전통)	Team B (vibe)
Production code	+32 / −16	+254 / −36
Unit test	+85 / −56	+115 / −49
System test	+170 / −138	+10 / −2
Simulator	+136 / −78	+217 / −40
합계	+423 / −288	+596 / −127

Part3 비교분석

: 설계 방식의 차이

양 팀 모두 front/left 2센서 → 둘 다 막히면 우회전 → 전방 재확인 → 막히면 후진 → 후진 중 left 우선 재탐색

	Team A (전통)	Team B (vibe)
유지보수 방식	기존 구조를 유지하며 최소 범위 수정	AI가 요구사항 흐름을 명시적 상태/함수로 확장
장점	변경량이 적고 기존 코드 흐름을 보존	요구사항 의도가 코드와 테스트에 잘 드러남
위험성	변경 의미가 내부 상태(ex. checkR)에 묻힐 수 있음	구조가 커지고 추가 검증이 필요함

Part3 비교분석

: 설계 방식의 차이

Team A(전통)

```
        if (checkR)
        { // 우회전 후 전방 센서 확인 상황
            if (!dir[0])
                direction = Direction::FRONT;
            else
            {
                direction = Direction::BACK;
            }
            checkR = false;
        }
        else if (fw)
        { // 전진상황
            if (!dir[0])
                direction = Direction::FRONT;
            else if (!dir[1])
                direction = Direction::LEFT;
            else
            {
                direction = Direction::RIGHT;
                checkR = true;
            }
        }
        else
        { // 후진(장애물 탈출)
            if (!dir[1])
                direction = Direction::LEFT;
            else
            {
                direction = Direction::RIGHT;
                checkR = true;
            }
        }
        currentMode.checkIsMoving(direction, motor);
    }

} // namespace rvc

```

Team B (vibe)

```
#include "../include/rvc/ObstacleProcessor.hpp"

#include "../include/rvc/ObstacleSensorDriver.hpp"

namespace rvc
{

// [변경] right sensor 입력을 읽지 않고 front/left만으로 1차 회피 방향을 결정한다.
Direction ObstacleProcessor::decideDirection(const ObstacleSensorDriver& obstacleSensorDriver) const
{
    if (!obstacleSensorDriver.front)
    {
        return Direction::Forward;
    }

    if (!obstacleSensorDriver.left)
    {
        return Direction::Left;
    }

    return Direction::Right;
}

```

Part3 비교분석

: Insight

[각 방식에서 드러난 결함·위험]

- 전통: System Test 20개를 수동 일괄 수정 → 휴먼 에러 여지 (3센서 시나리오가 20개 파일에 흩어져 있어 전파됨)
- Vibe: 1차 산출물에 후진 후 전진 버그 존재, 추가 커밋으로 별도 수정 → AI 결과물도 검증 후 수정 사이클이 필요했음

  → 양쪽 다 "한 번에 완성"이 아니었고, 비용의 발생 위치만 달랐다.

[비용은 사라지지 않고 이동]

- 전통: 초기 패치는 적음(+32) — 대신 변경 전파(ST 20개)와 검증을 사람이 떠안음
- Vibe: 초기 비용 큼(+254) — 대신 명시적 상태·추적성을 한 번에 산출 + 결함 수정 사이클 발생

  → 핵심: "어느 쪽의 변경 리소스가 적은지"가 아니라 **비용의 발생 시점** (: 초기 작성에서 발생 ↔ 변경 전파·검증 시점)

Part3 비교분석

: 핵심 인사이트 AI Agent로 구현하되, 유지보수 가능한 코드로 남는 방법

[명시적 상태·구조는 AI의 강점]

- Vibe는 동작을 숨은 의존성(모터 방향) 대신 명시적 플래그로 드러내 검증·인수인계가 쉬움

["그럴듯한 1차 산출물"은 반드시 사람이 검증]

- Vibe도 후진 버그를 사람이 잡아 수정. AI 산출물은 "완성"이 아니라 "검토 대상"

["최소 변경으로"를 명시적으로 제약]

- AI는 "이 변경을 최소로 끝내라"가 아니라 "이 변경을 잘 해라"로 해석하는 경향.
- 소규모 패치로 끝낼 수도 있는 변경이, 명시적 설계를 요구시에 구조 확장(+254)로 이어질 수 있음

[Insight]

단순 패치 속도는 전통방식, 명시적 구조·검증 가능성은 Vibe.
AI Agent에게 맡기되 사람이 결함과 과설계만 잡아내면 — 속도와 유지보수성을 동시에 가져갈 수 있다.

Part4

: 변경사항을 적용해 본 경험

[UP/OOAD]

- 기존 구조 안에서 작은 패치로 요구사항 변경을 진행 할 수 있었음
- 변경 영향이 System Test 20개 처럼 여러 산출물에 퍼져 있어, 사람이 직접 전파 및 검증
- 설계의도는 코드 안에 압축 → 빠르게 수정 가능

→ 유지보수에서 중요한 것이 '적은 패치로 코드 수정'이 아닌, 변경 요구사항의 테스트 및 문서까지의 일관적 반영여부임을 느낌

[Vibe Coding]

- AI로 변경 요구사항을 코드, UT, ST, simulator까지 빠르게 확장 적용 가능함을 확인함
- right sensor 단순삭제가 아닌, 삭제 후의 명시화된 동작을 프롬프트 하는 것의 중요성을 느낌
- 1차 구현 이후 예상치 못한 버그를 수동 테스트로 확인, 수동 테스트의 중요성을 체감함

→ AI는 변경 적용 속도를 높여주지만, 완성도는 테스트와 사람의 검증으로 결정된다는 것을 느낌

감사합니다