

RVC 제어 소프트웨어

구조적 분석(SA) 완성본

Homework #6: Structured Analysis for RVC Control Software

202211285 김지태

202411348 이시현

202111373 정일혁

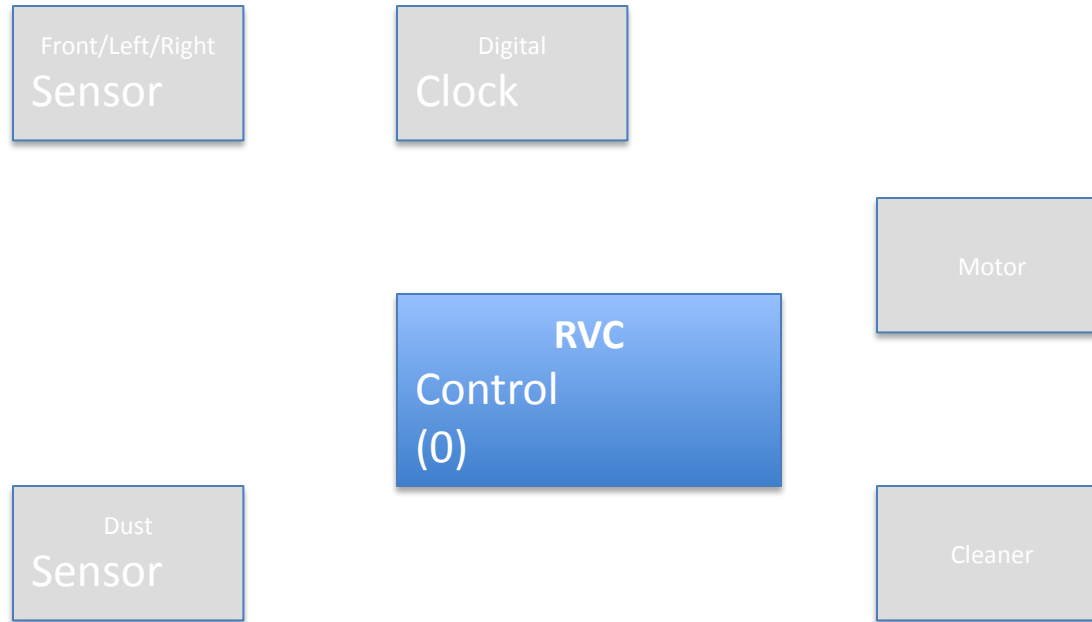
목차 (Table of Contents)

- 1. 개요 및 문제점 분석
- 2. System Context Diagram
- 3. DFD Hierarchy (Level 0~4)
- 4. FSM 설계 (Version 1 & Version 2)
- 5. Process Specifications (전체)
- 6. Data Dictionary (완전판)
- 7. 문제 해결 검증

1. 개요 및 문제점 분석

- 해결해야 할 문제점들:
- ✕ Stop 상태 Deadlock
 - Pause + Backwarding 상태로 분리하여 탈출 경로 제공
- ✕ Dust 미고려
 - Dust Cleaning 상태 추가 및 CN2 FSM 분리
- ✕ 좌/우 회피 우선순위 없음
 - 좌회전 우선 정책 명시
- ✕ Backward 미고려
 - All Blocked 상황에서 Backward 명령 추가
- ✕ Interface 불일치
 - Motor/Cleaner Interface를 일관성 있게 재설계

2. System Context Diagram (SCD)

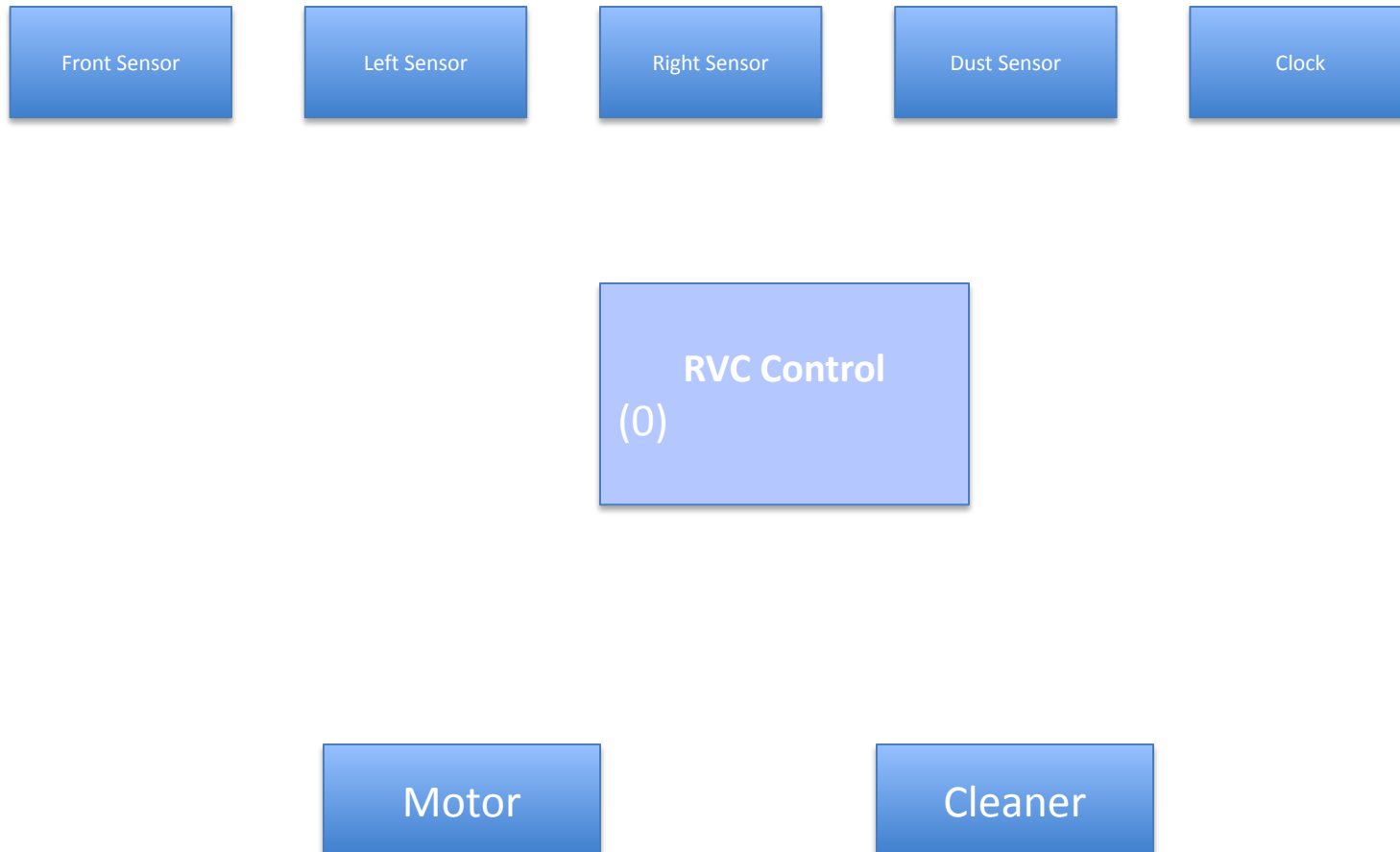


※ SCD는 시스템과 외부 환경의 경계를 명확히 정의합니다.

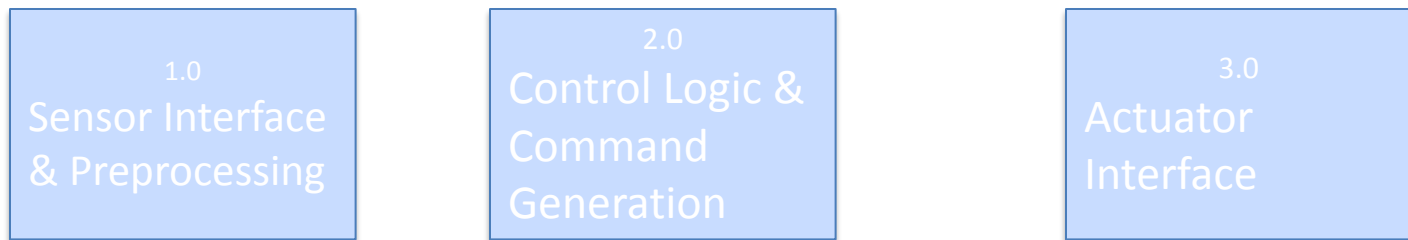
2-1. Event List (SCD 상세)

- **[Input] Front_Sensor_Input**
 - 전방 장애물 감지 (Boolean, Interrupt)
- **[Input] Left_Sensor_Input**
 - 좌측 장애물 감지 (Boolean, Periodic)
- **[Input] Right_Sensor_Input**
 - 우측 장애물 감지 (Boolean, Periodic)
- **[Input] Dust_Sensor_Input**
 - 먼지 존재 감지 (Boolean, Periodic)
- **[Input] Tick**
 - 주기적 시간 신호 (Temporal Event)
- **[Output] Direction**
 - 모터 방향 명령 (Forward/TurnLeft/TurnRight/Stop/Backward)
- **[Output] Clean_Command**
 - 청소기 제어 명령 (On/Off/Power-Up)

3. DFD Level 0: 시스템 개요



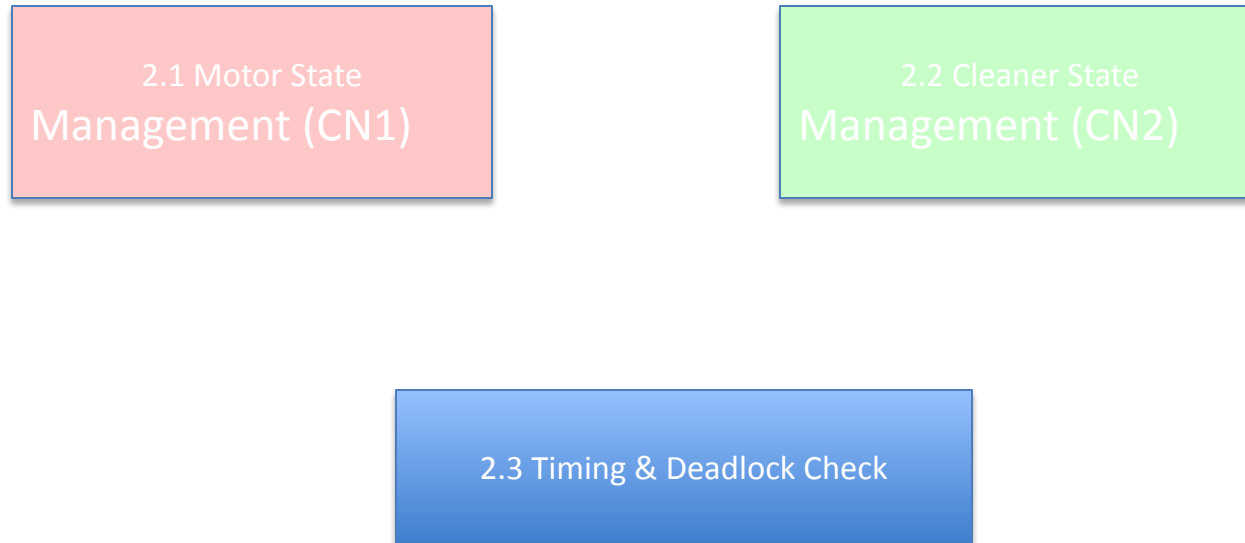
3-1. DFD Level 1: 주요 기능 분해



주요 데이터 흐름:

- 1.0 → 2.0: Obstacle_Location, Dust_Existence
- 2.0 → 3.0: Motor_Command, Cleaner_Command
- 2.0 내부: Cleaner_Trigger ↔ Motor_Status (상호작용)

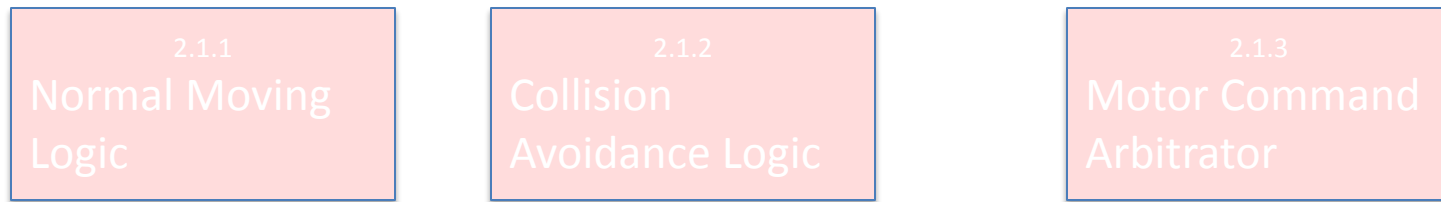
3-2. DFD Level 2: Control Logic 상세 (2개 CN)



CN 간 상호작용:

- CN2 → CN1: Cleaner_Trigger (청소 중 정지 요청)
- CN1 → CN2: Motor_Status (이동 상태 확인)
- 2.3: Tick 기반 Deadlock 감지 및 탈출 신호 생성

3-3. DFD Level 3: Motor State 상세



데이터 흐름:

- 2.1.1: Forward_Command 생성
- 2.1.2: Obstacle_Location → Avoidance_Command (우선순위 적용)
- 2.1.3: Forward, Avoidance, Cleaner_Trigger 중 최종 결정

3-4. DFD Level 4: Collision Avoidance 상세



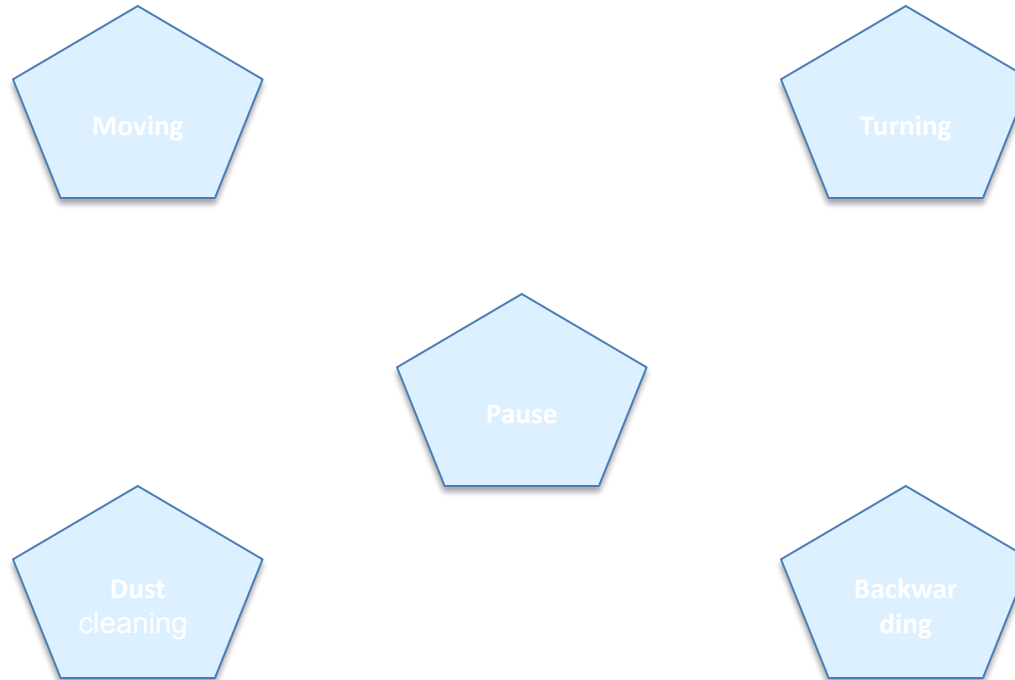
회피 로직:

- 2.1.2.1: Front 센서 체크
- 2.1.2.2: Left 우선 정책 적용 (Left Free → Turn Left, else → Turn Right)
- 2.1.2.3: 전방향 막힘 → Backward 또는 Pause
- 2.1.2.4: 최종 회피 명령 통합 출력

4. FSM Version 1: 단일 FSM (문제 해결)

- 상태 정의:
- • **Moving**: 정상 전진 및 청소 중
 - 명령: Forward / On
- • **Turning**: 장애물 회피 회전 중
 - 명령: TurnLeft/Right / On
- • **Backwarding**: 후진 중 (전방향 막힘)
 - 명령: Backward / On
- • **Dust Cleaning**: 먼지 집중 청소 (정지)
 - 명령: Stop / Power-Up
- • **Pause**: 일시 정지 (탈출 대기)
 - 명령: Stop / On

4-1. FSM Version 1: 상태 전이도



주요 전이: *Moving ⇌ Turning ⇌ Backwarding, Moving → Dust Cleaning, Any → Pause*

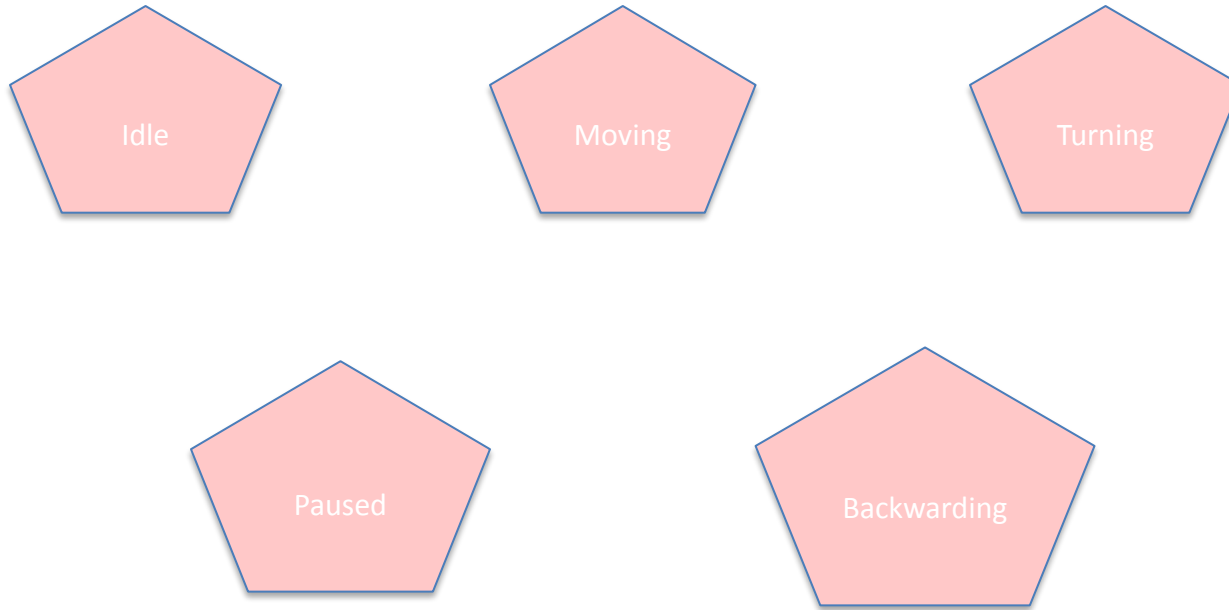
4-2. FSM Version 1: 상태 전이 테이블

| 현재 상태 | 전이 조건 | 다음 상태 | 명령 |
|-------------|----------------|--------------------|---------------|
| Moving | Front Obstacle | Turning | Stop/On |
| Moving | Dust Detected | Dust Cleaning | Stop/Power-Up |
| Turning | All Blocked | Backwarding | Backward/On |
| Turning | Left Free | Moving (Turn Left) | Forward/On |
| Backwarding | Back Free | Turning | Stop/On |

4-3. FSM Version 2: 2개 Control Node (분리)

- 설계 개선 목표: 일관성 및 유지보수성 향상
- CN1: Motor Control FSM
 - 상태: Idle → Moving → Turning → Backwarding → Paused
 - 역할: 이동 및 회피 전담
- CN2: Cleaner Control FSM
 - 상태: Off → Normal Cleaning → Power-Up Cleaning
 - 역할: 청소 모드 및 먼지 감지 전담
- 상호작용:
 - CN2 Power-Up 필요 시 → CN1에 Paused Trigger 전송
 - CN1 Moving 상태 → CN2에 Motor_Status 전달

4-4. CN1: Motor Control FSM



주요 전이:

- Idle → Moving (시작), Moving → Turning (장애물), Turning → Backwarding (전방향 막힘)
- Paused ← Cleaner_Trigger (CN2 요청)

4-5. CN2: Cleaner Control FSM



주요 전이:

- Off → Normal Cleaning (시작)
- Normal → Power-Up (먼지 감지) → Paused Trigger to CN1
- Power-Up → Normal (청소 완료)

5. Process Specifications (프로세스 명세)

- 각 DFD 프로세스별 상세 명세
 - 1.0 Sensor Interface & Preprocessing
 - 2.0 Control Logic & Command Generation
 - 3.0 Actuator Interface
 - 2.1 Motor State Management (CN1)
 - 2.2 Cleaner State Management (CN2)
 - 2.3 Timing and Deadlock Check
 - 2.1.1 Normal Moving Logic
 - 2.1.2 Collision Avoidance Logic
 - 2.1.3 Motor Command Arbitrator
 - 2.1.2.1 ~ 2.1.2.4 (회피 상세)

5-1-1. Process Spec: 1.0 Sensor Interface

Ref. No.: 1.0

Name: Sensor Interface & Preprocessing

Input:

- Front_Sensor_Input, Left_Sensor_Input, Right_Sensor_Input (Boolean)
- Dust_Sensor_Input (Boolean)
- Tick (Temporal Signal)

Output:

- Obstacle_Location: Record[Front, Left, Right: Boolean]
- Dust_Existence: Boolean

Process Description:

주기적으로 각 센서의 Raw 입력을 읽어들이어 논리적 상태로 변환한다.

5-1-2. Process Spec: 1.0 Sensor Interface

Algorithm:

WHILE system_running DO

ON Tick:

Obstacle_Location.Front := Read(Front_Sensor_Input)

Obstacle_Location.Left := Read(Left_Sensor_Input)

Obstacle_Location.Right := Read(Right_Sensor_Input)

Dust_Existence := Read(Dust_Sensor_Input)

Send Obstacle_Location to 2.0

Send Dust_Existence to 2.0

END WHILE

5-2-1. Process Spec: 2.0 Control Logic

Ref. No.: 2.0

Name: Control Logic & Command Generation

Input:

- Obstacle_Location: Record[Front, Left, Right: Boolean]
- Dust_Existence: Boolean
- Tick: Temporal Signal

Output:

- Motor_Command: Enum(Forward, TurnLeft, TurnRight, Stop, Backward)
- Cleaner_Command: Enum(On, Off, Power-Up)

Process Description:

FSM을 실행하여 센서 상태에 따른 모터 및 청소기 명령을 생성한다.
CN1(Motor FSM)과 CN2(Cleaner FSM)이 병렬로 실행되며 상호작용한다.

5-2-2. Process Spec: 2.0 Control Logic

Algorithm:

```
INITIALIZE CN1_State := Idle, CN2_State := Off
```

```
WHILE system_running DO
```

```
  ON Tick:
```

```
    // CN1: Motor Control
```

```
    CN1_State, Motor_Command := Execute_Motor_FSM(  
      CN1_State, Obstacle_Location, Cleaner_Trigger)
```

```
    // CN2: Cleaner Control
```

```
    CN2_State, Cleaner_Command, Cleaner_Trigger := Execute_Cleaner_FSM(  
      CN2_State, Dust_Existence, Motor_Status)
```

```
    Send Motor_Command to 3.0
```

```
    Send Cleaner_Command to 3.0
```

```
END WHILE
```

5-3-1. Process Spec: 3.0 Actuator Interface

Ref. No.: 3.0

Name: Actuator Interface

Input:

- Motor_Command: Enum(Forward, TurnLeft, TurnRight, Stop, Backward)
- Cleaner_Command: Enum(On, Off, Power-Up)

Output:

- Direction: String (HW 제어 신호)
- Clean_Command: String (HW 제어 신호)

Process Description:

논리적 명령을 하드웨어 액추에이터가 이해할 수 있는 신호로 변환한다.

5-3-2. Process Spec: 3.0 Actuator Interface

Algorithm:

WHILE system_running DO

ON Receive(Motor_Command):

CASE Motor_Command OF

Forward: Direction := "MOVE_FORWARD"

TurnLeft: Direction := "TURN_LEFT_45"

TurnRight: Direction := "TURN_RIGHT_45"

Stop: Direction := "STOP"

Backward: Direction := "MOVE_BACKWARD"

END CASE

Send Direction to Motor HW

ON Receive(Cleaner_Command):

CASE Cleaner_Command OF

On: Clean_Command := "VACUUM_NORMAL"

Off: Clean_Command := "VACUUM_OFF"

Power-Up: Clean_Command := "VACUUM_TURBO"

END CASE

Send Clean_Command to Cleaner HW

END WHILE

5-4-1. Process Spec: 2.1 Motor State (CN1)

Ref. No.: 2.1

Name: Motor State Management (CN1)

Input:

- Obstacle_Location: Record[Front, Left, Right: Boolean]
- Cleaner_Trigger: Boolean
- Tick: Temporal Signal

Output:

- Motor_Command: Enum(Forward, TurnLeft, TurnRight, Stop, Backward)
- Motor_Status: Enum(Moving, Turning, Paused)

Process Description:

이동 및 회피 상태를 관리하는 FSM을 실행한다.

Cleaner_Trigger가 True일 경우 Paused 상태로 전이한다.

5-4-2. Process Spec: 2.1 Motor State (CN1)

Algorithm:

STATE := Idle

WHILE system_running DO

ON Tick:

CASE STATE OF

Idle:

IF start_signal THEN STATE := Moving

Moving:

IF Cleaner_Trigger THEN STATE := Paused

ELSE IF Obstacle_Location.Front THEN STATE := Turning

ELSE Motor_Command := Forward

Turning:

IF All_Blocked(Obstacle_Location) THEN STATE := Backwarding

ELSE IF NOT Obstacle_Location.Left THEN

Motor_Command := TurnLeft; STATE := Moving

ELSE

Motor_Command := TurnRight; STATE := Moving

Backwarding:

Motor_Command := Backward

IF Back_Clear THEN STATE := Turning

Paused:

Motor_Command := Stop

IF NOT Cleaner_Trigger THEN STATE := Moving

END CASE

Motor_Status := STATE

END WHILE

5-5-1. Process Spec: 2.2 Cleaner State (CN2)

Ref. No.: 2.2

Name: Cleaner State Management (CN2)

Input:

- Dust_Existence: Boolean
- Motor_Status: Enum(Moving, Turning, Paused)
- Tick: Temporal Signal

Output:

- Cleaner_Command: Enum(On, Off, Power-Up)
- Cleaner_Trigger: Boolean

Process Description:

청소 모드를 관리하는 FSM을 실행한다.

먼지 감지 시 Power-Up 모드로 전이하고 CN1에 정지 요청을 전송한다.

5-5-2. Process Spec: 2.2 Cleaner State (CN2)

Algorithm:

STATE := Off

power_up_timer := 0

WHILE system_running DO

ON Tick:

CASE STATE OF

Off:

IF start_signal THEN

STATE := Normal_Cleaning

Cleaner_Command := On

Normal_Cleaning:

IF Dust_Existence AND Motor_Status = Moving THEN

STATE := Power_Up_Cleaning

Cleaner_Trigger := True

power_up_timer := 5 // 5 ticks

ELSE

Cleaner_Command := On

Cleaner_Trigger := False

Power_Up_Cleaning:

Cleaner_Command := Power-Up

power_up_timer := power_up_timer - 1

IF power_up_timer <= 0 THEN

STATE := Normal_Cleaning

Cleaner_Trigger := False

END CASE

END WHILE

5-6-1. Process Spec: 2.3 Timing & Deadlock

Ref. No.: 2.3

Name: Timing and Deadlock Check

Input:

- Tick: Temporal Signal
- Motor_Status: Enum(Moving, Turning, Paused)
- Obstacle_Location: Record[Front, Left, Right: Boolean]

Output:

- Deadlock_Signal: Boolean

Process Description:

시스템이 동일 상태에서 장시간 머물 경우 Deadlock으로 판단하여 탈출 신호를 생성한다.

5-6-2. Process Spec: 2.3 Timing & Deadlock

Algorithm:

stagnant_counter := 0

prev_state := Idle

DEADLOCK_THRESHOLD := 20 // 20 ticks

WHILE system_running DO

 ON Tick:

 IF Motor_Status = prev_state THEN

 stagnant_counter := stagnant_counter + 1

 ELSE

 stagnant_counter := 0

 prev_state := Motor_Status

 IF stagnant_counter >= DEADLOCK_THRESHOLD THEN

 IF All_Blocked(Obstacle_Location) THEN

 Deadlock_Signal := True

 // Force random turn or alert user

 END IF

 stagnant_counter := 0

 ELSE

 Deadlock_Signal := False

 END IF

END WHILE

5-7. Process Specs: 2.1.1 ~ 2.1.3

2.1.1 Normal Moving Logic

Input: Motor_Status

Output: Forward_Command

Description: Moving 상태일 때 Forward 명령 생성

Algorithm: IF Motor_Status = Moving THEN Forward_Command := Forward

2.1.2 Collision Avoidance Logic

Input: Obstacle_Location

Output: Avoidance_Command

Description: 장애물 감지 시 회피 명령 생성 (좌측 우선)

Algorithm:

```
IF Obstacle_Location.Front THEN
  IF NOT Obstacle_Location.Left THEN
    Avoidance_Command := TurnLeft
  ELSE IF NOT Obstacle_Location.Right THEN
    Avoidance_Command := TurnRight
  ELSE
    Avoidance_Command := Backward
  END IF
```

2.1.3 Motor Command Arbitrator

Input: Forward_Command, Avoidance_Command, Cleaner_Trigger

Output: Motor_Command

Description: 우선순위에 따라 최종 명령 결정

Algorithm:

```
IF Cleaner_Trigger THEN Motor_Command := Stop
ELSE IF Avoidance_Command IS_SET THEN Motor_Command := Avoidance_Command
ELSE Motor_Command := Forward_Command
```

5-8. Process Specs: 2.1.2.1 ~ 2.1.2.4

2.1.2.1 Front Collision Check

Input: Obstacle_Location.Front

Output: Front_Blocked: Boolean

Description: 전방 충돌 여부 확인

Algorithm: Front_Blocked := Obstacle_Location.Front

2.1.2.2 Turning Priority Decision

Input: Obstacle_Location.Left, Obstacle_Location.Right

Output: Turn_Direction: Enum(Left, Right, None)

Description: 좌측 우선 회전 방향 결정

Algorithm:

IF NOT Obstacle_Location.Left THEN Turn_Direction := Left

ELSE IF NOT Obstacle_Location.Right THEN Turn_Direction := Right

ELSE Turn_Direction := None

2.1.2.3 All Blocked Handler

Input: Turn_Direction, Front_Blocked

Output: Escape_Command: Enum(Backward, Pause)

Description: 전방향 막힘 시 탈출 명령 생성

Algorithm:

IF Front_Blocked AND Turn_Direction = None THEN

Escape_Command := Backward

END IF

2.1.2.4 Avoidance Command Merger

Input: Turn_Direction, Escape_Command

Output: Avoidance_Command

Description: 최종 회피 명령 통합

Algorithm:

IF Escape_Command IS_SET THEN

Avoidance_Command := Escape_Command

ELSE

Avoidance_Command := Turn_Direction

6. Data Dictionary (완전판)

- **DFD 및 FSM에서 사용되는 모든 데이터 정의**
- • 입력 데이터 (Sensor Inputs)
- • 중간 처리 데이터 (Internal Flows)
- • 출력 데이터 (Actuator Commands)
- • 제어 흐름 (Control Flows)
- • 상태 변수 (State Variables)

6-1. Data Dictionary: 입력 데이터

| 데이터 이름 | 설명 | 형식/유형 |
|--------------------|--------------------------|---------|
| Front_Sensor_Input | 전방 장애물 감지 신호 (Interrupt) | Boolean |
| Left_Sensor_Input | 좌측 장애물 감지 신호 (Periodic) | Boolean |
| Right_Sensor_Input | 우측 장애물 감지 신호 (Periodic) | Boolean |
| Dust_Sensor_Input | 먼지 존재 감지 신호 (Periodic) | Boolean |
| Tick | 주기적 시간 신호 (Temporal) | Signal |

6-2. Data Dictionary: 중간 처리 데이터

| 데이터 이름 | 설명 | 형식/유형 |
|-------------------|-----------|-------------------------|
| Obstacle_Location | 장애물 위치 정보 | Record[F,L,R: Boolean] |
| Dust_Existence | 먼지 존재 여부 | Boolean |
| Forward_Command | 전진 명령 | Enum(Forward) |
| Avoidance_Command | 회피 명령 | Enum(TurnL/R/Backward) |
| Turn_Direction | 회전 방향 결정 | Enum(Left, Right, None) |
| Front_Blocked | 전방 막힘 여부 | Boolean |
| Escape_Command | 탈출 명령 | Enum(Backward, Pause) |

6-3. Data Dictionary: 출력 데이터

| 데이터 이름 | 설명 | 형식/유형 |
|-----------------|---------------|-------------------------------------|
| Motor_Command | 모터 제어 명령 (최종) | Enum(Forward/TurnL/R/Stop/Backward) |
| Cleaner_Command | 청소기 제어 명령 | Enum(On/Off/Power-Up) |

HW Interface:

- Direction: Motor_Command를 HW 신호로 변환 (String)
- Clean_Command: Cleaner_Command를 HW 신호로 변환 (String)

6-4. Data Dictionary: 제어 흐름

| 데이터 이름 | 설명 | 형식/유형 |
|-----------------|------------------|-----------------------------|
| Cleaner_Trigger | CN2→CN1 정지 요청 신호 | Boolean (Control Flow) |
| Motor_Status | CN1→CN2 이동 상태 전달 | Enum(Moving/Turning/Paused) |
| Deadlock_Signal | Deadlock 탈출 신호 | Boolean (Control Flow) |

※ *Control Flow*는 데이터를 전달하는 것이 아니라 제어 신호를 전달하여 다른 프로세스의 상태 전이를 유발합니다.

6-5. Data Dictionary: 상태 변수

| 데이터 이름 | 설명 | 형식/유형 |
|-----------|-------------------|--|
| CN1_State | Motor FSM 현재 상태 | Enum(Idle/Moving/Turning/Backwarding/Paused) |
| CN2_State | Cleaner FSM 현재 상태 | Enum(Off/Normal_Cleaning/Power_Up_Cleaning) |

7. 문제 해결 검증

- 초기 문제점 → 해결 방안 검증
- **[Stop Deadlock]**
 - ✓ Pause + Backwarding 상태로 분리
 - ✓ 2.3 Deadlock Check 프로세스 추가
- **[Dust 미고려]**
 - ✓ CN2 FSM에 Dust Cleaning 상태 추가
 - ✓ Power-Up Cleaning 모드 구현
- **[좌/우 우선순위]**
 - ✓ 2.1.2.2에서 좌회전 우선 정책 명시
 - ✓ FSM 전이 조건에 반영
- **[Backward 미고려]**
 - ✓ Backwarding 상태 추가
 - ✓ 2.1.2.3 All Blocked Handler 구현
- **[Interface 불일치]**
 - ✓ Motor/Cleaner Interface 일관성 확보
 - ✓ Process Spec 2.0, 3.0에서 명확히 정의

7-1. SA 산출물 완성도 체크

- 과제 요구사항 달성 현황
- System Context Diagram: ✓ 완료 (Slide 4-5)
- DFD Hierarchy (Level 0~4): ✓ 완료 (Slide 6-10)
- FSM 설계: ✓ 완료 (Slide 11-16)
- Process Specifications: ✓ 완료 (Slide 17-25)
- Data Dictionary: ✓ 완료 (Slide 26-31)
- 문제점 해결: ✓ 완료 (모든 요구사항 반영)

8. 요약 및 결론

- **RVC 제어 소프트웨어 구조적 분석 결과:**
- ✓ System Context Diagram으로 시스템 경계 명확히 정의
- ✓ 4단계 DFD Hierarchy로 Top-Down 분해 완료
- ✓ 2가지 FSM 버전 제시 (단일 / 2개 CN 분리)
- ✓ 모든 프로세스에 대한 상세 명세 작성
- ✓ 완전한 Data Dictionary 구축
- ✓ 초기 5가지 문제점 모두 해결

- → **Complete & Consistent한 SA 설계 달성**
- → **IEEE Std 830-1998 기반 SRS 작성 준비 완료**