工學碩士 學位論文

최적 경로선정 알고리즘 기반의 300mm WAFER 이송용 OVERHEAD HOIST TRANSPORT 최적 제어관리에 관한 연구

A Study on the Development of the Overhead Hoist Transport Control and Management for 300mm Wafer Transfer Based on the Optimal Flow Path Algorithm

指導敎授 吳 珍 錫

2003 년 2월

韓國海洋大學校 大學院
機關시스템工學科 電氣電子制御 專攻
金 鶴 愃

本 論文을 金 鶴 愃의 工學碩士 學位論文으로 認准함

主審 : 工學博士 이 상 태 印

副腎 : 工學博士 노 창 주 印

副審 : 工學博士 오 진 석 印

2003年 2月

韓國海洋大學校 大學院

機關시스템工學科 電氣電子制御 專攻

金 鶴 愃

A Study on the Development of the Overhead Hoist Transport Control and Management for 300mm Wafer Transfer based on the Optimal Flow Path Algorithm

by Kim Hak-Sun

Department of Maritime Engineering
The Graduate School of Korea Maritime University
Pusan, Republic of Korea

Abstract

This paper explain Overhead Hoist Transport, Overhead Hoist Transport Control System and OPC(One-board Programmable Controller). Overhead Hoist Transport is the device that transfer wafer though rail installed on the ceiling. OHT Control System is the system that integrate and control to the best optimal and manage the many OHT. It is used in the factory that produced semiconductor wafer.

OHT is operated from an upper group by the transport order to be ordered.

Consequently, this paper is designed the module that consist is command entity and operate entity. And Wireless LAN communication and protocol is designed. It is ensured against risks which the minimize could communication error and fail.

The experiment observe SEMI Standsed scenario made the OHT for research

and was the low to the experiment method to be produced. Therefore, OPC that is made by the design verified the adequacy of a system design.

In this paper, All final result developed so that the contentment could order various driving condition and characteristic that contain a semicoductor standard.

제 1장 서론

1.1 연구배경

대한민국의 반도체 산업은 전체 수출의 14.1%('99년), 15%('00년)을 차지할 정도로 비중이 높다. 특히 메모리 분야에 대해서는 세계적으로 선진 수준을 점유하고 있을 정도로 생산 수량이 상당하다. 하지만 반도체 산업 관련 제조장비의 국산화율은 약 11.7%로 매우 저조한 편이다. 특히, 반도체 제조 시스템을 구성하는 장치중 핵심이라 할 수 있는 지능형 물류 시스템은 미국, 일본 등에서 전량 수입에 의존하고 있는 실정이다. 하지만 국내 반도체 생산 업계는 관련 산업이 상당히 기술 집약적이고, 고 부가가치를 창출할 수 있는 산업이기에 생산 설비 및 생산 방법에 관한 노-하우가 외부로 노출되는 것을 상당히 꺼려하는 특성을 지니고 있다.

웨이퍼(wafer) 생산에 관한 전 세계적인 동향은 현행 200mm 구경 웨이퍼(wafer)에서 300mm 웨이퍼(wafer)로 전환되어 가고 있는 추세이며 이러한 전환은 생산량이 2.25배 증가하고 생산원가는 약 30%가량 절감될 수 있음에 기인한다. 현재 국내의 반도체 업계 또한 이와 같은 동향으로 전환되어 가고 있는 추세이며 각 학계 및 재계의 연구 기관에서는 관련 장치의 연구 개발이 활발히 진행되고 있다.

반도체 관련 지능형 물류 시스템은 크게 5분야로 나눌 수 있다. 첫 번째는 전체 제조공정을 관리하는 MCS(Manufacturing Control System)이며 두 번째는 제조설비 제어 시스템을 관리하는 MES(Material Execution System)이다. 세 번째는 지상물류 이송 장치인 AGV(Automated Guided Vehicle)을 제어 관리하는 ACS(AGV Control System)이며 네 번째는 천장에 설치된 레일(rail)을 주행하며 물류를 이송하는 OHT(Overhead Hoist Transport)를 제어관리 하는 OCS(OHT Control System)이며 다섯 번째는 자동창고 개념의 Stocker를 제어관리 하는 SCS(Stocker Control System)이다. 1990년대부터 현재에 이르기까지 자동항만에서의 물류관리・이송 분야 및 자동차 부품 생산 공정과 같이 대규모 다품종 생산 공정에서 물류를 효과적으로 관리하기 위해서 스토커(stocker) 및 AGV에 관한 연구개발이 활발히 진행되어 왔으며 현재 대부분 국산화가 완성 되었다.

본 논문에서는 위에 명시한 지능형 반도체 물류 자동화 시스템 중 현재 국내에서 연구개발 사례가 미약한 분야인 OCS 및 OPC(OHT의 각 엑츄에이터(actuator)를 제어하는 One Board Programmable Controller 개발에 관하여 연구하고자 한다.

1.2 연구개요

본 논문의 주요 연구 목적 및 목표는 웨이퍼(wafer) 생산 공장의 천장에 설치된 레일(rail)을 주행하며 300mm 대구경 웨이퍼(wafer)를 이송하는 차세대 지능형 이송 장치인 OHT를 최적으로 제어관리 하기 위함이다. 본 연구목적 및 목표를 달성하기 위해서 중점적으로 연구해야 할 분야를 크게 두 가지로 나눌 수 있다. 첫 번째는 웨이퍼(wafer) 생산 공정에 웨이퍼(wafer)를 이송할 목적으로 투입될 수십대에서 수백대에 이르는 다수의 OHT를 통합제어관리 하는 OCS(OHT Control System) 분야이며 두 번째는 OHT를 구성하는 각 엑츄에이터(actuator)를 직접 제어하는 OPC(One Board Controller) 분야이다.

OCS는 반도체 생산 공정에 투입될 다수의 OHT를 최적으로 통합 제어관리 하기 위해서 이송 시간을 최소화하기 위한 최단거리의 이송경로를 탐색 할 수 있는 알고리즘 및 이송중인 OHT 상호간에 충돌을 방지 할 수 있는 OHT의 최적경로 선정에 관한 알고리즘이 탑재 되어야 한다.

OPC는 OHT를 구성하는 핵심 엑츄에이터(actuator)인 주행용 AC 서보모터 및 승강(hoist)용 AC 서보모터, 선회 및 횡행용 스테핑(stepping) 모터의 변위 및 속도를 신뢰성있게 제어를 할 수 있는 하드웨어 및 소프트웨어가 탑재되어야 한다. 또한 통합 제어관리의 신뢰도를 향상시키기 위해서는 OCS와 OPC 간에 통신 에러(error) 및 페일(fail)를 최소화 할 수 있는 안전성이 확보된 통신 기법을 적용하여야 한다.

본 논문에서는 위에 명시한 바와 같이 OHT를 최적으로 제어관리 하기 위한 핵심 알고리즘 및 제어 기법에 관하여 연구하고 실험 및 시뮬레이션을 통하여 이를 검증한다. 또한 모든 연구 결과물은 세계반도체 장비협회(SEMI)에서 국제적으로 규격화하고 규정한 반도체 관련 장치들이 지녀야 할 다양한 운전조건 및 특성을 만족시킬 수 있도록 개발한다.

본 논문의 2장에서는 본 논문에서 개발하고자하는 시스템의 개요 및 구성요소에 대해서 논의하며 3장에서는 OHT를 최적으로 제어관리 할 수 있는 알고리즘을 구축하고 이를 구현한다. 4장에서는 제안된 알고리즘을 구현할 수 있는 시스템을 구성하고 시스템 구성 모듈의 구조를 상세하게 설계한다. 5장에서는 실험 장치에 대하여 논의하고 결과를 고찰하며 6장에서는 실험 결과를 토대로 결론을 도출한다.

제 2장 300mm 대구경 웨이퍼(wafer) 제조공정과 OHT

2.1 웨이퍼(wafer) 제조공정

300mm 대구경 웨이퍼(wafer) 생산 공정에 있어서 모든 관련 설비들이 갖추어야 할 운전 조건 및 특성은 SEMI(세계반도체장비협회)에서 규격화하여 규정하고 있다. 웨이퍼(wafer)에 관련된 모든 제조 공정은 클린룸(clean room)에서 진행되며 각 장치간의 인터페이스 및 통신 방식 등은 SEMI 규격에서 규정하고 있다.

Fig. 2.1은 300mm 대구경 웨이퍼(wafer) 생산 공정의 차세대 지능형 물류 시스템을 도시화한 것이다. 전체 생산현장의 공정을 통합 관리하는 MCS(Manufacturing Control System)는 목표한 생산량을 달성하기 위하여 각 제조장치 및 물류시스템을 통합적으로 관리한다. 웨이퍼(wafer) 제조 장치를 통합 관리하는 MES(Material Executionl System)는 모든 웨이퍼(wafer) 제조 작업장(Job Station)에 작업 명령을 지령하고 작업 상황을 모니터링한다. 또한 작업명령의 지령 및 작업완료 상황을 상위 그룹인 MCS에 보고하며 보고를 받은 MCS는 다음 제조 작업을 위해 작업이 완료된 웨이퍼(wafer)의 공정간 이동 또는 장치간 이동을 해당 물류 시스템에 지령한다. AGV(Automated Guided Vehicle)를 통합 제어관리하는 ACS(AGV Control System)는 웨이퍼(wafer) 제조 공정중에 물류를 적재・관리하는 자동창고 역할을 담당하는 각 Stocker간의 웨이퍼(wafer) 이송을 위해 AGV의 이송작업을 스케쥴링(Scheduling)하여 각 AGV에 이송 명령을 지령한다. ACS로부터 이송명령을 받은 AGV는 명령받은 이송 경로를 통하여 각 Stocker 간에 이송 작업을 수행한다. AGV에 의해서 Stocker로 이송되어 적재된 웨이퍼(wafer)는 SCS(Stocker Control System)에 의해서 공정 로트(lot)별로 통합 관리된다. MCS는 Stocker에 적재된 웨이퍼(wafer)의 공정간 이송 및 장치간 이송을 OCS(OHT Control System)에 지령한다. OCS는 MCS로부터 이송 명령을 받은 해당 공정 및 작업장치로의 이송 명령을 OHT에 지령한다. OCS로부터 이송 명령을 받은 OHT는 해당 공정 및 작업장에서 요청된 이송작업을 수행한다.

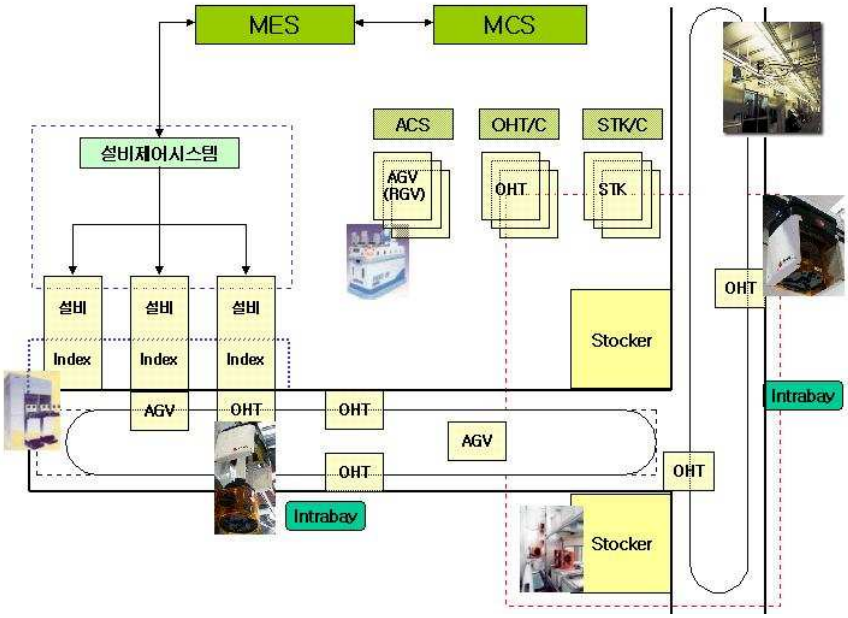


Fig. 2.1 Drawing of 300mm Wafer Manufacturing Process

2.2 OCS 개요

OHT는 천장에 설치된 레일(rail)을 주행하며 원거리(遠距離) 이송작업을 하는 특성을 지니고 있으므로 유선식 통신 방식을 적용할 수 없다. 따라서 이송작업 명령을 지령하는 OCS와 무선통식방식으로 인터페이스 된다.

OCS는 MCS로부터 이송에 관련한 명령을 전달받는다. OCS는 OHT에게 최단시간에 이송작업을 완료할 수 있도록 하기 위해서 출발지에서 목적지까지의 최단 경로를 탐색하고 이송 작업을 수행하기에 최적의 위치에 있는 OHT를 선정하여 이송명령을 지령한다. 이때, 이송중인 OHT 상호간의 충돌을 방지하기 위해서 이송구간의 레일(rail)위에 존재하는 타 OHT에게 정체 및 충돌을 방지 할 수 있는 위치로 이동 명령을 지령한다. MCS 및 OCS, OHT 상호간의 통신 인터페이스는 다음의 Fig. 2.2와 같다. OCS는 OHT를 이용하여 원활한 물류를 이송할 수 있도록 하기 위해서 다음의 Fig. 2.3과 같이 OCS를 구성하는 요소들 간의 상호 인터페이스 관계를 갖추어야 한다. OCS는 Fig. 2.3에서 보는 바와 같이 크게 MCS Interface Module, GUI(General User Interface) Module, Scheduling Engine Module, OHT Interface Module로 구성된다. 각각의 모듈(Module)에 대한 설명은 다음과 같다.

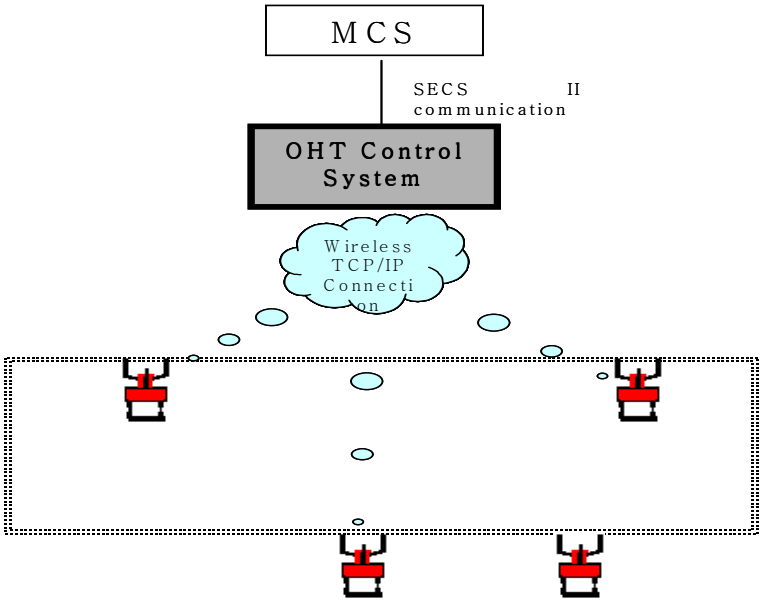


Fig. 2.2 Drawing of Communication Interface between MCS and OCS and OHT

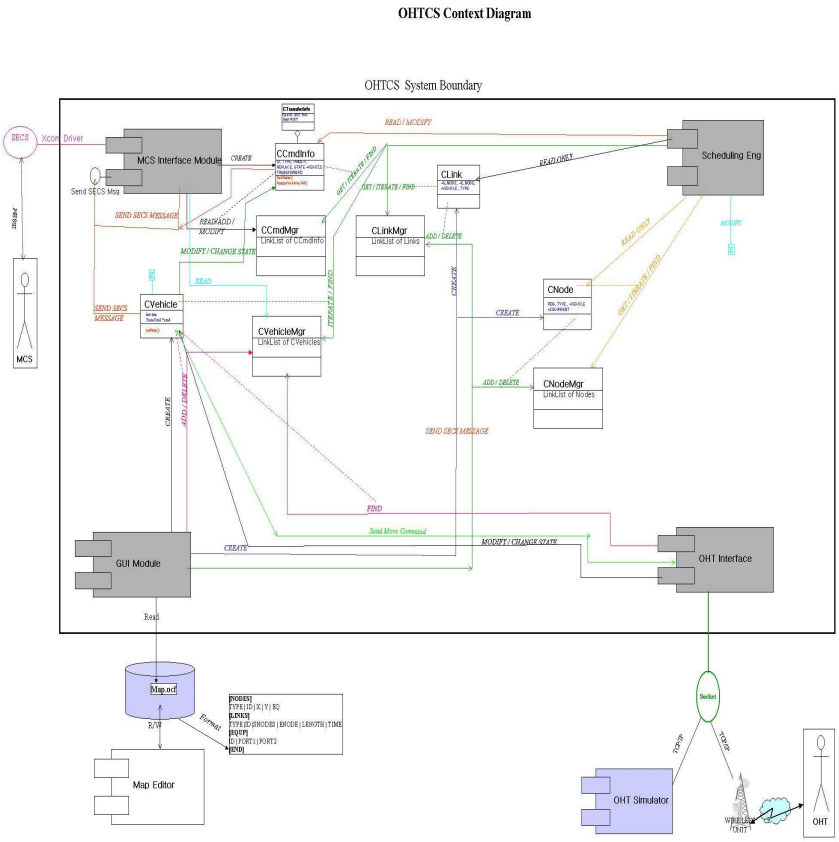


Fig. 2.3 Drawing of Software Module Interface between MCS and OCS and OHT

2.2.1 MCS 인터페이스 모듈

MCS에서는 OHT에 의한 물류의 이동이 필요하다고 판단될때 임의의 작업 Station의 물류(웨이퍼(wafer))를 다른 작업 Station으로 이송하라는 장치간의 이송 또는 공정간의 이동을 OCS에게 지령한다. 이때 MCS 및 OCS간에 원활한 통신을 수행하기 위해서는 신뢰성있는 소프트웨어 통신 인터페이스 모듈이 필요하다. MCS 인터페이스 모듈은 나날이 발달하는 반도체 생산 분야의 특성에 충분히 대처 할 수 있도록 충분한 유연성을 갖추기 위하여 소프트웨어 모듈 형식을 갖추어야 한다.

2.2.2 GUI 모듈

OCS는 이송작업중인 모든 OHT를 통합 관리하는 시스템이다. 관리자는 GUI 모듈을 통하여 공정내에 존재하는 수십에서 수백대의 OHT를 감시하고 상위그룹(MCS)과의 통신 상황을 감시할 수 있다. 따라서 OCS에서 GUI 모듈이 레일(rail)에 존재하는 모든 OHT의 이송상황 및 상태를 모니터링 하고 MCS 및 OHT 상호간에 통신 상황을 모니터링 하기 위해서는 Fig. 2.2에 나타나 있는 것과 같이 각 모듈은 해당 프로그램의 링크(Link) 함수들과 연동되어야 한다.

추후 반도체 공정의 증설이나 OHT의 추가적인 투입에 대비하여 충분한 유연성을 갖추도록 하기 위해서 소프트웨어 모듈 형식으로 개발 되어야 한다.

2.2.3 스케줄링 엔진 모듈

MCS에서는 물류의 이동 정보만을 OCS에게 지령할 뿐 이송할 OHT를 직접 지령하지 않는다. 따라서 MCS로부터 물류의 이송정보를 지령 받은 OCS는 해당 OHT를 호출하여 물류 이송을 직접 지령한다.

현재 반도체 생산 분야의 국제적인 동향은 “생산 시간의 단축”이다. 따라서 물류 시스템 또한 물류의 이송시간을 최소화하여야 한다. 따라서 OCS에서는 이송에 관련한 출발지에서 도착지까지의 최단경로를 탐색하고 최단경로에서 가장 가까이 있는 OHT를 호출하여 이송 명령을 지령한다. 이때, 작업 중인 OHT 상호간에 충돌을 방지할 수 있도록 하기 위하여 이송 경로상의 OHT를 실시간으로 탐색하여 충돌의 가능성이 있는 OHT에게 충돌을 방지할 수 있도록 이송 명령을 지령하거나 분기·

합류점을 경유하도록 이송 경로를 지령하여 충돌을 방지하도록 한다.

스케쥴링(scheduling) 알고리즘에서는 최적의 물류 이송 경로를 탐색할 수 있는 알고리즘이 탑재되어야 하며 추후 주행 레일(rail)의 구조 변경이나 증설, OHT의 추가 투입에 대비하여 충분히 능동적이고 유연적인 알고리즘 이여야 한다.

2.2.4 OHT 인터페이스 모듈

MCS에서 지령된 물류(웨이퍼(wafer)) 이송정보를 기반으로 OCS에서는 최적경로를 탐색하여 해당 OHT를 선택·호출하고 이송 명령을 지령한다. 이때 OCS와 OHT의 상호간 통신 방법은 TCP/IP 프로토콜 기반의 무선랜(Wireless LAN) 방식이 적용된다. 추후 추가 투입될 수 있는 OHT를 감안하여 충분한 유연성을 지닌 소프트웨어 모듈 형식을 갖추어야 한다.

2.3 OHT 개요

OHT는 OCS로부터 지령 받은 이동정보를 기반으로 웨이퍼(wafer)를 이송하는 장치로써 흔히 반도체 관련 물류 시스템에서는 비-클(vehicle)이라 명칭되기도 한다.

OHT가 이송하는 웨이퍼(wafer)는 300mm 대구경 웨이퍼(wafer)를 이송하기 위하여 웨이퍼(wafer)를 저장하는 웨이퍼(wafer) 카세트(Cassette)인 FOUP(Front Opening Unified Pod)에 담겨져 이송된다.

Fig. 2.4는 실제 OHT의 모형을 나타내며 Fig. 2.5는 OHT가 FOUP을 웨이퍼(wafer)제조 장치간에 이송하는 과정을 도시화한 것이다. 이송 OHT가 FOUP을 이송하기 위하여 갖추어야 할 기본 기능을 Table. 2.1에서 정리 하였다.

Table. 2.1 Basic Function of OHT

기능	내용
이송	각 station위치는 teaching을 통하여 data 저장한다.
	정위치 검출 방법: 레일(rail)에 설치된 barcode를 읽어 현재 station 정보 획득한다.
	신호기: 통과 가능인 경우 광 fiber를 점등, 통과 불가능인 경우 광 Fiber를 소등한다.
	주행 중 충돌/추돌 방지를 위한 원/근거리 sensor를 설치한다.
	전방의 장애물 검출을 위한 main/sub Sensor 설치한다.
승강(상승/하강)	Roll Shaft를 구동하여 pulley를 통하여 belt를 감거나 풀어준다
	주행중 전자 Brake를 ON 하여 위치를 고정한다.
	중심이탈 sensor on시 승강동작을 정지한다.
	아래방향의 장애물 검출을 위한 4개의 lock down sensor를 설치하여 장애물 검출시 동작을 멈춘다.
선회 (CW/CCW)	정회전, 역회전 Lim 검출 sensor 와 stopper를 설치하여 오동작을 방지한다.
횡행(좌/우)	좌, 우 lim 검출 sensor 와 stopper를 설치하여 오동작을 방지한다.
Chuck (open/close)	chuck open/close를 sensor로 확인
	단순 제하 sensor가 설치되어 FOUP의 존재여부를 확인
낙하방지 (open/close)	낙하방지 센서를 front와 rear에 각각 2개씩 설치하여 foup의 낙하를 검출한다.
	낙하방지 장치의 동작유무를 검출하기 위한 4개의 sensor 설치하여 낙하방지 장치가 동작하였는지 확인한다.
	하강 over run을 방지하기 위한 Photo Sensor 설치하여 Foup Hoist 동작중 Over Run 검출시 동작을 멈춘다.
수동조작	hand terminal을 사용하여 data를 무선랜으로 송수신한다.
전원공급	무접촉 전원방식인 HID(high frequency induction device 사용하여 OHT에 전원을 공급한다.
	litz wire를 레일(rail)에 설치하여 이송중인 OHT가 고주파 전원을 Pick-up 할 수 있도록 한다.
	pick-up coil을 OHT본체에 설치하여 이송 중에 고주파 전원을 pick-up 한다.
광전송 장치	equipment와의 광통신을 통한 loading/unloading
Cleaning Unit	OHT에서 발생하는 분진 흡수, 청정도 유지

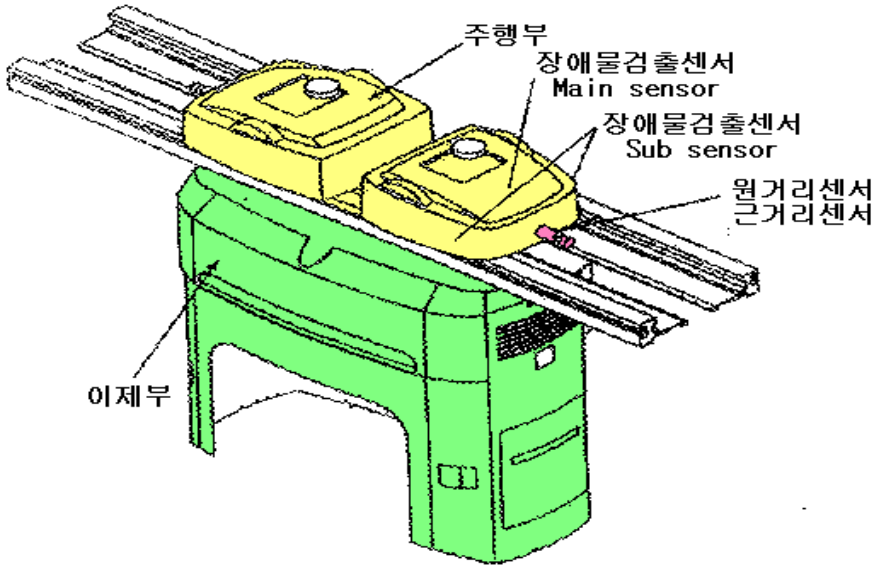


Fig. 2.4 Modeling of OHT

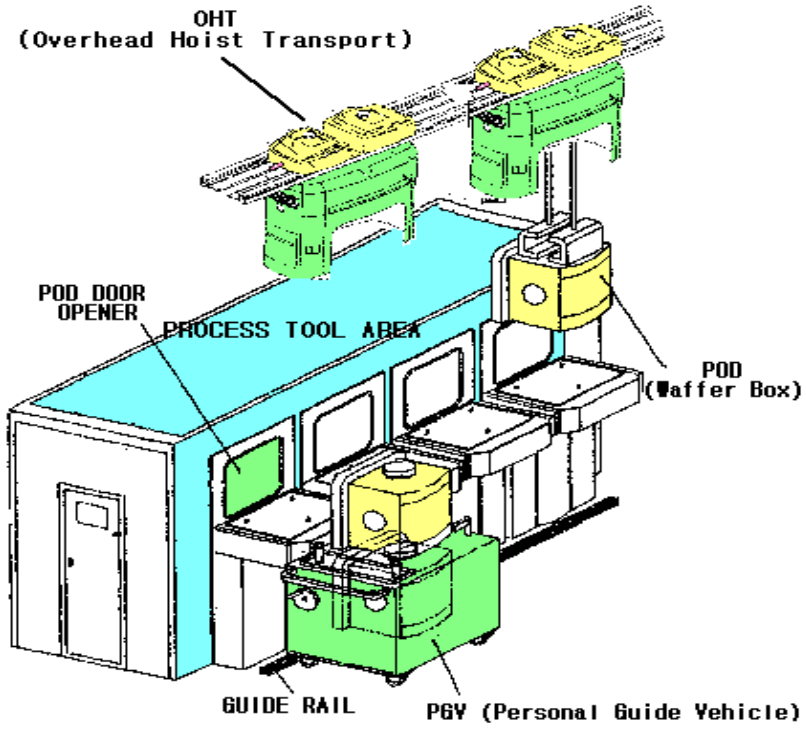


Fig. 2.5 Drawing of Foup Transport by OHT

제 3장 알고리즘 구축 및 구현

3.1 최적 이송경로 선정 알고리즘

이미 앞장에서 언급하였듯이 전 세계적인 반도체 생산 공정의 동향은 제조 시간의 단축이다. 따라서 천장에 설치된 레일(rail)을 주행하며 Foup을 이송하는 OHT 또한 이송시간을 최대한 단축 할 수 있는 경로로 운행되어야 한다.

OCS는 MCS로부터 지령 받은 이송정보를 토대로 최적의 이송경로를 스케쥴링(scheduling)하여 OHT에 이송 명령을 지령한다. 이때, 효율적이고 합리적인 이송명령을 지령하기 위해서 OCS는 다음과 같은 기능을 갖추어야 한다.

- MCS 작업을 수행하기 위한 최적의 OHT 선정
- 최단거리를 적용한 최적경로 선택
- 레일의 맵(Map) 구조에 유연한 알고리즘
- 최적의 작업수행을 위한 작업관리 및 OHT 이송명령 할당
- OHT들 간의 충돌방지

3.1.1 Floyd-Warshall 알고리즘[2,3,17)]

본 논문에서는 OHT의 최단 거리를 적용한 최적 이송경로를 선정하기 위한 알고리즘으로 Floyd-Warshall 알고리즘을 채택했다. 본 알고리즘은 1962년에 Floyd-Warshall에 의해 개발된 알고리즘으로써 네트워크의 모든 두 교점간의 최단경로(Shortest Path between All Paires of 노드(node)를 비교적 간편한 계산식으로 구할 수 있으며 쉽게 프로그래밍 할 수 있는 장점이 있다.

Floyd -Warshall 알고리즘의 결과로 2개의 행렬이 제공되는데 첫 번째 행렬에는 해당 지점까지 도달하는데 걸리는 가장 짧은 길이가 각 정점 대 정점별로 저장되어 있고, 두 번째 행렬에는 현재 정점에서 목표 정점까지 도달하기 위해 인접 정점중 어디로 가야 하는지를 정의해 주는 정점 일련번호가 저장되어 있다. 만약 현재 정점에서 목표 정점까지 도달할 수 있는 경로가 없을 경우 첫 번째 행렬의 해당 경로 길이가 무한대 값이 된다. 하지만 실제 프로그램에서는 무한대의 표현이 불가능하다. 따라서 본 논문에서는 최대 경로 길이 보다 긴 특정 값을 무한대 값으로 설정

하여 프로그래밍 한다. 따라서 특정 경로 요구에 대해 해당 경로가 존재하는지의 여부를 추가하는 연산 없이 곧바로 확인할 수 있다. Floyd -Warshall 알고리즘은 N개의 정점이 있다고 가정할 때 N(N-1)(N-2)회의 덧셈과 N(N-1)(N-2)회의 비교 연산이 요구된다.

Floyd-Warshall 알고리즘을 이용한 최단경로 선정 절차는 다음과 같다.

여기에서,

$\pi_{ij}^{(m)}$=교점 i에서 j까지의 최대 m개의 호를 사용한 최단 경로의 길이라 하고

$P_{ij}(0)$=교점 i에서 j까지의 최대 m개의 호를 사용한 최단 경로에서 최초 중간 교점이라 정의한다.

1단계 : 초기화.

$\pi^{(0)}$ [d_{ij}] (d_{ij}=0, if, 호 (i,j)가 없으면 dij=∞)

$P^{(0)}$ ← [j]

m ← 0

2단계 : $\pi^{(m-1)}$ 에서 행 m과 열 m을 표시함.

m ← $m+1$

$\pi(m-1)$에서 행 m과 열 m을 표시함.

3단계 : $\pi(m)$을 계산함, 즉 표시된 행과 열을 사용하여 삼각연산을 수행함.

표시된 행 m과 열 m 이외의 모든 원소 $\pi_{ij}^{(m-1)}(i \neq j \neq m)$을 행 m과 열 m의 원소의 합인 $\pi_{im}^{(m-1)} + \pi_{mj}^{(m-1)}$과 비교하여, 후자가 작으면 $\pi_{ij}^{(m-1)}$을 $\pi im^{(m-1)} + \pi_{mj}^{(m-1)}$로 수정하고 $P_{ij}^{(m-1)}$을 $Pim^{(m-1)}$로 수정함.

m=N이면 단계 4로 감, 그렇지 않으면 단계 1로 감.

4단계 : 끝냄

3.2 OHT 주행 레일(rail) 기본구조

Fig. 3.1은 OHT의 물류 이동시 주행로 역할을 하는 레일(rail)의 기본적인 구조를 나타낸다. 여기서, 노드(node)란 일반적인 레일상의 정점을 의미하며 알고리즘 상에서 주행 거리를 연산하는 기본 정보[노드(node)와 노드(node) 사이의 거리]가 된다. 여기서, 노드(node)와 노드(node)의 연결구간을 링크(link)라 표현한다.

Stop Station은 대기(Free) 상태인 OHT의 정지 지점이 되거나 주행중인 OHT 상호간에 충돌을 회피하기 위해서 잠시 정지하는 정지 지점이다.

Port는 Equipment라고도 일컬어지며 OHT가 Foup을 로딩(loading) / 언로딩(unloading) 하는 작업장을 의미하고 Fig. 2.5에 잘 나타나 있다. 따라서 OHT는 OCS가 지령하는 임의의 포트[port(From)]에서 포트[port(To)]로 물류를 이송한다. OHT의 물류이송 작업을 위해 OCS에서는 코드형식의 작업명령을 지령한다. 이송명령 코드 및 그에 따른 작업내역을 Table. 3.1에서 정리하였다. 사용코드는 반도체 공정내 장치간의 통신규약인(IBSEM)에 기초한다.

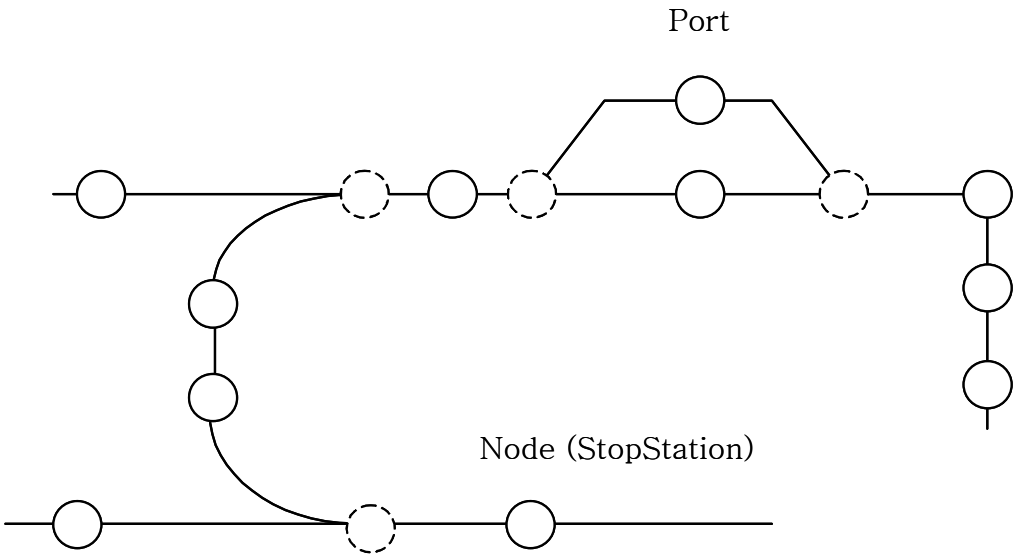


Fig. 3.1 Structure of OHT Traveling Rail

Table. 3.1 OHT Operating contents by Transport commend code

이송명령 코드 (OCS→OHT)	명령	OHT 작업내용
02	move	OHT의 현재 위치에서 Source 노드(node)까지의 이송
03	move & loading	Source 노드(node)에서의 Loading
04	move & unloading	Source 노드(node)에서 Destination 노드(node)로의 이송
		Destination 노드(node)에서의 unloading

3.3 알고리즘 구현

본 절에서는 3.1.1절에서 논의한 Floyd-Warshall 알고리즘을 기반으로 OHT를 효율적으로 운용할 수 있는 OHT 최적경로 선정에 관한 알고리즘의 구현에 관하여 논의 하고자 한다. 본 알고리즘의 구현은 Pentium Ⅳ급 PC상에서 Visual C++ 6.0을 사용하여 프로그래밍 한다. 본 알고리즘을 구현함으로써 제안된 알고리즘을 시뮬레이션 할 수 있으며 이와 같은 시뮬레이션을 수행함으로써 제안된 알고리즘의 성능을 평가하고 타당성 또한 검증 할 수 있을 것이다. 알고리즘 구현의 절차 및 내용은 Fig. 3.2와 같다.

① 전처리 과정
모든 노드에 대한 N x N 행렬의 인접행렬(W)을 작성한 후 Floyd 알고리즘을 이용하여 최단거리 행렬(D)과 경로행렬(P)을 생성한다. 각 노드에 대하여 최단거리 순으로 정렬한 노드 행렬(N)을 만든다

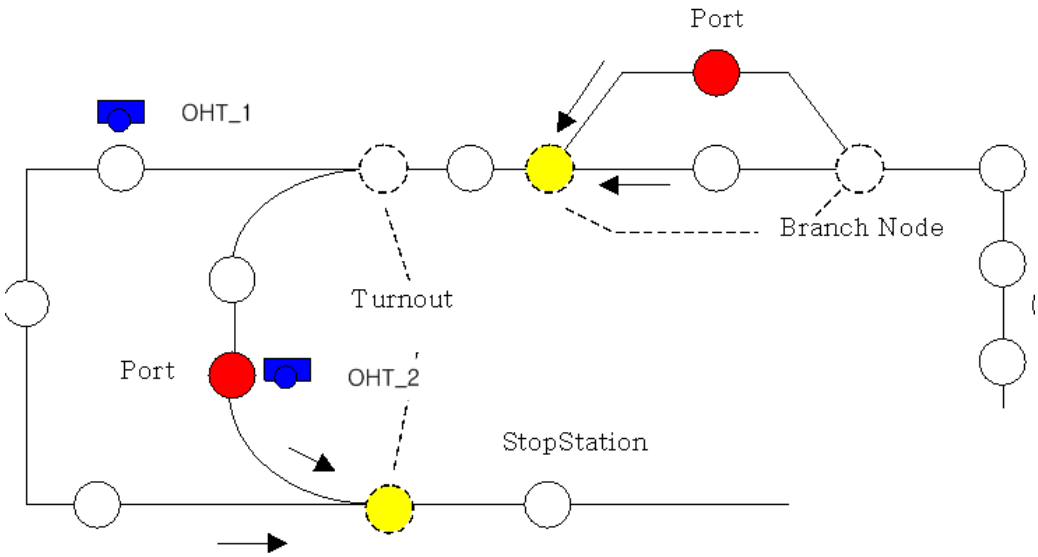
② OHT 선정
MCS 작업명령을 수행하기 위해 Source 노드에 가장 빨리 올 수 있는 OHT를 선택한다. 이때, OHT는 현재 하고 있는 작업을 끝내고 오거나 작업중이 아니라야 한다. 이를 위해 도착시간(OHT가 Source 노드(node)까지 오는데 걸리는 시간)을 계산하여 이 시간이 가장 작게 걸리는 OHT을 선택한다.

③ OHT 경로 선정
전처리 과정에서 만들어진 경로행렬(P)를 통해 임의의 Source 노드로부터 Destination 노드까지의 최단 거리 경로를 얻을 수 있다. 여기에서 얻어진 경로는 선택한 OHT의 현재 위치에서 Source Port까지 가기 위한 경로와 Source Port에서 Destination Port까지 가기 위한 경로와 같다.

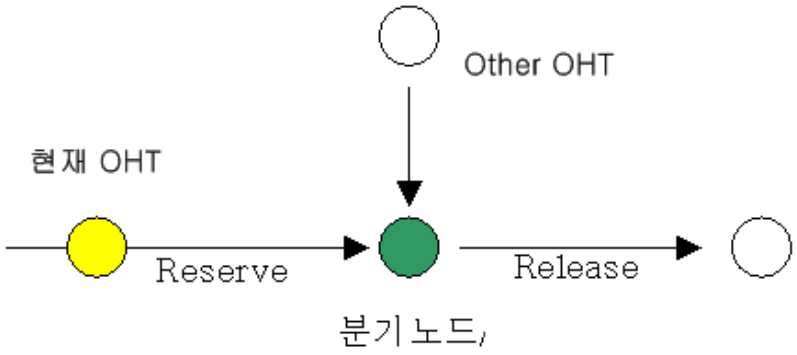
Fig. 3.2 Procedure of Algorithm Realization

3.3.1 충돌방지 로직(Logic)

실제 반도체 생산 공정에 투입되어 물류 이송을 수행하는 OHT는 수십에서 수백대에 이른다. 따라서 본 논문의 3.1.1에서 제시한 최적경로 선정 알고리즘만을 단독으로 OCS에 탑재하여 운용한다면 충돌 사고를 피하지 못할 것이다. 따라서 선정된 최적경로와 충돌방지 로직(Logic)을 동시에 고려하여야 한다. 이송작업 중인 OHT가 상호간에 충돌할 수 있는 경우는 아래에 도시화 하였듯이 Case 1과 case 2의 경우이다. 본 절에서는 각각의 경우에 충돌을 회피하기 위한 해법을 제시한다.



Case 1, Case 2



충돌방지 해법

여기서,

[Case 1] : 현재 스케줄링 중인 OHT의 이동해야 할 노드가 Turnout 노드 중 동시에 두 대 이상의 OHT가 동시에 진입할 수 있는 노드인 경우

[Case 2] : 현재 스케줄링 중인 OHT의 현재 노드위치가 진입점이 두 개 이상인 경우(이 경우 노드의 바로 이전 노드라면 해당 노드의 예약 상태에 따라 다음의 해법과 같은 과정을 따른다.)

3.3.2 충돌방지 해법

본 논문에서는 이송작업중인 OHT 상호간의 충돌을 회피 할 수 있도록 하기 위해서 최적경로 탐색 알고리즘에 다음과 같은 충동방지 로직을 추가하여 충돌을 방지할 수 있도록 시스템을 구현한다.

3.3.1절에서 언급한 충돌이 발생할 수 있는 상황이 발생하였을 경우 다른 OHT와의 충돌을 방지하기 위해 그 Turnout 노드 앞까지만 이동 시킨 후 현재 이동하려는 노드가 예약되어 있지 않으면 현재 스케줄링 중인 OHT가 그 노드를 예약하고 이동한 후 그 노드를 지날 때 Release 한다.

현재 이동하려는 노드가 다른 OHT에 의해서 예약되어 있으면 일정시간 만큼 대기한 후 다시 이동 경로에 대한 목적지 노드를 스케쥴링한다.

3.3.2.1 충돌방지 해법에 의한 명령전송(From_OCS To_OHT)

스케줄링 중인 OHT의 현재 경로 즉, OHT의 현재 위치에서부터 현재 경로상의 목적지(Source or Destination) 노드까지의 경로 상에서 다른 OHT와의 충돌을 고려하여 최대한 이동할 수 있는 노드를 찾아 최대이동 목적지의 경로정보와 함께 OHT에게 이송명령을 내린다. OHT는 자율 주행 방식으로 운행되기 때문에 이송을 지령을 받아 주행 중인 OHT는 주행경로에 존재하는 모든 OHT를 고려하여야만 한다. 레일(rail) 위에 존재하는 모든 OHT는 다음의 Case 1의 상태와 Case2의 상태로 정의된다. 따라서 본 프로세싱(Processing)은 이송경로에 존재하는 이송중인 OHT와 작업을 종료하고 정지해 있는 OHT 경로상에 있는 노드들(링크들)을 차례로 조사하면서 다른 OHT의 상태(Case 1, Case 2)에 따라 다음과 같이 처리한다.

여기서,

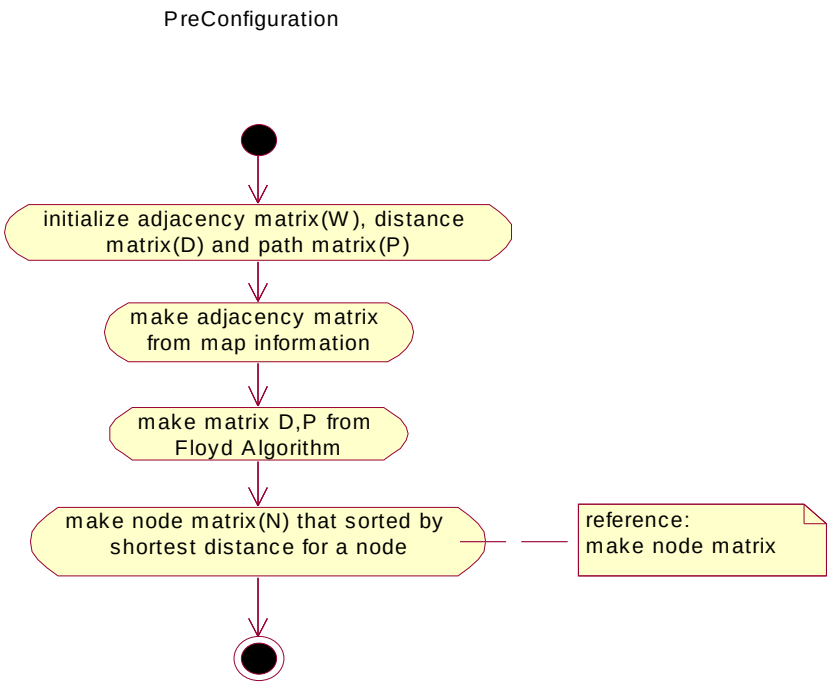
Case 1 : 다른 OHT의 상태가 Free인 경우
Case 2 : 다른 OHT의 상태가 Working인 경우

Case 1의 경우 Free OHT을 단순이동시키며 이때, 단순 이동시킬 목적지와 단순 이동 로직 안에는 이동시킬 목적지까지의 경로를 구한다. 그 경로 안에서 OHT가 갈 수 있는 노드를 검색한 후 단순 이동명령을 수행한다. 이 과정에서 Free 상태인

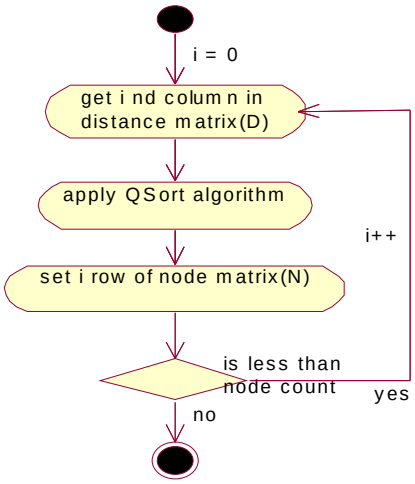
OHT가 존재하면 마찬가지로 단순이동 명령을 해당 OHT에게 지령한다.(이 단순이동명령이 내려진 후에 이전의 단순이동명령이 순서대로 수행되도록 해야 한다)

Case[2]의 경우 다른 OHT의 단위 이송 목적지가 현재 OHT의 경로 안에 있다면 다른 OHT의 단위 이송 목적지 바로 이전 노드까지가 최대 이동 가능한 노드로 판단한다. 만일 다른 OHT의 단위 이송 목적지가 현재 OHT의 경로를 벗어난다면 다른 OHT의 경로(단위 이송명령에 의한 경로)와 현재 작업스케줄중인 OHT의 경로를 서로 비교하면서 다른 OHT의 경로와 분리되는 지점(노드 A)까지는 안전하게 이동할 수 있으므로 노드 A부터 다시 다음 노드로의 이송가능 여부를 판단한다.

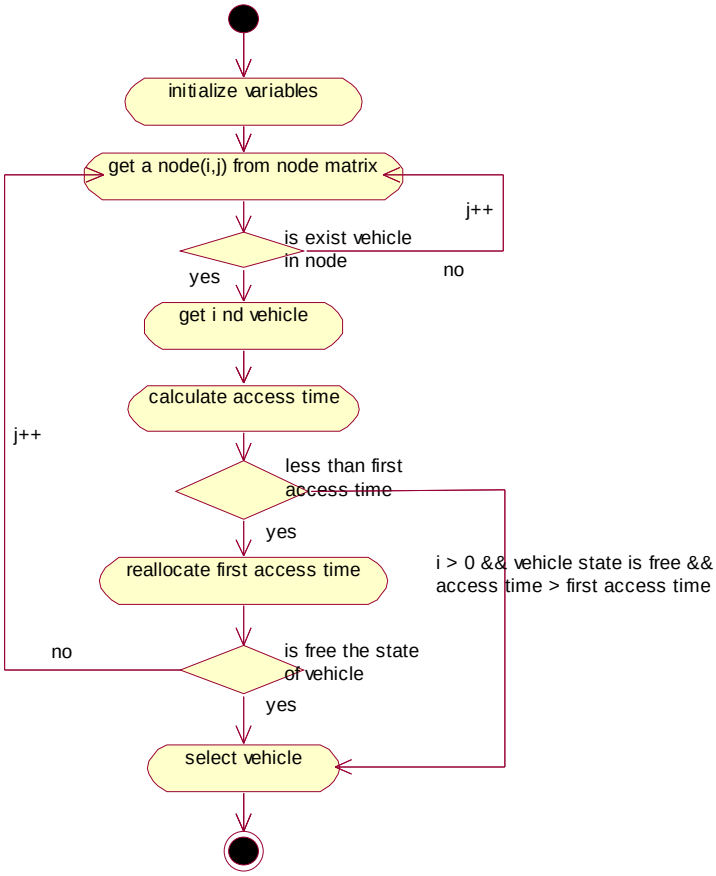
타 OHT 전방에 존재하는 OHT들을 차례로 조사하면서 해당 OHT의 목적지와 경로를 마찬가지로 비교하여 이송 가능한 노드를 확인한다. 이때, 이동 가능한 노드는 최소한 노드 A 이상이다. 다음의 Fig. 3.3은 알고리즘을 구현하기 위하여 설계한 Flow-chart 이며 설계된 Flow-Chart는 다음의 Fig. 3.4와 같이 프로그램 된다. Fig. 3.4는 알고리즘을 소프트웨어(C언어) 적으로 구현한 소스코드의 핵심부분을 발췌한 것이다.



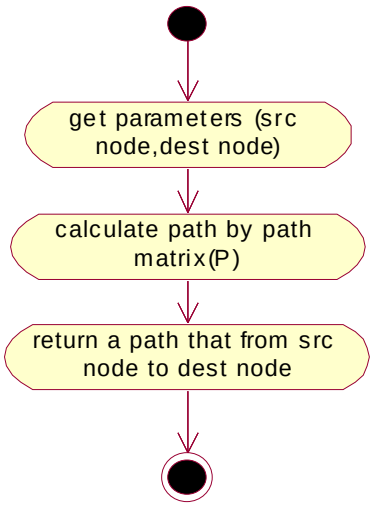
make node matrix(N)



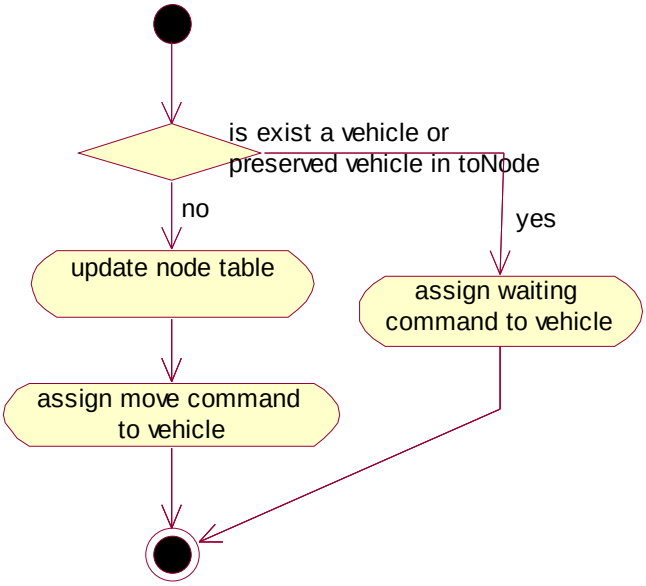
select Vehicle



calculate Path



collision prevention



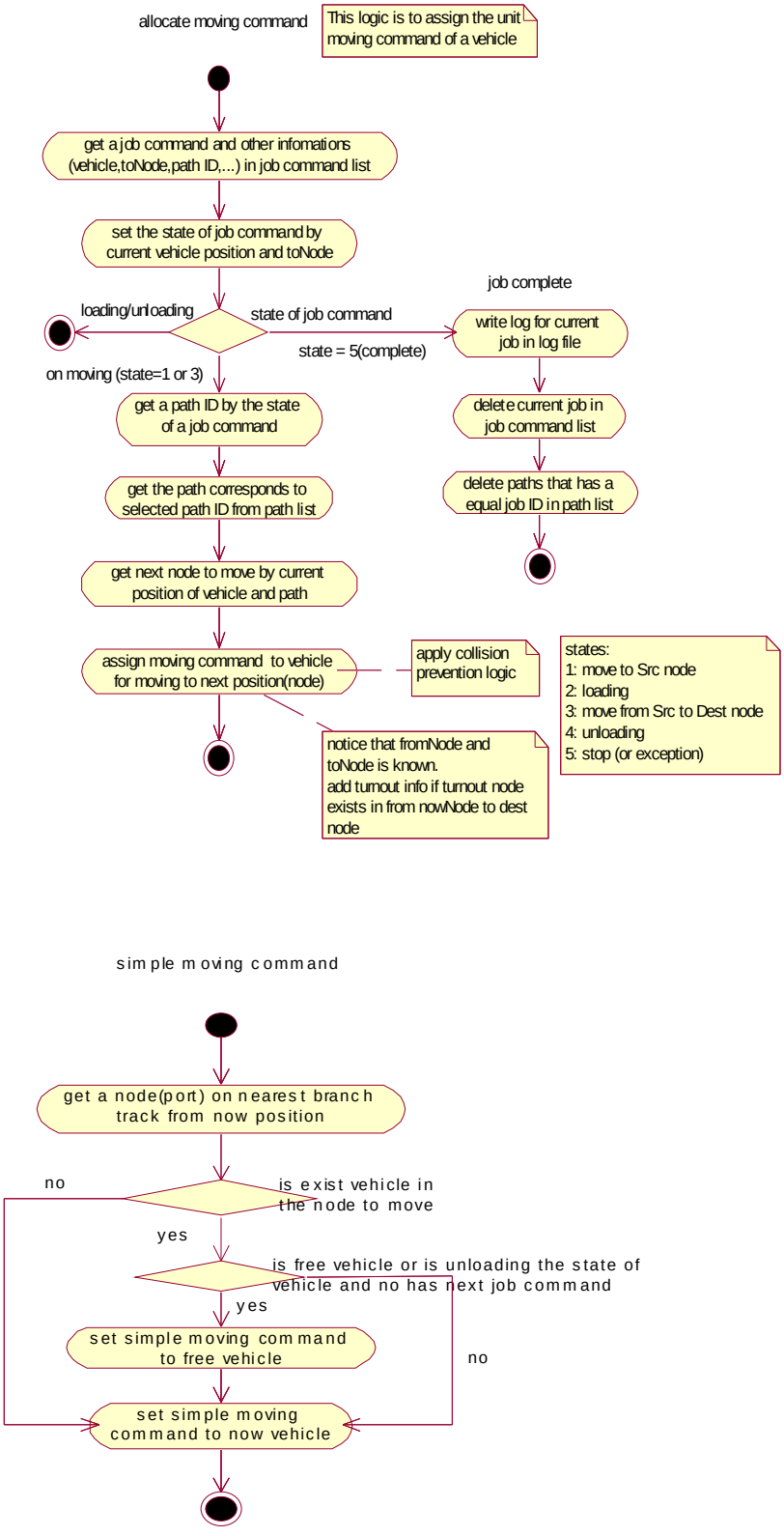


Fig.3.3 Flow-Chart for Algorithm Realization

```
int i, j, k; //변수선언
for(i=1; i<=n; i+1) //행의 개수(인접행렬)
{
    for(i=1; j<=n; j+1) //열의 개수(인접행렬)
    {
        D[i][j]=A[i][j]; //인접행렬[A]를 copy하여 거리경로 계산을 위한 행렬[D]를
                         //생성
        P[i][j]=A[i][j]; //인접행렬[A]를 copy하여 거리경로 계산을 위한 행렬[P]를
                         //생성
    }
}
for(k=1; k<=n; k+1) //행의 개수
{
    for(i=1; i<=n; i+1) //열의 개수
    {
        for(j=1; j<=n; j+1) //최소 거리를 계산하기 위한 비교 노드수
        {
            if(D[i][j]>D[i][j]+D[k][j]) //i~j, j~k, k~j의 거리를 비교하여 작을 경우
            {                           //D에는 거리의 합을 갱신하고 P에는 경유점 k
                D[iπj]=D[i][j]+D[k][j];//를 갱신함
                  P[i][p]=k;
            }
        }
    }
}
```

Fig. 3.4 Source code of Algorithm Realization

3.4 OCS 시뮬레이터 구현

OCS는 MCS로부터 지령된 물류 이송 정보를 바탕으로 OHT로 하여금 최적의 이송경로를 주행하며 Foup을 이송하기 위한 이동명령을 지령한다. 본 절에서는 본 논문에서 제안한 최적경로 선정 알고리즘 및 충돌방지 로직(logic)에 대한 타당성을 검증하기 위해서 구현한 OCS 시뮬레이터에 대하여 논의하고자 한다. 본 논문에서 구현한 OCS 시뮬레이터는 PC 상에서 시뮬레이션 할 수 있도록 소프트웨어적으로

구현되었다. 다음의 Fig. 3.5는 본 논문에서 구현한 시뮬레이션 프로그램의 전체 프로세싱 구조를 도시화한 것이다.

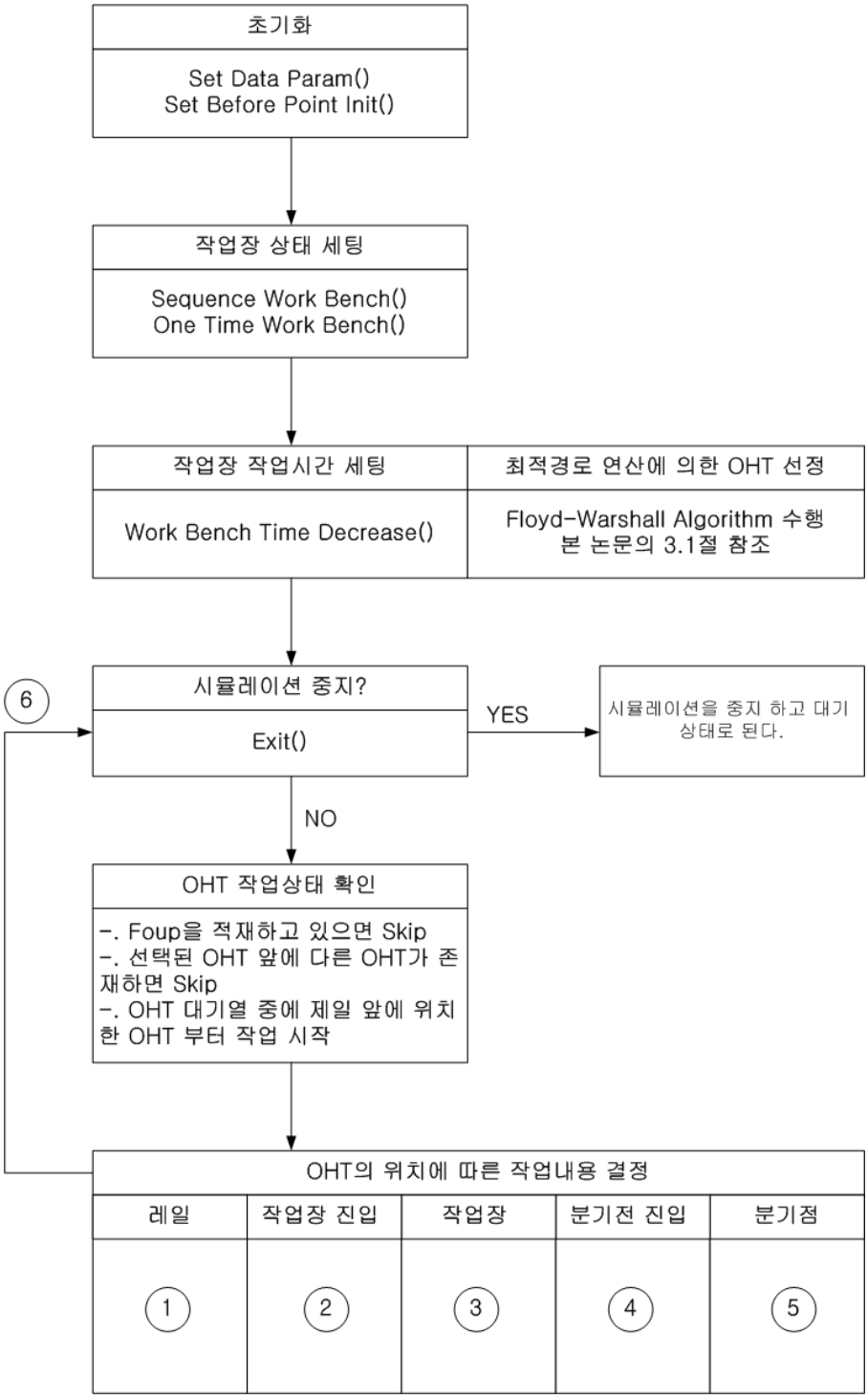


Fig. 3.5 Drawing of Simulation Procedure

본 시뮬레이터는 Fig. 3.5에 나타나 있듯이 GUI(General User Interface) 구성 툴(tool)을 이용하여 시뮬레이션할 전체 작업 공정을 시뮬레이션 PC의 모티터 상에 구성하며 시뮬레이션을 수행한다. 이송 명령이 발생하면 최적경로를 연산하고 이송 명령을 지령하기에 최적의 위치에 있는 OHT를 선정한다. 이후에 OHT는 이송 동작을 수행하며 시뮬레이션이 시작된다. 본 시뮬레이션은 주행동작을 수행하는 레일 및 작업을 수행하는 작업장, 방향 전환을 하는 분기점에서 각각의 다른 연산을 수행하여야 하기 때문에 본 시뮬레이터는 OHT가 이송을 동작을 수행함에 있어 OHT의 위치(레일(rail), 작업장 진입, 작업장 도착, 분기점 진입, 분기점 도착)에 따라서 별도의 시뮬레이션 순서도를 설계하였으며 이를 기반으로 시뮬레이션을 수행하도록 구현한다. 시뮬레이션을 중지할 때까지 시뮬레이션은 무한히 반복된다. OHT의 위치에 따른 시뮬레이션 순서도는 다음의 Fig. 3.6, 3.7, 3.8, 3.9, 3.10과 같으며 설계된 순서도는 시뮬레이터를 구성하는 주요 프로그램 클래스 함수의 구조를 도시화한 것과 같다. Fig. 3.11은 설계된 클래스 함수의 구조를 프로그램 상에서 실제로 구현한 코드의 주요 부분을 발취한 것이다..

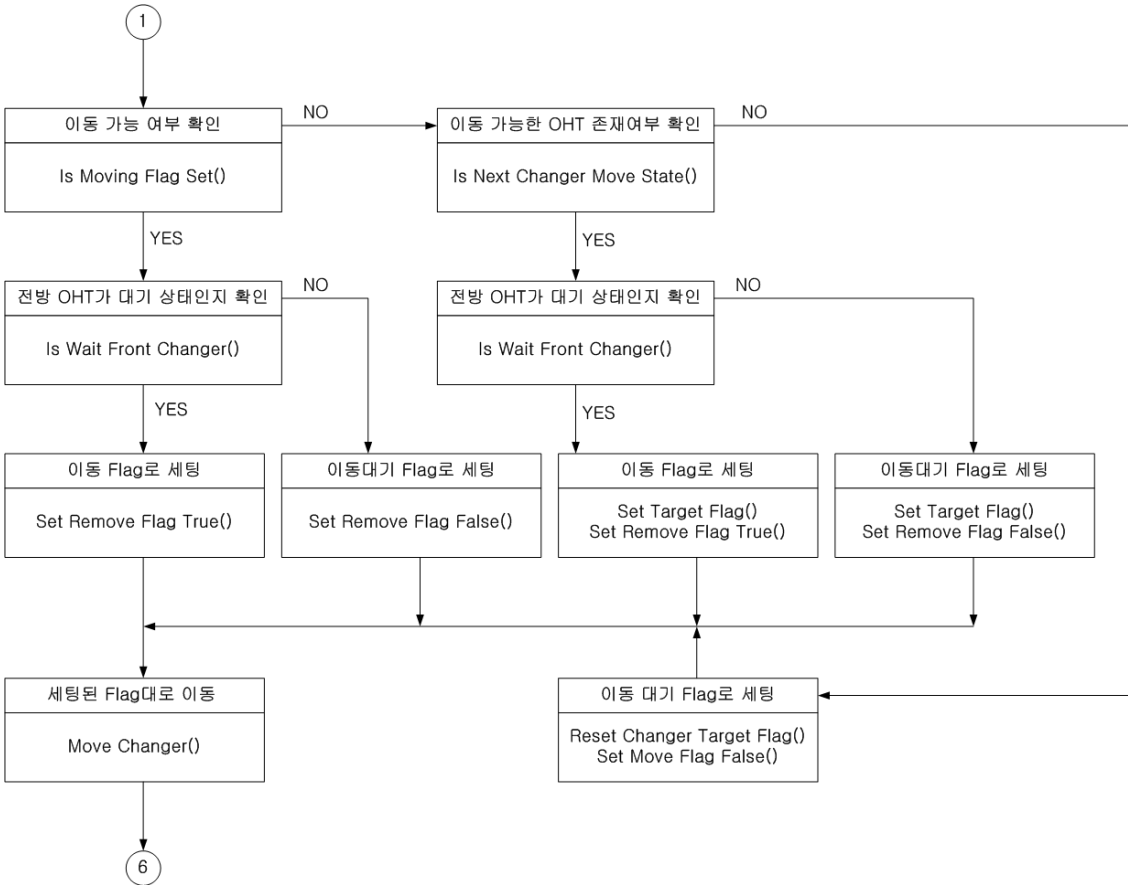


Fig. 3.6 Simulation Flow-Chart of OHT Traveling Rail

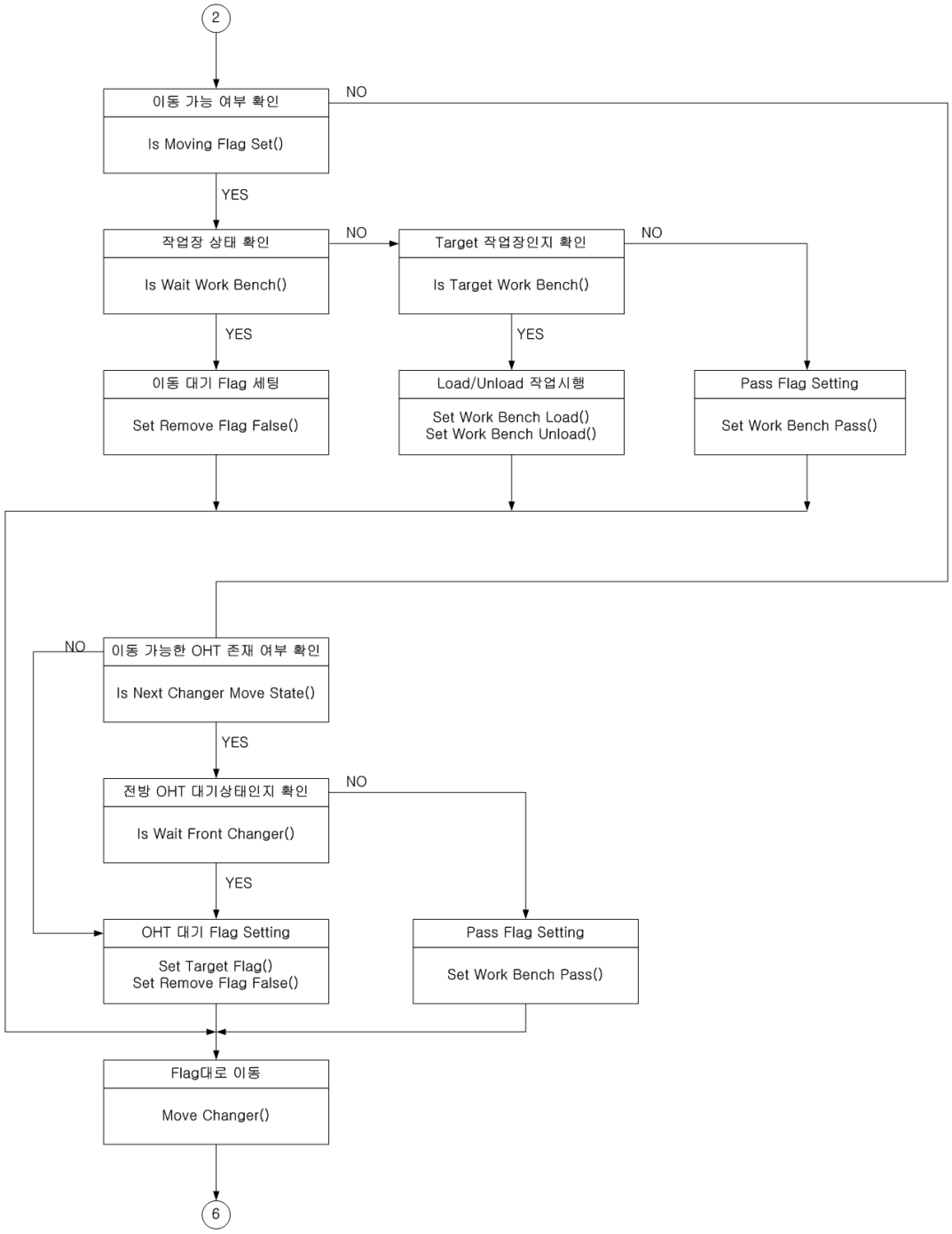


Fig. 3.7 Simulation Flow-Chart of OHT into Job Station

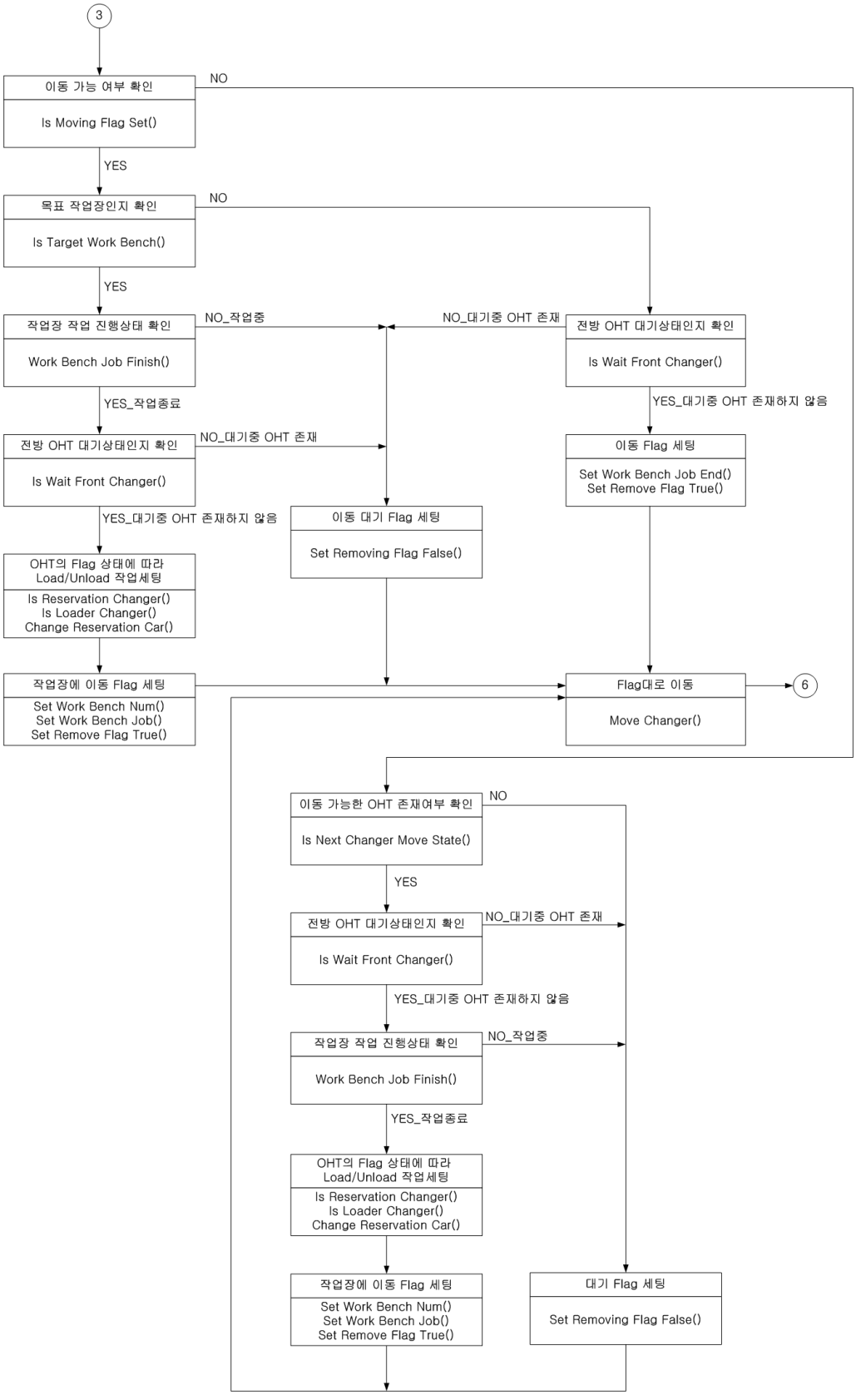


Fig. 3.8 Simulation Flow-Chart of OHT Arrived Job Station

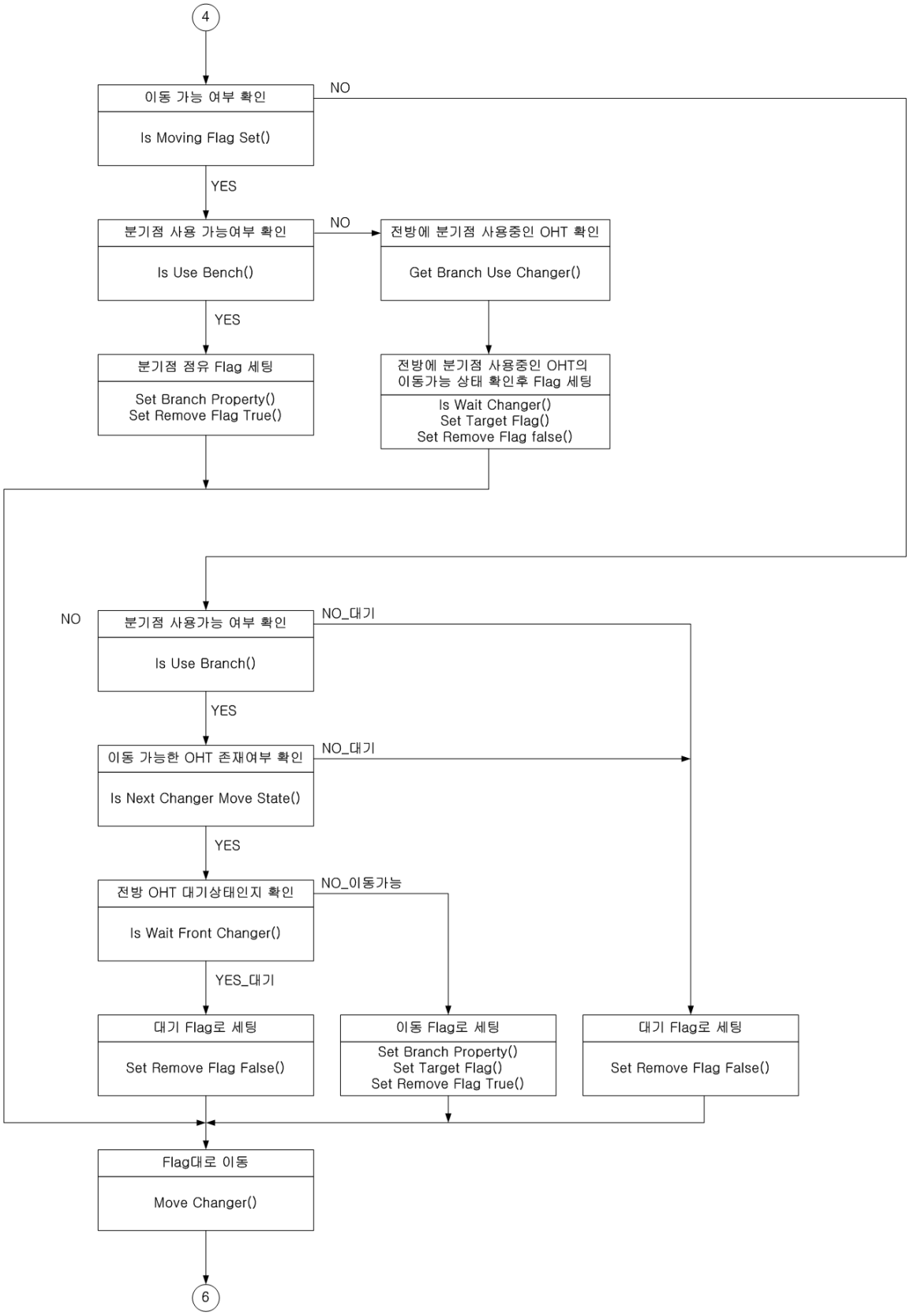


Fig. 3.9 Simulation Flow-Chart of OHT into Junction Point

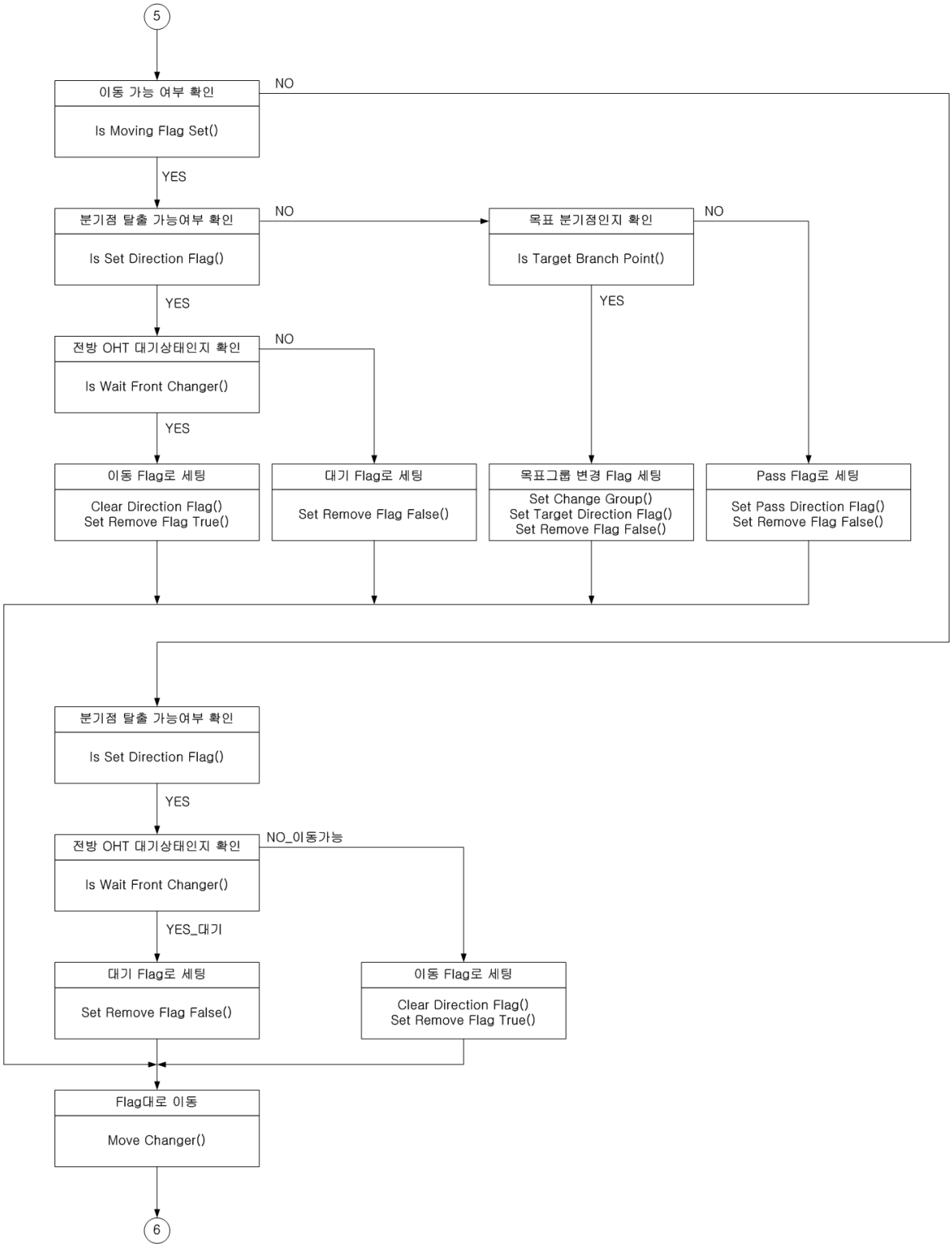


Fig. 3.10 Simulation Flow-Chart of OHT arrived Junction Point

```
// TRouteAlgorithm.h: interface for the CTRouteAlgorithm class.
//
//////////////////////////////////////////////////////////////////////

#if
!defined(AFX_TROUTEALGORITHM_H__E3BC88F3_DBBF_4A37_B588_8874BFCC26AE_
_
            INCLUDED_)
#define
AFX_TROUTEALGORITHM_H__E3BC88F3_DBBF_4A37_B588_8874BFCC26AE__
            INCLUDED

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#include "TRouteData.h"

class CTRouteAlgorithm : public CObject
{
public:
        CTRouteAlgorithm();
        virtual ~CTRouteAlgorithm();

public:
        CT레일(rail)GroupData          *m_p레일(rail)GroupData;
        CTChangerData                  *m_pChangerData;
        CTWorkBenchData                *m_pWorkBenchData;
        CTBranchData                   *m_pBranchData;
        CTSignalData                   *m_pSignalData;
        CTJoinCountData                *m_pJoinCountData;
        CTGroupCountData               *m_pGroupCountData;

public:
void SetDataParam(CTrailGroupData *prailGroupData, CTChangerData *pChangerData,
                    CTWorkBenchData *pWorkBenchData,CTBranchData
                   *pBranchData, CTSignalData *pSignalData, CTJoinCountData
                   *pJoinCountData, CTGroupCountData *pGroupCountData);

        BOOL FrontChanger(int iChangerNum);
        // 전방 OHT 존재여부 확인
```

```
void RearChanger(int iChangeNum, int *iTempChanger, int *iTempCount);
        //현재위치에서 최후방 OHT를 찾아 번호와 위치를 리턴함

        void MoveChanger(int iChangerNum);
        // OHT를 다음위치로 이동함(분기점이동포함)

        int  ChoiceNowPoint(int iChangerNum);
        // 현재 OHT 위치를 반환함(분기점,작업장, 레일)

        BOOL WorkBenchTimeDecrease();
        // 작업시간 감소함수

        BOOL IsLoaderChanger(int iChangerNum);
        // OHT가 Foup을 싣고있는 OHT인지 확인

        BOOL IsTargetWorkBench(int iChangerNum, int iPositionFlag);
        // 타그룹에서 목표그룹으로 찾아가기 위해 레일그룹을 확인하는 함수

        void IsUnloadJob(int iChangerNum);
        // OHT가 Foup을 내려놓고 OHT와 작업장 세팅

        void IsLoadJob(int iChangerNum);
        // OHT가 Foup을 싣고  OHT와 작업장세팅

            BOOL ReallocationWorkBench1(int iChangerNum, int *iReservationCar, int
                                                    *iWorkBenchNum);
        // 현재 작업장에 할당된 OHT번호와 target작업장 번호를 리턴함

       BOOL ReallocationWorkBench2(int iChangerNum, int *iReservationCar, int
                                        *iWorkBenchNum);
        // OHT의 다음위치  작업장에 할당된 OHT번호와 target작업장 번호를 리턴함

       BOOL bIsExitFrontChanger(int iChangerNum1, int iChangerNum2,
                                                int iWorkBenchNum);
        // 진입 작업장 앞에 OHT가 존재하는지 확인

       void ChangeReservationCar(intiChangerNum1, int iChangerNum2, int
                                                iWorkBenchNum);
        // 전방 OHT에 작업장번호를 넘김
```

```
BOOL ReallocationGroup(int iChangerNum);
// 레일간 OHT의 할당 갯수를 맞추기 위한 함수.

void SetChangeGroup(int iChangerNum);
// OHT가 이동한 후에 위치 포인트 증가함수

BOOL IsWaitWorkBench(int iChangerNum);
// 다음위치가 OHT를 기다리고 있는 작업장인지 확인

BOOL IsWaitNowWorkBench(int iChangerNum);
// 현재위치가 OHT를 기다리고 있는 작업장인지 확인

void SetWorkBenchUnload(int iChangerNum);
// 다음위치가 Foup을 unload할 Target작업장인지 확인

void SetWorkBenchLoad(int iChangerNum);
// 다음위차가 Foup을 load할 target작업장인지 확인

void SetWorkBenchPass(int iChangerNum);
// 다음작업장이 생산작업을 시작하게 함

void ChangerhInitAssignment();
// 작업장에 OHT할당

void ChangerhInitAssignment(int iChangerNum);
// 빈OHT 작업장 지날때 재 할당함수

BOOL IsWaitFrontChanger(int iChagerNum);
// OHT의 다음위치에 다른 OHT가 있는지 확인

BOOL IsWaitChanger(int iChangerNum);
// 예약을 받지 못한 빈 OHT인지 확인

BOOL SetMoveChanger(int iChangerNum);
// 이동 포인트에 OHT가 존재시 대기상태로 세팅

void SetMovingFlag(int iChangerNum);
// OHT를 움직이기 위한 플래그 세팅
```

```
void SetOnePointMove(int iChangerNum);
// OHT가 한칸을 움직인후 위치포잊트 저장

BOOL CheckNextWorkBench(int iChangerNum);
// 현재 지나가는 작업장의 옆에 작업장이 작업중인지 확인

BOOL IsMovingFlagSet(int iChangerNum);
// 현재 OHT가 장애물에 대해 움직일수 있는 상태인지확인

void SetWorkBenchNum(int iChangerNum);
// OHT에 지금 지나온 작업장의 번호를 기억시킴

BOOL IsUseBranch(int iChangerNum);
// 분기점에 OHT가 이동가능한 상태인지 확인

int  GetBranchUseChanger(int iChangerNum);
// 분기점을 사용하고 있는 OHT번호를 얻음

void SetBranchProperty(int iChangerNum);
// 외길의 진행방향을 세팅함

void ClearDirectionFlag(int iChangerNum, int iPositionFlag);
// 분기점에서 OHT가 지나가고 난후 초기화로 세팅

void SetTargetDirectionFlag(int iChangerNum);
// 분기점에서 OHT가 탈출후 새로운 현재위치를 갱신함

void SetPassDirectionFlag(int iChangerNum);
// 분기점이 이동 가능한 상태에서 분기점을 통과 후 다음 위치를 OHT에 저장

BOOL IsSetDirectionFlag(int iChangerNum);
// 분기점의 방향상태가 이동 가능한 상태인지 확인

BOOL IsTargetBranchPoint(int iChangerNum);
// 현재 분기점이 OHT의 목표지점으로 갈 수 있는가를 확인

void SequenceWorkBench();
// 작업장의 동작순서를 세팅 안하였을 때 순서적으로 작업장이 가동되도록 함
```

```
void OnetimeWorkBench();
// 작업장이 단 한번만 진행되도록 세팅하는 함수

BOOL WorkBenchJobFinish(int iChangerNum);
// 현재 OHT가 존재하는 작업장의 작업이 끝났는지 확인

BOOL IsReservationChanger(int iChangerNum);
// OHT가 예약상태인가를 확인

void SetWorkBenchJob(int iChangerNum);
// OHT가 목표작업장에 도착하여 Foup을 load/unload시 작업장 플래그 및
// OHT 플래그 갱신

void SetBeforPoint();
// OHT의 이동전 좌표 세팅

void SetBeforPointInit();
// 현재 OHT좌표의 전위치로 복귀

void SetReMoveFlageFalse(int iChangerNum);
// OHT가 정지대기상태에 들어갈때 플래그 세팅

void SetReMoveFlageTrue(int iChangerNum);
// OHT 정지상태에서 움직일때 플래그 갱신

BOOL IsNextChangerMoveState(int iChangerNum);
// 앞OHT가 작업대기 상태인지 확인

void SetTargetFlag(int iChangerNum);
// 현재 OHT가 작업대기중에 뒤의 OHT가 이동요청이 들어오면 뒷 OHT의 목
//표 좌표를 받음

BOOL IsWorkBenchWaiteState(int iChangerNum);
// OHT가 목표작업장에 도착하였는데 작업장이 Foup을 OHT에게 load 가능
//한 상태인지 확인함.

void SetWorkBenchJobEnd(int iChangeNum);
// 작업장에서 OHT가 load한 후 작업장 플래그갱신
```

```
        void FindEqualPoint(int iChangerCount, int iChangerNum, int *iEqualCount, int
                                                *iEqualNum);
        // OHT의 위치가 실제 레일에 존재하는 위치인지를 확인

        void ResetChangerTargetFlag(int iChangerNum);
        // OHT에게 목표작업장의 좌표를 reset시킴

        BOOL IsWaiteTimeOver(int iChangerNum);
        // 분기점에서 대기 시간을 카운트함.

        public:
                void InitData();

        }
```

Fig. 3.3.7 Class Source Code for OCS Simulator Realization

제 4장 시스템 구성 및 구현

4.1 전체 시스템 구성

OHT의 최적 제어관리 시스템은 물리적, 논리적으로 시스템을 나누어 구성 된다. 여기서, 물리 시스템은 Fig. 4.1과 같이 OHT 시스템과 외부 시스템으로 나누어서 구분한다. OCS로부터 물류 이송 명령을 받아 실제로 OHT를 운전하며 이송을 수행하도록 하는 시스템이 OHT 시스템이며 OHT를 구동하여 물류를 이송하기 위한 전반적인 물류 이송 정보에 관한 명령을 지령하는 시스템이 외부 시스템이다. 본 논문에서는 사용자가 외부 시스템을 사용하여 OHT 시스템을 제어할 수 있도록 한다. 다시 말하면 사용자로 하여금 MCS 역할을 할 수 있도록 시스템을 구성한다. 이러한 물리 시스템을 서로 연동 할 수 있도록 하기 위해서는 OPC(One Board Programmable Controller) 시스템, 핸드-터미널(hand-terminal) 시스템, 프로그램 메니져(program manager) 시스템의 3가지 논리 시스템이 구성되어야 한다. 각 논리 시스템은 서버(server)와 클라이언트(client)의 기능 개체를 갖는다. 각 논리 시스템에서 기능 개체의 한 쪽은 구동 개체이고, 나머지 한 쪽은 명령 개체이다. 이러한 각 시스템 구성에 따른 기능개체에 대한 구분을 Table. 4.1에서 정리하였다.

Fig. 4.1 Structure of OHT Control and Management System

Table. 4.1 Sorting of OHT Control and Management System

물리 시스템 구분	논리 시스템	기능 개체
OHT 시스템	OPC 시스템	OCS Client (구동개체)
	핸 드 - 터 미 널 (hand-terminal) 시스템	핸드-터미널(hand-terminal) Server (구동개체)
	프로그램 메니져(program manager) 시스템	프로그램 메니져(program manager) Server (구동개체)
외부 시스템	OPC 시스템	OCS Server (명령개체)
	핸 드 - 터 미 널 (hand-terminal) 시스템	핸드-터미널(hand-terminal) Client(명령개체)
	프로그램 메니져(program manager) 시스템	프로그램 메니져(program manager) 클라이언트(client) (명령개체)

OPC 시스템은 물류 이송을 위해 OHT를 구동하기 위한 시스템이며 OCS 클라이언트(client)와 OCS 서버(server)의 기능 개체로 구성된다. OCS 클라이언트(client)는 구동 개체로 OHT 시스템에 포함되어 OCS로부터 이송명령을 지령 받아 OHT를 구성하는 각 엑츄에이터(actuator)를 직접 제어하며 물류 이송을 수행할 수 있도록 하며 OCS 서버(server)는 명령 개체로 외부 시스템에 포함되어 공정 내에 존재하는 모든 OHT를 통합 관리하며 물류 이송을 지령한다. 또한 OCS 서버(server)에는 본 논문의 핵심인 최적경로 탐색 알고리즘이 탑재되어 OHT로 하여금 최단 시간에 원활한 이송작업을 수행 할 수 있도록 모든 작업내용의 전반적인 사항을 스케쥴링(scheduling)하여 적절한 OCS 클라이언트(client)에 작업 명령을 지령하는 명령개체이다.

핸드-터미널(hand-terminal) 시스템은 구동 정보 학습 및 테스트를 위해 사용되는 시스템이다. 초기에 OHT를 레일(rail)에 로드(load)하여 각 작업장의 주행 및 승강, 선회, 횡행 동작에 관한 티칭(teaching) 및 구동 학습 작업을 원격으로 수행하고 관련 작업을 수행하며 얻어진 데이터를 OHT에 입력하기 위한 시스템이다. 핸드-터미널(hand-terminal) 시스템은 핸드-터미널(hand-terminal) 서버(server)와 핸드-터미널(hand-terminal) 클라이언트(client)의 기능개체로 구성된다. 핸드-터미널(hand-terminal) 서버(server)는 구동 개체로 OHT 시스템에 포함되고, 핸드-터미널(hand-terminal) 클라이언트(client)는 명령 개체로 외부 시스템에 포함된다. 핸드-터미널(hand-terminal) 시스템의 서버(server)와 클라이언트(client)는 OHT의 수동 조작 모드에서 무선랜(Wireless LAN) 인터페이스 방식으로 상호간에 접속이 가능하며 수동 조작자는 핸드-터미널(hand-terminal) 클라이언트(client)를 통하여 OHT

원격 조작할 수 있다. 따라서 핸드-터미널(hand-terminal) 서버(server)는 구동 정보 학습 및 테스트를 위해 작업 명령에 따라 OHT를 구동하는 구동 개체이며, 핸드-터미널(hand-terminal) 클라이언트(client)로부터 작업 명령을 수신한다.

프로그램 메니져(program manager) 시스템은 OHT의 직접적인 구동동작과는 무관하며, 다른 논리 시스템(OPC 시스템, 핸드-터미널(hand-terminal) 시스템)을 관리하기 위한 시스템이다. 프로그램 메니져(program manager) 시스템은 프로그램 메니져(program manager) 서버(server)와 프로그램 메니져(program manager) 클라이언트(client)의 기능 개체로 구성된다. 프로그램 메니져(program manager) 서버(server)는 구동 개체로 OHT 시스템에 포함되고, 프로그램 메니져(program manager) 클라이언트(client)는 명령 개체로 외부 시스템에 포함된다.

프로그램 메니져(program manager) 서버(server)는 다른 논리 시스템(OPC 시스템, 핸드-터미널(hand-terminal) 시스템)을 강제로 종료하고 실행할 수 있는 기능과 OHT 시스템을 강제로 셧-다운/리부팅(Shutdown/Reboot)할 수 있는 기능을 가진 구동 개체이고, 프로그램 메니져(program manager) 클라이언트(client)로부터 작업 명령을 수신한다. 프로그램 메니져(program manager) 클라이언트(client)는 OHT 시스템을 관리하기 위해 프로그램 메니져(program manager) 서버(server)에 작업 명령을 보내는 명령 개체이다.

이와 같은 논리적인 시스템은 유지보수 및 설치 시운전 상황을 고려하여 충분한 유연성을 갖출 수 있도록 구현 되어야 한다. 따라서 본 논문에서는 위에서 언급한 각 논리적인 시스템의 특성을 충분히 확보 할 수 있도록 하기 위해서 소프트웨어 모듈 형식으로 구현한다.

4.1.1 OCS 시스템

본 논문에서 구성한 OCS는 물리적인 개체의 하드웨어 및 논리적인 개체인 소프트웨어로 구분된다. OCS는 MCS로부터 지령된 물류 이송 정보를 기반으로 하여 전체 OHT의 작업 스케쥴을 연산하고 물류 이송명령을 지령하는 명령 개체이다. 이러한 OCS의 기능을 극대화시키고 방대한 연산 시간을 최소화하기 위하여 OCS의 하드웨어 및 소프트웨어를 다음의 Table. 4.2.와 같이 구성하였다.

Table. 4.2. Construction of OCS

Hardware	CPU	Intel PentiumⅣ 2.4Ghz
	Memory	DDR 1GB
	Video Memory	64MB
	Cash Memory	512KB
	Lan Interface Module	Card : USB 10Mbps
		Access Pint : 49Mbps
Software	OS	Window 2000 서버(server)
	Programing Tool	Microsoft Visual C++ 6.0

4.1.2 OHT의 시스템

OHT의 시스템은 OPC 시스템의 구동 개체, 핸드터미널(hand-terminal) 시스템의 구동 개체가 주를 이룬다. 또한 OCS와의 통신 인터페이스를 위하여 OCS에서 사용한 통신 장치와 동일한 장치를 사용한다. 원활한 물류 이송을 위하여 OHT가 필수적으로 갖추어야 할 기능을 정리하면 다음의 Table. 4.3과 같다.

Table. 4.3 Control object by OHT Function

OHT의 필수적인 기능	제어 대상
주행	주행 동작을 위한 구동기는 AC 서보모터를 사용한다.
승강	승강 동작을 위한 구동기는 AC 서보모터를 사용한다.
횡행	횡행 동작을 위한 구동기는 Stepping 모터를 사용한다.
선회	선회 동작을 위한 구동기는 Stepping 모터를 사용한다.
분기	레일(rail)의 분기점에서 분기 동작을 위한 구동기는 DC 모터를 사용한다.
낙하방지	Foup이 OHT에 상태에서 Load된 상태에서 이송시 낙하방지 셔터를 작동시키기 위한 구동기는 DC 모터를 사용한다.
Chuck	Foup의 Clamping 동작을 위한 구동기는 DC 모터를 사용한다.
Barcode Reader	OHT가 레일(rail)을 주행하며 각 작업장에 부착된 Barcode No를 Read 하기 위한 방식은 RS-232를 사용한다.

본 논문에서는 위에서 언급한 바와 같은 OHT의 기능을 견실하게 갖추도록 하기 위하여 Fig. 4.2와 같이 OHT 제어 시스템을 구성 하였다. 주행 및 승강용 AC 서보 모터와 선회 및 횡행용 스테핑(stepping) 모터의 변위 및 속도를 제어하기 위한 용도로 4축 모션 컨트롤 하드웨어 모듈(4-axis motion control hardware module)을 채택한다. 분기 및 낙하방지, Chuck용 DC의 정·역 방향 제어를 제어하기 위한 용도로 64ch Isolation Digital I/O Hardware Module을 사용하였으며 OHT의 동작 상태 확인용 TR Type 센서의 상태를 체크하기 위한 용도로 64ch Isolation Digital Input Hardware Module을 채택하였다. 바코드 리더(barcode reader)는 OHT 콘트롤(control) 시스템의 CPU 모듈에 창착되어 있는 직렬통신 포트[Comm Port(RS-232)]와 인터페이스 할 수 있는 모듈을 채택하였다.

OHT를 구성하는 각 제어 대상을 제어하기 위한 시스템의 모든 하드웨어 모듈은 많은 양의 데이터를 고속으로 처리하기에 적합한 방식인 PCI 버스 인터페이스 방식을 채택하여 실시간(real-time) 기반에서 모든 제어 관리에 관련된 데이터 처리가 가능하도록 시스템을 설계 및 구축하였고 각 하드웨어를 원활히 제어하기 위한 소프트웨어는 Window 2000 OS 기반에서 Microsoft Visual C++ 6.0을 사용하여 개발한다. OHT 제어용 하드웨어 모듈의 구성을 도시화하면 다음의 Fig. 4.2와 같다.

Fig. 4.2 Drawing of OHT Control Hardware Module

4.1.3 무선통신 시스템

명령 개체인 OCS와 지령받은 정보를 기반으로 물류이송을 수행하는 구동개체인 OHT간의 물류 이송에 관한 모든 정보 및 데이터 교환은 통신 인터페이스를 기반으로 수행된다. OHT는 레일(rail)을 주행하며 물류를 이송하는 장치이므로 유선식 통신 방식을 적용하기란 불가능하므로 무선 통신 기법을 적용하여야 한다. 본 논문에서 채택한 OCS와 OHT간의 무선 통신 기법은 TCP/IP 프로토콜을 기반으로 하는 무선 통신 기법인 무선랜(wireless lan) 기법을 채택하였다. 본 논문의 모든 물리 시스템에는 USB Wireless LAN Adapter를 탑재하여 상호간에 무선랜(wireless lan)으로 통신 인터페이스 될 수 있도록 하였으며 실제 반도체 공정의 방대한 면적에서도 본 시스템이 무리없이 적용될 수 있음을 검증하기 위하여 무선 통신의 중간 개체로 엑세스 포인트(access point)를 설치 적용하여 시스템을 구성한다. 여기서, 엑세스 포인트(access point)란 대다수의 통신 개체가 불특정 다수로 통신 접속 시 데이터 손실 없이 상호간에 통신을 할 수 있도록 하며 엑세스 포인트(access point) 상호간에 무선으로 통신을 할 수 있는 기능이 있기 때문에 통신 거리의 한계를 극복할 수 있다. 다음의 Table. 4.4에서 엑세스 포인트(access point)에 대한 기능 및 설명을 정리하였다.

Table. 4.4 Function of Access Point

항 목	기 능 설 명
Store-and-forward capability	확장된 LANs의 구성을 가능하게 한다. 다른 LANs과 프레임을 송수신하며, 검사한다.
Frame filtering based on address	주소 데이터베이스와 수신된 프레임의 수신지/목적지의 주소를 사용하여, LANs간에 통신이 필요하지 않은 경우를 파악한다. 이렇게 해서, 전체 데이터 흐름을 줄이고 대역폭의 효율을 증가시킨다.
Data Link layer relay	OSI (Open System Interconnection) 모델의 Data Link layer에서 동작하며, 프로토콜에 관계없이 LAN 연결 서비스의 투명성을 유지한다. 프로토콜에 대한 투명성은 확정된 LAN 서비스에서 중요한 요소이다.
Dynamic address learning	주소 데이터베이스에 새로운 수신지 주소를 자동으로 추가한다. 주소 학습은 프로토콜과 관리 개

	체에 독립적이다. 또한, 주소 데이터베이스에 주소를 기억하는 시간(Aging Timer interval)을 설정할 수 있다. 이 기간을 초과하도록 데이터의 소통이 없는 주소는 주소 데이터베이스에서 제거된다.
Workgroup bridge mode	이 모드에서 Access Point는 Wireless Client와 통신하고, 주소 데이터베이스에 기반해서 패킷을 LANs간에 전송한다.
LAN-to-LAN endpoint bridge mode LAN-to-LAN multipoint bridge mode	이 모드에서 Access Point는 Access Point간에 통신하고, 주소 데이터베이스에 기반해서 패킷을 LANs간에 전송한다.

본 논문에서 사용한 무선통신(Wireless LAN) 시스템은 USB Wireless LAN Device(모든 물리적인 시스템에 적용함)와 엑세스 포인트(access point) 2개를 사용하여 워크그룹 브릿지 모드(workgroup bridge mode) 기반에서 시스템을 구성 및 설계한다. Fig. 4.3에 나타나 있듯이 본 모드는 셀(Cell) 간에 채널은 3 이상으로 분리하는 것이 유리하며, 로밍(roaming)을 위해서 영역에 반드시 중첩구간을 두어야 한다. 본 논문에서는 Fig. 4.3에 나타나 있는 Wireless Client를 각 OHT로 설정하였고 무선 방식으로 상호간에 데이터를 송수신 하므로 레일(rail)을 자율적으로 주행하며 물류를 이송하는 작업을 수행함에 있어 제한이 없도록 한다. 또한 엑세스 포인트(access point)의 오토 로밍(Auto Roaming) 서비스 기능은 데이터의 손실이 없이 엑세스 포인트(access point)간에 자유로운 이동을 지원한다. 엑세스 포인트(access point)를 추가함에 의해, 무선 통신 서비스 영역을 계속 확장해 갈 수 있는 유리함이 있어 통신 거리의 한계를 극복할 수 있다.

Fig. 4.3 Drawing of Wireless LAN Workgroup Bridge Mode

4.2 시스템 구현 상세 구조설계

본 절에서는 구성된 시스템을 실제로 구현하고 실험 및 고찰을 통하여 제안된 알고리즘의 타당성을 검증하기 위하여 시스템을 실제로 구현하기에 앞서 필히 선행하여야 할 과정인 시스템의 구현을 위한 상세적인 구조 설계에 대하여 논의 하고자 한다.

4.1절에서 논의 하였듯이 모든 시스템을 물리적, 논리적인 개체로 구분하여 정의하였다. 시스템 구분에 의한 모든 개체는 구동 개체와 명령 개체로 최종 구분하였다. OHT의 물류에 이송에 관련한 모든 작업은 각 시스템별 기능 개체간의 명령에 의해서 수행되므로 신뢰성과 견실함이 충분히 확보된 소프트웨어 통신 구조가 기반

되어야 한다. 더불어, 지령된 명령과 동일하게 이송 작업을 수행할 수 있고 각 개체들 간에 충돌 문제 등으로 인한 시스템 에러(error)를 최소화할 수 있도록 신뢰성과 견실함이 충분히 확보된 소프트웨어 구동개체 구조 또한 기반되어야 한다.

본 논문에서 상세적으로 설계한 각 시스템별 개체의 구조에 대한 내용은 다음과 같으며 모든 개체들 간의 구조 설계는 본 논문의 주요 연구 대상인 OHT를 중심으로 수행 한다.

4.2.1 통신 소프트웨어

SEMI Standard 통신 규약에서는 상위 명령 개체로부터 지령을 받은 하위 개체는 명령의 수신을 올바르게 완료되었다는 ACK 메시지를 보내야 함을 통신상에서 지켜야 할 기본 규칙으로 규정하고 있다.

본 논문에서는 SEMI Standard 규정을 준수하고 각 개체들 간의 원활한 통신을 수행할 수 있도록 하기 위해서 Fig. 4.4와 같이 소프트웨어 메인(main) 통신 구조를 설계 한다. Command Receiving Thread가 명령을 수신한 경우, 수신 명령을 동작 모듈로 보내며 동작모듈에서는 수신된 명령과 동일한 동작을 수행한다. 이때, 동작 모듈에서 명령 전송이 필요한 경우, 명령을 Command Sending Thread로 보낸다. 본 과정을 수행함에 있어서 동작 모듈이 대기 상태에 놓이지 않도록 Mid Buffer(Uploading Buffer)를 사용해서 Command Sending Thread로 보낸다. Command Sending Thread에서 명령을 송신한 후 Command Receiving Thread가 응답을 수신하는 경우에는 응답의 수신을 Command Sending Thread에 알려 명령 처리가 완료되었음을 인식하도록 한다. Command Receiving Thread가 명령을 수신한 경우에는 수신한 명령에 대한 응답을 Acknowledge Sending Thread로 보내서 명령을 지령한 개체에 Acknowledge Message를 보내도록 한다. 여기서, Command Receiving Thread는 Socket의 상태를 감시하고 명령과 응답을 수신한다. Command Receiving Thread는 Socket의 Monitoring을 신속히 하기 위해, 수신한 명령의 응답을 바로 보내지 않고, Queue를 사용해서 Acknowledge Sending Thread로 보낸다.

Fig. 4.4 Design of Communication Software Module Structure

4.2.2 통신 소프트웨어 모듈 동기화

각 명령 개체와 구동개체간의 올바른 명령 송·수신을 실현하기 위해서는 모든 통신 소프트웨어가 동기화되어 통신 프로세스(Process)가 진행되어야 한다. 본 논문에서는 각 개체 간에 통신에 의한 명령 처리 과정이 원활히 진행되고 통신 프로토콜의 충돌 등으로 인한 통신 페일(Fail)을 충분히 방지 할 수 있도록 통신 모듈간의 동기화 구조를 설계하였으며 통신 모듈의 정규 동기화 과정을 사용하여 명령과 응답을 송수신할 수 없는 상황을 따로 설정하여 발생하는 상황별로 동기화를 구현할 수 있도록 통신 모듈간의 동기화 구조를 설계한다. 본 논문에서는 통신 모듈의 동기화 설계를 정규 동기화 부분과 비정규 동기화 부분으로 나누어 수행하였으며 그에 대한 구체적인 내용은 다음의 4.2.2.1절과 4.2.2.2절에서 논의 한다.

4.2.2.1 정규 동기화

4.2.1절에서 논의 하였듯이 Command Receiving Thread는 Socket을 감시하는 기능을 담당한다. Socket을 감시하는 과정에 있어서 시간을 요하는 작업을 회피하기 위해 다음의 Fig. 4.5에 나타나 있듯이 명령을 수신하는 경우 Mid Buffer(Pending

Buffer)를 사용하여 동작 모듈로 수신된 명령을 보낸다. 동작 모듈은 새로운 명령을 확인하고, Execution Buffer에 저장한다. 이때, Command Receiving Thread와 동작 모듈은 수신된 명령을 처리하기 위해 비동기로 동작하며 Mid Buffer(Pending Buffer)를 통해서 동기를 맞출 수 있도록 소프트웨어 통신 모듈의 정규 동기화 모드(Mode)를 설계한다. 여기서, “Pending”은 Command Receiving Thread는 수신된 명령의 수용여부를 확인하고 명령에 종류에 따라 해당 Mid Buffer(Pending Buffer)에 저장하며 “Apply”는 동작 모듈이 새로운 명령을 확인하고, 해당 Execution Buffer에 저장한다.

정규 Pending/Apply Flow를 따르지 않을 경우, 동기화 문제를 부분적으로 다루어야 한다. 그리고 동시에 수행할 수 있는 명령(병렬로 처리할 수 있는 명령)을 파악하여 종류별로 Buffer를 따로 두어야 하고, 새로운 명령이 추가될 경우에 속성을 파악하여 적절한 Buffer 구조로 포함시켜야 한다. 그렇지 않으면 동시에 수행할 수 없기 때문에 명령 수행의 효율이 떨어진다. 다음의 Fig. 4.5와 같이 개체가 명령을 수신 하였을 경우 정규 Pending/Apply Flow에 대한 구조를 설계 한다.

Fig. 4.5 Design of Pending/Apply Flow

명령을 전송하는 기능을 수행함에 있어 동작 모듈이 통신으로 인해 대기상태에

놓이는 것을 방지하기 위해 Mid Buffer(Uploading Buffer)를 사용하여 Command Sending Thread로 전송할 명령을 보낸다. 이때, 동작 모듈과 Command Sending Thread는 명령을 전송하기 위해 비동기로 동작하며 Mid Buffer(Uploading Buffer)를 통해서 동기를 맞춘다. 여기서, “Uploading”은 동작 모듈은 명령을 전송해야 하는 경우, 전송이 가능한지를 확인하고 완전한 명령을 만들어 Mid Buffer(Uploading Buffer)에 저장하며. “Apply”은 Command Sending Thread는 새로운 명령을 확인하고, Execution Buffer에 저장한다. 그리고 통신 프로토콜에 맞추어 전송한다. 이후, Command Sending Thread는 새로운 명령을 확인하고, Execution Buffer에 저장한다. 그리고 이미 설계 완료된 통신 프로토콜에 맞추어 전송을 수행하도록 설계한다.

정규 Uploading/Apply Flow를 따르지 않을 경우, 동기화 문제를 부분적으로 다루어야 한다. 그리고 동시에 전송할 수 있는 명령(병렬로 처리할 수 있는 명령)을 파악하여 종류별로 Buffer를 따로 두어야 하고, 새로운 명령이 추가될 경우에 속성을 파악하여 적절한 Buffer 구조로 포함시켜야 한다. 그렇지 않으면 동시에 전송할 수 없기 때문에 명령 전송의 효율이 떨어진다. 다음의 Fig. 4.6과 같이 개체가 명령을 송신할 경우 정규 Uploading/Apply Flow에 대한 구조를 설계 한다.

Fig. 4.6 Design of Uploading/Apply Flow

응답을 수신하는 기능을 수행함에 있어서는 Mid Buffer(Pending Buffer)를 사용

하는 Pending/Apply Flow대신, Thread Event Object를 사용하는 Retry Sending Flow를 사용하도록 설계 한다. Command Sending Thread는 명령을 전송한 후에 응답을 기다리기 위해 대기한다. Command Receiving Thread는 Command Sending Thread에서 전송한 명령과 일치하는 응답을 받은 경우, Command Sending Thread에 Event Object의 Signal을 보낸다. Command Sending Thread는 Event Object의 Signal을 받고 대기 상태에서 해제되며, 응답을 확인해서 현재 명령 전송을 종료할 것인지 재전송할 것인지를 결정하며 명령과 응답의 쌍을 구성하기 위해 자동 생성되는 Order Number를 사용한다. 전송한 명령과 수신한 응답의 Order Number가 정확히 일치할 때, 응답을 수신한 것으로 인식한다. 본 논문에서는 다음의 Fig. 4.7과 같이 개체가 명령을 송신한 후 응답을 수신하는 경우의 통신 소프트웨어의 동기화를 위한 Retry Sending Flow에 대한 구조를 설계 한다.

Fig. 4.7 Design of Retry Sending Flow

명령에 대한 응답을 전송하는 기능을 수행함에 있어서는 Mid Buffer(Uploading Buffer)를 사용하는 Uploading/Apply Flow대신, Queue를 사용하는 Acknowledge Sending Flow를 사용하도록 설계 한다. 앞 절에서 이미 언급 하였듯이 Command Receiving Thread는 Socket을 감시하는 기능을 수행한다. 그러므로 시간을 요하는 작업을 회피하기 위해 Queue를 사용하여 Acknowledge Sending Thread로 명령에 대한 응답을 보낸다. Acknowledge Sending Thread는 Queue에서 전송할 응답을 확인하고, 전송한다. 명령과 응답의 쌍을 구성하기 위해 자동 생성되는 Order Number를 사용한다. 그러므로 수신한 명령에 대한 응답을 전송할 때, 수신한 명령과 같은 Order Number를 사용한다. 다음의 Fig. 4.8과 같이 개체가 명령을 수신한

후 응답을 송신하는 경우의 통신 소프트웨어의 동기화를 위한 Retry Sending Flow 에 대한 구조를 설계 한다.

Fig. 4.8 Design of Acknowledge Sending Flow

4.2.2.2 비정규 동기화

통신 모듈의 정규 동기화 과정을 사용하여 명령과 응답을 송수신할 수 없는 상황이 발생하는 경우, 각 상황별로 동기화를 구현해야 한다. 이와 같은 상황을 다음의 3가지 경우로 설정 한다.

- -. 정규 Pending/Apply Flow없이, 동작 모듈에서 강제로 Mid Buffer(Pending Buffer)를 변경하는 경우
- -. 정규 Uploading/Apply Flow없이, 동작 모듈에서 강제로 Mid Buffer(Uploading Buffer)를 변경하는 경우
- -. 정규 Acknowledge Sending Flow없이, 동작 모듈에서 강제로 Queue를 변경하는 경우

본 논문에서는 전체 시스템을 구현하고 운전하는 도중 위와 같은 상황이 발생할 수 있는 실제 상황과 그에 대한 대처 방안을 고려한 설계 내용을 다음의 Table. 4.5에서 정리 하였다.

Table. 4.5 Substitute of Communication software module irregular synchronization

소프트웨어 통신 모듈의 정규 동기화에 의한 명령의 송·수신이 불가능한 실제 상황	각 상황에 따른 대처방안
통신이 두절된 후, 다시 연결하는 상황	회복의 요구 수준에 따라 통신만 연결하고 그대로 진행할 수 있는 동기화 구조가 요구되지만 본 논문에서는 Command Sending Thread/ Acknowledge Sending Thread/동작 모듈을 동기화 하여 본 상황에 대처 할 수 있도록 설계 한다.
Abnormal Command 수행 중 이전에 수행중인 이전에 수행중인 Normal Command를 변경하는 상황	Command Sending Thread/ Acknowledge Sending Thread/동작 모듈을 동기화 하여 본 상황에 대처 할 수 있도록 설계 한다.
돌발적인 Alarm이 발생된 상황	회복의 요구 수준에 따른 동기화 구조가 요구된다. Command Sending Thread/ Acknowledge Sending Thread/동작 모듈을 동기화 하여 본 상황에 대처 할 수 있도록 설계 한다.

4.2.2.3 명령 실행 사이클 동기화

본 절에서는 통신을 매체로 한 각 명령 실행의 동기화에 관해 논의 하고자 한다. 명령 실행 사이클을 동기화하기 위해, 통신 프로토콜 수준에서 동기가 가능한 2가지 동기화 방법을 제안한다. 첫 번째 방법은 명령의 안전한 전송을 확인하기 위한 응답(Acknowledge)이고, 나머지 하나는 명령 실행의 종료를 확인하기 위한 보고(Report)이다. 실제로 명령 실행 사이클의 동기화는 보고(Report)에 의해 가능하게 되며, 새로운 명령이 수행되어 종료될 때까지 많은 명령과 응답이 송수신될 수 있다. 보고(Report)는 명령의 수행 과정에서 필요한 정보를 보고함을 의미하는 과정 보고와 명령의 수행이 정상적으로 종료되거나 비상 상황에 의해 종료되는 경우, 명령 실행 사이클이 종료되었음을 보고하는. 종료 보고로 나눌 수 있다. 종료 보고(Report)는 현재 실행 중인 명령의 실행 사이클은 종료된다. 다시 말해, 새로운 명

령을 다시 실행할 수 있는 상태가 된다. 본 논문에서는 무선 통신을 매체로 한 명령을 송・수신함에 있어서 각 명령 사이클 간의 동기화를 실현하기 위해서 다음의 Fig. 4.9와 같이 명령 실행 사이클을 설계 한다.

Fig. 4.9 Design of Commend practice cycle synchronization

4.2.3 통신구조 설계

본 절에서는 OHT를 최적으로 제어관리하기 위해 설계된 각 시스템간의 통신 구조에 대하여 논의하고자 한다. 이미 앞 절에서 논의 하였듯이 본 논문에서는 OHT를 최적으로 제어관리하기 위한 시스템을 물리적인 시스템과 논리적인 시스템으로 분류하였으며 이러한 분류를 또 다시 개체들로 나누어 정의하였다. 또한 각 개체들을 명령 개체와 구동 개체로 분류하여 전체 시스템을 설계하였다. 이러한 개체 들 간의 명령 처리 수단은 모두 통신을 기반으로 이루어진다. 따라서 신뢰성 있는 OHT 제어관리 시스템을 개발하기 위해서는 견실한 통신 구조가 설계가 요구되며 이와 같은 요구를 충족하기 위하여 설계한 통신 구조 항목은 OCS와 OHT 간의 통신 구조, OHT와 핸드-터미널(hand-terminal) 장치 간의 시스템의 통신 구조, OHT 프로그램 메니져(program manager) 시스템의 통신 구조와 같으며 구체적인 설계 내용은 본 논문의 4.2.3.1, 4.2.3.2, 4.2.3.3 절에서 다루도록 한다.

4.2.3.1 OCS 및 OHT 시스템

본 논문에서는 다음의 Fig. 4.10과 같이 OCS와 OHT의 통신 구조를 설계한다. OCS 서버(server)에서는 Foup 이송에 관한 정보를 OHT에게 지령하기 위하여 Normal Transport Command, Abnormal Transport Command, Request Command, Acknowledge Command의 명령을 송신한다. 여기서, Normal Transport Command란 정규 동작 명령을 의미하며, Abnormal Transport Command란 비정규 동작 명령을 의미한다. 그리고 Request Command란 OHT의 현재 상태를 요구하는 명령이며, Acknowledge Command란 각 명령에 대한 응답을 의미한다.

OHT의 OPC에 탑재된 OCS 클라이언트(client)에서는 OCS로부터 지령된 명령을 올바르게 수행하고 각 명령에 대한 응답 및 상호간에 통신상의 동기화를 구현하기 위하여 Event Report Command, Response Command, Acknowledge Command의 명령을 송신한다. 여기서, Event Report Command란 Normal/Abnormal Transport Command의 수행 상황을 보고하는 명령을 의미하며, Response Command란 OHT의 현재 상태를 보고하는 명령을 의미한다. 그리고 Acknowledge Command란 각 명령에 대한 응답을 의미한다. 본 논문에서 설계한 OCS 및 OHT 상호간의 통신 구조는 Semi-Standard 시나리오에 근거를 둔 명령과 응답을 송·수신 하도록 설계되었다.

Fig. 4.10 Design of Communication structure between OCS and OHT

4.2.3.2 핸드-터미널(hand-terminal) 시스템 및 OHT 시스템

다음의 Fig. 4.11과 같이 OHT와 의 핸드-터미널(hand-terminal) 시스템 간의 통신 구조를 설계 한다. OHT의 OPC에 탑재된 핸드-터미널(hand-terminal) 서버(server)에서는 핸드-터미널(hand-terminal) 시스템으로부터 지령된 각 모드(mode)별 명령에 따라 수동운전 및 티칭(teaching)작업을 올바르게 수행하기 위하여 Monitoring Command, Response Command, Acknowledge Command, Event Report Command 의 명령을 송신한다. 여기서, Monitoring Command란 현재 OHT의 하드웨어 및 소프트웨어의 상태를 보고하는 명령을 의미하며, Response Command, Acknowledge Command, Event Report Command은 본 논문의 4.2.3.1 절에서 이미 논의 하였다.

핸드-터미널(hand-terminal) System에 탑재된 핸드-터미널(hand-terminal) Client에서는 OHT를 수동으로 조작하고 각 작업장별 티칭(teaching) 작업을 올바르게 수행하도록 하기 위하여 Hand Mode Command, Jog Mode Command, Jog Operation Command, Normal Transport Command, Abnormal Transport Command, Request Command, Acknowledge Command의 명령을 송신한다. 여기서, Hand Mode Command란 Run/Teach 운전의 모드(mode) 변경에 관련된 명령을 의미하며, Jog Mode Command란, 티치 모드(teach mode)에서의 구동부 선택 명령을 의미한다. 그리고 Jog Operation Command,란 개별 구동부에 대한 동작 명령을 의미 한다. Normal Transport Command란 Abnormal Transport Command, Request Command, Acknowledge Command에 대한 내용은 본 논문의 4.2.3.1절에서 이미 논의하였다. 이와 같이 본 논문에서 설계한 OHT 및 핸드-터미널(hand terminal) 시스템 상호간의 통신구조는 OHT에 탑재된 OPC 시스템에서 사용하는 명령과 응답을 모두 포함하며 여기에 구동 정보 학습과 테스트를 위한 독자적인 명령과 응답을 추가하여 설계하였으며 본 설계 또한 Semi-Standard 시나리오에 근거를 둔 명령과 응답을 송・수신 하도록 설계되었다.

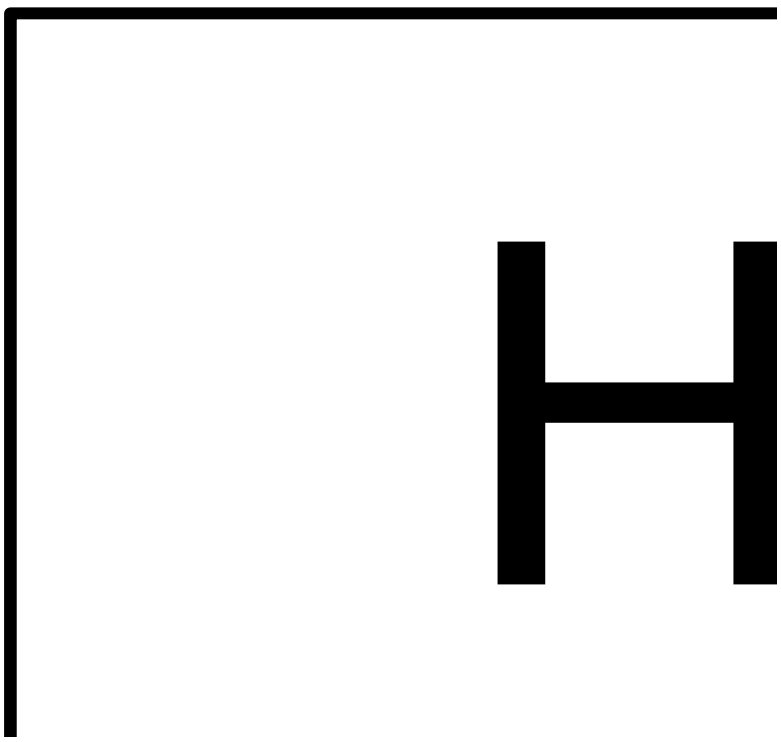


Fig. 4.11 Design of Structure between OHT and Hand-Terminal System

4.2.3.3 프로그램 메니저(program manager) 시스템 및 OHT 시스템

다음의 Fig. 4.12와 같이 OHT의 OPC에 탑재된 프로그램 메니저(program manager) 서버(server)와 프로그램 메니저(program manager) 시스템에 탑재된 프로그램 메니저(program manager) 클라이언트(client)간의 통신 구조를 설계한다. 프로그램 메니저(program manager) 시스템에 탑재된 프로그램 메니저(program manager) 클라이언트(client)와 OHT의 OPC에 탑재된 프로그램 메니저(program manager) 서버(server)가 무선 네트워크로 인터페이스(interface)되어 온라인(on-line)상태가 되면 프로그램 메니저(program manager) 클라이언트(client)에서 Program Manager Command, Acknowledge Command를 송신하여 OHT의 OPC에 탑재된 모든 소프트웨어를 원격으로 유지보수 할 수 있다. 여기서, Program Manager Command란 다른 논리 시스템의 구동 개체인 OCS Client와 핸드-터미널(hand-terminal) 서버(server)와 물리 시스템(OHT 시스템)을 관리하기 위한 명령을 의미하며, Acknowledge Command란 프로그램 메니저(program manager) 서버(server)로부터 수신된 각 명령에 대한 응답을 의미한다. 프로그램 메니저(program

manager) 서버(server)에서는 프로그램 메니져(program manager) 클라이언트(client)에서 지령한 Command에 따라서 Monitoring Command, Acknowledge Command를 송신하는데 여기서, Monitoring Command란 프로그램 Program Manager Command의 수행 상태를 보고하는 명령을 의미하며, Acknowledge Command란 프로그램 메니져(program manager) 클라이언트(client)로부터 수신된 각 명령에 대한 응답을 의미한다.

Fig. 4.12 Design of Structure between OHT and Program Manager System

4.2.4 프로토콜(Protocol) 설계

본 논문의 4.2.3절에서는 상위 그룹으로부터 지령되는 명령에 의해서 모든 물류 이송 작업을 수행하는 OHT를 최적으로 제어하기 위한 시스템을 구현하기 위하여 각 시스템별로 견실한 통신 구조를 설계하였다. 본 절에서는 각 시스템별로 설계된 통신 구조에 의거하여 실제로 시스템 간에 주고받는 통신 프로토콜(protocol) 설계에 관하여 논의 하고자 한다. 본 논문에서 설계된 프로토콜(protocol)은 반도체 관련 물류 장치가 갖추어야할 규정을 준수하기 위하여 관련 규격을 면밀히 조사·분석하여 설계한다.

본 논문에서 설계한 프로토콜(protocol)은 OHT에 탑재된 OPC 시스템, 핸드-터미

널(hand-terminal) 시스템, 프로그램 메니져(program manager) 시스템의 세 가지 프로토콜(protocol)로 구분된다. OPC 시스템은 프로토콜(protocol)은 실제 물류 이송 작업에 관련된 내용이 대부분이며 Semi-Standard 시나리오에 부합하는 프로토콜(protocol)로 구성된다. 핸드-터미널(hand-terminal) 시스템은 OPC 시스템의 프로토콜(protocol)을 모두 포함하며, 이에 추가하여 수동모드에서의 개별 구동과 티칭 데이터를 얻기 위한 구동 정보 학습, 테스트에 필요한 프로토콜(protocol)을 설계하였으며 프로그램 메니져(program manager) 시스템은 논리 시스템의 구동 개체인 OCS 클라이언트(client) 및 핸드-터미널(hand-terminal) 서버(server)와 물리 시스템(OHT 시스템)을 관리하기 위한 시스템 이므로 이와 관련된 프로토콜(protocol)을 설계한다. 본 논문에서 설계한 프로토콜(protocol)의 기본 형식을 다음의 Table. 4.6에서 정리 하였으며 본 형식은 모든 프로토콜(protocol)에 동일하게 적용된다. 하지만 본 프로토콜(protocol)은 명령의 종류에 따라 데이터(data)의 내용과 형식에 차이가 있을 수 있으며 헤더(header), 테일(tail), 데이터(data)는 여러 개의 필드로 구성된다. 헤더(header)와 테일(tail)은 모든 명령에 대해 필수적이며, 데이터(data)는 명령에 따라 존재하지 않는 경우도 있다.

Table. 4.6 Basic Form of Communication protocol

명령의 기본 형식		
Header	Data	Tail
특수 문자 정의		
시작 문자	'* '(명령의 시작을 인식한다.)	
종료 문자	'$' (명령의 종료를 인식한다.)	
필드 구분자	'-' (개별 필드를 인식한다.)	
명령의 계층 구조 (Header에 포함된다.)		
PCMD (Primary Command)	상위 명령을 표현하며, 필수적으로 요구된다. (Ex. _PCMD_MOVE_)	
ACMD (Auxiliary Command)	하위 명령을 표현하며, 상위 명령의 종류에 따라 존재하지 않는 경우도 있다. (Ex. _ACMD_MOVE_, _ACMD_MOVELOAD, _ACMD_MOVEUNLOAD_, _ACMD_MOVEBACK_)	
Header의 필드 구성 (모든 명령에서 동일하며, 필드사이에는 구분자가 있다.)		
명령 길이	PCMD부터 모든 필드와 구분자, 종료문자를 포함한 총 명령 길이	
PCMD	명령의 상위 종류 (Mandatory)	
ACMD	명령의 하위 종류 (Optional)	
CMDID	상위 시스템에서 보내온 명령의 인식 문자	
SEQ	명령의 순차적인 일련 번호 (명령과 응답의 쌍을 만들기 위한 필드)	
VHID	명령을 수행하는 OHT의 인식 문자	
Tail의 필드 구성 (모든 명령에서 동일하며, 필드사이에는 구분자가 있다.)		
Send Time	명령의 전송 시간	
Retry Count	명령의 재전송 횟수	

4.2.4.1 OCS 및 OHT 시스템

본 절에서는 OCS와 OHT에 탑재된 OPC의 개별 명령 구조에 따라 설계된 프로토콜에 대해서 논의하고자 한다. 본 프로토콜은 OCS에 탑재되며 명령 개체이면서 개별적인 외부 시스템에 속하는 OCS 서버(server)와 OHT에 OPC 형식으로 탑재되어 OHT를 직접 제어관리하는 구동개체인 OCS Client의 통신을 위하여 설계되었다. 본 프로토콜은 명령의 송신 및 수신 체계에 따른 명령 구조에 따라서 설계되었다. OCS에 탑재된 OCS 서버(server)에 의해 송신되어 OHT에 탑재된 OCS Client로 수신되는 명령은, 전체 시스템이 정상적인 경우 이송 작업을 위한 동작 명령인 Normal Transport Command와 OPC 시스템 비정상적인 경우의 이송 작업을 위한 명령인 Abnormal Transport Command, OCS 서버(server)에서 오류회복이나 정보 재 수신을 위해 OHT의 현재 상태를 요구한기 위한 명령인 Request OHT Information Command, 다른 명령 군의 수신 시에 동기화를 위한 응답으로 사용되는 OCS Acknowledge Command로 구성된다. 또한, OHT에 탑재된 OCS Client에 의해 송신되어 OHT에 탑재된 OCS 서버(server)로 수신되는 명령은, Normal/Abnormal Transport 명령의 수행 상태를 단계별로 OCS에 보고하기 위한 Event Report Command, OCS 서버(server)에서 오류회복이나 정보 재 수신을 위해 OHT의 현재 상태를 요구하기 위한 Response OHT Information Command, 다른 명령군의 수신 시에 동기화를 위한 응답으로 사용되는 OPC Acknowledge Command로 구성된다.

본 논문에서 설계된 프로토콜은 SEMI-Standard 시나리오에 입각한 명령을 기반으로 설계되었으며 각 명령에 따라 설계된 프로토콜은 다음의 Table. 4.7~4.13에서 정리 하였다.

Table. 4.7 Protocol design of Normal Transport Command

- OCS Server->OCS Client : 명령 개체->구동 개체 : 외부 시스템->OHT 시스템. - 정상적인 경우의 작업을 위한 동작 명령에 속한다. - Normal Transport 명령이 수행 중인 경우, 다른 Normal Transport 명령을 수신할 수 없다. - Abnormal Transport 명령이 수행 중인 경우, Normal Transport 명령을 수신할 수 없다.			
Header	PCMD	10	
	ACMD	02	Move(단순 이동 명령)
		03	Move and Load(이동 후 Load)
		04	Move and Unload(이동 후 Unload)
		05	Move Back(후진)
Data	Destination	목적지	
	Carrier ID	상위 시스템에서 수신한 Foup의 일련 문자	
	Path	단일 Path	[위치][분기정보] 분기정보: N(없음)|L(좌분기)|R(우분기)
		구분자	':'
		내용	현재 위치에서 목적지까지의 경로를 구분자와 단일 Path의 조합으로 구성한다.> (Ex. 2N:3N:4N:5L:6N => 현재 위치 2부터 목적지 6까지의 Path 정보)

Table. 4.8 Protocol design of Abnormal Transport Command

<table>
<tr><td colspan="4">- OCS Server->OCS Client : 명령 개체->구동 개체 : 외부 시스템->OHT 시스템
- 비정상적인 경우의 작업을 위한 명령
- Abnormal Transport 명령이 수행중인 경우, 다른 Abnormal Transport 명령을 수신할 수 없다.
- Abort 명령의 수행 후에는, 반드시 Move/MoveUnLoad 명령이 따라야 한다.
- Pause 명령의 수행 후에는, 반드시 Resume 명령이 따라야 한다.
- Data Part가 존재하지 않는 명령이다.</td></tr>
<tr><td rowspan="5">Header</td><td>PCMD</td><td colspan="2">20</td></tr>
<tr><td rowspan="4">ACMD</td><td>21</td><td>Cancel : Move,/MoveLoad Normal Transport 명령이 수행 중이고 승강 이전의 주행 상태에서 Normal Transport 명령을 완전히 취소할 수 있다.</td></tr>
<tr><td>23</td><td>Abort : Move/MoveUnLoad Normal Transport 명령이 수행 중이고 승강 이전의 주행 상태에서 명령을 취소하고, 목적지를 바꾼 다른 Normal Transport 명령으로 대체 할 수 있다.>
다음 Normal Transport 명령은 Move./MoveUnload 명령이어야 한다.</td></tr>
<tr><td>24</td><td>Alarm : OHT->OCS로 Alarm을 Report한다.
이 후로 상세 구현을 미룬다.
Abnormal Transport가 아닌 다른 명령 군으로 바꾸어야 한다.</td></tr>
<tr><td>25</td><td>Pause : Move/MoveLoad/MoveUnLoad Normal Transport 명령이 수행 중이고 승강 이전의 주행 상태에서, 현재 상태를 유지하면서 작업을 일시 중지할 수 있다. 다음 명령은 Resume 명령이어야 한다.</td></tr>
<tr><td>26</td><td>Resume : Pause 상태에서 유지된 현재 상태로 작업을 재계한다.</td></tr>
</table>

Table. 4.9 Protocol design of Request OHT Information Command

- OCS Server->OCS Client : 명령 개체->구동 개체 : 외부 시스템->OHT 시스템 - OCS Server에서 오류회복이나 정보 재 수신을 위해 OHT의 현재 상태를 요구한다. - 한번에 하나의 명령만 처리할 수 있다. - 다른 명령 군과 관계없이 수행할 수 있다.			
Header	PCMD	30	
	ACMD	NC (의미가 없다)	
Data	Receive Flag	Y	응답수신: OHT는 Response OHT Information을 전송함.
		N	응답수신 거부 : OHT는 Response OHT Information을 전송하지 않음.

Table. 4.10 Protocol design of OCS Acknowledge Command

- OCS Server->OCS Client : 명령 개체->구동 개체 : 외부 시스템->OHT 시스템 - 다른 명령 군의 수신 시에 동기화를 위한 응답으로 사용된다. - Data Part가 존재하지 않는 명령이다.			
Header	PCMD	40	
	ACMD	01	ACK1 : 정상적인 수신, 정상적인 명령 수행
		02	ACK2 : 동일한 명령 (정상적인 수신, 정상적인 명령 수행)
		03	NAK1 : 현재 정상적으로 명령을 수행할 수 없음.
		04	NAK2 : 명령을 완전히 거절하며, 다시 보내어서는 안됨.

Table. 4.11 Protocol design of Event Report Command

- OCS Client->OCS Server : 구동 개체->명령 개체 : OHT 시스템->외부 시스템 - Normal/Abnormal Transport 명령의 수행 상태를 단계별로 OCS로 보고한다.			
Header	PCMD	50	
	ACMD	51	Position : 주행 상태에서 위치가 바뀔 때, 위치를 보고함.
		52	Normal : Normal Transport 명령에서, 단계별 상태를 보고함.
		53	Installed : OHT이 Install될 때, 보고함.
		54	Removed : OHT이 Remove될 때, 보고함.
		55	Abnormal : Abnormal Transport 명령에서, 단계별 상태를 보고함.
Data	Carrier Id	현재 싣고 있는 Foup의 일련 문자	
	Event	Position : 의미 없음	
		Normal : IBSEM Scenario에 따른 작업 상황 정보 AcquireStarted: 05 , AcquireCompleted: 07 Departed: 08. DepositStarted: 10, DepositCompleted: 12 MoveArrived: 15, MoveLoadArrived: 16, MoveUnloadArrived: 17 MoveBackArrived: 18	
		Installed : 의미 없음	
		Removed : 의미 없음	
		Abnormal : IBSEM Scenario에 따른 작업 상황 정보 CancelCompleted: 01, AbortCompleted: 02, AlarmCompleted: 03, PauseCompleted:04, ResumeCompleted: 05	
	Position	현재 OHT의 위치정보	

Table. 4.12 Protocol Design of Response OHT Information Command

- OCS Client ->OCS Server : 구동 개체->명령 개체 : OHT 시스템->외부 시스템 - OCS 서버(server)에서 오류회복이나 정보 재수집을 위해 OHT의 현재 상태를 요구한다. - 한번에 하나의 명령만 처리할 수 있다. - 다른 명령 군과 관계없이 수행할 수 있다.		
Header	PCMD	60
	ACMD	NC (의미가 없다)
Data	Status	Normal : IBSEM Scenario에 따른 작업 상황 정보 AcquireStarted: 05 , AcquireCompleted: 07 Departed: 08. DepositStarted: 10, OHTDepositCompleted: 12 MoveArrived: 15, MoveLoadArrived: 16, MoveUnLoadArrived: 17, MoveBackArrived: 18
		Abnormal : IBSEM Scenario에 따른 작업 상황 정보 CancelCompleted: 01, AbortCompleted: 02, AlarmCompleted: 03, PauseCompleted:04, ResumeCompleted: 05
	Destination	현재 수행중인 명령의 목적지 (Normal Transport인 경우)
	Carrier Id	현재 싣고 있는 Foup의 일련 문자 (Normal Transport인 경우)
	Path	현재 수행중인 명령의 Path 정보 (Normal Transport인 경우)

Table. 4.13 Protocol design of OPC Acknowledge Command

- OCS Client ->OCS Server : 구동 개체->명령 개체 : OHT 시스템->외부 시스템 - 다른 명령군의 수신 시에 동기화를 위한 응답으로 사용된다. - Data Part가 존재하지 않는 명령이다.			
Header	PCMD	70	
	ACMD	01	ACK1 : 정상적인 수신, 정상적인 명령 수행
		02	ACK2 : 동일한 명령 (정상적인 수신, 정상적인 명령 수행)
		03	NAK1 : 현재 정상적으로 명령을 수행할 수 없음.
		04	NAK2 : 명령을 완전히 거절하며, 다시 보내어서는 안됨.

4.2.4.2 핸드-터미널(hand-terminal) 시스템 및 OHT 시스템

본 절에서는 핸드-터미널(hand-terminal) 시스템과 OHT 시스템의 개별 명령에 따라 설계된 프로토콜에 대해서 논의 하고자 한다. 본 프로토콜은 핸드-터미널(hand-terminal) 시스템에 탑재되며 명령 개체이면서 개별적인 외부 시스템에 속하는 핸드-터미널(hand-terminal) 클라이언트(client)와 OHT에 실제로 OPC 형식으로 탑재되어 핸드-터미널(hand-terminal) 클라이언트(client)에서 지령된 명령대로 OHT를 직접 제어관리 하는 구동개체인 핸드-터미널(hand-terminal) 서버(server)의 통신을 위하여 설계되었다. 본 프로토콜은 명령의 송신 및 수신 체계에 따른 명령 구조에 따라서 설계되었다. 핸드-터미널(hand-terminal) 클라이언트(client)에 의해 송신되어 OHT에 탑재된 핸드-터미널(hand-terminal) 서버(server)로 수신되는 명령은, 수동조작 모드 및 티칭 모드 상호간의 모드 변경을 위한 명령인 Hand Mode Conversion Command와 각 구동부의 개별 동작을 위해 구동부를 선택하기 위한 명령인 Jog Operation Mode Conversion Command, 선택된 구동부에 대해 실제로 수동동작을 수행하기 위한 명령인 Jog Operation Command, 선택된 구동부의 학습 정보(티칭정보)를 저장하기 위한 명령인 Jog Save Command, 선택된 구동부의 구동 설정 값을 바꾸기 위한 명령인 Setting MC Parameter Command로 구성된다.

또한, OHT에 탑재된 핸드-터미널(hand-terminal) 서버(server)에 의해 송신되어 핸드-터미널(hand-terminal) 시스템에 탑재된 핸드-터미널(hand-terminal) 클라이언트(client)로 수신되는 명령은, 핸드-터미널(hand-terminal) 명령의 수행 상태를 보고하기 위한 명령인 Event Report Command와 OHT의 구동부 상태, 디지털 입출력 포트 상태, 바코트 리더(barcode reader) 상태, 주행 상태 등을 실시간으로 보고하기 위한 핸드-터미널(hand-terminal) 시스템 Monitoring Command로 구성되며 서버(server)와 Client 상호간에 다른 명령군의 수신 시에 동기화를 위한 응답으로 사용되는 명령인 Acknowledge Command로 구성된다.

위에서 언급한 각 명령에 따라 설계된 프로토콜은 다음의 Table. 4.14~4.21에서 정리 하였으며 본 시스템에서 실제로 쓰이는 프로토콜이지만 본 논문의 4.2.4.1절에서 이미 논의한 내용에 관해서는 본 절에서 서술하지 않는다.

Table. 4.14 Protocol Design of Hand Mode Conversion Command

- Hand Terminal Client -> Hand Terminal Server : 명령 개체->구동 개체 : 외부 시스템 ->OHT 시스템 - Data Part가 존재하지 않는 명령이다.			
Header	PCMD	100	
	ACMD	01	Run Mode : OPC 시스템과 동일한 명령을 수행하기 위한 모드
		02	Teach Mode : Hand-Terminal의 독자적인 명령을 수행하기 위한 모드

Table. 4.15 Protocol Design of Jog Operation Mode Conversion Command

- Hand Terminal Client -> Hand Terminal Server : 명령 개체->구동 개체 : 외부 시스템 ->OHT 시스템 - 각 구동부의 개별 동작을 위해 구동부를 선택하는 명령이다. - Data Part가 존재하지 않는 명령이다.			
Header	PCMD	110	
	ACMD	01	주행부를 선택한다.
		02	분기부를 선택한다.
		03	낙하 방지부를 선택한다.
		04	횡행부를 선택한다.
		05	선회부를 선택한다.
		06	승강부를 선택한다.
		07	척부를 선택한다.

Table. 4.16 Protocol Design of Jog Operation Command

- Hand Terminal Client -> Hand Terminal Server : 명령 개체->구동 개체 : 외부 시스템 ->OHT 시스템 - 선택된 구동부에 대해 다양한 동작을 명령한다. - Data Part가 존재하지 않는 명령이다.			
Header	PCMD	120	
	ACMD	01	CW (Open) 정 방향 이동: 주행부, 분기부, 횡행부, 선회부, 승강부 Open : 낙하 방지부, 척부
		02	CCW (Close) 역 방향 이동: 주행부, 분기부, 횡행부, 선회부, 승강부 Close : 낙하 방지부, 척부
		03	Home (TR Step) 원점 이동 : 횡행부, 선회부, 승강부 단위 이동 : 주행부
		04	Stop

Table. 4.17 Protocol Design of Jog Save Command

- Hand Terminal Client -> Hand Terminal Server : 명령 개체->구동 개체 : 외부 시스템 ->OHT 시스템 - 선택된 구동부의 학습 정보를 저장한다.		
Header	PCMD	130
	ACMD	NC (의미 없음)
Data	Station	현재 위치를 이름으로 한 파일에 선택된 구동부의 학습 정보를 저장한다. 현재 위치가 정해지지 않은 경우에 Station을 파일이름으로 사용한다. 학습 정보가 필요한 구동부 : 주행부, 횡행부, 선회부, 승강부
	Port	NC (의미 없음)

Table. 4.18 Protocol Design of Setting MC Parameter Command

- Hand Terminal Client -> Hand Terminal Server : 명령 개체->구동 개체 : 외부 시스템 ->OHT 시스템 - 선택된 구동부의 구동 설정 값을 바꾼다.			
Header	PCMD	140	
	ACMD	01	Set Parameter : 구동부의 전체 설정 값을 바꾼다.
		02	Change Drive Speed : 구동부의 구동 속도를 바꾼다.
Data	TS	Curve Mode (T-Curve, S-Curve)	
	SV	Initial Speed	
	DV	Drive Speed	
	MDV	Maximum Drive Speed	
	AC	Acceleration Factor	
	AK	Acceleration Jerk Factor	

Table. 4.19 Protocol Design of Event Report Command

- Hand Terminal Server -> Hand Terminal Client : 구동 개체->명령 개체 : OHT 시스템->외부 시스템 - Hand-Terminal 명령의 수행 상태를 보고한다.			
Header	PCMD	50	
	ACMD	52	Normal : Hand-terminal Command의 수행 상태 동기화를 위해 Semi-Standard Command를 차용함.
Data	Carrier Id	NC (의미 없음)	
	Event	Normal : Hand-Terminal Command의 수행 상태 OperationCompleted: 105, OperationError: 106	
	Position	NC (의미 없음)	

Table. 4.20 Protocol Design of Monitoring Command

- Hand Terminal Server -> Hand Terminal Client : 구동 개체->명령 개체 : OHT 시스템->외부 시스템 - OHT의 구동부 상태, Digital Input 및 Output Port 상태, Barcode Reader 상태, 주행 상태 등을 실시간으로 보고한다.			
Header	PCMD	160	MC Status : 각 구동부의 상태를 보고한다.
		170	MC Position : 각 구동부의 위치를 보고한다.
		180	MC Teaching Data : 각 구동부의 Teaching Data를 보고한다.
		190	DO Status : 각 Digital Output Channel의 상태를 보고한다.
		200	DI Status : 각 Digital Input Channel의 상태를 보고한다.
		210	IR Status : NC (의미 없음)
		220	BR Status : Barcode와 주행 상태를 보고한다.
	ACMD	NC (의미 없음)	
Data	Status1	저장된 정보와 사용하는 필드의 개수는 보고의 종류에 따라 다르다.	
	Status2		
	Status3		
	Status4		
	Status5		
	Status6		
	Status7		
	Status8		

Table. 4.21 Protocol Design of Acknowledge Command

- Hand Terminal 서버(server)<->Hand Terminal Client - 명령 개체<->구동 개체 : 외부 시스템<->OHT 시스템 - 다른 명령군의 수신 시에 동기화를 위한 응답으로 사용된다. - Data Part가 존재하지 않는 명령이다.			
Header	PCMD	230	
	ACMD	01	ACK1 : 정상적인 수신, 정상적인 명령 수행
		02	ACK2 : 동일한 명령 (정상적인 수신, 정상적인 명령 수행)
		03	NAK1 : 현재 정상적으로 명령을 수행할 수 없음.
		04	NAK2 : 명령을 완전히 거절하며, 다시 보내어서는 안됨.

4.2.4.3 프로그램 메니저(program manager) 시스템 및 OHT 시스템

본 절에서는 프로그램 메니저(program manager) 시스템에서 논리 시스템에 속하는 구동 개체인 OCS Client와 핸드-터미널(hand-terminal) 서버(server) 및 물리 시스템에 속하는 OHT 시스템을 관리하기 위해 상호간에 송·수신 하여야 하는 개별 명령에 따라 설계된 프로토콜에 관하여 논의하고자 한다. 본 프로토콜은 프로그램 메니저(program manager) 시스템 장치에 탑재되며 명령 개체이면서 개별적인 외부 시스템에 속하는 프로그램 메니저(program manager) 클라이언트(client)와 OHT에 실제로 OPC 형식으로 탑재되어 프로그램 메니저(program manager) 클라이언트(client)에서 지령된 명령대로 OCS Client와 핸드-터미널(hand-terminal) 서버(server) 및 물리 시스템에 속하는 OHT 시스템을 관리하기 위한 통신을 수행하기 위하여 설계되었다. 본 프로토콜은 명령의 송신 및 수신 체계에 따른 명령 구조에 따라서 설계되었다. 프로그램 메니저(program manager) 클라이언트(client)에 의해 송신되어 OHT에 탑재된 프로그램 메니저(program manager) 서버(server)로 수신되는 명령은, 논리 시스템의 구동 개체인 OCS Client 및 핸드-터미널(hand-terminal) 서버(server)를 원격으로 실행·종료 시키고 물리 시스템인 OHT 시스템을 원격으로 Shutdown/Reboot 시키기 위한 명령인 Operation Command로 구성된다.

또한, OHT에 탑재된 프로그램 메니저(program manager) 서버(server)에 의해 송신되어 프로그램 메니저(program manager) 시스템에 탑재된 프로그램 메니저

(program manager) 클라이언트(client)로 수신되는 명령은, 프로그램 메니져(program manager)의 수행 상태를 보고하고, 명령 수행을 동기화하기 위하여 사용되는 Monitoring Command로 구성되며 프로그램 메니져(program manager) 서버(server)와 프로그램 메니져(program manager) 클라이언트(client)가 다른 명령 군의 명령을 수신 시에 통신상의 동기화를 위하여 사용하는 Acknowledge Command로 구성된다.

위에서 언급한 각 명령에 따라 설계된 프로토콜은 다음의 Table. 4.22～4.24에서 정리 하였다.

Table. 4.22 Protocol Design of Operation Command

<table>
<tr><td colspan="4">- Program Manager Client -> Program Manager server
- 명령 개체->구동개체 : 외부 시스템->OHT 시스템
- 다른 논리 시스템의 구동 개체(OCS Client/Hand-Terminal Server)를 강제로 실행과 종료할 수 있다.
- 물리 시스템(OHT 시스템)을 강제로 Shutdown/Reboot할 수 있다.
- Data Part가 존재하지 않는 명령이다.</td></tr>
<tr><td rowspan="5">Header</td><td>PCMD</td><td colspan="2">150.</td></tr>
<tr><td rowspan="4">ACMD</td><td>01</td><td>Shutdown : 물리 시스템(OHT 시스템)을 Shutdown 한다.</td></tr>
<tr><td>02</td><td>Reboot : 물리 시스템(OHT 시스템)을 Reboot 한다.</td></tr>
<tr><td>03</td><td>Execute : 다른 논리 시스템의 구동 개체(OCS Client/Hand-terminal Server)를 실행한다.</td></tr>
<tr><td>04</td><td>Exit : 다른 논리 시스템의 구동 개체(OCS Client/Hand-Terminal Server)를 종료한다..</td></tr>
</table>

Table. 4.23 Protocol Design of Monitoring Command

- Program Manager Sever -> Program Manager Client - 구동개체->명령 개체 : OHT 시스템->외부 시스템 - Program Manager의 수행 상태를 보고하고, 명령을 수행을 동기화하기 위해 사용한다.			
Header	PCMD	210	IR Status : 초기 정보전송 동기화에 사용된다.
		220	BR Status : 명령 수행에 따른 상태 변화를 보고한다..
	ACMD	NC (의미 없음)	
Data	Status1	저장된 정보와 사용하는 필드의 개수는 보고의 종류에 따라 다르다.	
	Status2		
	Status3		
	Status4		
	Status5		
	Status6		
	Status7		
	Status8		

Table. 4.24 Protocol Design of Acknowledge Command

- Program Manager server <-> Program Manager Client - 명령 개체<->구동개체 : 외부 시스템<->OHT 시스템 - 다른 명령 군의 수신 시에 동기화를 위한 응답으로 사용된다. - Data Part가 존재하지 않는 명령이다.			
Header	PCMD	230	
	ACMD	01	ACK1 : 정상적인 수신, 정상적인 명령 수행
		02	ACK2 : 동일한 명령 (정상적인 수신, 정상적인 명령 수행)
		03	NAK1 : 현재 정상적으로 명령을 수행할 수 없음.
		04	NAK2 : 명령을 완전히 거절하며, 다시 보내어서는 안됨.

4.2.5 통신 구현

본 절에서는 본 논문에서 설계한 각 개체간의 통신 구조 및 통신 프로토콜의 구현에 대하여 논의하고자 한다. Fig. 4.13은 소프트웨어 상에서 통신을 구현하기 위한 무선 소켓통신 기본 클래스를 도시화한 것이다.

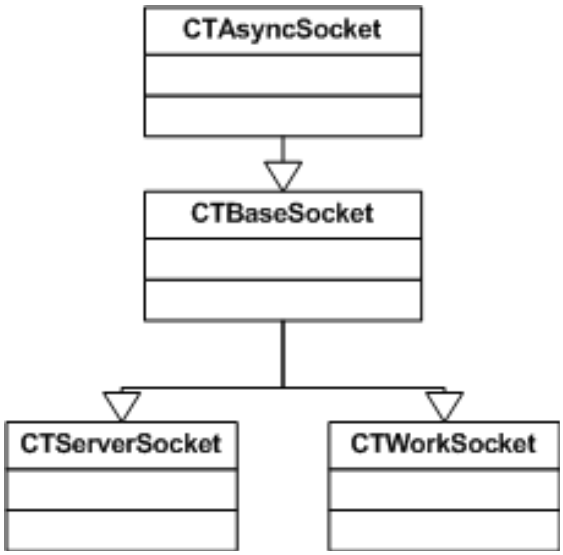


Fig. 4.13 Basic class of Communication

여기서, CTAsyncSocket는 비동기 방식의 무선 통신 코어 기능을 하는 클래스이며 CTBaseSocket는 소켓 상위 클래스이다. CT Server Socket은 연결 요청을 위한 모니터 소켓이며 CTWorkSocket은 작업 명령을 송수신하기 위한 작업 소켓이다. 이와 같이 구성된 통신 기본 클래스는 다음의 Fig. 4.14 4.15, 4.16, 4.17과 같이 실제로 프로그램 상에서 구현하였다. CTAsyncSocket은 일반 소켓 코어이므로 생략하고 나머지 클래스에 대한 소스는 대해서만 표기한다. 상세 코드는 내용이 방대하여 생략하고 관련된 부분의 정의만 포함한다.

```
////////////////////////////////////////////////////////////////////////////
// CTBaseSocket Class ////////////////////////////////////////////////
class CTBaseSocket : public CTAsyncSocket
{
public:
        CTBaseSocket();
        virtual ~CTBaseSocket();
protected: // 데이터 멤버
        BOOL                    m_bConnect;
        _SOCKETNOTIFYPROC_      m_pNotifyProc;
        CTSocketParam           m_tParam;
public: // 연결 설정/종료/모니터 함수
        void SetNotifyProc(LPVOID pvdObject, _SOCKETNOTIFYPROC_
                           pNotifyProc);
        BOOL IsConnect();
        void SetConnectionFlag(BOOL bConnect);
        void SocketSuspendClose();
        BOOL SocketClose(BOOL bWait=FALSE);
protected:
        virtual void OnClose(int iErrCode);
        virtual void OnCriticalError();
};
```

Fig. 4.14 Source Code of CTBaseSocket Class

```
////////////////////////////////////////////////////////////////////////////
// CT서버(server)Socket Class ////////////////////////////////////////////
class CT서버(server)Socket : public CTBaseSocket
{
public:
        CT서버(server)Socket();
        virtual ~CT서버(server)Socket();
public: // Sever 함수
        BOOL InitCreate(UINT uiPortNum);
        BOOL InitListen();
        BOOL InitParam(UINT uiSocketKind, UINT uiSocketID);
protected:
        virtual void OnAccept(int iErrCode);
};
```

Fig. 4.15 Source Code of CT Server Cocket Class

```
//////////////////////////////////////////////////////////////////////////
// CTWorkSocket Class ////////////////////////////////////////////////
class CTWorkSocket : public CTBaseSocket
{
public:
        CTWorkSocket();
        virtual ~CTWorkSocket();
protected: // 데이터 멤버
        CString m_strPeerIP;
        BOOL   m_bSend1;
        BOOL   m_bSend2;
public: // 연결 설정 함수
        BOOL    InitCreate();
        BOOL InitConnect(CString str서버(server)Addr,
                                UINT uiPortNum, DWORD dwTimeout);
        BOOL   InitParam(int iBufSizeIn, int iBufSizeOut,
                               UINT uiSocketKind, UINT uiSocketID);
        CString GetPeerIP();
public: // 데이터 송수신 함수
        BOOL              ReadData(CTWLProtocol   *pWLReadData,   DWORD
dwByteLen);
        BOOL     SendData1(CTWLProtocol *pWLSendData);
        BOOL     SendData2(CTWLProtocol *pWLSendData);
public: // 송수신 플래그 함수
        void    SetSend1(BOOL bSend){ m_bSend1 = bSend; }
        void    SetSend2(BOOL bSend){ m_bSend2 = bSend; }
        BOOL    CanSend1(){ return m_bSend1;  }
        BOOL    CanSend2(){ return m_bSend2;  }
protected: // 모니터 함수
        virtual void OnRead(int iErrCode, DWORD dwByteLen);
        virtual void OnSend(int iErrCode);
        virtual void OnReadComplete(int iErrCode, DWORD dwByteLen);
        virtual void OnSend1Complete(int iErrCode, DWORD dwByteLen);
        virtual void OnSend2Complete(int iErrCode, DWORD dwByteLen);
};
```

Fig. 4.16 Source Code of CTWorkSocket Class

```
//////////////////////////////////////////////////////////////////////////
// CTWorkSocket Class ////////////////////////////////////////////////
class CTWorkSocket : public CTBaseSocket
{
public:
        CTWorkSocket();
        virtual ~CTWorkSocket();
protected: // 데이터 멤버
        CString m_strPeerIP;
        BOOL   m_bSend1;
        BOOL   m_bSend2;
public: // 연결 설정 함수
        BOOL    InitCreate();
        BOOL InitConnect(CString str서버(server)Addr,
                                UINT uiPortNum, DWORD dwTimeout);
        BOOL   InitParam(int iBufSizeIn, int iBufSizeOut,
                               UINT uiSocketKind, UINT uiSocketID);
        CString GetPeerIP();
public: // 데이터 송수신 함수
        BOOL              ReadData(CTWLProtocol   *pWLReadData,   DWORD
dwByteLen);
        BOOL     SendData1(CTWLProtocol *pWLSendData);
        BOOL     SendData2(CTWLProtocol *pWLSendData);
public: // 송수신 플래그 함수
        void    SetSend1(BOOL bSend){ m_bSend1 = bSend; }
        void    SetSend2(BOOL bSend){ m_bSend2 = bSend; }
        BOOL    CanSend1(){ return m_bSend1;  }
        BOOL    CanSend2(){ return m_bSend2;  }
protected: // 모니터 함수
        virtual void OnRead(int iErrCode, DWORD dwByteLen);
        virtual void OnSend(int iErrCode);
        virtual void OnReadComplete(int iErrCode, DWORD dwByteLen);
        virtual void OnSend1Complete(int iErrCode, DWORD dwByteLen);
        virtual void OnSend2Complete(int iErrCode, DWORD dwByteLen);
};
```

Fig. 4.17 Source Code of CTWorkSocket Class

Fig. 4.18은 소프트웨어 상에서 통신을 구현하기 위한 무선 소켓통신 작업 클래스를 도시화한 것이다. 여기서, Global member인 m_tWLOCSClientDevice는 통신 작업을 위한 소켓 객체이며 CTOPCTestView 클래스는 OCS 서버(server)에서 명령과 응답을 수신한다. CTOCSClientThread 클래스는 OCS 서버(server)로 명령을 전송하는 기능을 담당하며 작업 수행의 대기 상태를 막기 위해 Thread로 분리한다. CTOCSAckThread는 OCS 서버(server)에 명령에 대한 응답을 전송하는 기능을 담당하며 통신 모니터의 대기 상태를 막기 위해 Thread로 분리한다. 이와 같이 구성된 통신 작업 클래스는 다음의 Fig. 4.19, 4.20, 4.21, 4.22과 같이 실제로 프로그램상에서 구현 하였다. 상세 코드는 내용이 방대하여 생략하고 관련된 부분의 정의만 포함한다.

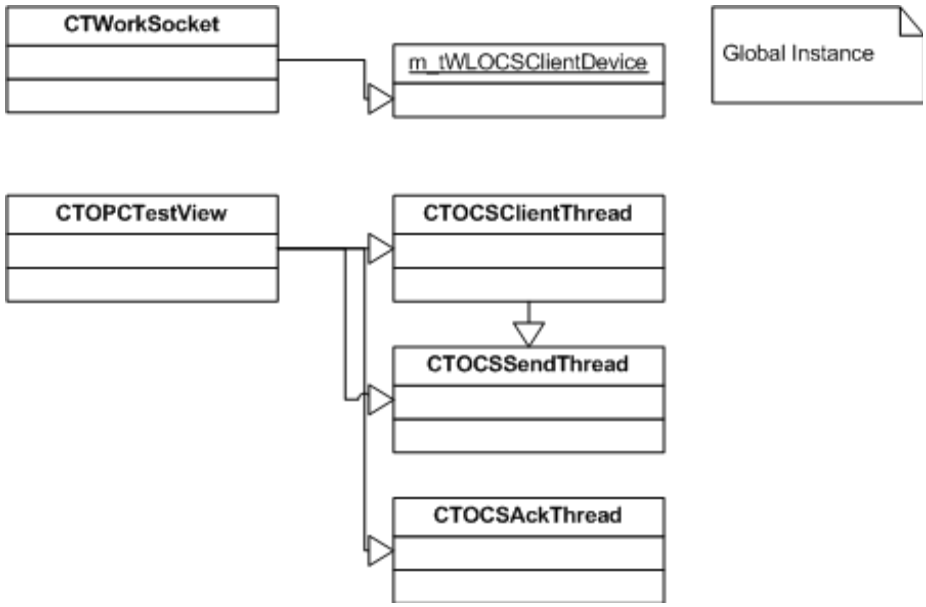


Fig. 4.18 Class of Communication working

```
////////////////////////////////////////////////////////////////////////////
// CTOPCTestView Class /////////////////////////////////////////////////////
class CTOPCTestView : public CFormView
{
protected:
        CTOPCTestView();
        // ~ 생략
public:
        // 통신 소켓 객체
        CTWorkSocket          *m_pWLOCSClientDevice;
        // ~ 생략
public: // 통신 버퍼 객체
        CTWLCmdSeq *m_pWLOCSSendCmdSeq;
        CTWLCmdSeq   *m_pWLOCSReadCmdSeq;
        CTWLQueue     *m_pWLOCSQueue;
        CTWLQueue     *m_pWLOCSAckQueue;
        CTWLProtocol          *m_pWLOCSSendData;
        CTWLProtocol          *m_pWLOCSAckData;
        CTWLProtocol          *m_pWLOCSReadData;
        // ~ 생략
public: // 명령 수신 함수
        void ReadOCSCommand(DWORD dwByteLen);
        BYTE PendOCSCommand();
public: // 소켓 통신 기본 함수
        BOOL OnWLNotify(CTSocketParam *pParam);
        DEFINE_SOCKET_NOTIFY(CTOPCTestView, OnWLNotify)
        void OnReadComplete(CTSocketParam *pParam);
        void OnSend1Complete(CTSocketParam *pParam);
        void OnSend2Complete(CTSocketParam *pParam);
        void OnCriticalError(CTSocketParam *pParam);
        void OnAccept(CTSocketParam *pParam);
        void OnRead(CTSocketParam *pParam);
        void OnSend(CTSocketParam *pParam);
        void OnClose(CTSocketParam *pParam);
};
```

Fig. 4.19 Source code of CTOPCTestView Class

```
////////////////////////////////////////////////////////////////////////////
// CTOCSClientThread Class ////////////////////////////////////////
class CTOCSClientThread : public CTBaseThread
{
public:
        CTOCSClientThread();
        virtual ~CTOCSClientThread();
public: // 통신 소켓 객체
        CTWorkSocket    *m_pWLOCSClientDevice;
        // ~생략
public: // 통신 버퍼 객체
        CTWLCmdSeq          *m_pWLOCSSendCmdSeq;
        CTWLCmdSeq          *m_pWLOCSReadCmdSeq;
        CTWLQueue           *m_pWLOCSQueue;
        CTWLQueue           *m_pWLOCSAckQueue;
        CTWLProtocol        *m_pWLOCSSendData;
        CTWLProtocol        *m_pWLOCSAckData;

public: // 작업 로직 데이터
        CTPendCommandStatus         *m_pPendCommandStatus;
        CTUpLoadCommandStatus       *m_pUpLoadCommandStatus;
        CTOpCmdExeStage             *m_pOpCmdExeStage;
        CTMCTeachDataItem           *m_pMCTeachDataItem;

protected: // 통신/작업 관련 모듈
        CTOCSSendThread     m_tOCSSendThread;
        CTOCSAckThread      m_tOCSAckThread;
        CTDoOpMode          m_tDoOpMode;
        CTDoAbnormalMode    m_tDoAbnormalMode;
        // ~ 생략
protected: // OCS 서버(server)와 연결 설정 및 작업 명령 수행 함수
        BOOL CheckThreadEnd();
        BOOL TryConnection(BOOL b서버(server));
        BOOL RunInstall();
        //~생략
};
```

Fig. 4.20 Source Code of CTOCSClientThread Class

```
////////////////////////////////////////////////////////////////////////////
// CTOCSSendThread Class ///////////////////////////////////////////
class CTOCSSendThread : public CTBaseThread
{
public:
        CTOCSSendThread();
        virtual ~CTOCSSendThread();
protected: // 소켓 통신 객체
        CTWorkSocket          *m_pWLOCSClientDevice;

protected: // 통신 버퍼 객체
        CTWLCmdSeq            *m_pWLOCSSendCmdSeq;
        CTWLProtocol          *m_pWLOCSSendData;
protected: // 통신 로직 객체
        CTPendCommandStatus         *m_pPendCommandStatus;
        CTUpLoadCommandStatus       *m_pUpLoadCommandStatus;
        CTOpCmdExeStage             *m_pOpCmdExeStage;
        // ~ 생략
protected: // 명령 생성/송신 함수
        BOOL CheckThreadEnd();
        BOOL ApplyUpLoadReportCommand  ();
        BOOL ApplyUpLoadResponseCommand();
        BOOL SendIBSEMCommand();
        BOOL TryCommandSend  (BYTE byCmdKind);
        BYTE RetryCommandSend(BYTE byCmdKind);
        BOOL MakeReportCommand  (DWORD dwRetryCnt);
        BOOL MakeResponseCommand(DWORD dwRetryCnt);
};
```

Fig. 4.21 Source Code of CTOCSSendThread Class

```
////////////////////////////////////////////////////////////////////////////
// CTOCSAckThread Class ////////////////////////////////////////////
class CTOCSAckThread : public CTBaseThread
{
public:
        CTOCSAckThread();
        virtual ~CTOCSAckThread();
protected: // 소켓 통신 객체
        CTWorkSocket *m_pWLOCSClientDevice;
protected: // 통신 버퍼 객체
        CTWLQueue     *m_pWLOCSAckQueue;
        CTWLProtocol *m_pWLOCSSendData;
        CTWLProtocol *m_pWLOCSAckData;
        // ~ 생략
protected: // 응답 전송 함수
        BOOL CheckThreadEnd ();
        BOOL SendAcknowledge();
};
```

Fig. 4.22 Source Code of CTOCSAckThread Class

4.2.6 OHT 구동방식 상세설계

본 절에서는 OCS에서 지령된 물류 이송 정보를 바탕으로 OHT를 실제로 운전하여 물류 이송작업을 하기위한 OHT 구동에 방식에 관하여 구체적으로 논의하고자 한다. 본 시스템에서 OHT가 원활한 물류 이송 작업을 수행할 수 있도록 하기 위해서는 전체 시스템과의 견실한 연동성이 고려되어야 하며 물류 이송에 관련한 각 기능별 구동 방식이 모듈별로 상세하게 정립되어야 한다. 따라서 본 절에서는 OHT의 구동과 관련된 각 시스템과의 연동성에 관하여 고찰된 OHT의 구동개념에 관하여 논의하며 OHT의 물류 이송작업과 관련된 각 기능을 원활하게 수행할 수 있도록 하기위해 상세하게 설계된 OHT 구동방식에 관하여 논의한다.

4.2.6.1 구동개념 설계

본 논문의 4.1절에서는 통합 시스템을 효율적으로 관리하고 시스템간에 견실한 연동성을 갖출 수 있도록 하기 위하여 전체 시스템을 각 기능별 개체로 구분하여 설계하였다. OHT 최적 제어관리를 위한 전체 시스템 중 OHT의 구동에 관련된 부분은 OPC 시스템의 구동 개체인 OCS 클라이언트(client)와 핸드-터미널(hand-terminal) 시스템의 구동 개체인 핸드-터미널(hand-terminal) 서버(server)다. 이 2가지의 구동 개체는 실제 Application의 구현에 있어 OHT 구동 어플리케이션(application)에 통합된다. 본 논문에서는 OHT 구동 어플리케이션(application)에서의 구동 로직(logic) 모듈을 Run Mode Logic Module과 Teach Mode Logic Module로 구분하였으며 Run Mode Logic Module은 Semi-Standard 시나리오에 부합하는 작업 명령을 수행하기 위한 모듈 동작으로 구성한다. 반면 Teach Mode Logic Module은 구동 정보 학습을 위한 개별 구동부의 조그(jog) 동작과 홈(home) 동작, STEP 동작을 수행 하도록 설계하며 OCS 클라이언트(client)는 OHT에 탑재되어 OHT를 실제로 구동 시스템으로 Run Mode Logic Module만 사용하도록 설계한다. 핸드-터미널(hand-terminal) 서버(server)는 구동 정보 학습 시스템으로 Run Mode Logic Module과 Teach Mode Logic Module을 모두 사용할 수 있도록 설계 한다. 다음의 Fig. 4.23은 OHT 구동 Application과 연관된 각 개체별 연동 관계를 도시화 한 OHT 구동 개념도이며 Table. 25에서 OHT 구동에 관련된 모드별 기능 및 구조에 대한 설명을 정리하였다.

Fig. 4.23 Drawing of OHT Driving Concept

Table. 4.25 OHT Driving Function of Run Mode and Teach Mode

Run Mode Logic Module		
사용 기능 개체	OPC 시스템의 OCS Client 핸드-터미널(hand-terminal) 시스템의 핸드-터미널(hand-terminal) 서버(server)	
주요 기능	Semi-Standard 시나리오에 부합하는 작업 명령을 수행한다. Normal Transport, Abnormal Transport	
구동 구조	모듈 동작	
Teach Mode Logic Module		
사용 기능 개체	핸드-터미널(hand-terminal) 시스템의 핸드-터미널(hand-terminal) 서버(server)	
주요 기능	구동 정보 학습을 위한 작업 명령을 수행한다.	
구동 구조	JOG 동작	개별 구동부의 수동 조작을 위한 동작 구조이다. CW/CCW (정 방향/역 방향) 동작 : 주행부, 횡행부, 선회부, 승강부 Open/Close(열림/닫힘) 동작: 분기부, 낙하방지부, 척부
	HOME 동작	구동 위치를 초기화 하기 위한 동작 : 횡행부, 선회부, 승강부
	STEP 동작	주행에 있어 바코드 단위로 이동하기 위한 동작 : 주행부

Run Mode Logic Module의 구동 개념은 SEMI-Standard 시나리오에 부합하는 작업 명령을 수행하기 위한 각 구동부의 모듈 동작이다. 주행부, 횡행부, 선회부, 승강부의 모듈 동작은 학습된 구동 정보를 이용한다.

Teach Mode Logic Module의 구동 개념은 구동 정보 학습을 위한 각 구동부의 JOG/HOME/STEP 동작이다. 구동 정보 학습이 필요한 구동부에는 주행부, 횡행부, 선회부, 승강부가 있다. 나머지 구동부는 단순한 이진 논리 동작이므로 구동 정보 학습을 요구하지 않는다.

4.2.6.2 시스템 오류 처리기법 설계

본 절에서는 각 구동부의 Run Mode Logic Module 및 Teach Mode Logic Module 동작 중에 오류가 발생하였을 시 이에 대처하기 위하여 설계된 시스템 오류 발생시 처리 기법에 관하여 논의하고자 한다. 각 구동부의 Run Mode Logic Module 동작 중에 시스템 오류가 발생하였을 시 회복 가능한 오류는 동작을 일시 정지하고 회복이 완료될 때까지 대기한다. 오류가 회복되면 수행중인 동작을 재계한다. 회복 불능인 오류는 현재 동작을 완전히 멈추고 종료한다. 다음의 Fig. 4.24는 Run Mode Logic Module 동작 중에 오류가 발생하였을 시 이에 대처하기 위하여 설계된 처리기법을 도시화한 것이다.

Fig. 4.24 Drawing of Error Settling Technique on
Run Mode Logic Module

각 구동부의 Teach Mode Logic Module 동작 중에 오류가 발생하면 위와 같은 절차를 따른다. 오류의 종류에 관계없이 현재 동작을 완전히 멈추고 종료한다.

오류의 확인은 각 구동부와 관련된 Digital Input의 논리값으로 판단한다. 각 구동부의 동작 구조를 설명하면서 오류 상황과 관련된 Digital Input에 대해 자세히 설명한다. 그 외에 논리적인 오류도 있으나, 본 장에서는 순수한 구동 하드웨어의 동작과 관련된 부분만 설명하므로 제외한다.

각 구동부의 Teach Mode Logic Module 동작 중에 시스템 오류가 발생 하였을 시 오류의 종류에 관계없이 현재 동작을 완전히 멈추고 종료한다. 오류의 확인은 각 구동부와 관련된 Digital Input의 논리 값으로 판단한다. 오류 상황과 관련된 Digital Input에 관한 상세한 내용은 본 논문의 OHT를 구성하는 각 구동부의 동작 구조를 논의하는 절에서 자세히 다루도록 하겠다. 다음의 Fig. 4.25는 Teach Mode Logic Module 동작 중에 오류가 발생하였을 시 이에 대처하기 위하여 설계된 처리기법을 도시화한 것이다.

Fig. 4.25 Drawing of Error Settling Technique on
Run Mode Logic Module

4.2.6.3 주행부 구동방식 설계

OHT는 천장에 설치된 레일(rail)을 주행하며 물류 이송 작업을 수행한다. 다음의 Fig. 4.26은 주행 레일(rail)의 구조 및 주행에 관련된 센서를 도시화한 것이며 그림에서 알 수 있듯이 주행과 관련된 센서블록에는 바코드 표와 정위치 센서가 설치된다. OHT는 주행하며 바코드 센서를 통하여 바코드 표의 넘버(number)를 리드(read) 함으로써 현재 위치 정보를 확인 할 수 있으며 디지털 입, 출력 포트를 감시함으로써 정위치 센서를 감지하고 센서가 감지되었을 경우에 각 스테이션(station) 별로 이미 저장된 티칭(teaching) 데이터(data)만큼 거리를 주행한 후 정확한 위치에서 주행을 멈출 수 있다.

Fig. 4.26 Drawing of OHT Travelling Rail and Sensor

본 논문에서는 Run Mode Logic Module에서의 주행부 구동 방식을 다음의 Fig. 4.27와 같이 설계한다. OHT가 물류 이송을 위하여 주행기능을 수행하던 중 목적지 이전의 바코드를 리드(read)하면 감속을 하며 목적지의 바코드를 감지하면 극 감속을 하고 이후에 정위치 센서를 감지하면 미리 저장된 티칭 데이터만큼 주행한 후

최종위치에서 정지하도록 설계한다. 이와 같은 구동방식은 다음의 Fig. 4.28과 같은 플로우 챠트(flow-chart)를 설계함으로써 소프트웨어적으로 구현이 가능하다.

Fig. 4.27 Drawing of OHT Travelling Method
on Run Mode Logic Module

Fig. 4.28 Flow-chart of OHT Travelling Module Action
on Run Mode Logic Module

본 논문에서는 Teach Mode Logic Module에서의 주행부 구동 방식을 두 가지 모드(mode)로 나누어 설계한다. 다음의 Fig. 29는 OHT를 단순 수동조작 및 구동정보 학습을 위하여 설계된 조그(jog) 모드(mode)에서의 구동 방식을 나타내며 Fig. 4.30은 주행에 관련한 구동정보 학습(teaching)을 위하여 설계된 스텝(step) 모드(mode)

에서의 구동방식을 도시화한 것이다. 이와 같은 구동은 모두 핸드-터미널(hand-terminal) 클라이언트(client)에서 지령된 명령에 의해서 실제 구동이 가능하다. 조그(jog) 모드에서 주행부를 전진/후진으로 구동하면 정지할 때까지 계속 구동하게 된다. 구동 정보 학습을 위해 주행부의 속도를 바꾸어 가며 목표하는 정확한 위치에 도달할 수 있다. 주행부의 스텝(step) 동작은 바코드(barcode) 위치 단위로 이동하기 위한 동작이다. 각 센서 블록의 정 위치에 정확하게 이동하기 위해 사용한다. 구동 정보 학습에 있어 최종 위치에 대한 구동 정보는 정위치를 기점으로 산출된다. 정위치에서 주행부의 위치 값은 0이 된다. 우선 스텝(step) 동작으로 목적지에 정위치하고, 조그(jog) 동작으로 최종 위치를 맞춘다. 결과적으로 주행부는 정위치를 기점으로 최종 위치까지의 위치 값을 학습하게 된다. 본 논문에서는 이와 같은 구동정보 학습을 통틀어 티칭(Teaching) 작업이라 표기 하였으며 티칭 작업을 통하여 산출된 각 작업 스테이션(station)별 정위치 데이터를 티칭 데이터라 표기한다. Step 모드에서의 구동방식은 다음의 Fig. 4.31과 같은 플로우챠트(flow-chart)를 설계함으로써 소프트웨어적으로 구현이 가능하다.

Fig. 4.29 Drawing of Jog Mode Travelling Method
on Teach Mode Logic Module

Fig. 4.30 Drawing of Step Mode Travelling Method
on Teach Mode Logic Module

Fig. 31 Flow-chart of OHT Travelling Module Action
on Teach Mode Logic Module

만약, OHT가 주행 기능을 수행하던 도중 전방 충돌을 감지하던 센서인 충격감지용 압소바(absorber) 센서가 감지되거나 주행부 구동 장치인 서보 모터에서 알람 메시지가 감지되었을 경우에는 주행부 오류 상황으로 간주하고 동작 모드에 따라서 대처 할 수 있도록 한다. Run Mode Logic Module의 모듈 동작에서 오류가 발생하면 현재 진행 상태에서 주행을 멈추고 오류 회복을 기다리고 오류가 회복되면 이어서 작업을 진행한다. Teach Mode Logic Module의 동작에서 오류가 발생하면 주행을 완전히 멈추고 현재 동작을 종료한다.

4.2.6.4 승강부 구동방식 설계

주행부에 의하여 목적지에 도착한 OHT는 Foup을 싣거나 내려는 동작을 수행하여야 한다. 승강부는 Foup을 작업 스테이션(station)에 싣거나 내리는 호이스트(hoist) 기능을 담당하며 AC 서보모터에 의해서 구동된다. Run Mode Logic Module의 동작 방식은 아래의 Fig. 4.32에 나타나 있듯이 하강과 상승으로 구분되며, 각각은 3단계의 동작으로 구분된다. 각 단계의 구분은 승강의 안전을 위한 센서와 관계가 있으며 안전한 승강을 위해 각 단계별로 여러 가지 센서를 확인하도록 구동 방식을 설계한다. 본 설계에서는 안전한 승강부 구동을 위해 감시해야 할 센서의 종류 및 감지되었을 시 대처방안에 대하여 Table. 4.26에서 정리하였으며 각 동작 단계별로 감시해야 할 센서의 종류는 Table. 4.27에서 정리 하였다. Run Mode Logic Module에서의 승강부 구동은 Fig. 4.33과 같은 플로우-챠트(flow-chart)를 설계함으로써 소프트웨어적으로 구현이 가능하다.

Table. 4.26 Kind and Function of Safety Sensor on OHT Hoist

센서종류	감지 시 대처방안
승강 궤적에 장애물 침입	승강을 멈추고 회복을 기다림.
High Cut	고속 출발의 가능 여부 확인. 가능하지 않은 경우 승강을 멈추고 회복을 기다림.
Center Break	중심 이탈. 승강을 멈추고 회복을 기다림.
Overrun	하강에 오버런 발생. 승강을 멈추고 회복을 기다림.
Foup Before	Foup의 존재 여부 확인. 존재하지 않을 경우, 승강을 멈추고 회복을 기다림.
Servo Alarm	승강 서보 모터에 알람 발생. 승강을 멈추고 회복을 기다림.

Fig. 4.32 Drawing of Hoist Driving Method on Run Mode Logic Module

Fig. 4.33 Flow-chart of OHT Hoist Driving Module Action
on Run Mode Logic Module

Table. 4.27 Kind of Observation Sensor During Hoist driving

Run Mode Logic Module에서 하강의 각 단계별 감지 오류 센서						
	Emg. Stop	High Cut	Center Break	Over Run	Foup Before	Servo Alarm
1단계	O					O
2단계 전	O	O	O			O
2단계	O		O	O		O
3단계 전	O		O	O	O	O
3단계	O		O	O		O
Run Mode Logic Module에서 상승의 각 단계별 감지 오류 센서						
	Emg. Stop	High Cut	Center Break	Over Run	Foup Before	Servo Alarm
3단계	O		O			O
2단계 전	O		O			O
2단계	O		O			O
1단계 전	O	O	O			O
1단계	O		O			O
Servo Motor Alarm						
	Emg. Stop	High Cut	Center Break	Over Run	Foup Before	Servo Alarm
JOG 동작	O					O
HOME 동작	O					O

Teach Mode Logic Module의 승강부 구동은 주행부와 마찬가지로 조그(jog) 모드와 홈(home) 동작 모드의 두 가지 동작 모드로 구분하여 설계한다. 승강부의 조그(jog) 모드에서 하강/상승으로 구동하면 정지 명령을 지령 할 때까지 계속 구동하게 된다. 사용자는 구동 정보 학습을 위해 승강부의 속도를 바꾸어 가며 목표하는 정확한 위치에 도달할 수 있다. 승강부의 홈(home) 동작 모드는 원점(초기 위치)으로 이동하기 위한 모드이다. 왜냐하면 구동 정보 학습은 원점을 기점으로 이루어져

야 하기 때문이다. 원점에서 승강부의 위치 값은 0이며 처음에 홈(home) 동작으로 원점복귀를 하고, 조그(jog) 동작으로 정확한 위치를 맞추어야 한다. 결과적으로 승강부는 원점을 기점으로 최종 위치까지의 위치 값을 학습하게 된다. Fig. 4.34는 Teach Mode Logic Module에서의 조그(jog) 모드 구동방식을 도시화 한 것이며 Fig. 4.35 Teach Mode Logic Module에서의 홈(home) 동작 모드 구동방식을 도시화한 것이다.

Fig. 4.34 Drawing of Jog mode Hoisting Method on Teach Mode Logic Module

Fig. 4.35 Drawing of Home Mode Hoisting Method on Teach Mode Logic Module

4.2.6.5 횡행부 구동방식 설계

승강부는 주행부에 의하여 목적지에 도착한 후 Foup을 싣고 내리는 호이스트(hoist) 기능을 수행한다. 호이스트(hoist) 동작을 수행하기에 앞서 OHT는 작업 해당 작업 스테이션(station)에 Foup을 안전하고 정확하게 싣고 내리기 위하여 호이스트(hoist) 장치 자체를 횡방향으로 이송(횡행부)할 수 있으며 양 방향으로 회전(선회부) 할 수 있다. 이러한 기능은 각 스테이션(station) 별로 Foup이 놓여있는 위치가 횡 방향 및 시계방향, 반 시계방향으로 약간씩 차이가 날 수 있기 때문에 꼭 필요한 기능이다. 본 절에서는 호이스트(hoist) 장치를 횡방향으로 이송시키는 기능을 담당 하는 횡행부 동작에 관하여 논의하고자 한다. 본 논문에서는 횡행부 구동 방식을 Fig. 4.36과 같이 설계한다. 스테핑 모터(stepping motor)에 의해서 구동되는 횡행부는 리미트(limit) 센서가 양단에 설치되어 최대 유효 구동 거리가 정해져 있다. Run Mode Logic Module에서의 횡행부의 구동은 Teach Mode Logic Module에서의 학습된 구동정보를 기반으로 구동하며 Fig. 4.37와 같은 플로우-챠트(flow-chart)를 설계함으로써 소프트웨어적으로 구현이 가능하다.

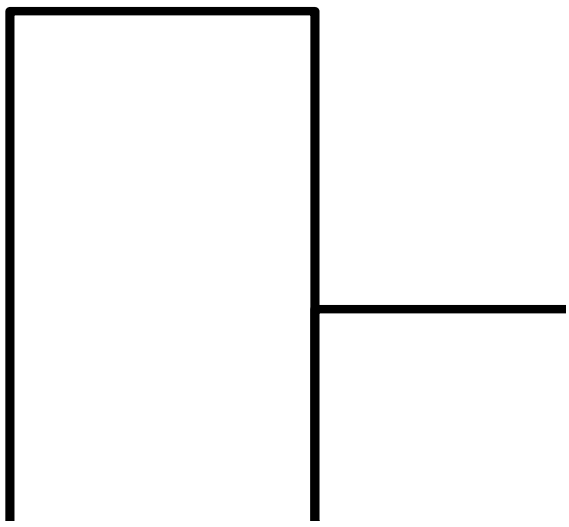


Fig. 4.36 Drawing of Horizontal Axis Driving Method on Run Mode Logic Module

Fig. 4.37 Flow-chart of Horizontal Axis Driving Module Action on Run Mode Logic Module

횡행부의 Teach Mode Logic Module에서의 구동방식은 승강부와 마찬가지로 조그(jog) 모드 및 홈(home) 동작 모드로 구분하여 설계한다. 아래의 Fig. 4.38은 조그(jog) 모드에서의 횡행부 구동을 도시화한 것이며 조그(jog) 모드에서 이동/복귀로 구동하면 정지 명령을 지령할 때까지 계속 구동하며 구동 정보 학습을 위해 횡행부의 속도를 바꾸어 가며 목표하는 정확한 위치에 도달할 수 있다.

Fig. 4.38 Drawing of Jog Mode Horizontal Axis Driving Method on Teach Mode Logic Module

횡행부의 Teach Mode Logic Module에서의 횡행부 홈(home) 동작 모드는 원점(초기 위치)으로 이동하기 위한 모드이다. 구동 정보 학습은 원점을 기점으로 이루어져야 하며 원점에서 횡행부의 위치 값은 0이 된다. 각 스테이션(station)별 티칭(teaching)작업을 하기 위해서는 우선, 홈(home) 동작으로 원점복귀를 하고, 조그(jog) 동작으로 정확한 위치를 맞춘다. 결과적으로 횡행부는 원점을 기점으로 최종 위치까지의 위치 값을 학습하게 된다. 구동 방식은 승강부의 방식과 동일하다.

4.2.6.6 선회부 구동방식 설계

선회부의 기능 및 수행 목적에 대한 내용은 4.2.6.5절에서 이미 논의 하였다. 본 논문에서는 선회부 구동 방식을 다음의 Fig. 4.39와 같이 설계 한다. 스테핑 모터(stepping motor)에 의해서 구동되는 선회부는 횡행부와 마찬가지로 Limit 센서가 양단에 설치되어 최대 유효 구동 거리가 정해져 있다. Run Mode Logic Module에서의 선회부의 구동은 Teach Mode Logic Module에서의 학습된 구동정보를 기반으로 구동하며 Fig. 4.40과 같은 플로우-챠트(flow chart)를 설계함으로써 소프트웨어적으로 구현이 가능하다. Teach Mode Logic Module에서의 동작 모드에 따른 구동 방식은 승강부 및 횡행부에서 논의한 내용과 대동소이(大同小異) 하므로 본 절에서는 생략한다.

Fig. 4.39 Drawing of Turning Axis Driving Method
on Run Mode Logic Module

Fig. 4.40 Flow-chrat of Turning Axis Driving Module Action
on Run Mode Logic Module

4.2.6.7 기타 구동부의 구동방식 설계

OHT의 주행부, 승강부, 선회부, 횡행부를 제외한 나머지 구동부는 분기점에서 좌측 및 우측의 분기 방향 전환용 DC 모터(motor)를 구동하는 분기부, 이송중인 Foup의 낙하를 방지하기 위한 낙하방지 셔터용 DC 모터(motor)를 구동하는 낙하방지부, 호이스트(hoist)에 의해 Foup을 싣고 내리는 동작을 수행하기 위해 Foup을 클램핑(clamping)하는 DC Motor를 구동하는 Chuck부이다.

Fig. 4.41은 분기부의 Run Mode Logic Module 및 Teach Mode Logic Module에서의 구동 방식을 설계하여 도시화한 것이며 본 구동은 단순한 이진 논리 연산에 의해서 구동된다. Run Mode Logic Module의 모듈 구동 방식과 Teach Mode Logic Module의 조그(jog) 모드 구동 방식은 모두 동일하며 구동 정보 학습을 요구하지 않는다. 분기 방향의 판별은 Normal Transport 작업 명령의 Path 정보를 이용한다.

Fig. 4.41 Drawing of Junction Action Driving Method

Fig. 4.42는 낙하 방지부의 Run Mode Logic Module 및 Teach Mode Logic Module에서의 구동 방식을 설계하여 도시화한 것이며 본 구동부 또한 DC 모터의 정·역 회전 방향 제어만을 필요로 하므로 단순한 이진 논리연산에 의해서 구동된다. Run Mode Logic Module의 모듈 구동 방식과 Teach Mode Logic Module의 조그(jog) 모드 구동 방식은 모두 동일하며 구동 정보 학습을 요구하지 않는다. 또한 구동 정보 학습을 요구하지 않는다. Normal Transport 작업 명령이 완료되면 항상 닫힘(close) 상태가 된다.

Fig. 4.42 Drawing of Foup Drop Prevention Action Driving Method

다음의 Fig. 4.43은 Chuck부의 Run Mode Logic Module 및 Teach Mode Logic Module에서의 구동 방식을 설계하여 도시화한 것이며 본 구동부 또한 분기부 및 낙하 방지부와 마찬가지로 DC 모터의 정·역 회전 방향 제어만을 필요로 하므로 단순한 이진 논리연산에 의해서 구동된다. Run Mode Logic Module의 모듈 구동 방식과 Teach Mode Logic Module의 조그(jog)모드 구동 방식은 모두 동일하며 구동 정보 학습을 요구하지 않는다. MOVELOAD 작업에서 CLOSE(Clamping) 동작이 수행되고 MOVEUNLOAD 작업에서 OPEN(Release) 동작이 수행된다.

Fig. 4.43 Drawing of Foup Clamping and Release Action Driving Method

Chuck부는 구동중에 항상 중심이탈 센서(center break sensor)를 감시하여야 한다. 본 센서는 Foup이 클램핑(clamping) 장치의 중앙에 위치하고 있을 때에만 클램핑(clamping) 하기 위함이다. 만약 다른 구동부의 동작 중에도 진동 등의 외란으로 인하여 Foup이 Chuck의 중앙에서 이탈 할 수 있으므로 Foup을 OHT에 싣는 과정 및 이송과정 중에 계속해서 감시하여야 한다. 본 센서의 감시는 센서가 인터페이스 되어있는 티지털 입, 출력 포트를 감시함으로써 가능하다. 만약, Run Mode Logic Module에서 중심이탈 센서(center break sensor)에 의해서 Foup의 이탈을 감지하면 즉시 동작을 멈추고 회복을 기다려야하며 정상 상태로 회복되면 동작을 재계한다. Teach Mode Logic Module에서 이와 같은 상황이 발생하면 동작을 완전히 멈추고 종료한다.

4.2.7 OHT 구동 구현

본 절에서는 본 논문의 4.2.6절에서 이미 논의한 바 있는 OHT 구동 구조에 대한 상세 설계 내용을 기반으로 하는 OHT 구동 소프트웨어 구현에 대해서 논의하고자 한다. 본 논문에서는 OHT 구동의 구현을 위하여 다음의 Fig. 4.44와 같이 OHT 구동에 관한 기본 클래스를 구성한다. 여기서 CTBRDevice 클래스는 바코드를 읽는 기능을 담당하며 CTDICMDevice 클래스는 일반적인 입력을 위해 사용한다. CTDITRDevice 클래스는 주행부와 관련된 입력을 위해 사용하며 CTDIBRDevice 클래스는 분기부와 관련된 입력을 위해 사용한다. CTDIFPDevice 클래스는 낙하방지부와 관련된 입력을 위해 사용하며 CTDIRADevice 클래스는 횡행부와 관련된 입력을 위해 사용한다. CTDIRODevice 클래스는 선회부와 관련된 입력을 위해 사용하며 CTDIUDDevice 클래스는 승강부와 관련된 입력을 위해 사용하고 CTDICHDevice 클래스는 척부와 관련된 입력을 위해 사용한다. CTDOCMDevice 클래스는 각종 모터의 브레이크 등의 일반 출력 제어에 사용하며 CTDOBRDevice 클래스는 분기 DC 모터를 제어하기 위해 사용한다. CTDOFPDevice 클래스는 낙하방지 DC 모터를 제어하기 위해 사용하며 CTDOCHDevice 클래스는 척 DC 모터를 제어하기 위해 사용한다. CTMCTRDevice 클래스는 주행 서보 모터를 제어하기 위해 사용하며 CTMCRADevice 클래스는 횡행 스텝 모터를 제어하기 위해 사용하고 CTMCRODevice 클래스는 선회 스텝 모터를 제어하기 위해 사용한다. CTMCUDDevice 클래스는 승강 서보 모터를 제어하기 위해 사용한다. 각 클래스에 대하여 구현된 프로그램 코드는 내용이 방대하므로 생략하도록 한다.

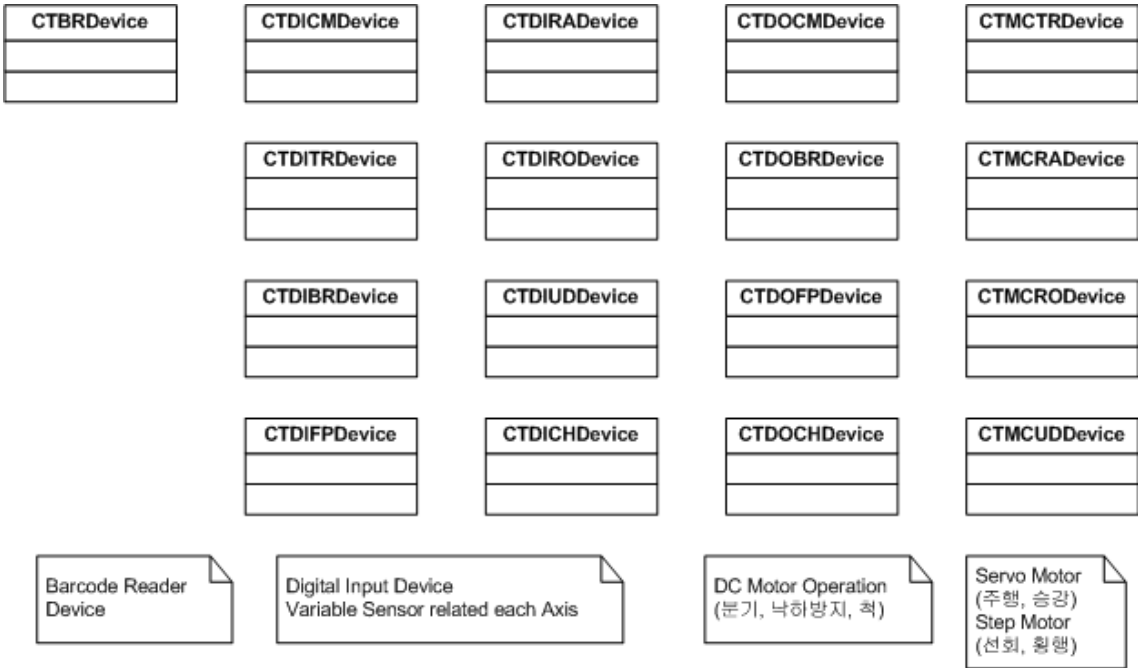


Fig. 4.44 Drawing of Basic Class for OHT Operating

다음의 Fig. 4.45는 구동작업을 위한 클래스의 구성을 도시화 한 것으로써 CTOCSClientThread 클래스는 구동 작업의 Main Thread 기능을 하며 CTDoOpMode 클래스는 정규 구동 작업 모듈의 기능을 담당하고 CTDoAbnormalMode 클래스는 비정규 구동 작업 모듈의 기능을 담당한다. CTDoTeachMode 클래스는 구동 정보 학습 모듈 기능을 담당한다. 이와 같이 구성된 통신 작업 클래스는 다음의 Fig. 4.46, 4.47, 4.48, 4.49와 같이 실제로 프로그램 상에서 구현하였다. 상세 코드는 내용이 방대하여 생략하고 관련된 부분의 정의만 포함한다.

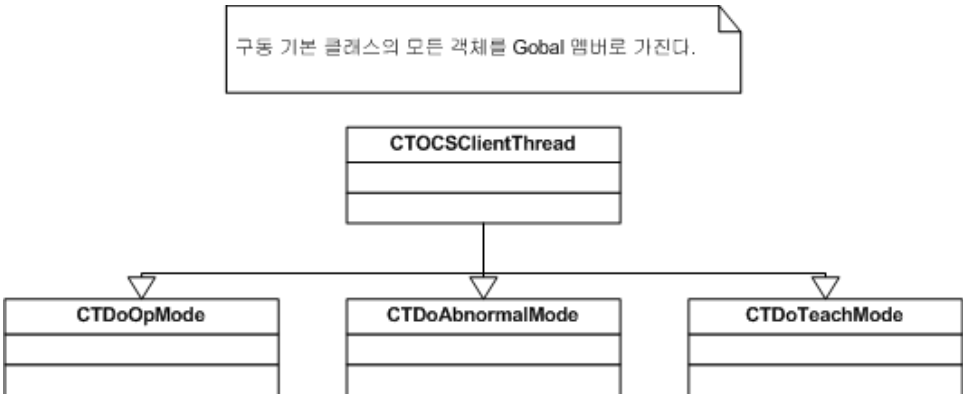


Fig. 4.45 Drawing of Working Class for OHT Operating

```
//////////////////////////////////////////////////////////////////////////
// CTOCSClientThread Class ///////////////////////////////////////
class CTOCSClientThread : public CTBaseThread
{
public:
	CTOCSClientThread();
	virtual ~CTOCSClientThread();
	// ~ 생략
protected: // 구동 기본 클래스 객체
	// DO (PCI-1752)
	CTDOCMDevice		*m_pDOCMDevice;		CTMCBRDevice
*m_pMCBRDevice;
	CTMCFPDevice *m_pMCFPDevice; CTMCCHDevice *m_pMCCHDevice;
	// DI (PCI-1754)
	CTDICMDevice *m_pDICMDevice; CTDITRDevice *m_pDITRDevice;
	CTDIBRDevice *m_pDIBRDevice; CTDIFPDevice *m_pDIFPDevice;
	CTDIRADevice *m_pDIRADevice; CTDIRODevice *m_pDIRODevice;
	CTDIUDDevice *m_pDIUDDevice; CTDICHDevice *m_pDICHDevice;
	// MC (PCI-1240)
	CTMCTRDevice *m_pMCTRDevice; CTMCRADevice
*m_pMCRADevice;
	CTMCRODevice *m_pMCRODevice; CTMCUDDevice
*m_pMCUDDevice;
	// COM (BL-180)
	CTBRDevice *m_pBRDevice;
	// ~생략
public: // 구동 로직 객체
	CTPendCommandStatus	*m_pPendCommandStatus;
	CTUpLoadCommandStatus *m_pUpLoadCommandStatus;
	CTOpCmdExeStage		*m_pOpCmdExeStage;
	CTMCTeachDataItem	*m_pMCTeachDataItem;
protected: // 구동 로직 모듈
	CTDoOpMode		m_tDoOpMode;
	CTDoAbnormalMode m_tDoAbnormalMode;
	// ~ 생략
```

Fig. 4.46 Source code of CTOCSClientThread Class

```
////////////////////////////////////////////////////////////////////////////
// CTDoOpMode Class ////////////////////////////////////////////////
class CTDoOpMode : public CObject
{
public:
        CTDoOpMode();
        virtual ~CTDoOpMode();
protected: 기본 구동 클래스 객체
        // DO (PCI-1752)
        CTDOCMDevice *m_pDOCMDevice; CTMCBRDevice *m_pMCBRDevice;
        CTMCFPDevice *m_pMCFPDevice; CTMCCHDevice *m_pMCCHDevice;
        // DI (PCI-1754)
        CTDICMDevice *m_pDICMDevice; CTDITRDevice *m_pDITRDevice;
        CTDIBRDevice *m_pDIBRDevice; CTDIFPDevice *m_pDIFPDevice;
        CTDIRADevice *m_pDIRADevice; CTDIRODevice *m_pDIRODevice;
        CTDIUDDevice *m_pDIUDDevice; CTDICHDevice *m_pDICHDevice;
        // MC (PCI-1240)
CTMCTRDevice *m_pMCTRDevice; CTMCRADevice *m_pMCRADevice;
        CTMCRODevice        *m_pMCRODevice;        CTMCUDDevice
*m_pMCUDDevice;
        // COM (BL-180)
        CTBRDevice *m_pBRDevice;
protected: // 구동 로직 객체
        CTPendCommandStatus    *m_pPendCommandStatus;
        CTUpLoadCommandStatus *m_pUpLoadCommandStatus;
        CTOpCmdExeStage        *m_pOpCmdExeStage;
        CTMCTeachDataItem      *m_pMCTeachDataItem;
protected:
        CTDoSubMode *m_pDoSubMode;
        // ~ 생략
public: // 구동 시작 함수
        BOOL DoOpMode();
//~생략
```

Fig. 4.47 Source Code of CTDoOpMode Class

```
////////////////////////////////////////////////////////////////////////////
// CTDoAbnormalMode Class ////////////////////////////////////////
class CTDoAbnormalMode : public CObject
{
public:
        CTDoAbnormalMode();
        virtual ~CTDoAbnormalMode();
protected: // 기본 구동 클래스 객체
        // DO (PCI-1752)
        CTDOCMDevice            *m_pDOCMDevice;         CTMCBRDevice
*m_pMCBRDevice;
        CTMCFPDevice *m_pMCFPDevice; CTMCCHDevice *m_pMCCHDevice;
        // DI (PCI-1754)
        CTDICMDevice *m_pDICMDevice; CTDITRDevice *m_pDITRDevice;
        CTDIBRDevice *m_pDIBRDevice; CTDIFPDevice *m_pDIFPDevice;
        CTDIRADevice *m_pDIRADevice; CTDIRODevice *m_pDIRODevice;
        CTDIUDDevice *m_pDIUDDevice; CTDICHDevice *m_pDICHDevice;
        // MC (PCI-1240)
        CTMCTRDevice            *m_pMCTRDevice;         CTMCRADevice
*m_pMCRADevice;
        CTMCRODevice            *m_pMCRODevice;         CTMCUDDevice
*m_pMCUDDevice;
        // COM (BL-180)
        CTBRDevice *m_pBRDevice;
protected: // 구동 로직 객체
        CTPendCommandStatus    *m_pPendCommandStatus;
        CTUpLoadCommandStatus *m_pUpLoadCommandStatus;
        CTOpCmdExeStage         *m_pOpCmdExeStage;
        CTMCTeachDataItem       *m_pMCTeachDataItem;
protected:
        // Sub-Module Object
        CTDoSubMode *m_pDoSubMode;
        // ~ 생략
public: // 구동 시작 함수
        BOOL DoAbnormalMode();
//~생략
```

Fig. 4.48 Source Code of CTDoAbnormalMode Class

```
////////////////////////////////////////////////////////////////////////////
// CTDoTeachMode Class ////////////////////////////////////////////
class CTDoTeachMode : public CObject
{
public:
        CTDoTeachMode();
        virtual ~CTDoTeachMode();
protected: // 기본 구동 클래스 객체
        // DO (PCI-1752)
        CTDOCMDevice            *m_pDOCMDevice;         CTMCBRDevice
*m_pMCBRDevice;
        CTMCFPDevice *m_pMCFPDevice; CTMCCHDevice *m_pMCCHDevice;
        // DI (PCI-1754)
        CTDICMDevice *m_pDICMDevice; CTDITRDevice *m_pDITRDevice;
        CTDIBRDevice *m_pDIBRDevice; CTDIFPDevice *m_pDIFPDevice;
        CTDIRADevice *m_pDIRADevice; CTDIRODevice *m_pDIRODevice;
        CTDIUDDevice *m_pDIUDDevice; CTDICHDevice *m_pDICHDevice;
        // MC (PCI-1240)
        CTMCTRDevice            *m_pMCTRDevice;         CTMCRADevice
*m_pMCRADevice;
        CTMCRODevice            *m_pMCRODevice;         CTMCUDDevice
*m_pMCUDDevice;
        // COM (BL-180)
        CTBRDevice *m_pBRDevice;

protected: // 구동 로직 객체
        CTPendCommandStatus     *m_pPendCommandStatus;
        CTUpLoadCommandStatus *m_pUpLoadCommandStatus;
        CTOpCmdExeStage         *m_pOpCmdExeStage;
        CTMCTeachDataItem       *m_pMCTeachDataItem;
protected:
        // Sub-Module Object
        CTDoSubMode *m_pDoSubMode;
// ~ 생략
```

Fig. 4.49 Source Code of CTDoTeachMode Class

제 5장 실험 및 고찰

5.1 실험장치

본 논문의 2장, 3장, 4장에서 논의한 내용을 기반으로 최적경로 탐색 알고리즘 기반의 OHT 최적 제어관리 시스템을 Fig. 5.1과 5.2와 같이 제작하였다. Fig. 5.1의 (a)는 명령개체인 OCS 서버(server), 핸드-터미널(hand-terminal) 클라이언트(client), 프로그램 메니져(program manager) 클라이언트(client)가 탑재된 컴퓨터 장치이다. 본 장치의 본체에는 OHT와의 무선통신을 위하여 무선랜 모듈(wireless lan module)이 USB로 인터페이스(interface)되어 있다. Fig. 5.2는 실험용으로 제작된 OHT로써 (a)는 OHT 주행부, (b)는 OHT 승강, 횡행, 선회부가 합쳐진 호이스트(hoist) 이며, (c)는 OCS 클라이언트(client), 핸드-터미널(hand-terminal) 서버(server), 프로그램 메니져(program manager) 서버(server)가 탑재된 OPC, (d)는 Foup을 클램핑(clamping)하는 Chuck부, (e)는 300mm 웨이퍼(wafer) 이송용 카세트인 Foup, (f)는 이송중인 Foup의 추락을 방지하는 낙하방지부이다.

Fig. 5.1 Computer for Experiment

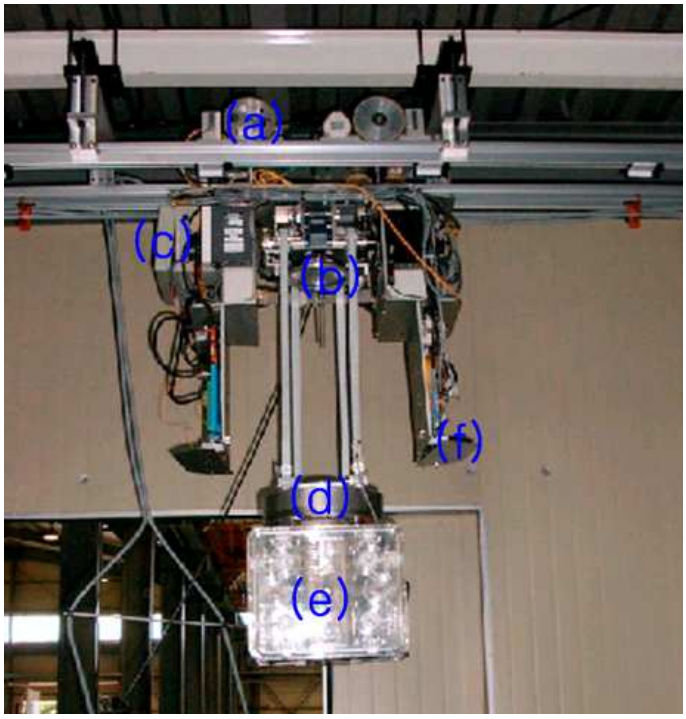


Fig. 5.2 OHT and Foup

Fig. 5.3은 다채널의 무선랜 어뎁터(wireless lan adapter)의 중계기 역할과 오토 로밍 서비스(auto roaming service)를 동시에 지원하는 엑세스 포인트(access point) 이며 Fig. 5.4는 무선랜 어뎁터(wireless lan adapter), Fig. 5.5는 엑세스 포인트 (access point) 간의 수신 감도 향상을 위한 안테나(antenna)이다.

Fig. 5.3 Access Point

5.4 Wireless LAN Adapter

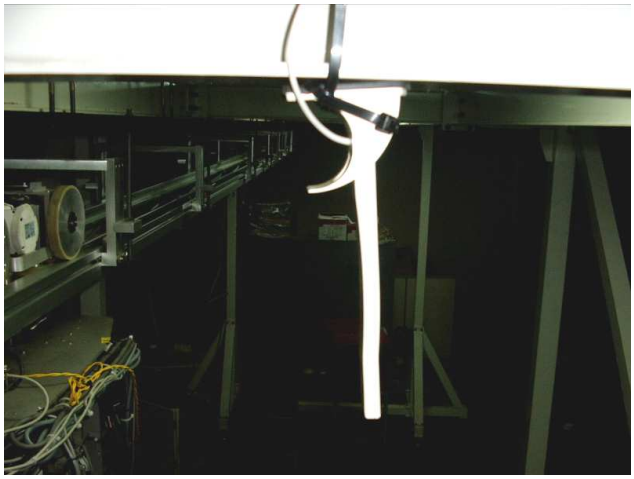


Fig. 5.5 Wireless LAN Antenna

5.2 시뮬레이션 실험

본 논문의 3장에서 제안한 최적경로 탐색 및 충돌방지 알고리즘의 타당성을 검증하기 위하여 다음의 Table. 5.1과 같은 조건을 설정하여 시뮬레이션을 수행 하였다.

Table. 5.1 OHT System Scope and OHT specification

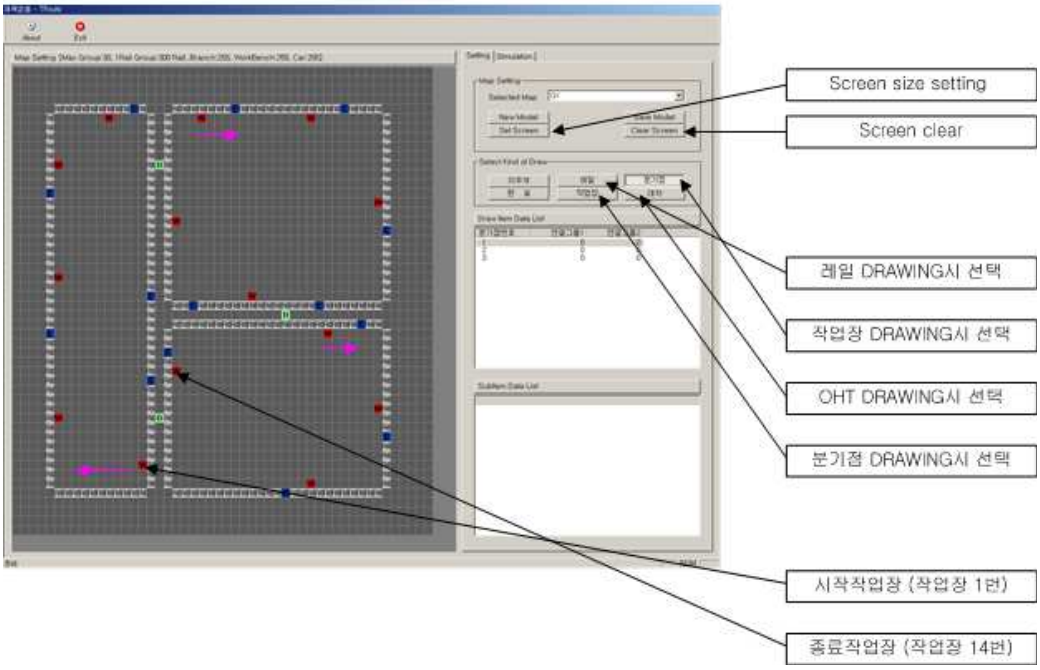


Fig. 5.6 Simulation Condition Creating Window

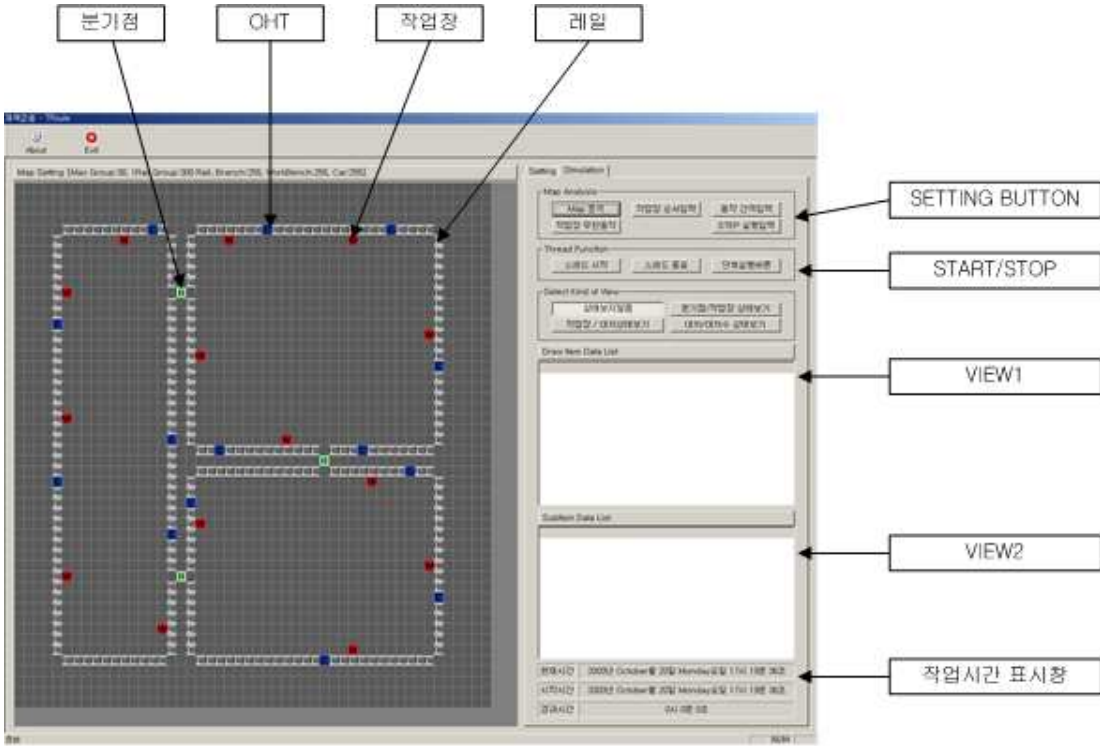


Fig. 5.7 Simulation Condition Creating Finish Window

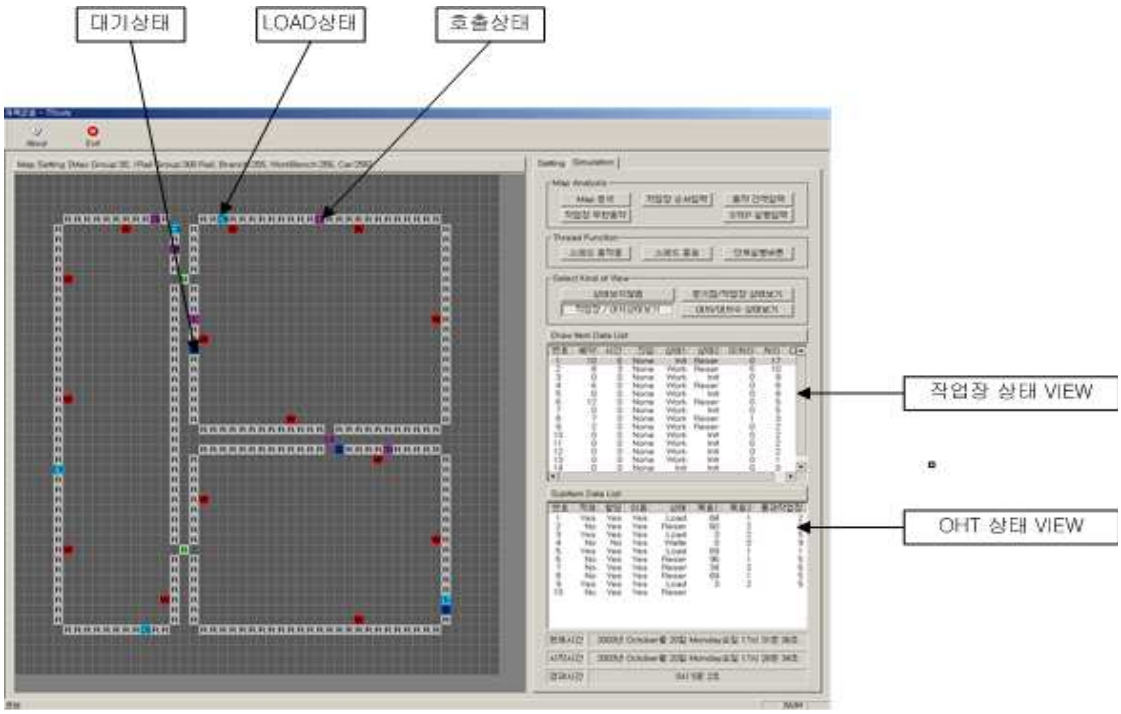


Fig. 5.8 Simulation Processing Window

Fig. 5.6은 시뮬레이션 환경을 구성하는 윈도우(window) 이며 Fig. 5.7은 시뮬레이션 환경 구성을 완성하여 시뮬레이션을 시작할 준비가 완료된 윈도우(window)를 나타낸다. 시뮬레이션은 일정시간(여기서는 1초) 간격으로 작업 요구가 발생 되도록 한다. 작업장으로부터 물류 이송에 관한 요구가 발생하였을 때 3장에서 제안한 알고리즘에 의해서 최적경로를 탐색하고 작업장에서 가장 가까이 있는 OHT를 호출하여 이송 작업을 수행하도록 한다. 이때, 본 논문에서 제안한 충돌방지 로직(logic) 또한 연동하도록 한다. 본 시뮬레이션에서는 작업장의 물류공급을 무한정으로 공급하도록 설정하여 시뮬레이션상의 모든 OHT가 대기 상태 없이 연속 운전을 하도록 하였으며 300시간 이상의 연속 시뮬레이션을 수행하였다. 시뮬레이션을 시작한 이후 현재까지의 모든 시뮬레이션 결과는 Fig. 5.8의 작업장 상태 뷰(view)와 OHT 상태 뷰(view)에 기록된다. 시뮬레이션을 수행한 결과 제안된 알고리즘에 의해 OHT가 최적의 경로로 주행하며 물류 이송작업을 수행함을 관찰할 수 있었으며 작업 수행 중에 OHT 상호간에 충돌이나 분기점에서의 교통 장애가 전혀 발생되지 않았으며 각 작업장 별로 이송하여야 할 물류가 적체되지 않음을 확인할 수 있었다. 시뮬레이션을 수행하여 이와 같은 결과를 도출함으로써 본 논문에서 제안한 알고리즘이 타당함을 검증하였다.

5.3 OHT 구동 실험

OHT의 구동 실험은 다음과 같은 세미 스텐다드 시나리오(SEMI-standard scenario)를 기반으로 수행하였다. OHT에 의한 물류 이송방식은 스텐다드 시나리오(SEMI-standard scenario)에서 규정하는 물류 이송규칙을 반드시 준수 하여야 하므로 본 실험 방법을 채택한다.

Fig. 5.9는 Move and Load Transport의 시나리오로써 OHT가 Foup을 Load하기 위한 목적지로 자율 주행하여 Foup을 로드(load) 하기까지의 시나리오이다. Fig. 5.17은 Move and Unload Transport 시나리오로써 Move and Load Transport 시나리오(scenario)에 의해서 로드(load)된 Foup을 다음 목적지까지 자율 주행하여 이동한 다음 해당 목적지에 언로드(unload)하는 시나리오이다.

Fig. 5.9 Move and Load Transport Scenario

본 실험을 수행하기에 앞서 Fig. 5.10의 핸드-터미널(Hand-Terminal) 시스템을 사용하여 구동 정보 학습을 수행하여 각 스테이션(station)의 티칭 데이터를 OHT에 입력하였다. 여기서, 명령개체와 구동개체를 무선랜(wireless lan)으로 인터페이스(interface)하기 위한 IP주소 설정은 다음의 그림 5.11과 같이 한다. Move and Load

Transport 시나리오(scenario)에 의한 OHT 구동실험에서 OCS로부터 Foup을 로드(load)하기 위한 목적지로 주행하라는 명령을 지령 받으면 OHT는 명령에 대한 응답을 하고 주행모듈의 동작에 의해서 주행 동작을 수행하며 결과는 다음의 Fig. 5.12와 같다. 여기서, 무선랜(wireless lan)을 이용한 명령의 송수신 수행 과정은 Fig. 5.13에서 확인한다.

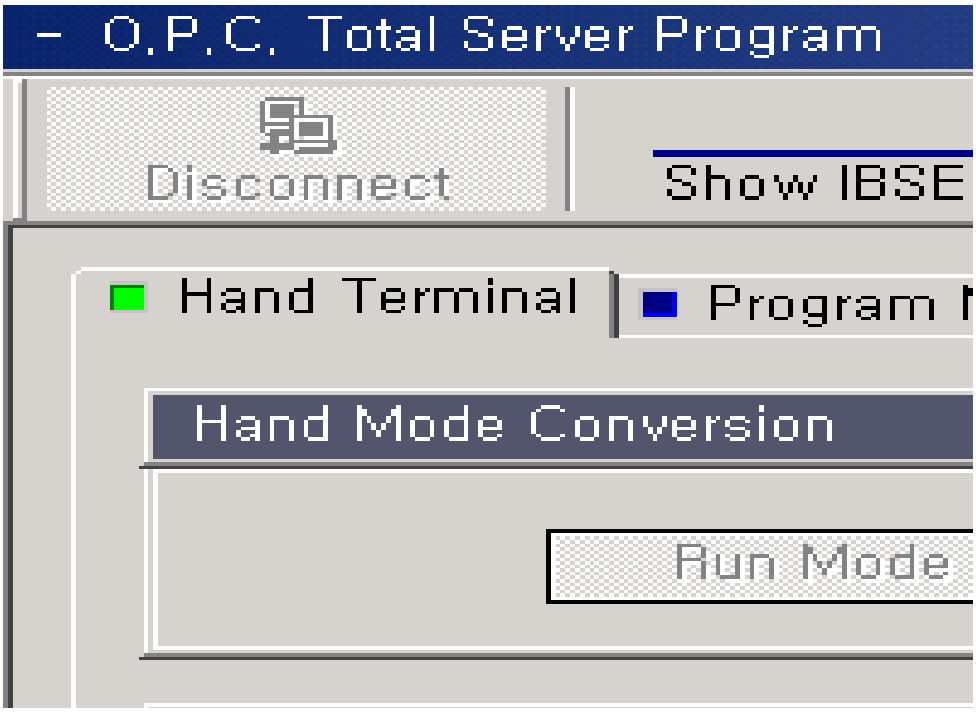


Fig. 5.10 Main Window of Hand-Terminal System

Fig. 5.11 Wireless LAN IP Address Setting Window

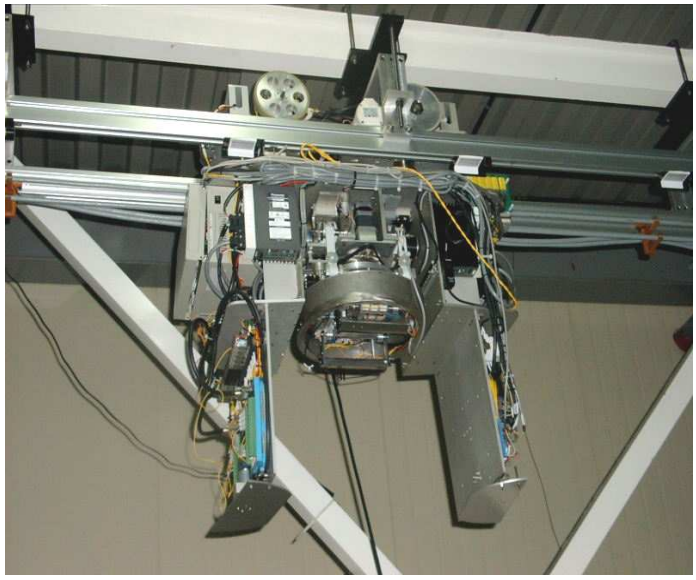


Fig. 5.12 OHT Travelling for Foup Load

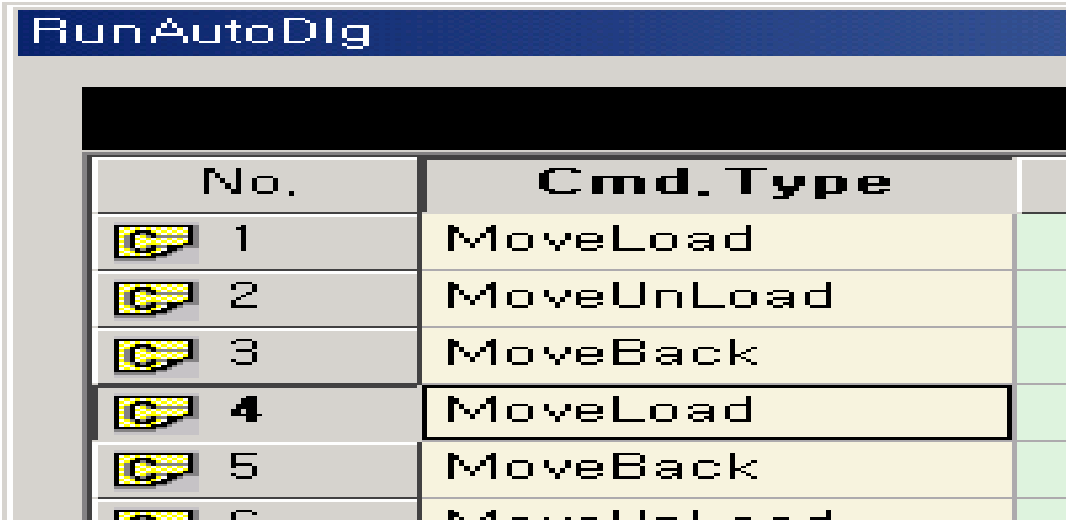


Fig. 5.13 Run Auto Mode Monitoring Window

이후에 OHT가 Foup을 로드(load)하기 위한 목적지에 도착하면 도착 상황을 OCS 서버(server)에 보고한다. 이후에 OCS 서버(server)로부터 Foup을 로드(load)하라는 명령을 받으면 미리 입력된 티칭(teaching) 정보를 기반으로 횡행부 및 선회부가 구동하고 이후에 승강부에 의해서 Chuck이 하강되며 하강이 완료되면 Foup을 클램핑(clamping)한다. Foup을 로드(load)하기 위하여 목적지에 도착한 후 Foup을 클램핑(clamping)하는 구동실험에 대한 결과는 다음의 Fig. 5.14와 같다. 여기서, 명령의 송신 및 수신 상태의 확인은 Fig. 5.13에서 가능하다.

Fig. 5.14 OHT Foup Clamping

Foup 클램핑(clamping) 작업이 종료되면 Foup을 언로드(unload)하기 위한 목적지로 이송하기 위해서 Foup을 OHT 내부로 완전히 로딩(loading)하여야 하고 주행중 Foup의 낙하를 방지하기 위해서 낙하방지부를 구동한다. 다음의 Fig. 5.15는 Foup을 OHT 내부로 로딩(loading)하는 실험의 결과를 나타내며 Fig. 5.16은 로딩(loding)이 완료된 후 횡행부 및 선회부가 원점복귀 및 낙하방지부 구동실험에 대한 결과이다. 모든 구동부 실험과 연동되어 수행되는 무선랜(wireless lan) 통신실험에 대한 결과는 Fig. 5.13에서 확인한다. 이후에 Move and Unload Transport 시나리오(scenario)에 의해서 Foup을 언로드(unload)하기 위한 목적지로의 이송명령이 지령되면 OHT는 주행부를 구동하여 목적지로 Foup을 이송한다.

Fig. 5.15 OHT Hoist Driving for Foup Load

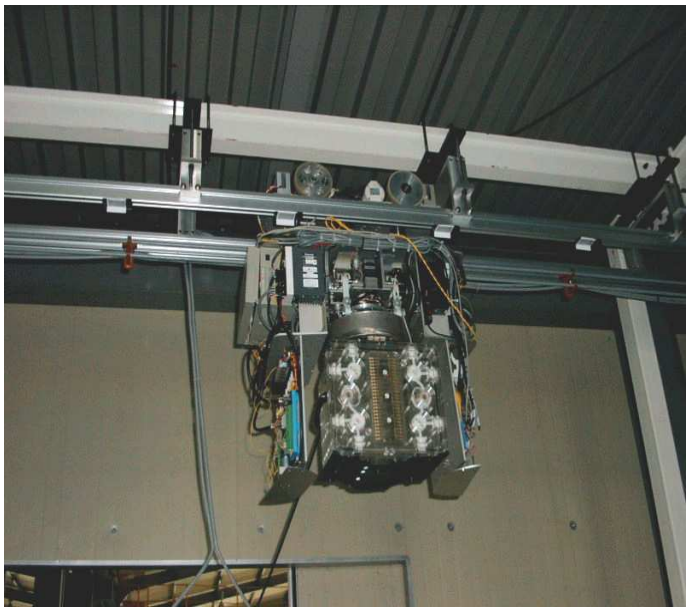


Fig. 5.16 Foup Load Finsh

Fig. 5.17 Move and Unload Transport Scenario

OCS 서버(server)로부터 Foup 언로드(unload) 명령이 지령되면 OHT는 목적지를 향해서 주행한다. 목적지에 도착한 OHT는 Fig. 5.17에 나타나 있듯이 도착 상황을 OCS 서버(server)에 보고한 후 낙하방지부, 횡행부, 선회부를 구동한다. 이와 같은 구동에 관한 실험 결과는 Fig. 5.18과 같다. Fig. 5.19는 Foup을 언로드(unload)하기 위한 승강부 구동실험의 결과이며 Fig. 5.20은 Foup을 완전히 언로드(unload)하기 위해서 Foup을 릴리즈(release)하기 위한 Chuck부의 구동실험 결과이다. 이후에 수행한 실험인 원점복귀를 위한 승강부, 횡행부, 선회부의 구동 실험에 대한 결과는 Move and Unload Transport 시나리오(scenario)에 의한 구동실험 결과와 중복되므로 생략한다. 무선랜(wireless lan) 통신 실험에 대한 결과는 Move and Load Transport 시나리오(scenario)에서의 실험과 같은 방법으로 확인이 가능하다. 다음의 Fig. 5.21은 본 논문에서 구현한 프로그램 메니져 클라이언트(program manager Client)의 메인(main) 윈도우(window)를 나타낸다.

Fig. 5.18 OHT Destination Arrived for Foup Unload

Fig. 5.19 OHT Hoist Driving for Foup Unoad

Fig. 5.20 OHT Hoist Driving for Foup Load

Fig. 5.21 Program Manager Client Main Window

본 논문에서는 위와 같은 OHT 구동 실험을 300시간 이상 연속으로 수행하였다. 본 실험을 수행하면서 시스템 페일(system fail) 현상이 한번도 발생하지 않았으며 시스템을 구성하는 각 개체간의 통신 및 구동소프트웨어 모듈 간에 충돌 현상이 발생하지 않음을 확인 할 수 있었다. 따라서 본 논문에서 제안한 통신구조 및 프로토콜, OHT 구동구조에 대한 설계가 OHT를 세미 스텐다드 시나리오(Semi-standard scenario)에 만족하며 운전 할 수 있도록 설계되었음을 검증하였다.

제 6장 결론

본 논문에서 구현한 OHT 최적 제어관리 시스템의 OCS는 MCS로부터 물류 이송에 관한 정보를 받아 OHT가 최적의 경로를 경유하여 물류를 이송하기 위한 최적 경로 계산 알고리즘을 기반으로 설계 및 구현되었다. 본 논문에서는 제안된 알고리즘에 대한 타당성을 검증하기 위하여 컴퓨터상에서 이를 시뮬레이션하였다. 본 시스템에서는 제안된 알고리즘을 기반으로 구동하는 OHT를 제작하여 개별 구동실험 및 무선 통신 시스템과 연동한 통합적인 실험을 수행하였다. 이때 OHT 운전에 관한 세미 스텐다드 시나리오(SEMI-standard scenario)를 분석하였으며 이를 기반으로 OHT를 구동 실험을 수행하여 본 논문에서 OHT 최적 제어관리를 위하여 설계 및 구현한 전체 시스템 구성 및 통신구조, 통신 프로토콜(Protocol), OHT 구동구조에 대하여 그 타당성을 검증하고자 하였다. 실험을 통하여 다음과 같은 결론을 얻었다.

1. OHT가 최적 경로를 경유하여 물류를 이송하기 위한 알고리즘을 제안 하였으며 소프트웨어적으로 이를 구현하여 시뮬레이션을 수행하였으며 그 결과, 만족스러운 결과를 얻었다.

2. OHT가 알고리즘에 의해서 계산된 최적 경로를 주행할 때 발생할 수 있는 충돌 및 분기점에서의 정체 현상을 방지하기 위한 로직(logic)을 설계하였으며 소프트웨어적으로 이를 구현하였으며 제안된 알고리즘과 연동하여 시뮬레이션을 수행하였다. 그 결과, 만족스러운 결과를 도출할 수 있었다.

3. 실제 반도체 공정에서 사용될 OHT의 필수적인 운전 조건을 구현하기 위하여 OHT 운전에 관한 세미 스텐다드 시나리오(SEMI-standard scenario)를 분석하였으며 이를 기반으로 통신 및 OHT 구동에 관한 상세적인 구조를 설계하였고 구현하였다. 본 설계의 타당성은 세미 스텐다드 시나리오(SEMI-standard scenario)를 기반으로 하는 OHT 구동 및 통신에 대한 통합실험 플로우(Flow)를 작성하여 이를 수행함으로써 검증하였다.

본 연구를 통하여 국내에서 아직 개발 및 상품화 사례가 거의 없는 OHT를 이용한 반도체 물류 시스템 분야에 대한 국산화 기술을 확보하는 기반을 마련하였다.

향후 과제로는 실제 반도체 공정에 본 시스템을 적용시켜 대다수의 OHT를 제어 관리하는 실제적인 통합 실험을 수행함으로써 보다 견실한 신뢰성을 확보하고 본 논문에서 제안한 알고리즘을 OHT에 직접 탑재함으로써 OCS에 집중된 시스템 부하를 적절히 분담하여 보다 유연적인 시스템을 구축하고자 한다. 또한 AGV와 같이 자율적으로 이동하며 물류를 이송하는 유사 시스템에 본 시스템을 적용시킬 수 있도록 시스템을 재구성하여 보다 범용성을 지니도록 하는 것을 과제로 남긴다.

참고문헌

(1) T. Yamashita, "Start of Automous Mobile Robots" Operation in Clean Room", IEEE/RSJ International Workshop on Intelligent Robotics and Systems pp512-519, 1989.

(2) O. Causse and J. L. Crowley, " Navigation with Constrains for an Autonomous Mobile Robot", Proc. of the 1994 IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 1899-1905, 1994.

(3) R.W. Hall, "The Fastest Path through a Network with Random Time-dependent Travel Times", Transportation Science, Vol. 20. No. 3, 1986.

(4) A.K. Ziliaskopoulos, H.S Mashmassani, " A Time-dependent Shortest Path Algorithm for Real-time Intelligent Vehicle/Highway Systems Applications", TRB, 1993.8.

(5) 변재욱, "Optimal AGV Management in an Automated Harbor using the Estimation Method of Future Gridlock and Collision" 울산 대학교 석사학위 논문, 2000.

(6) 손원익, "A Study on the Flow Path Design of Unidirectional AGV System" 한국 산업공학회지 19권, 2호 pp43～51, 1993.

(7) 노영식, "A Central Navigation Control for Large-Scale Free-Ranging AGV System" 울산 대학교 학술 논문집 25권, 2호, pp. 113～125, 1994.

(8) 송성훈, " Dynamic Shortest Path Algorithm for Real-Time Vehicle Route Decision" 홍익대학교 과학기술연구소 논문집 10권, 2호, pp 557～574, 1999.

(9) 이병훈, “ Development of the SECS Protocol between Equipment and Host in a Semiconductor Process" 명지대학교 산업기술 연구소 논문집 20권, 1호, pp 192～196, 2001.

(10) 오삼권, "The Implementation of SECS(SEMI Equipment Communication Standard) Module" 호서대학교 공업기술 연구소 논문집 17권, 1호, pp 231～241, 1998.

(11) 유정운, “A study on the AGV deadlock avoidance and collision-free algorithm by using graph-theoretic approach” 서울대학교 석사학위 논문, 1998.

(12) Lyn robison, "Visual C++6.0 Database Programming", SAMS, 1998.

(13) Write Stevens, "TCP/IP Illustrated, vol 2", Addison Wesley Publishing company, 2000.

(14) Steve Oualling "Practical C++ Programming", O'REILLY, 2000.

(15) 이상엽, “ Windows Programming bible", 영진출판사, 1999.

(16) 전병선, “Microsoft Visual C++6.0 MFC Programming" 삼양출판사, 1999.

(17) 강맹규, "네트워크와 알고리듬", 박영사, pp.107-121, 1991.